Ferramenta para correção automática de exercícios de programação em ambientes de aprendizagem, com plugin para Moodle

ARIEL MARTINI

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação ao Instituto de Matemática e Estatística da Universidade de São Paulo.

Desenvolvido com apoio financeiro da CAPES

Orientador: Prof. Dr. Carlos Hitoshi Morimoto

SÃO PAULO

2012

Resumo

Este trabalho descreve a pesquisa e o desenvolvimento de uma ferramenta para correção automática de exercícios de programação que foi integrada ao ambiente de aprendizagem Moodle, no formato de plugin.

Um dos principais problemas enfrentados por cursos de programação em geral é a dificuldade e demora na correção de exercícios, quando feita manualmente. A correção automática de exercícios, oferecida como um serviço online, estimula o aprendizado do aluno, pois facilita o acesso ao material com acompanhamento constante, permitindo que ele identifique e supere rapidamente suas dúvidas e assim obtenha um maior domínio sobre os tópicos.

A ferramenta foi projetada para ter boa usabilidade tanto para o professor quanto para os alunos. Também é fácil de instalar, sendo compatível com a grande e crescente base instalada do Moodle, disponível para qualquer pessoa com acesso à internet, facilmente configurável, rápida e segura.

Nesse trabalho apresentamos o resultado de dois ciclos de desenvolvimento. O primeiro foi aplicado a uma turma de uma disciplina de introdução à computação no IME-USP, onde conseguiu atender todos os tópicos do curso e seus exercícios específicos, e serviu de base para as melhorias introduzidas no segundo ciclo.

Abstract

This work describes the research and development of a tool for automatic grading of programming exercises, which has been integrated to the learning environment Moodle, as a plugin.

One of the main problems faced by programming courses are the lengthy and difficult assessment of exercises, when done manually. The automatic assessment of exercises, offered as an online service, encourages student learning, as it facilitates access to material with constant monitoring, allowing him to quickly identify and overcome his doubts, thus achieving a greater mastering over the topics.

The tool has been designed to be easy to use and install, compatible with the large and growing Moodle installed base, available to any person with internet access, easily configurable, fast and safe.

In this work we present the result of two cycles of development. The first has been applied to a programming introduction course in IME-USP, where it managed to address all course topics and corresponding exercises, and served as a base to the improvements introduced in the second cycle.

Sumário

1 Introdução

Uma estratégia para incentivar os alunos a manterem o interesse em uma disciplina é fornecer listas de exercícios a serem resolvidos constantemente. Avaliar o aluno é fundamental para que ele identifique e possa dar enfoque apropriado às áreas que tem maior dificuldade [Foubister et al. 1997]. E isso precisa ser feito periodicamente para fixar os conceitos, principalmente para os alunos mais fracos e pouco confiantes.

Mas se a correção dos exercícios for manual, cria-se uma carga de trabalho adicional para o professor ou monitor. Um trabalho cansativo e repetitivo, que toma tempo que poderia ser dirigido para outras tarefas, como assistência a alunos e preparação de material didático. E se o número de alunos for muito grande, a falta de recursos pode tornar inviável a aplicação de exercícios em volume adequado. Sempre é possível incentivar os alunos a fazerem os exercícios voluntariamente, mas isso é menos vantajoso para o resultado do curso que exercícios obrigatórios [Saikkonen et al. 2001].

Felizmente, como os exercícios de programação são originalmente feitos para serem interpretados e executados por uma máquina, corrigi-los de maneira automática não requer um mecanismo complexo. Assim, é possível economizar recursos humanos e ao mesmo tempo deixar mais material disponível.

Outra vantagem da correção automática é que o aluno recebe o resultado instantaneamente, oposto ao caso da correção manual, quando a resposta pode demorar vários dias. Passado esse tempo, o aluno pode ter se esquecido de como resolveu o exercício, e as recomendações do avaliador podem não ser plenamente aproveitadas.

Uma reclamação frequente da correção manual é que é possível surgir diferentes critérios de avaliação, principalmente se a tarefa for distribuída para ser feita por pessoas diferentes. Devido ao caráter repetitivo do trabalho, mesmo um único avaliador pode

corrigir um programa de maneiras diferentes, dependendo do humor, nível de atenção e outros fatores humanos. A correção automática elimina esse fator, aplicando sempre o mesmo critério a todos os alunos.

Também é possível que um programa funcione bem para um conjunto de testes e não funcione para outro. Assim, se for testado manualmente, pode ocorrer de serem empregados apenas um desses conjuntos de testes, gerando um resultado que não reflete a realidade. Com a avaliação automática, podemos criar um conjunto mais completo e abrangente de testes, evitando essa situação.

1.1 Dificuldades da avaliação automática

Apesar das muitas vantagens, é preciso notar que a avaliação automática também possui algumas desvantagens. A maior delas é que é muito difícil avaliar aspectos qualitativos como mantenibilidade, comentários, adequação dos nomes das variáveis, endentação e modularidade.

É possível criar métricas que estimam a qualidade destes fatores, mas é difícil serem perfeitas, pois são bastante subjetivas. Essas métricas dependem do paradigma, mas algumas que podem ser avaliadas são: número de caracteres por linha, proporção de linhas de comentários, proporção de endentação, proporção de linhas em branco, espaços por linha, tamanho médio do módulo, número de palavras reservadas, tamanho dos identificadores, uso de GOTO, número de arquivos *include*, proporção de *define*, entre outros [Jackson 2000].

Porém, quando a correção é manual, como esses aspectos de mantenibilidade são mais fáceis de serem observados pelo avaliador, podem acabar tendo mais peso na nota final que a eficiência e corretude do programa em si. Isso gera um efeito colateral onde os alunos acabam gastando muito tempo em fatores acessórios, como interface ou mensagens amigáveis, uma vez que eles nunca têm certeza se fizeram o suficiente para conquistar a nota máxima [Cheang et al. 2003].

Outro fator que pode passar despercebido é se alguma técnica que havia sido solicitada no enunciado não foi empregada. Por exemplo, se for solicitado que se imprima dez caracteres utilizando um loop, alguém poderia resolver com dez comandos consecutivos. Ou também podem ser empregadas técnicas escusas de programação, como construções

difíceis de serem compreendidas, efeitos colaterais, invasão de memória, entre outros. Esses aspectos não impedem que o programa funcione corretamente aos olhos do avaliador automático, mas certamente devem ser considerados na avaliação.

Uma maneira de verificar se o programa possui estrutura semântica adequada é realizando uma avaliação estática, normalizando e comparando com uma ou mais estruturas modelo [Wang, T. et al. 2011]. Essa análise também pode gratificar programas que, mesmo não executando adequadamente, possuam algum valor na sua estrutura. Isso pode ajudar programadores novatos a terem mais confiança.

Mais um problema pode surgir na comparação do resultado retornado pelo programa do aluno com os valores corretos estipulados pelo criador do exercício. Se for uma simples comparação caractere a caractere, um aluno que modifique minimamente a saída pode ter o resultado anulado, mesmo se aos olhos de um humano o resultado estivesse correto.

Para tentar amenizar esse problema podemos especificar o tipo do resultado. Se for numérico, convertemos o resultado para número antes de comparar. Isso evita erros como diferença de precisão, número de casas decimais, zeros à esquerda, caracteres de espaço ou quebra de linha. Podemos também trabalhar com uma comparação mais abrangente, eliminando espaços e ignorando maiúsculas e minúsculas. Também é possível utilizar casamento de padrões ou expressões regulares, onde podemos especificar variações da saída que não alterem o resultado.

Outra questão difícil de ser resolvida é o caso de trabalhos em grupo. Porém, a avaliação automática é mais indicada para exercícios e problemas de pouca complexidade e pequeno porte, que seriam aplicados em grande número, para cursos introdutórios. Os trabalhos em grupo, apesar de mais complexos, são em menor quantidade. Assim, sua correção não gera uma carga de trabalho repetitiva para os monitores. Portanto, o benefício de automatizar essa correção não seria tão grande.

Por fim, é importante que a resposta a qualquer tipo de erro seja bastante clara e compreensível. Alguns erros de compilação podem ser de interpretação bastante difícil por programadores novatos. Assim, é importante identificar as maiores dificuldades, suas soluções, e incorporar estas informações no relatório do corretor automático.

1.2 Motivações

O uso do corretor automático reduz a carga de trabalho dos professores e monitores, liberando-os para outras atividades, como prestar assistência presencial aos alunos que têm mais dificuldade, ou elaborar novos exercícios.

Também possibilita trabalhar com um número maior de exercícios de diferentes graus de dificuldade, contemplando alunos de diferentes níveis. É possível deixar os exercícios disponíveis não só para os alunos do curso, mas para toda comunidade acadêmica, ou mesmo para o público em geral, se for desejado.

A correção automática fornece um resultado coerente e instantâneo, livre de variações ou favorecimentos, e diferentemente da correção manual, não pode ser influenciada pelo aspecto estético. É possível reenviar o programa para correção quantas vezes for preciso. Se o aluno precisar depurar o programa, os dados para teste utilizados na correção automática são ótimos pontos de partida.

O aluno iniciante não precisa lidar com configuração de parâmetros do compilador e sistema operacional, o sistema está pronto para compilar e avaliar programas instantaneamente. Além disso, o relatório do corretor é claro e abrangente, facilitando a identificação de eventuais problemas.

Por fim, o sistema é projetado para ser seguro, rápido e robusto, podendo atender muitas requisições simultâneas. É resistente a código malicioso graças ao encapsulamento da máquina virtual, e se utiliza de ferramentas gratuitas ativamente desenvolvidas, amplamente testadas e seguras.

1.3 Descrição do trabalho

Na Universidade de São Paulo, USP, mais de dois mil alunos cursam as disciplinas de introdução à computação todos os anos. Este número tende somente a crescer, pelo aumento natural de vagas disponíveis na universidade ou pelo alcance crescente do ensino de programação a novos cursos. Com três ou quatro exercícios a serem entregues por cada aluno ao longo do curso, um imenso volume de trabalho é gerado. Até 2007, a correção destes exercícios era feita manualmente, por professores e monitores. E, como já dito, é sujeita a erros, podendo não ser perfeitamente uniforme ou abranger todos os

testes necessários. Além disso, os resultados retornavam aos alunos apenas um mês depois da entrega.

Para facilitar este processo, alguns professores se valiam de *scripts* feitos por eles mesmos, que automatizam algumas tarefas. Mesmo assim, grande parte do trabalho permanecia manual. Como será descrito na seção 3, corretores automáticos já existem desde a década de 1960. Apesar disso, nenhum dos corretores existentes em 2007 atendia plenamente as necessidades do IME-USP. Alguns deles eram voltados para competições de programação, como os juízes online, e focam apenas no resultado do programa, certo ou errado, o que não é o mais indicado a alunos iniciantes que não conseguem compreender os motivos dos erros. Alguns corretores são também desvinculados dos ambientes de aprendizagem, hoje em dia representados pelos LMS (*Learning Management System*) que são sistemas de gerenciamento de cursos acessíveis pela internet, dentre os quais o Moodle e o Sakai são utilizados na USP.

Assim, com o apoio do projeto TIDIA-Ae da Fapesp, foi desenvolvida uma ferramenta para correção automática de exercícios de código aberto. Esta ferramenta foi focada na disciplina de introdução à computação, pois é a que possui maior número de alunos, e consequentemente o maior volume de trabalho. Exercícios para todos os tópicos dessa disciplina deveriam ser suportados, mas a ferramenta também deveria ser expansível, podendo contemplar exercícios para outras disciplinas.

Consultando os professores, foi decidido que a ferramenta não deveria ser usada apenas em laboratório, e sim estar disponível diretamente para os alunos. Para isso, deveria ser acessível de qualquer lugar, através da internet. Isso também poderia atender os alunos de ensino à distância. Pelo número de alunos envolvidos, deveria suportar muitos acessos simultâneos, e mesmo assim ter resposta rápida.

Por causa disso, os LMS se mostraram o ambiente ideal para localizar a interface da ferramenta. Eles fornecem boa parte da infraestrutura necessária, e já contêm módulos para distribuição de atividades, que podem ser estendidos através de plugins. Também já são amplamente utilizados em meio acadêmico.

Como inicialmente o projeto era apoiado pelo TIDIA-Ae, em um primeiro momento a ferramenta foi desenvolvida para o Sakai, que era o LMS escolhido pelo projeto. Em um segundo momento, foi escolhido o LMS Moodle, que possui mais de 72 mil instalações ao

redor do mundo[1], inclusive no IME-USP, e mais de 50 milhões de usuários cadastrados. Estes dois LMS são gratuitos e de código aberto.

Apesar do Sakai ser uma ferramenta promissora, o Moodle é mais maduro, mais ativamente desenvolvido, possui maior base instalada e melhor documentação. Porém o principal fator para sua escolha foi seu legado no IME-USP, onde vem sendo usado há mais de dez anos, e uma grande base de conhecimento dos cursos anteriores já está presente.

Nossa ferramenta deverá estar disponível para qualquer pessoa que queira utilizá-la, em repositórios de plugins na internet, e ser de fácil instalação e configuração.

1.3.1 Busca por uma melhor usabilidade

Para a integração com o LMS foi projetado um plugin com foco na usabilidade (ISO 13407) . Os usuários são os alunos, resolvendo os exercícios, e os professores ou monitores, criando exercícios e revisando-os. Os requisitos a seguir foram levantados.

Ao resolver um exercício, o aluno precisa ter o domínio sobre a ferramenta. A interface deve ser simples, o relatório deve retornar rapidamente e ser de fácil compreensão, e a operação deve focar na resolução do exercício, com o mínimo de distrações possível. Para reduzir o número de ações necessárias para enviar um programa para correção, um editor de texto com realce de sintaxe foi incorporado ao plugin.

O aluno pode enviar o código quantas vezes julgar necessário para ser corrigido, até estar satisfeito com o resultado, ou quando o prazo de envio estipulado pelo professor terminar. É desnecessário restringir o número de vezes que o aluno pode enviar um código, até por que ele poderia editá-lo em outro ambiente para testes. E nosso objetivo é justamente fornecer um ambiente único, para facilitar a vida do estudante.

Para auxiliar os alunos iniciantes, os erros de compilação devem ser traduzidos e acompanhados de dicas para facilitar sua resolução.

Para o professor, a tela de acompanhamento também deve permitir visualizar todos os trabalhos enviados, as estatísticas e as notas atribuídas automaticamente, tudo em uma

[1] http://moodle.org/stats/

única tela. Deve ser possível analisar o conteúdo de cada exercício, acrescentar comentários e alterar a nota.

O professor ou monitor pode revisar os exercícios antes das notas serem consideradas finais, pois a correção automática pode não ser perfeita. Pequenos erros podem fazer com que o corretor atribua nota mínima ao programa do aluno, por exemplo, se a saída tiver conteúdo certo, mas formato incorreto. Por outro lado, um programa mal escrito e de baixa mantenibilidade, pode produzir os resultados esperados pelo corretor e obter nota máxima.

A confecção de um exercício é composta por duas partes: o enunciado e a configuração. Deve ser possível digitar o enunciado em um editor de hipertexto no próprio navegador e, se for necessário, usando estilos de formatação e figuras. Como o exercício será compilado automaticamente, não é necessário aos alunos, principalmente aos iniciantes, lidarem com configuração de compilador e parâmetros de linha de comando. O aluno precisa apenas saber como escrever um programa que seja compilado com sucesso.

A configuração de cada exercício é feita em um arquivo XML, e contém parâmetros como linguagem de programação alvo, tempo máximo de execução, além da definição dos diferentes testes que serão aplicados. Cada teste tem um nome, uma nota máxima, os dados de entrada e os dados de saída esperados. A nota máxima é dada pela soma das notas de todos os testes.

No relatório cada teste pode ser visível, secreto ou parcialmente visível, onde é possível esconder a saída esperada ou o resultado. Assim, podemos ter tanto testes visíveis pelo aluno, que podem ajudá-lo a desenvolver o programa, como testes secretos, que servem apenas para avaliá-lo e compor a nota final.

É importante que o professor monte um conjunto de testes bastante completo, prevendo o maior número de situações adversas possível, e também torne alguns testes invisíveis, evitando o uso de técnicas não permitidas para alcançar os resultados.

Mais sobre a interface na seção 3.1.5.

1.3.2 Corretor

Para efetivamente compilar a executar o código, o plugin se comunica com um programa que chamamos de Corretor. Ele foi desenvolvido em Java, orientado a objetos, visando à portabilidade e a expansibilidade. Ele recebe o programa do aluno e a configuração XML

do exercício, e retorna um relatório de erros. Para poder testar exercícios relativos ao curso de introdução à programação, ele deve suportar a entrada e saída de números inteiros, números reais e strings. Este tratamento deve ser feito de maneira a ignorar elementos como textos explicativos, e isolar apenas as respostas, a fim de compará-las com o gabarito.

1.3.3 Segurança

Ao serem testados, os exercícios recebidos são efetivamente executados pelo corretor. Isto abre espaço para um cenário onde códigos maliciosos poderiam ser introduzidos no ambiente. A fim de evitar esta situação deve ser utilizada uma arquitetura onde o exercício é executado em uma máquina virtual, independente do servidor onde o LMS está sendo executado.

Além disso, esta máquina virtual não deve permitir acesso universal a partir da web, como Telnet ou SSH. Ela deve ser acessada através requisições com tratamento específico e sintaxe exclusiva ao corretor.

Por se tratar de pontos de entrada para potenciais invasores, as tecnologias utilizadas devem ser maduras, constantemente atualizadas, e terem uma grande base de desenvolvedores ativos. Felizmente é possível encontrar várias dessas tecnologias que também são gratuitas.

1.4 Contribuições

Como contribuição, este trabalho deixa o sistema desenvolvido, pronto para ser utilizado nos cursos introdutórios de computação do IME-USP. Da mesma maneira, poderia ser usado por qualquer curso de introdução à computação ministrado em C ou Java, ou outros cursos de programação mais avançados, trazendo a eles os benefícios da correção automática. Por ser baseado no Moodle, é facilmente integrado a qualquer das centenas de ambientes educacionais que já utilizam este sistema. De qualquer maneira, o Moodle é de fácil instalação e utiliza poucos recursos.

A ferramenta também pode ser estendida e modificada para atender outras necessidades do IME-USP. Por ser de código aberto, também pode ser modificada por terceiros, por exemplo, incorporando suporta a outras linguagens de programação.

Este trabalho traz uma síntese do estado da arte em corretores automáticos. E dele serão originados artigos disponíveis para a comunidade científica.

1.5 Organização do trabalho

No Capítulo 2 será detalhado o histórico dos corretores automáticos de programas, desde os primeiros corretores em lote, na década de 1960, passando pelos corretores interativos, até os mais atuais, com interface web. Também serão mencionados os trabalhos com juízes online e detectores de plágio. Por fim, será descrito o estado da arte nos ambientes de aprendizado LMS, dando foco ao Moodle.

No Capítulo 3, a metodologia utilizada para projetar e desenvolver a ferramenta será descrita, e as decisões tomadas para seguir as linhas guia já definidas. Também será explicado os componentes da ferramenta: o plugin, o corretor e a máquina virtual.

No Capítulo 4 serão descritos os testes utilizados, boas práticas para a criação de testes pra um curso de introdução à programação, e alguns resultados.

2 Trabalhos correlatos

Nesta seção listaremos alguns corretores automáticos de programas já existentes, desde os primeiros, desenvolvidos na década de 60, até os atuais, com interface Web. Também falaremos sobre detectores de plágio e os editores, compiladores e juízes online, que estão relacionados ao trabalho. Depois descreveremos o que são, como evoluíram e para que os ambientes LMS são usados. Daremos foco ao Moodle, citando suas características, vantagens e desvantagens.

2.1 Corretores automáticos de programas

2.1.1 Primeira fase: Cartões perfurados

Figura 1 - Um computador IBM 650 e sua operadora com os cartões perfurados [Computer-history.info 2011]

Encontramos na literatura exemplos de avaliação automática praticamente desde o momento que os educadores passaram a pedir que os alunos desenvolvessem programas. Um dos registros mais antigos data de 1960 [Hollingsworth 1960]. Este artigo inicia dizendo que a correção de vinte programas levou *apenas* cinco minutos. Em linguagem *assembly*, a mais próxima da linguagem de máquina, os programas entravam no computador *IBM 650* através de cartões perfurados (*figura 1*). Os resultados possíveis eram apenas dois: *resposta errada* ou *programa completo*.

Já naquela época o autor acreditava que usar o avaliador era mais eficiente para o aprendizado, mais rápido e menos dispendioso que aulas de laboratório. Mesmo sendo um dos primeiros artigos sobre o assunto, nele já havia comentários que seria possível um aluno enviar algum código malicioso, desenvolvido especificamente para danificar o sistema de avaliação.

Durante a década de 1960 os avaliadores para entrada via fita perfurada na linguagem *Algol* foram aperfeiçoados, incorporando relatórios mais detalhados e maior variedade de exercícios possíveis [Berry 1966; Braden & Perlis 1965; Forsythe & Wirth 1965; Naur 1964]. Em um sistema para a linguagem *PL/I,* era necessário que o próprio aluno invocasse funções do corretor para comparar os resultados [Temperly & Smith 1968].

2.1.2 Segunda fase: Processamento em lote

O *Basser Automatic Grading System*, ou *BAGS*, para o computador *KDF9*, já refletia o amadurecimento dos sistemas operacionais [Hext & Winings 1969]. Agora, a saída do programa era comparada com dados previamente estipulados, e um relatório detalhado era gerado com o resultado. Já nessa época surgia a ideia de tentar verificar a existência de plágio automaticamente, comparando tempo de execução e tamanho do programa, mas ainda não havia sido colocada em prática. O *BAGS* usava *Algol*, *Minigol* e *assembly*.

Mais tarde, em 1984, o sistema *Give* já rodava sob o sistema operacional *Unix* [Carrington et al. 1984]. Nele os programas eram testados com dados fornecidos e os resultados avaliados sob um processo definido pelo instrutor. Outro avaliador automático, também para *Unix*, era um *script* que varria um conjunto de diretórios procurando por arquivos a serem compilados em linguagem *C*, executava-os e comparava os resulta-

dos com um conjunto pré-determinado [Isaacson & Scott 1989]. Este processamento em lote era bem mais simples que os complexos programas da década de 60.

2.1.3 Terceira fase: Programas interativos

Desenvolvido no Rochester Institute of Technology, o sistema *TRY* podia ser usado pelos próprios alunos para testarem seus programas [Reek 1989]. Isso possibilitava o retorno imediato do resultado da execução. Até então os sistemas de avaliação eram sempre executados pelos professores.

No projeto *Kassandra,* introduzido em 1993 na Universidade ETH de Zurique [Matt 1994], o teste dos programas podia ser feito em um sistema remoto através de soquete de *internet*. Isso permitia o isolamento entre o gerenciador de atividades e o sistema de testes, aumentando a segurança.

Na década de 1990 alguns corretores contemplavam a linguagem Pascal, que era bastante usada para ensino de programação. O sistema *SPROUT* comparava resultados de programas escritos no *Turbo Pascal* para *MS DOS* [Pardoe & Vickers 1994], enquanto o *Pascal Trainer* era um programa escrito em *C++* para *DOS* que analisava respostas para perguntas específicas usando código escrito em *Pascal* [Webber 1996]. Ele não tratava de programas completos, e sim pequenos trechos de código que deveriam ser digitados pelo aluno.

Em 1996 o projeto *ASSIST* da Universidade de Liverpool foi introduzido [Jackson & Usher 1997]. Desenvolvido para o sistema de janelas *X Window*, permitia combinar o resultado dos testes de corretude com métricas de eficiência, estilo, complexidade e adequação para montar a nota final. Assim, para verificar o estilo e a complexidade, um cálculo era feito sobre variáveis como tamanho do módulo, número de comentários e aplicação de endentação.

2.1.4 Quarta fase: A web

Um dos primeiros sistemas a usar interface web foi o *BOSS*, da Universidade de Warwick [Joy & Luck 1998]. Ele também já permitia a correção de programas em linguagem *Java*, além da mais tradicional *C*. Ele possuía um servidor web foi integrado, mas apenas a parte utilizada pelos professores para revisão dos trabalhos era acessível por um navegador. A parte utilizada pelos alunos era através de uma interface gráfica.

Eventualmente o BOSS foi evoluído para trabalhar com o paradigma de testes JUnit [Beck 2003]. Assim era possível testar também aspectos dos objetos internos ao programa. Além disso, era possível avaliar conceitos de desenvolvimento de programas orientado a testes.

O *Ceilidh*, dos meados dos anos 1980, foi a base para o *CourseMaker*, desenvolvido uma década depois [Foubister et al. 1997; Higgins et al. 2003]. O *CourseMaker* possuía interface em X Window, Java ou web, um grande avanço sobre a interface em linha de comando que era usada pelo *Ceilidh*. Como outras ferramentas de avaliação automática, ele dava suporte ao desenvolvimento, execução e administração de exercícios em diversas linguagens de programação. Porém também possuía características de gerenciamento de conteúdo, como a estruturação de cursos em capítulos e exercícios associados, como os LMS. Permitia que os monitores acompanhassem todas as etapas de avaliação de um aluno, e possuía funções para distribuição de textos e administração de notas.

Nos anos 2000 outros trabalhos também se utilizaram da interface web, como o *Scheme-Robo,* que usava a linguagem funcional *Scheme* [Korhonen & Malmi 2000; Saikkonen et al. 2001], e o *RoboProf,* implementado como um servlet Java [Daly & Waldron 2004; Daly 1999].

2.1.5 Trabalhos paralelos

Um sistema desenvolvido na Universidade de Liverpool, em 2000, com interface em linha de comando, partia do princípio que a avaliação poderia ser automatizada, mas seria melhor com a supervisão de uma pessoa [Jackson 2000]. O aplicativo compilava e testava o programa enviado pelo aluno, além de aplicar métricas de qualidade automáticas. Mas também permitia ao operador qualificar outros aspectos como a qualidade dos comentários, avaliar eventuais falhas nos testes como formatação incorreta da saída, e comentar a qualidade do código em si, como por exemplo se foram empregados os conceitos requisitados no enunciado ou se continha técnicas inadequadas.

O Homework Generation and Grading (HoGG) foi utilizado desde 2001 na Rutgers University [Morris 2003]. Seu núcleo era implementado em Java e trabalhava em três etapas. Na primeira, o módulo *driver* carregava e executava o programa do aluno na mesma máquina virtual em que o avaliador estava sendo executado. Em seguida, o módulo *evaluator*, escrito em Perl, se utilizava de expressões regulares para comparar

os resultados. Por fim, o módulo repórter enviava os emails para os alunos e professores com os resultados.

O sistema JEWL é um arcabouço para programas com interface gráfica em *Java*, e também pode ser usado em um ambiente de avaliação automática [English 2004]. Ao invés de trabalhar com eventos, o programa se baseia em um loop explícito onde as mensagens recebidas pela interface gráfica são tratadas uma a uma.

Alguns avaliadores se utilizam de uma análise estática do programa, ou seja, não o executam [Rahman et al. 2007; Truong et al. 2005; Zamin et al. 2006]. Eles funcionam analisando a estrutura para transformar o código enviado pelo aluno em um pseudo código e posteriormente compará-lo com uma ou mais soluções possíveis enviadas pelo professor. Este método mostra claras desvantagens, como não contemplar uma possível solução correta não imaginada pelo professor, e não realizar testes de entrada e saída.

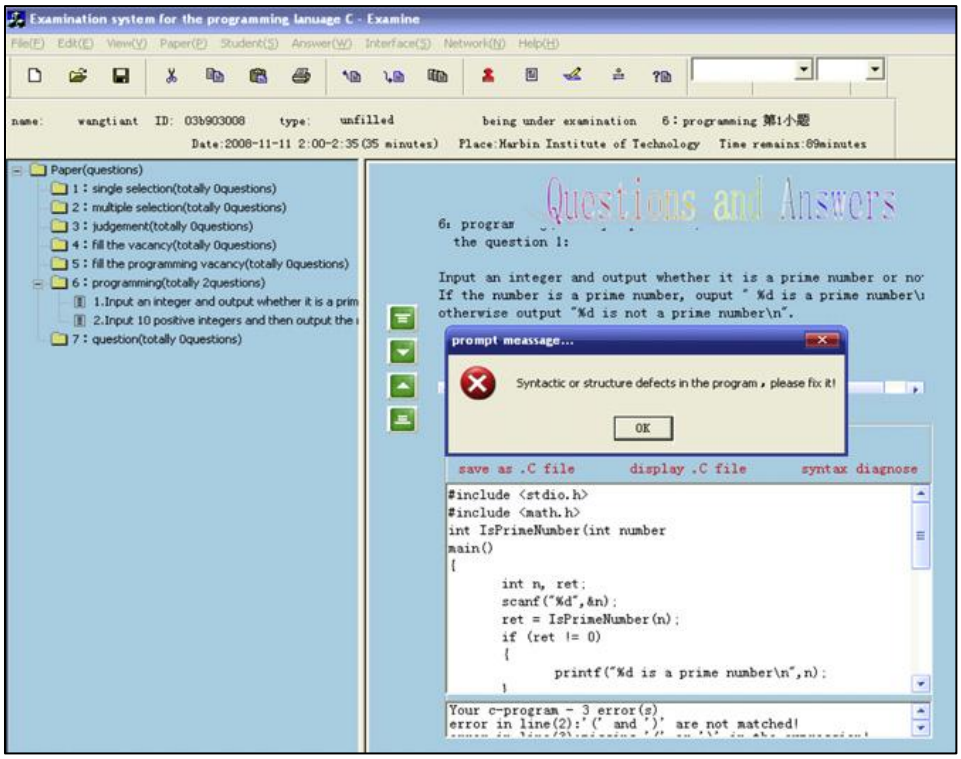

Figura 2 - Mensagem de erro na interface do AutoLEP [Wang, T. et al. 2011]

O *AutoLEP*, uma solução desenvolvida no Harbin IT, na China, incorpora tanto a análise dinâmica como a estática para tentar realizar uma avaliação mais completa do programa [Wang, T. et al. 2011]. Na análise estática, o código é normalizado, retirando partes redundantes, antes de se comparar com os modelos de referência. No caso do

código não se encaixar em nenhum, e se o avaliador julgar que está correto, é possível adicioná-lo como mais um modelo de referência. Este programa, porém, é desenhado para uso em aulas de laboratório, possui interface desenvolvida em C++ (*figura 2*) e não é acessível remotamente.

2.1.6 Juízes online

O nome juiz online, ou *online judge,* surgiu em 1999 na Universidade de Cingapura [Cheang et al. 2001, 2003]. Inicialmente utilizado para um curso de programação competitiva, este programa recebia os programas via email e exibia os relatórios da compilação, execução, corretude e desempenho em uma página web.

Mais tarde, esse nome se tornou sinônimo para os sistemas online usados durante competições de programação, e para praticar para essas competições. Para cada problema existe uma classificação das respostas, que foram testadas não só na corretude, comparando os resultados com dados pré-estabelecidos, mas também com relação ao tempo de execução e utilização de recursos.

Existem vários sistemas desse tipo disponíveis na web, normalmente desenvolvidos e mantidos por universidades, como o UVa Online Judge[2], desenvolvido na Universidade de Valladolid em 2005 [Revilla et al. 2008], ou o Sphere Online Judge (SPOJ)[3], da Universidade de Gdansk [Kosowski et al. 2008, 2005]. O SPOJ possui mais de dez mil problemas cadastrados e mais de cem mil usuários. A qualquer momento um usuário pode escolher um problema e tentar resolvê-lo, e sua solução é classificada junto às soluções de outros usuários.

Além dos sistemas disponíveis na web, também existem online judges que podem ser instalados localmente, como o PC, PKU JudgeOnline, DOMJudge e Mooshak [Georgouli & Guerreiro 2011].

Os juízes online mantêm muitas semelhanças com o corretor. Porém, apesar de poderem ser usados no ensino de programação, eles foram desenvolvidos tendo em vista as competições. Portanto, algumas características que consideramos importantes nem

---

[2] http://uva.onlinejudge.org

[3] http://spoj.pl

sempre estão presentes, como integração com livro de notas ou uma descrição mais didática dos códigos de erro.

2.1.7 Quinta fase: Integração com um LMS

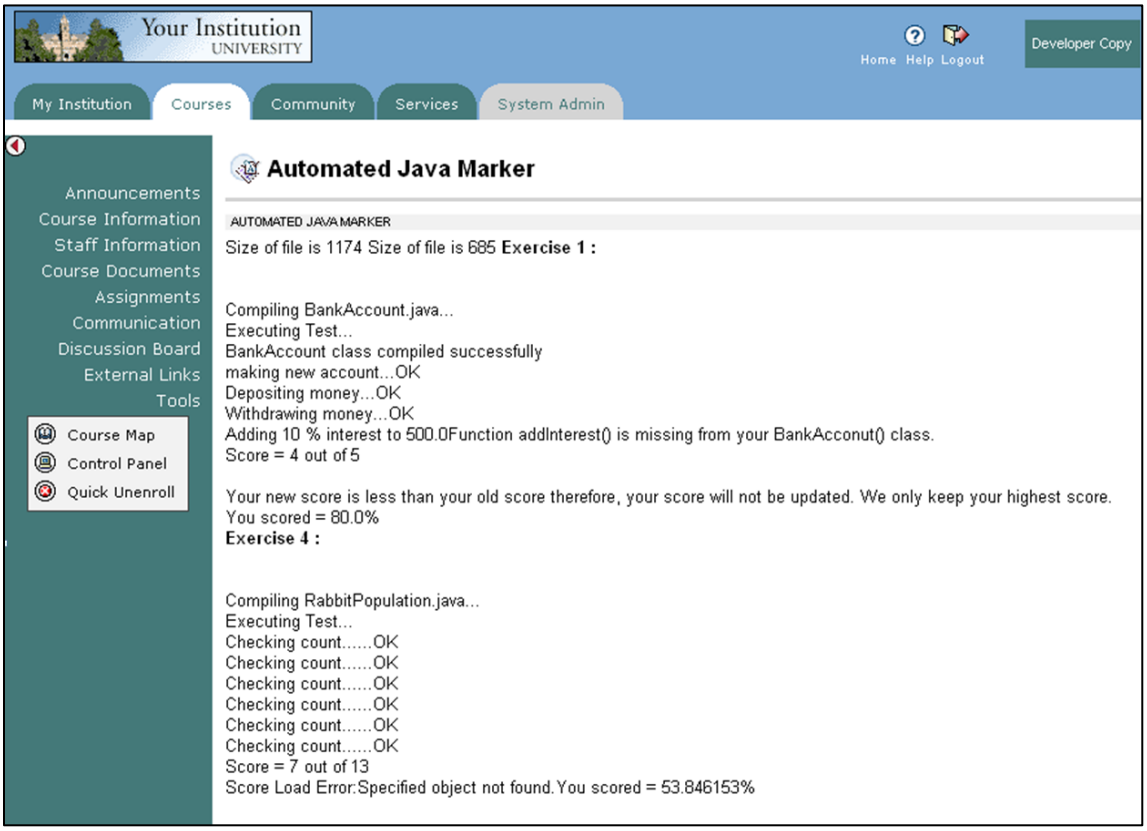

Figura 3 - Tela do ASAP rodando sob o Blackboard mostra o relatório de uma avaliação automática de um exercício [Douce et al. 2005]

Desenvolvido na Universidade de Kingston, o *ASAP, Automated System for the Assessment of Programming,* é um corretor automático modular que pode ser agregado a um LMS (*ver 2.4*) [Douce et al. 2005]. Desenvolvido para trabalhar com a linguagem *Java,* além de verificar a saída do programa, ele também testa funções individualmente através de testes de casos de uso. Inicialmente desenvolvido como um módulo para o LMS comercial *Blackboard*, teoricamente o *ASAP* poderia ser adaptado para outros LMS.

A correção do programa é feita separadamente, por um *WebService* que recebe o programa a ser testado e devolve um arquivo XML com o resultado. Para tentar identificar situações de plágio, o *ASAP* utiliza os serviços de um mecanismo detector externo chamado *JPlag*.

O *VPL, Virtual Programming Lab,* é uma ferramenta de código aberto para administrar exercícios de programação no ambiente LMS Moodle (*ver 2.4.3*) [Pino et al. 2010]. Desenvolvido na Universidade de Las Palmas de Gran Canaria a partir de 2009, ele é composto de três módulos: um plugin para o Moodle, um applet Java editor de código fonte e um servidor remoto que compila e executa os exercícios de maneira segura.

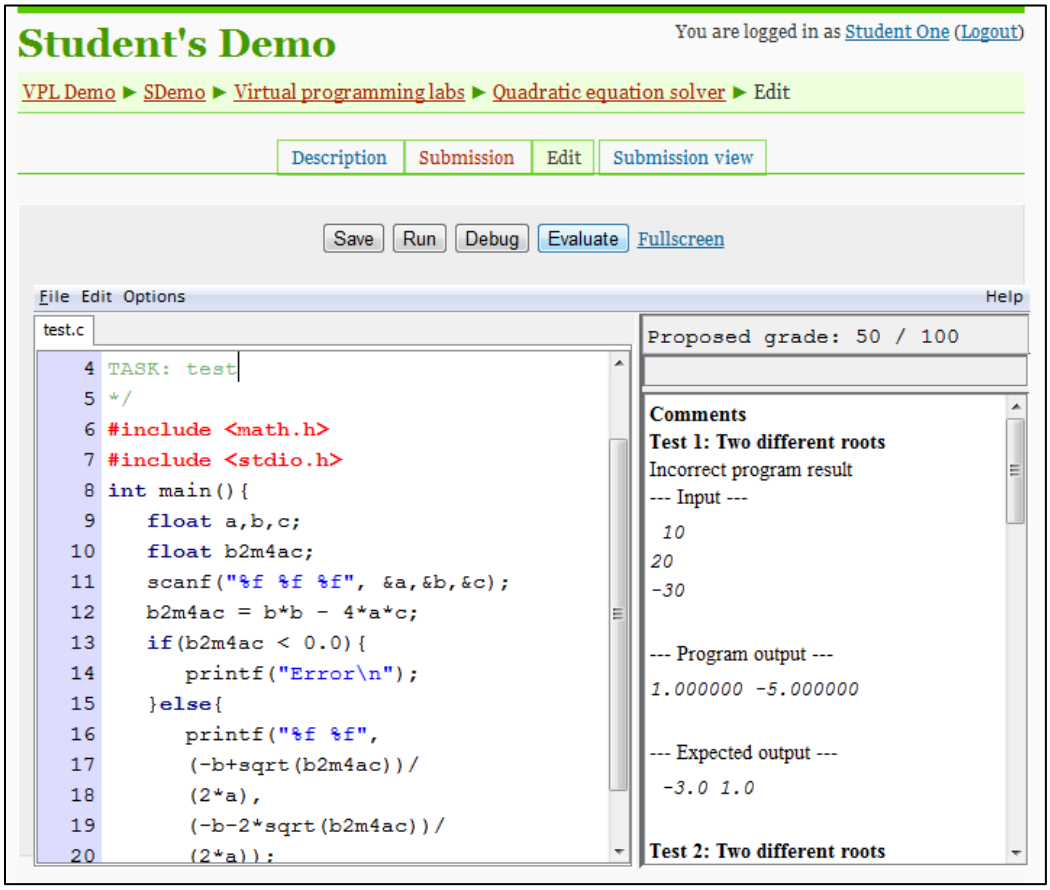

Figura 4 - Tela do VPL exibe o editor e o resultado de uma avaliação (http://vpl.dis.ulpgc.es)

A página principal para o aluno possui quatro abas: *description*, onde o aluno visualiza o enunciado; *submission,* onde é possível enviar um ou mais arquivos; *edit*, onde o applet Java pode ser usado para editar, depurar ou avaliar os arquivos; e *submission view*, onde são exibidos os arquivos enviados.

O applet editor permite realce de sintaxe e funções de edição simples como busca e substituição. Além disso, permite executar, depurar ou avaliar o programa (*Figura 4*). Ao executar, após a compilação, é exibido um console onde é possível interagir com o programa. Ao depurar, o depurador de linha de comando *gdb* [Stallman & Pesch 1991] é exibido em uma tela de console. E ao avaliar, o programa passa por testes de entrada e

saída, e um relatório é exibido na lateral da janela. Este relatório é armazenado para referência futura e a nota é armazenada como uma proposta de nota. Após a revisão de um professor ela pode ser transferida para o livro de notas.

Um projeto desenvolvido na Universidade do Algarve, em Portugal, integrou o online judge Mooshak no LMS Claroline, que são dois aplicativos gratuitos e de código aberto [Georgouli & Guerreiro 2011]. O Mooshak é escrito na linguagem TCL, e o Claroline PHP, e para a comunicação foi usado o comando curl do PHP, que obtém o conteúdo de uma página web.

Estes corretores integrados a LMS são os que mais se aproximam da ferramenta proposta neste trabalho. Mas nenhum deles possui todas as características que buscamos. O que mais se aproxima é o VPL. Porém além de ter uma interface complexa, com editor baseado em applet, que pode ser de difícil execução em alguns ambientes. O VPL também não possuía um sistema de correção automática funcional quando este trabalho foi iniciado.

2.2 Editores e compiladores online

No início de 1999 foi desenvolvido um aplicativo chamado *Intranet Compilers* que permitia compilar um programa em ambiente web. O usuário escolhia uma linguagem entre C, C++, Fortran, Java e Pascal, e um compilador entre GNU, Borland, Microsoft e Sun. O servidor era baseado em script CGI escrito em PERL [Malinowski & Wilamowski 2000]. Porém não ficou disponível para utilização pública.

Paralelamente surgiu o site *pastebin.com*, que existe pelo menos desde 2002[4]. Ele introduziu a ideia de site onde é possível compartilhar um texto com outras pessoas através de uma URL única gerada automaticamente. E o nome *pastebin* acabou virando sinônimo para este tipo de site. Devido à sua relação histórica com o protocolo de bate papo IRC, desde sua origem foi bastante usado para transmissão de trechos de código fonte. Assim, hoje podemos encontrar vários sites que permitem compilar e executar o conteúdo do *pastebin*, apesar do site original ainda não oferecer este tipo de suporte.

[4] http://techcrunch.com/2011/10/26/pastebin

De acordo com uma página na Wikipédia[5] que compara os *pastebins*, em 2011 existiam quarenta sites desse tipo. A grande maioria oferecia suporte a realce de sintaxe, ou seja, são apropriados para distribuição de trechos de código. E seis deles ofereciam a opção de interpretação ou compilação do código.

Podemos citar alguns nomes. O Ace[6], um editor Javascript de código aberto, usado internamente pelo Cloud9[7], um IDE online que permite interpretar Javascript e HTML/CSS. O CodeMirror[8] e o Ymacs[9] são outros editores de código escritos em javascript, com suporte a realce de sintaxe para diversas linguagens.

O Codepad[10] compila e executa treze linguagens diferentes, e permite compartilhar o código com outras pessoas. Porém não tem suporte a entrada de dados de teste, apenas saída. O jsFiddle[11] é um *pastebin* de código Javascript e HTML bastante simples, que permite executar o Javascript em tempo real e compartilhar código através de uma URL única gerada quando se salva o projeto.

O Kodingen[12] é um IDE online para aplicações web (PHP, Perl, Python, entre outras), assim como o CodeRun Studio[13] que também suporta ASP.Net, inclusive para depuração (*Figura 5*). Estes dois suportam projetos complexos e possuem editores variados, inclusive editores gráficos. Muitos trabalhos recentes propõem IDEs com interface web, como o *Adinda* [Deursen et al. 2010], o *WebIDE* [Janković & Gledec 2010], o *Collabode*, um IDE colaborativo para Java [Goldman & Little 2011] e o *Web Based IDE*, um IDE com foco em estrutura MVC [Palvai 2010]. E muitos outros devem surgir uma vez que é cada vez maior o uso de interface web para qualquer tipo de aplicativo.

---

[5] http://en.wikipedia.org/wiki/Comparison_of_pastebins

[6] http://ace.ajax.org

[7] http://cloud9ide.com

[8] http://codemirror.net

[9] http://ymacs.org

[10] http://codepad.org

[11] http://jsfiddle.net

[12] http://kodingen.com

[13] http://coderun.com

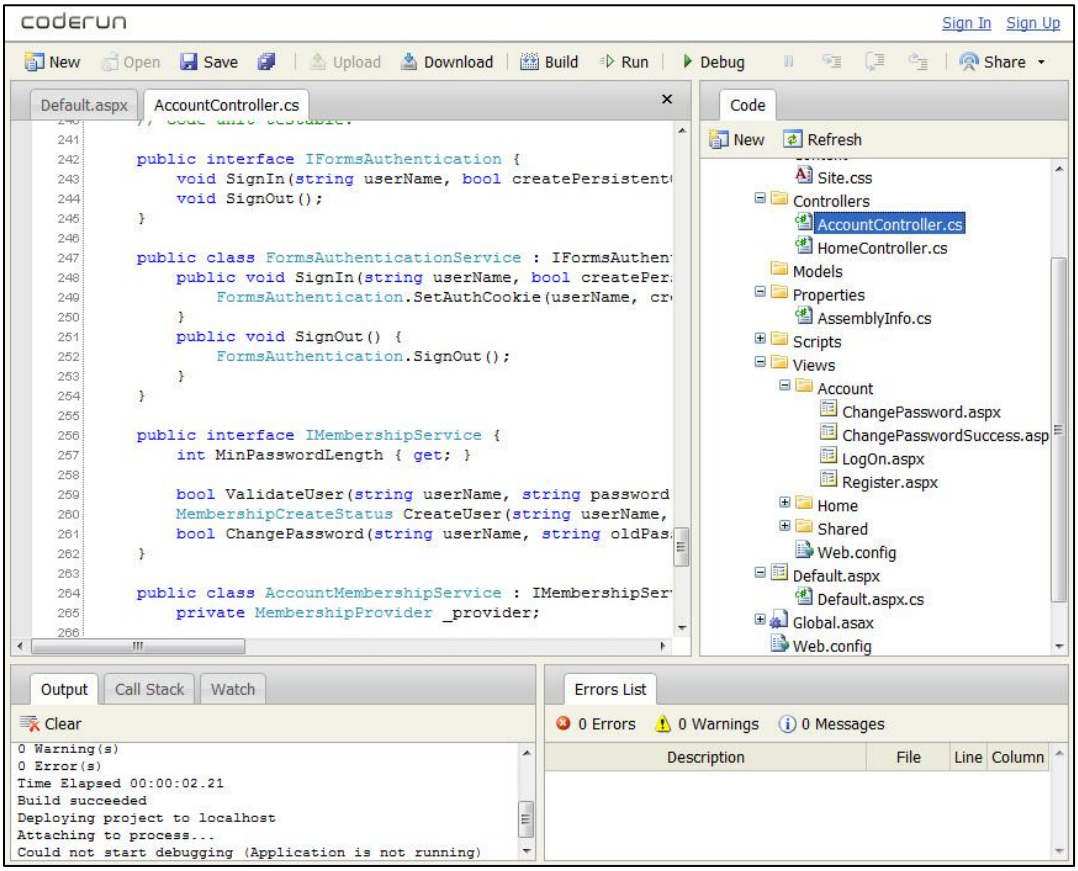

Figura 5 - Tela do Coderun Studio mostra a edição de um código fonte em C# dentro do browser (http://coderun.com)

O IdeOne[14] é um editor simples que permite compilar e executar o código em uma máquina virtual. Ele suporta mais de sessenta compiladores diferentes, e exibe o resultado no próprio navegador. Para testar o código é possível definir uma lista que será passada pelo *stdin* para o programa. O resultado inclui a saída do código, tempo de execução e memória utilizada. Também é possível compilar a executar o código em qualquer um dos compiladores disponíveis através de um WebService disponível gratuitamente, diretamente de outro site [Jianxia et al. 2011].

Estes editores online se relacionam com os corretores automáticos, mas são mais voltados para desenvolvimento do que ensino de programação. Apesar disso um destes editores, o Codemirror, foi aproveitado pelo nosso trabalho como base para o editor online de código.

[14] http://ideone.com

2.3 Detectores de plágio

O plágio como forma de apropriação de propriedade intelectual existe desde que o homem produziu trabalhos de arte e pesquisa [Lukashenko et al. 2007]. Pode ser definido como referenciar o trabalho de outra pessoa como se fosse seu próprio, sem a referência da fonte original.

No âmbito dos exercícios de programação, alguns professores podem ter a sensação que existam muitos casos de cópias de programas entre os alunos. Apesar disso, encontrar esses casos manualmente é uma tarefa extenuante [Prechelt et al. 2002]. Muitas vezes essa busca não é feita apropriadamente, e os casos só são identificados em casos extremos, quando um aluno se esquece de mudar o nome do autor original nos comentários de seu código, ou quando dois programas diferentes produzem os mesmos resultados não esperados.

Para facilitar este trabalho, existem diversas ferramentas que tentam encontrar casos de plágio automaticamente. Dentre os mais eficientes estão o YAP3 [Wise 1996], o Moss [Aiken 1998] e o Jplag [Prechelt et al. 2002]. Eles possuem pequenas diferenças de otimização e desempenho entre si, porém todos se baseiam no mesmo princípio, a comparação da estrutura dos programas. Para isso o detector transforma os programas em vetores de elementos atômicos (*tokens*), ignorando endentação, quebra de linha, comentários e outros elementos que podem ser alterados facilmente. Depois compara estes vetores entre si, buscando similaridades.

Esses detectores de plágio estão disponíveis para uso na internet, funcionando como serviços. Os arquivos a serem comparados são enviados pela interface web ou por um script, e o resultado pode ser visualizado online, através de um navegador. No relatório gerado, os programas são ordenados por similaridade, e podem ser comparados manualmente para averiguar o grau de similaridade.

A detecção automática de plágio é importante para nosso trabalho, porém ainda não foi totalmente integrada. Falamos mais sobre isso na seção 3.5.2, desenvolvimentos futuros.

2.4 Ferramentas LMS

LMS (*Learning Management System*), CMS (*Course Management System*) ou VLE (*Virtual Learning Environment*) são nomes diferentes para o mesmo tipo de software que oferece suporte ao ensino e aprendizado eletrônico [Botev et al. 2005].

A origem dos LMS foi basicamente para ser uma ferramenta de comunicação no meio acadêmico, porém hoje existe uma grande variedade de funções incorporadas, como divulgação de textos e ementas, avaliação, comunicação síncrona e assíncrona, envio e recebimento de conteúdo, organização de turmas, divulgação de notas, ferramentas colaborativas e até ambientes virtuais tridimensionais de convivência [Yueh 2008].

O ambiente onde é instalado, na maioria das vezes, é um portal acessível pela internet, podendo ser usado de qualquer lugar do mundo. E é empregado tanto para aprendizado à distância como para complementar o aprendizado presencial, que é chamado de *blended learning*, ou aprendizado mesclado.

2.4.1 Histórico

Em 1960 foi apresentado pela universidade de Illinois o primeiro LMS ou sistema instrucional auxiliado pelo computador, o Plato (*Programmed Logic for Automated Teaching Operation*) [Kumar et al. 2011]. Foi pioneiro em conceitos chave dos LMS como fóruns online, troca de mensagens, testes online, email, chat, troca de telas e jogos multiusuário.

No ano de 1997 foi introduzido o software comercial Blackboard, que atualmente é empregado em um grande número de universidades e também usado para ensino à distância. A empresa proprietária do Blackboard possui um histórico marcado por várias brigas judiciais. Mesmo com controvérsias, conseguiram patentear o conceito de LMS. E assim levaram adiante processos contra outras empresas fabricantes de software pela infração da patente, inclusive projetos de código aberto. Porém após algumas derrotas jurídicas, em 2007 o Blackboard afirmou que não mais processaria iniciativas de código livre, e em 2008 a justiça americana se mostrou a favor de invalidar a patente. Finalmente, em 2010, a empresa desistiu de tentar reativar a patente[15].

[15] http://insidehighered.com/news/2009/12/16/blackboard

No mesmo ano que foi lançado o Blackboard, em 1997, apareceu o WebCT, que é outra ferramenta bastante utilizada. Ele chegou a ser muito criticado por ser difícil de usar, não possuir uma estrutura uniforme, além de se basear fortemente em applets Java, que torna o uso lento e requer máquinas poderosas. Além disso, possui vários problemas de acessibilidade, não tendo passado em vários testes[16]. Em 2006 o WebCT foi adquirido e integrado ao Blackboard.

Em 2008 a comunidade Sakai Project envolvia mais de 100 instituições de ensino de todo o mundo. O software era utilizado por mais de 350 instituições[17]. Estes números tendem a crescer bastante, já que várias instituições que empregavam softwares comerciais estão optando por softwares de código aberto. O Sakai é um aplicativo orientado a serviços, desenvolvido em Java J2EE, projetado para ser escalável, confiável, interoperável e extensível.

Foi baseado no sistema CHEF (*CompreHensive collaborativE Framework*), desenvolvido pela Universidade de Michigan em 2002, que era um arcabouço para desenvolvimento de sistemas colaborativos. A primeira versão do Sakai foi apresentada em 2005. Seu nome é uma brincadeira com o nome CHEF, já que Sakai é um famoso chef de cozinha japonês.

O projeto brasileiro TIDIA (Tecnologia da Informação no Desenvolvimento da Internet Avançada), através do TIDIA-Ae (Aprendizado Eletrônico) adotou o Sakai, traduzindo e adaptando algumas partes. A partir do segundo semestre de 2008 ficou disponível para ser utilizado por toda comunidade acadêmica da USP no endereço http://tidia-ae.usp.br.

O ILIAS [Ilias.de 2011] foi desenvolvido pela Universidade de Colônia, na Alemanha, e é utilizado por diversas instituições ao redor do mundo. Já o Dokeos é mantido por uma comunidade internacional, e sua base instalada ultrapassa a marca de 1 milhão de usuários, e se concentra nas universidades da Holanda. Também fornece uma universidade gratuita em seu site, que possui mais de 200 mil alunos cadastrados.

---

[16] http://en.wikipedia.org/wiki/WebCT

[17] http://sakaiproject.org/organization-list

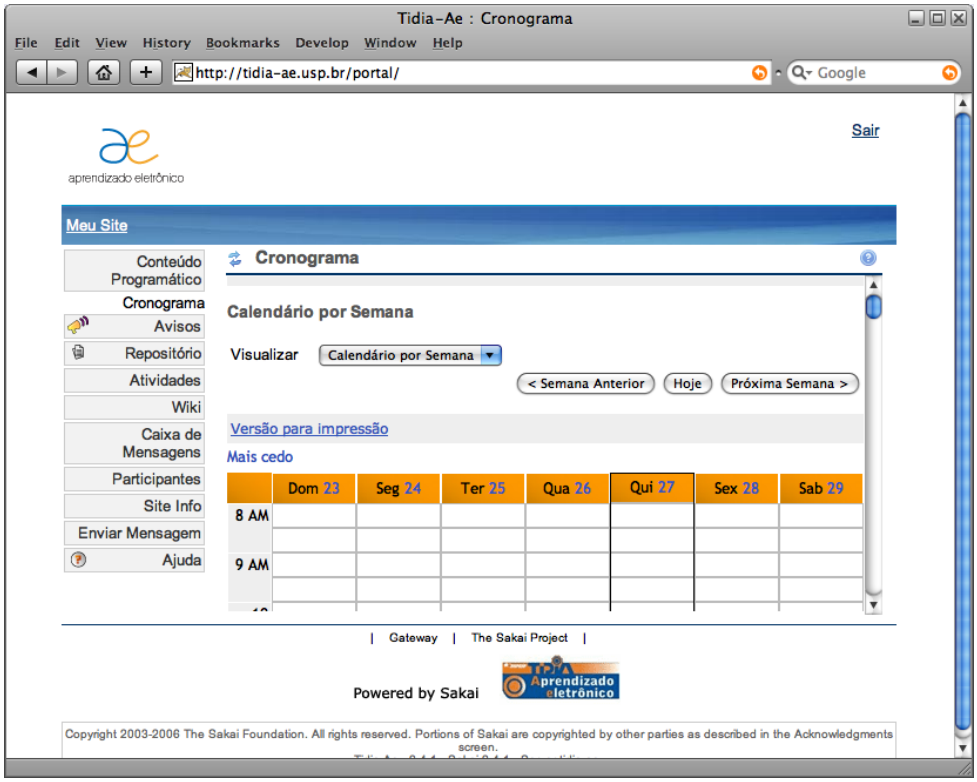

Figura 6 - Tela do Tidia Ae, baseado no Sakai, para uso dos alunos da USP. (http://tidia-ae.usp.br)

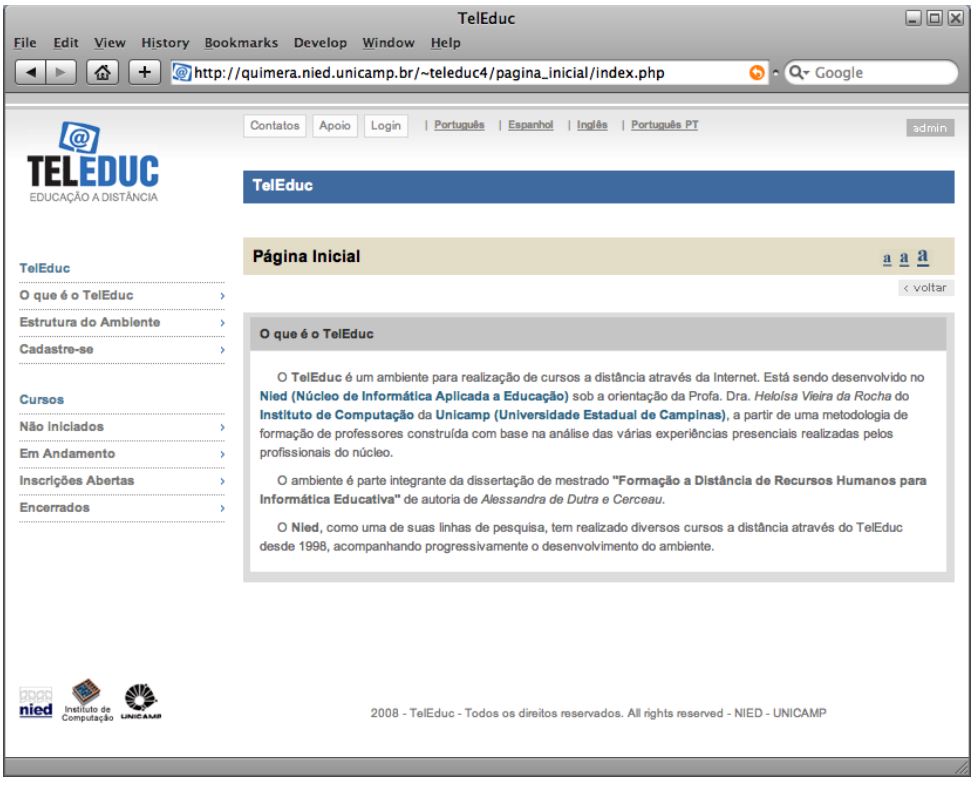

Figura 7 - Página inicial de uma instalação de teste do Teleduc, disponível no site do NIED. (http://nied.unicamp.br)

Um software desenvolvido no Brasil pelo NIED (Núcleo de Informática Aplicada à Educação), na Unicamp, é o Teleduc. Com foco na educação à distância, é desenvolvido desde 1998, e possui uma grande base de usuários. Foi desenvolvido baseado nas experiências com formação presencial de professores da Unicamp, traduzidas para formação à distância.

Outro software gratuito nacional é o AulaNet, desenvolvido na PUC Rio [Fuks & Cunha 2003]. Sua base é a colaboração proveniente da interação entre aluno e professor, entre alunos, e entre aluno e o conteúdo didático. Possui vários relatórios que exibem a participação do aluno, facilitando a avaliação ao final do curso.

Todos estes softwares são portais acessíveis pela Internet. Do ponto de vista das funcionalidades são semelhantes, com pequenas diferenças entre si. O que varia mais são os nomes das funções e a maneira de implantação e uso. Existem trabalhos apontando as diferenças entre os sistemas, porém isto foge ao escopo deste trabalho [Cheung 2007; Wainwright et al. 2007].

2.4.2 Adoção das ferramentas

Os LMS ajudam a desenvolver habilidades únicas quando os novos meios de comunicação e métodos colaborativos são bem empregados. Proporcionam acesso imediato ao conhecimento a pessoas em qualquer lugar do mundo. Também servem para substituir o livro texto tradicional, que é estático, por algo dinâmico e adaptado a cada aluno [Darbhamulla & Lawhead 2004].

É cada vez maior a adoção dos LMS e a quantidade de pessoas que os utilizam. Dados apontam [Malikowski et al. 2007] um grande crescimento do número de alunos que cursam disciplinas online, ministradas à distância. Mesmo assim, as ferramentas são utilizadas cerca de três vezes mais por alunos de cursos presenciais. Isso ocorre pois esse tipo de curso é mais comum, e é usado apenas como auxílio ao aprendizado. Já em um curso à distância a ferramenta é a base de toda interação que irá ocorrer, tornando sua implantação mais complexa e delicada.

De uma maneira geral, tanto LMS como outros softwares computacionais estão cada vez mais presentes nas universidades, escolas e outras instituições. Existem vários motivos para isso, por exemplo, por fornecer uma maior flexibilidade na vida dos estudantes,

possibilitando uma melhor conciliação dos estudos com a vida pessoal e profissional [Ginns & Ellis 2007].

2.4.3 Moodle

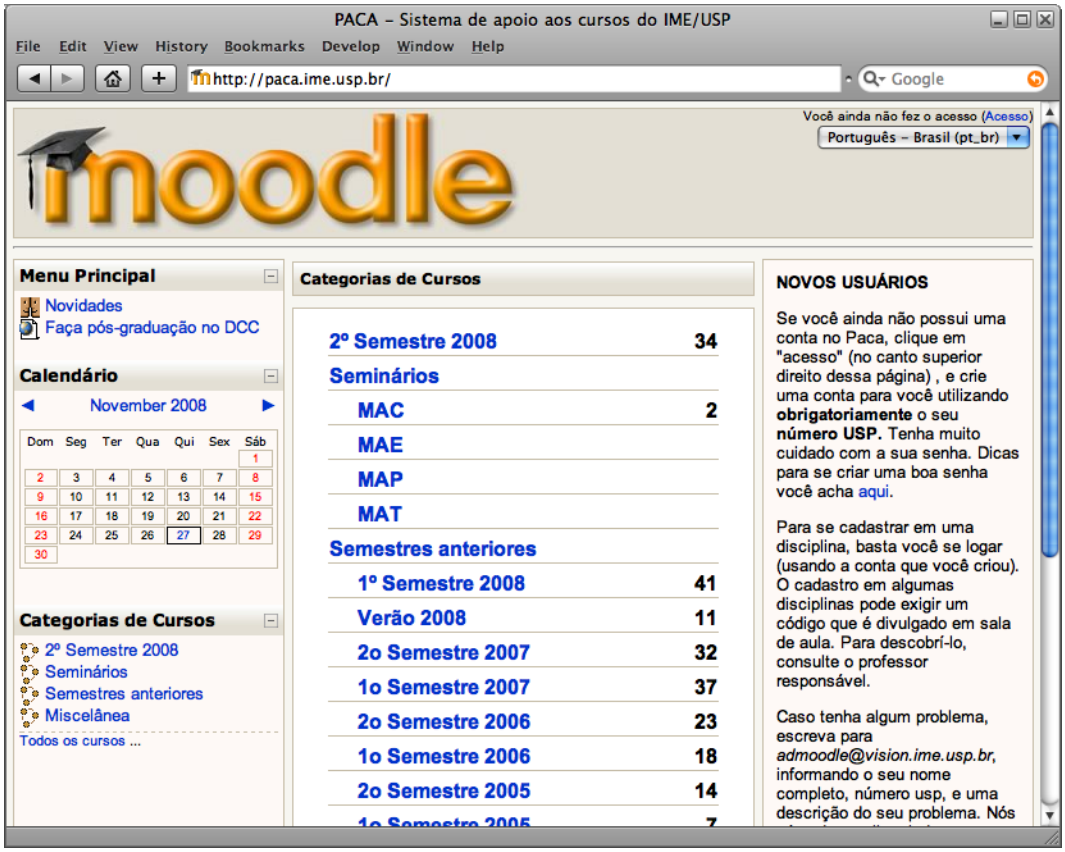

Figura 8 - Tela do moodle, conforme instalação do IME-USP, utilizada pelos alunos. (http://paca.ime.usp.br)

O Moodle, *Modular Object-Oriented Dynamic Learning Environment*, é outro programa LMS, gratuito e de código aberto. Foi desenvolvido a partir de 1999 por Martin Dougiamas. Em Dezembro de 2011 possuía 72 mil instalações e mais de 50 milhões de usuários registrados, e seu histórico indica grandes perspectivas de crescimento [Kumar et al. 2011; Moodle.org 2011]. Ele também é utilizado no instituto IME-USP.

O Moodle possui módulos para envio de atividades, fórum de discussão, repositório de arquivos, avaliação e atribuição de notas, mensagens instantâneas, calendário, novidades e anúncios, testes online, wiki, entre outros. Além dos módulos padrão, no site do

moodle existem mais de 700 plugins disponíveis, que permitem novos tipos de atividades, recursos, questões, autenticação, aparência, entre outros[18].

O Moodle foi a plataforma escolhida para dar suporte à ferramenta desenvolvida nesse trabalho. Leia mais sobre esta escolha na seção 3.1.3.

[18] http://moodle.org/plugins/

3 Ferramenta de correção automática

Este trabalho teve início em 2007, e foi parcialmente financiado pelo projeto TIDIA-Ae. A ferramenta foi concebida para ser inicialmente utilizada por cursos introdutórios, focado na usabilidade para alunos com pouca experiência.

Apesar do longo histórico dos corretores automáticos e das alternativas mais recentes, não existia, até o início deste trabalho, um aplicativo da maneira que propusemos. Ele deveria suportar a avaliação automática de exercícios com foco no ensino de programação, e ser totalmente integrado a um LMS. Algumas das alternativas têm outro foco, como os juízes online, que são mais voltados para competições de programação do que propriamente para o ensino (*Ver 2.1.6*). Outras não possuem todas as características desejadas, conforme as linhas gerais presentes no primeiro capítulo (*Ver 1.2*), onde a usabilidade e o suporte ao ensino de programação são os pontos principais.

3.1 Metodologia

Como este corretor nunca havia sido desenvolvido ou aplicado pela equipe do nosso instituto, foi definido um ciclo inicial de testes. Seria desenvolvida uma aplicação simplificada, porém funcional, e aplicada em uma turma de alunos. Seus requisitos foram levantados junto aos professores do curso de introdução à programação.

3.1.1 Ciclo inicial de testes

Baseadas nas linhas gerais, algumas decisões técnicas foram tomadas para este ciclo inicial. Apesar da baixa utilização e de se encontrar em fase de desenvolvimento inicial, o sistema LMS escolhido foi o Sakai. À época era um sistema promissor e apoiado oficialmente pelo projeto Tidia-AE da Fapesp, entre outras instituições internacionais.

Os testes dos programas enviados pelos alunos seriam executados em uma máquina virtual a fim de garantir seu encapsulamento. Para tal foi escolhido o mecanismo User-mode Linux[19], que permite a criação de uma máquina virtual executando qualquer distribuição Linux escolhida, com seu próprio sistema de arquivos e tarefas.

A extensão do Sakai foi feita modificando o módulo de atividades. Vale notar que, por se tratar de um estágio inicial, esta alteração foi feita diretamente sobre o código existente, e não modularmente no formato de um plugin.

Também foi decidido utilizar um arquivo de configuração XML para definir os testes. Para facilitar o desenvolvimento, este arquivo seria anexo pelo professor à atividade. Uma desvantagem que isto gerou era que o aluno possuía acesso ao arquivo, podendo inspecionar todos os testes, mesmo os secretos.

Os alunos deveriam enviar seu código através do upload do arquivo. Assim, eles precisavam editar o código em um editor externo, e depois enviar usando o navegador. A única linguagem suportada era o C, e apenas a saída do programa era analisada.

O módulo do Sakai transferia os arquivos necessários via SFTP para o sistema de arquivos da máquina virtual, a saber, o arquivo de configuração e o código a ser testado. Posteriormente, através de uma conexão SSH para o interior da máquina virtual, executava o corretor.

O teste foi realizado ao longo do primeiro semestre de 2011, com uma turma de 49 alunos que trabalharam 28 atividades. A matéria era de introdução à programação, para alunos lidando com programação pela primeira vez. De todos os alunos, apenas 15 seguiram utilizando a ferramenta até o final. O sistema se mostrou bastante lento, levando cerca de 10 segundos para retornar o relatório após cada envio.

Os exercícios foram ainda analisados pelo detector de plágio Moss, onde foram encontrados alguns casos de cópia entre alunos. Esta análise, porém, foi feita diretamente pelo professor junto ao site.

---

[19] http://user-mode-linux.sourceforge.net/

3.1.2 Decisões para o projeto final

Baseado nos resultados do primeiro ciclo e nas premissas definidas na seção 1.2, definimos nosso projeto final. As principais mudanças foram:

- Adoção do LMS Moodle.
- Desenvolvimento de um plugin modular.
- Nova interface.
- Substituição do User-mode Linux pelo Vserver.
- Adoção do protocolo XMLRPC para a comunicação do plugin com a máquina virtual.
- Configuração em arquivo XML invisível aos alunos.
- Corretor em Java com suporte a Java além de C, e correção de funções.

Cada um desses itens serão detalhados a seguir.

3.1.3 Adoção do Moodle

Ao invés do Sakai, escolhido na primeira etapa, o corretor foi desenvolvido para o LMS Moodle. Como já mencionado na seção 2.4.3, o Moodle é gratuito, amplamente utilizado e ativamente mantido e desenvolvido. O principal fator que levou à adoção do Moodle foi seu legado no IME-USP, onde é utilizado desde 2003, e uma grande base de material de cursos já está registrado.

Ele é feito na linguagem PHP, que é um script interpretado pelo servidor, normalmente Apache ou Lighttpd. Isso gera um custo adicional da interpretação quando comparado com linguagens compiladas, como é o caso do Java, utilizado pelo Sakai. Porém estudos recentes [Trent et al. 2008] mostram que este custo é pouco significativo, e a carga computacional do aplicativo em si é o componente mais importante do tempo total de execução, desde que o servidor web esteja configurado e calibrado de acordo com os recursos de hardware disponíveis.

O Moodle também é uma plataforma bastante documentada e com uma grande base de desenvolvedores ativos[20]. Seu código é estruturado em módulos e orientado a objetos. Isso torna o desenvolvimento de extensões e modificações relativamente fácil. O Sakai,

[20] http://docs.moodle.org/dev

por outro lado, possui poucos desenvolvedores e pequena base de conhecimento, e sua documentação ainda está incompleta.

O banco de dados do Moodle é um banco relacional simples. Existe uma camada de abstração, portanto é possível utilizar diversos mecanismos de banco de dados transparentemente, como MySQL, PostgreSQL ou Oracle, entre outros.

A estrutura de banco de dados do Moodle é bastante autoexplicativa, baseada em tabelas simples mantidas por cada módulo. A única tabela compartilhada é a de contextos, que é um código que referencia unicamente cada componente do site, como cursos, usuários, tarefas, etc. Já no Sakai, a estrutura de banco de dados é complexa e pouco documentada.

Para testar o código e fazer a atualização do servidor, basta substituir os arquivos de código fonte PHP e executar eventuais scripts de atualização. Com a ferramenta PDT (*PHP Development Tools*) para o IDE Eclipse é possível fazer depuração local ou remota. No Sakai é necessário compilar o código utilizando ferramentas de linha de comando.

Por fim, a instalação do Moodle é muito simples, tanto se for escolhido instalar um pacote completo, como se for utilizar código baixado do repositório GIT. Instalar um servidor Apache, um banco MySQL e configurá-los é uma tarefa de poucos minutos, independente da plataforma utilizada.

3.1.4 Desenvolvimento em formato de plugin

Ao desenvolver o corretor para o Sakai, até pelo caráter temporário, por ser tratar de um ciclo de testes, e também pelo Sakai ser mais limitado com relação ao desenvolvimento de plugins, o código fonte original foi alterado diretamente.

No Moodle, porém, é possível criar uma nova modalidade de atividade (*assignment type*), e tornar sua instalação modular[21]. Isso permite a incorporação do corretor em um ambiente já existente mais fácil e rápida, ao passo que a modificação de código existente requer condensação de código e, dependendo do caso, recompilação dos códigos fontes.

[21] http://docs.moodle.org/dev/Assignment_types

O Moodle é totalmente baseado em plugins independentes, de maneira que é fácil estender os mais diversos aspectos do software. De acordo com a documentação[22], porém, é necessário seguir algumas linhas guia ao escrever ou estender um plugin:

Seguir o mesmo estilo de código do projeto com relação a tags, espaços, endentação, nome de variáveis, comprimento das linhas, declaração de funções, documentação, comentários, exceções, entre outros;

Manter o padrão de segurança, distinguindo os papéis de administrador, professor, aluno e convidado separados. Autenticando o usuário a cada página, verificando o nível de acesso e permissões, verificando e limpando entradas e saídas, limpando registros antes de gravar no banco de dados, fazendo log dos comandos, entre outros.

Certificar que o HTML gerado é compatível com XHTML, e deixar todo o código de formatação e layout em um arquivo CSS separado.

Javascript deve ser usado para melhorar a usabilidade, e não para acrescentar funcionalidades, devendo ser possível utilizar o programa mesmo com o Javascript desligado. Além disso, a maioria do código deve ser inclusa em um arquivo separado, deixando para o corpo apenas chamadas simples de função.

Toda e qualquer frase ou nome exibido pelo programa deve ser referenciada em um arquivo de internacionalização, que será localizado dependendo do idioma da instância instalada. A língua padrão do Moodle é o inglês.

Padrões de acessibilidade e usabilidade devem ser seguidos. Acessibilidade significa que o software deve funcionar bem para o maior número de pessoas possível. Usabilidade, que deve ser facilmente usado pela maioria das pessoas.

O software deve ser escalável, ou seja, um pequeno aumento no número de instâncias de usuários, cursos, atividades ou outras variáveis deve provocar um pequeno aumento correspondente no tempo de processamento e uso de memória. Para isso alguns cuidados devem ser seguidos, como não colocar acessos ao banco de dados dentro de loops, limitar o tamanho de saídas repartindo em páginas, testar a carga com muitos cursos e usuários, tomar cuidado ao fazer chamadas remotas e de script.

---

[22] http://docs.moodle.org/dev/Coding

O acesso ao banco é através de uma camada de abstração chamada XMLDB, que padroniza tipos, chaves e índices para trabalhar transparentemente com MySQL, Postgre SQL, Oracle ou MS SQL Server. É preciso seguir os padrões de id, nomes de tabelas e campos, relacionamentos, entre outros.

Se for necessário comunicar com outros módulos, deve-se usar a API de eventos do Moodle. Por exemplo, existem eventos que são disparados quando um usuário é criado, alterado ou excluído, o mesmo para cursos, grupos, e assim por diante.

O Moodle incorpora uma maneira de realizar testes automáticos através de uma biblioteca chamada *simpletest*. Assim, é recomendado criar testes automatizados para cada funcionalidade incorporada.

Se estas recomendações forem seguidas, teremos um código que poderia ser incorporado à distribuição oficial do Moodle, e assim beneficiar toda a numerosa base instalada.

3.1.5 Projeto da interface

No formato anterior o aluno utilizava um editor de texto à sua escolha para escrever o programa, e depois o enviava para correção através de um upload no navegador. O problema deste procedimento é que, a cada vez que era necessário testar o programa, o aluno precisava salvar o arquivo, localizá-lo e enviá-lo.

Para agilizar este processo, um editor foi incorporado à interface, onde é possível escrever o código no próprio navegador. Assim, a seção mais importante da ferramenta para o aluno está em uma só página. Ela reúne o enunciado, a área para editar o código, o relatório de resultado e a nota. Não é necessário visitar várias páginas para compreender o exercício, resolvê-lo, e conferir o resultado. Para quem preferir utilizar outros editores, a opção de envio de arquivo foi mantida, nesta mesma página. Para melhorar a usabilidade, este editor possui realce de sintaxe. Leia mais sobre o editor na seção 3.2.2.

Ao enviar o código, o relatório com o resultado da correção automática deve retornar em poucos segundos. Este relatório contém as mensagens da compilação, o detalhamento de cada um dos testes de comparação de resultado, e a nota final. Sem sair da página é possível conferir e realizar as modificações, caso sejam necessárias.

Na figura 9 é possível ver a diferença entre a utilização de um editor externo e de um editor integrado ao corretor.

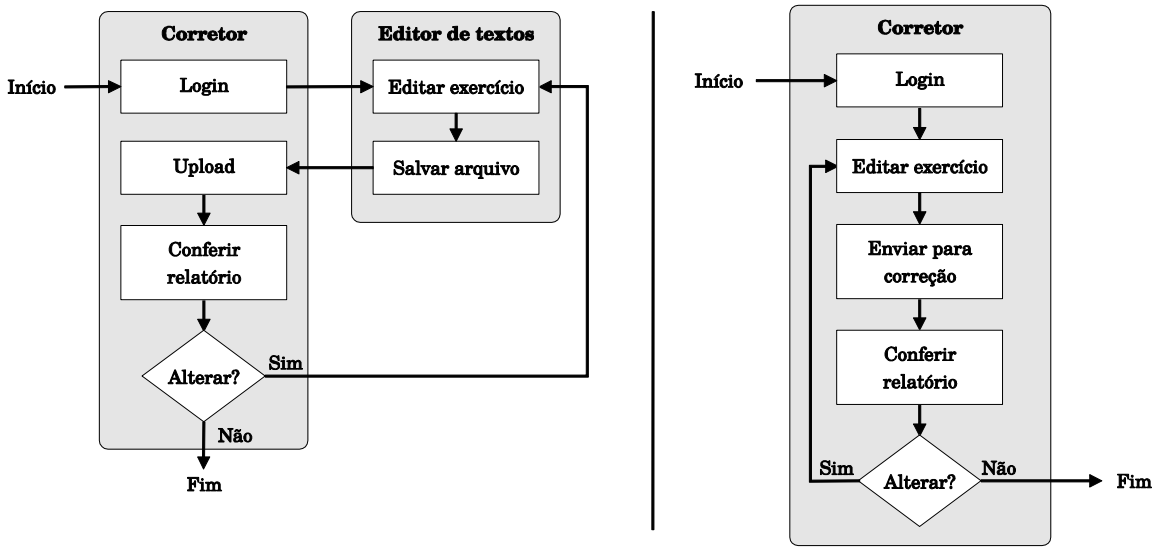

Figura 9 - Diagrama de uso com envio de arquivo (esquerda), e com o editor integrado (direita)

3.1.6 Máquina virtual

Como foi definido que, por motivos de segurança, o servidor que executa o Moodle deveria estar segregado do servidor que compila e executa os programas, foi escolhida uma arquitetura contendo uma máquina virtual.

A máquina virtual escolhida foi o Vserver[23]. Nos testes realizados se mostrou mais rápido que o User-mode Linux, e fundamental para que a resposta fosse praticamente instantânea, melhorando a percepção do aluno e sua experiência com o corretor.

O Vserver utiliza um conceito de particionamento baseado em contextos de segurança. Isto permite a criação de vários VPS (Virtual Private Server), totalmente independentes. Eles rodam simultaneamente em um único servidor físico, compartilhando os recursos de hardware.

Cada VPS é um ambiente operacional praticamente idêntico a um servidor Linux convencional, e qualquer serviço pode ser utilizado normalmente. Cada VPS possui seu próprio banco de dados de usuários e senha de root.

Este ambiente é ideal para execução dos exercícios enviados para serem corrigidos, uma vez que os recursos do servidor não ficam expostos.

[23] http://linux-vserver.org/

3.1.7 Comunicação via XMLRPC

No primeiro ciclo de testes, o plugin para o Sakai se comunicava com a máquina virtual através de conexões SSH. Primeiro, os arquivos eram enviados via SFTP[24]. Esses arquivos são o XML de configuração e o programa e ser executado. Depois, uma conexão SSH era estabelecida para executar o corretor.

Isto apresentava algumas desvantagens. Primeiro, a conexão SSH é mais lenta se comparada a uma conexão HTTP, pois envolve negociação de autenticação e chaves de segurança. E eram necessárias três conexões, duas para transferir os arquivos e a terceira para executar o corretor. Outra desvantagem era com relação à segurança. A porta SSH permitindo conexões entrantes deixava exposta uma quantidade maior de recursos, mesmo que apenas os recursos da máquina virtual.

A fim de melhorar este cenário, escolhemos o protocolo XMLRPC (XML Remote Procedure Call)[25] para a comunicação do plugin Moodle com o corretor. O servidor recebe do plugin, através de uma conexão HTTP, os arquivos que antes eram enviados via SFTP, encapsulados em um XML. Assim, a única porta exposta pela máquina virtual é a do servidor HTTP, que permite apenas conexões específicas para uso do protocolo XML-RPC.

O corretor é invocado através de um comando Shell, tomando como parâmetros os arquivos gravados temporariamente na máquina virtual. O resultado é retornado pela mesma conexão HTTP, também encapsulada em um XML, que é finalmente decodificada pelo corretor.

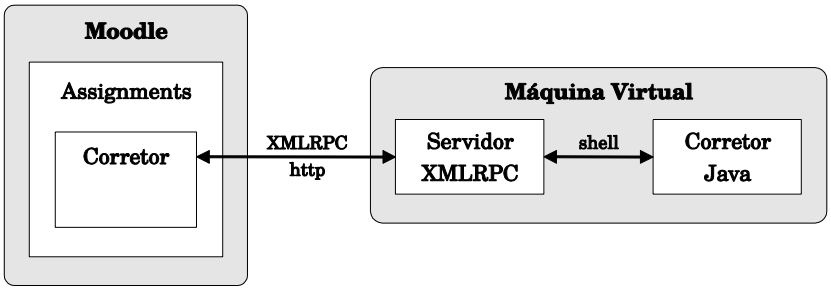

Figura 10 - Diagrama com os componentes do corretor automático

[24] http://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol

[25] http://xmlrpc.scripting.com/

Para executar o script PHP foi escolhido o servidor Apache. O LMS Moodle também é escrito em PHP e executado no Apache. Essas tecnologias, além de serem gratuitas e de código aberto, são bastante maduras. Por serem sempre atualizadas e terem uma grande base de desenvolvedores, são bastante seguras.

3.1.8 Configuração XML invisível

O arquivo XML de configuração (ver seção 3.4.5) contém todos os testes a serem executados pelo corretor sobre o exercício enviado. Para tornar a avaliação mais abrangente, alguns destes testes são apenas parcialmente visíveis no relatório para o aluno, por exemplo, não exibem qual a saída esperada, apenas se o teste foi bem sucedido. Assim, o arquivo não deve nunca estar disponível para consulta pelos alunos, pois seria fácil descobrir os parâmetros dos testes secretos. No primeiro plugin, até por uma limitação da implementação, o arquivo era visível pelos alunos. No novo plugin, para o Moodle, isso foi alterado, e os alunos não tem mais acesso ao arquivo.

A manutenção da configuração no mesmo padrão XML permite que o mesmo exercício seja facilmente transportado do plugin Sakai para o Moodle, e vice versa.

3.1.9 Corretor em Java

O corretor que executa no interior da máquina virtual agora possui suporte à linguagem Java, além da linguagem C, presente na fase de testes. Também prevê o teste de funções individuais, e não apenas da saída padrão do programa. Leia mais sobre o corretor na seção 3.4.

3.2 Plugin Moodle

O plugin foi baseado na ferramenta *assigments* (atividades) do Moodle. Na versão atual à época que o corretor foi projetado, essa ferramenta possuía quatro tipos de atividade: atividade offline, texto online, upload de um arquivo e upload de vários arquivos. Este módulo se mostrou conveniente para ser estendido, uma vez que já possuía recursos que seriam reaproveitados: lista de atividades enviadas por aluno e respectivas notas, integração com o livro de notas, data de início e data limite.

A criação de um novo tipo de atividade é prevista na documentação do Moodle, e pode ser incorporada em qualquer instalação facilmente, da mesma maneira que qualquer outro plugin disponível.

As configurações comuns a qualquer atividade são: título, enunciado em hipertexto, data de início, data limite, nota máxima, categoria de avaliação, modo de grupo. Além dessas, é possível durante o desenvolvimento do plugin especificar outros campos de configuração. A única configuração específica à nossa ferramenta é o arquivo de configuração XML, que deverá ser enviado em um campo para upload. Este arquivo está especificado na seção 3.4.5.

3.2.1 Página de criação ou alteração de atividade

Ao acrescentar uma atividade, o professor deve escolher a opção *add an activity* → *assignments* → *upload or edit source code*. A tela (*figura 11*) que aparecerá solicita os seguintes campos:

- *Assignment name*: Nome da atividade
- *Description*: Descrição em hipertexto, que pode conter formatação e figuras anexas.
- *Display description on course page*: Exibe a descrição complete na página do curso.
- *Available from*: Data em que a atividade passará a ser visível para o aluno.
- *Due date*: Data e hora limite para envio de respostas.
- *Prevent late submissions*: Aceita o envio de respostas após a data limite.
- *Grade*: Nota máxima.
- *Grade category*: Uma das categorias de nota que podem ser cadastradas para esse curso.
- *XML*: Arquivo de configuração (ver seção 3.4.5).
- *Group mode*: Agrupamento dos alunos do curso.

Ao alterar uma atividade é possível substituir o arquivo XML de configuração. Para isto basta enviar novamente. O código interno se encarrega de apagar o arquivo anterior.

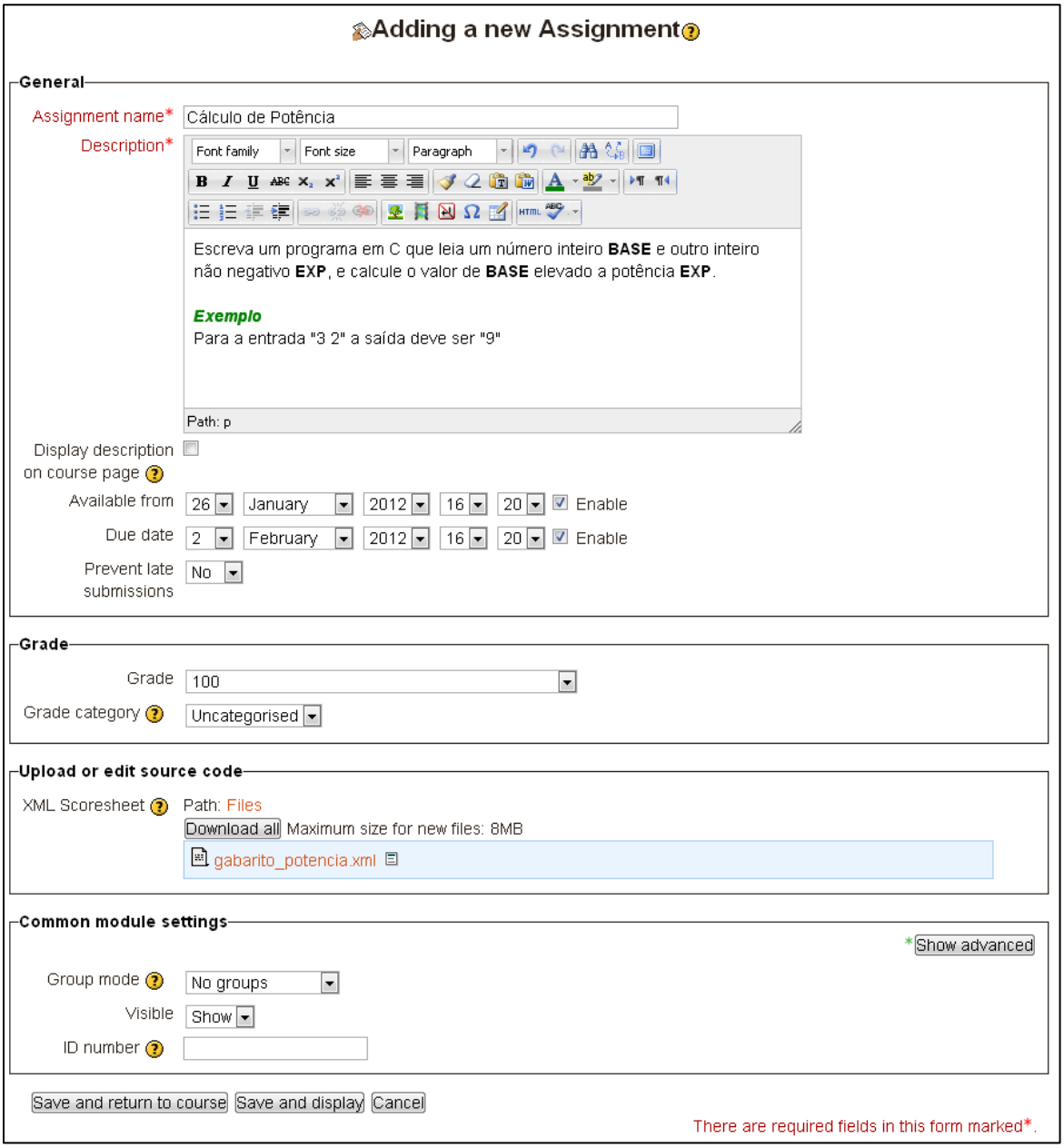

Figura 11 - Tela de configuração de uma atividade de correção automática

3.2.2 Página do aluno

A principal característica da página para o aluno deve ser a facilidade de edição do código, execução dos testes e análise do relatório. Visando diminuir o número de passos necessários para enviar ou reenviar um código, foi decidido manter todas as funções em uma única página.

Esta página reúne todas as informações necessárias: o título, enunciado do exercício, campo para digitação da resposta do exercício, campo para envio de arquivo, relatório de erros e avaliação, nota final e comentário do avaliador.

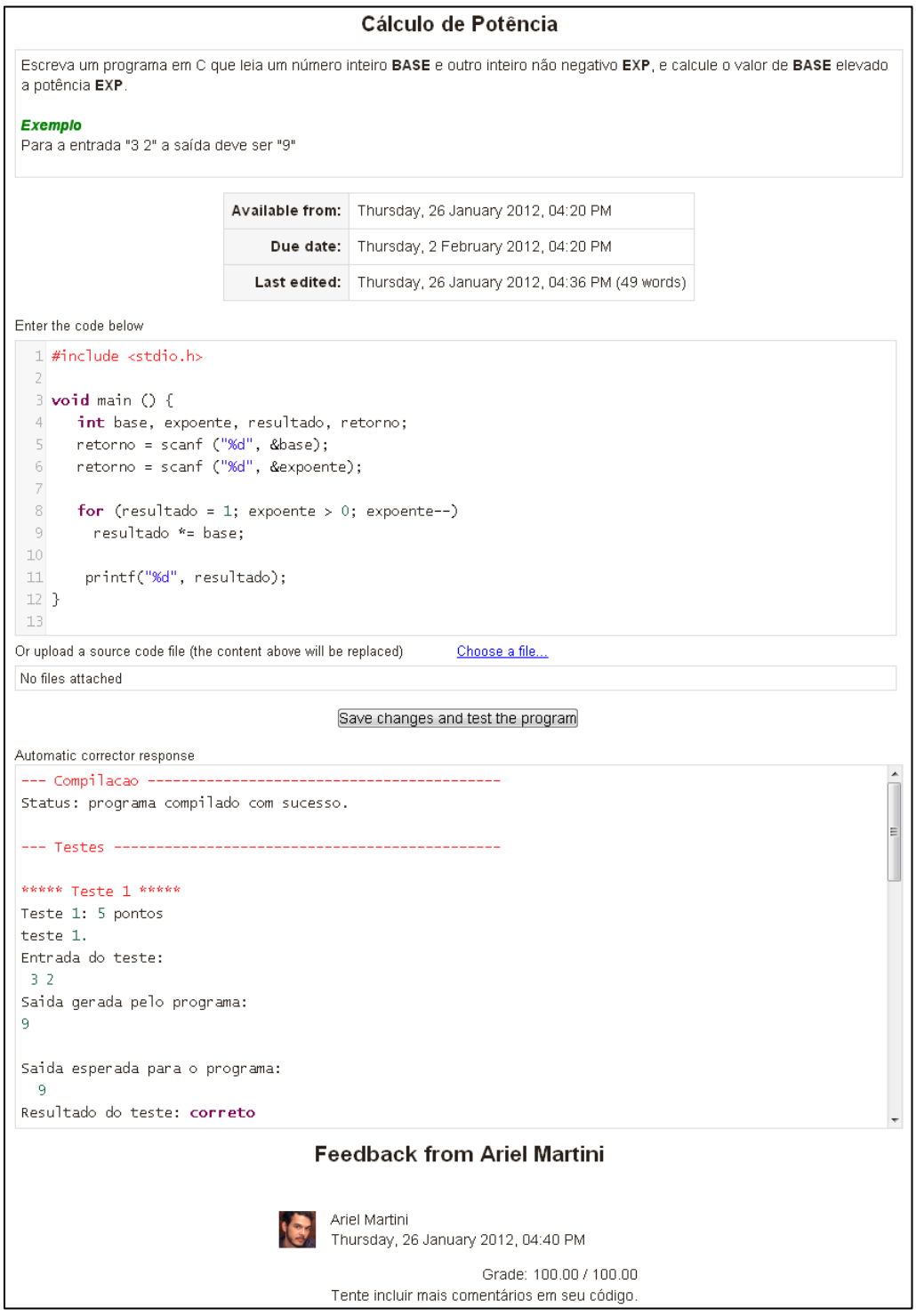

Figura 12 - Página principal para o aluno, com a descrição, área para digitação do código e relatório de erros

O campo onde o programa será editado é baseado no Codemirror[26], um editor implementado em linguagem Javascript, gratuito e de código aberto. Ele exibe o número das linhas numa coluna à esquerda, faz endentação automática, casamento de parênteses e realiza realce de sintaxe, ou seja, aplica cores diferentes conforme o código. Por exemplo, existem cores diferentes para comentários, palavras reservadas, operadores, números, cadeias de caracteres, etc. O Codemirror possui suporte a realce de sintaxe para diversas linguagens de programação diferentes. Foi adotado um padrão de cores fixo, porém é possível modificar o código para permitir personalização das cores.

Tudo isto torna a edição do código no navegador mais próxima da edição usando um IDE externo. Apesar disso, várias características não estão presentes, como depuração, pesquisa, hierarquia de funções ou objetos, complementação automática de símbolos, entre outras.

O campo para envio de arquivo permite o upload de um arquivo presente localmente no computador do usuário. O aluno deve usar esta função se já possuir o arquivo ou se preferir usar outro editor. Nesse caso, qualquer alteração utilizando o editor integrado será substituída pelo arquivo enviado.

O relatório de erros e testes é o enviado pelo corretor e exibido logo abaixo do editor de texto e campo para upload. Ele também é exibido utilizando o Codemirror. Foi desenvolvido um padrão de cores específico para o relatório, para facilitar sua leitura.

A nota é automaticamente extraída do relatório e gravada no banco de dados. Num momento posterior um monitor ou professor ainda pode incluir comentários e alterar esta nota. Estes comentários aparecem logo abaixo do relatório.

A cada vez que o aluno envia o programa para o teste, é gravada uma informação para posteriormente gerar estatísticas sobre a utilização do programa.

3.2.3 Página do professor ou monitor

A página principal para o professor é uma tabela com todos os alunos matriculados. Em cada linha é exibido o nome do aluno, se foi enviada uma solução para aquele exercício ou não, em que data enviou, qual a nota atribuída, e se já foram incluídos comentários. A partir da tabela o professor pode ver em detalhe o resultado em um popup, ou abrir o

[26] http://codemirror.com

item em uma página, onde é exibido o enunciado, o conteúdo da resposta e o relatório. Nessa página é possível incluir comentários ou modificar a nota.

3.3 Máquina Virtual

O Vserver utilizado foi montado baseado na distribuição Ubuntu. Nele foram instaladas as ferramentas necessárias para compilar os programas, como gcc, gpp, Java, entre outras. Após a instalação dos pacotes, a conexão do interior da máquina virtual com a internet ou com outras redes foi bloqueada. A única conexão aberta é a porta que aceita as conexões entrantes para um servidor Apache, que executa o servidor XMLRPC.

Para implementar este servidor escolhemos o pacote XML-RPC for PHP[27], que permite integrar facilmente funções de codificação e decodificação de pacotes XML na linguagem PHP.

O servidor recebe os arquivos de parâmetro e os grava temporariamente no sistema de arquivos da máquina virtual utilizando nomes gerados aleatoriamente. Estes são então passados como parâmetros de linha de comando ao invocar o programa corretor. Isso é feito usando o comando *shell_exec* do PHP[28]. O resultado do corretor, recuperado através da saída comum stdout, é então encapsulado em um XML de resposta e repassado para o plugin através da mesma conexão. Por fim, os arquivos temporários são excluídos.

3.4 Corretor

Para efetivamente compilar e executar o código, realizando os testes de entrada e saída, foi desenvolvido um corretor em linguagem Java. Visando sua futura extensão, a metodologia usada foi a de orientação a objetos. Ele foi projetado inicialmente para atender os exercícios relativos aos tópicos do curso de introdução à computação. Após uma análise, chegou-se à conclusão que o suporte à entrada e saída de um ou mais números inteiros, reais ou strings seria suficiente. Isso se mostrou correto, conforme veremos na seção 4.

O corretor recebe dois parâmetros pela linha de comando: O nome do arquivo XML de configuração e o nome do arquivo contendo o programa a ser executado. Ele analisa o

---

[27] http://phpxmlrpc.sourceforge.net/

[28] http://php.net/manual/en/function.shell-exec.php

XML, modifica o código inserindo rotinas de teste e utiliza um compilador de linha de comando para compilar o programa. Se for bem sucedido, o corretor executa o programa para cada teste contido no XML, alimentando a entrada com os dados de teste e comparando a saída com o gabarito. Se o teste for bem sucedido, soma a nota daquele teste à nota final. Por fim, o relatório com o resultado é emitido na saída padrão, stdout.

3.4.1 Linguagens suportadas

As linguagens inicialmente suportadas são C e Java. Porém, é possível adicionar qualquer linguagem que possua compilador linha de comando para Linux, e tenha suporte a entrada e saída via terminal ou linha de comando. A grande maioria das linguagens existentes atendem esses pré-requisitos. Para adicionar uma linguagem é necessário instalar o compilador na máquina virtual e modificar o código fonte do corretor para prever a nova linguagem, estendendo a classe de definição de linguagem.

3.4.2 Tradução de erros

Alguns erros de compilação podem ser de difícil compreensão por alunos iniciantes. Para auxiliá-los, os mais comuns foram traduzidos e, quando possível, melhor explicados e acrescidos de informações úteis para a solução do erro. Isto foi feito diretamente no arquivo de mensagens de erro do compilador, no interior da máquina virtual.

3.4.3 Formato de entrada

Os dados de entrada de um teste podem ser enviados para o programa de três maneiras diferentes: *stdin*, *args* ou função. Através do *stdin*, os dados são enviados como se estivessem sendo digitados pelo usuário, devendo ser capturados por funções como *scanf*. Através de args, os dados são passados como parâmetros na linha de comando da chamada do programa. E no caso de função, os dados são passados como parâmetros da função. Se houver mais de um parâmetro, eles devem estar separados por quebra de linha.

3.4.4 Formato da saída

Para tratar a saída esperada de um teste, o corretor separa os elementos usando espaços, vírgulas, ponto e vírgula ou barra. Cada elemento é processado individualmente.

Se for um número, ele compara numericamente após converter para ponto flutuante. No caso de números fracionais, existe um limite de precisão de 10^{-4}. Se a diferença for menor que este limite, os números são considerados iguais.

Se não for número, compara as cadeias de caracteres normalmente, desprezando a diferença entre maiúsculas e minúsculas.

3.4.5 Arquivo de configuração

O arquivo em formato XML possui a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?>
<gabarito>
  <global>
      <displayFinalGrade>true</displayFinalGrade>
      <displayStatistics>true</displayStatistics>
      <maxRunningTime>10000</maxRunningTime> <!-- milliseconds -->
      <maxOutputLength>50000</maxOutputLength> <!-- bytes -->
      <srcLanguage>java</srcLanguage>
      <registerFunc>float, soma, float, float</registerFunc>
  </global>
  <teste>
      <nome>Teste 1</nome>
      <descricao>Teste de potencia </descricao>
      <nota>10</nota>
      <entrada>
          3
          2
      </entrada>
      <saida>9</saida>
      <functionTest>true</functionTest>
      <functionName>potencia</functionName>
      <inputMethod>stdin</inputMethod>
      <displayOutput>true</displayOutput>
      <displayExpectedOutput>true</displayExpectedOutput>
      <displayResult>true</displayResult>
  </teste>
</gabarito>
```

A seguir descrevemos cada campo:

3.4.5.1 displayFinalGrade

Booleano. Indica se a nota final estará presente no relatório

3.4.5.2 displayStatistics

Booleano. Indica se as estatísticas estarão presentes no relatório

3.4.5.3 maxRunningTime

Tempo máximo de execução do programa, em milissegundos, para cada teste executado. Caso esse tempo seja ultrapassado, o programa é terminado e o relatório resulta em erro.

3.4.5.4 maxOutputLength

Tamanho máximo da saída de cada teste que será capturado, em bytes. Se ultrapassar, o resultado é truncado.

3.4.5.5 srcLanguage

Opcional. Indica a linguagem alvo do exercício. Valores permitidos: *c*, *java*. Valor padrão: *c*.

3.4.5.6 registerFunc

Especifica o cabeçalho de uma função no formato “tipo de saída, nome da função, tipo de entrada 1, tipo de entrada 2 (opcional), tipo de entrada 3 (opcional), etc.”.

3.4.5.7 nome

Nome do teste

3.4.5.8 descricao

Descrição do teste

3.4.5.9 nota

Nota adicionada à nota final caso este teste seja bem sucedido. A nota máxima possível é a soma de todos os campos *nota*.

3.4.5.10 entrada

Dados que serão passados ao programa. Veja 3.4.3.

3.4.5.11 saida

Dado de retorno esperado. Veja 3.4.4.

3.4.5.12 functionTest

Booleano. Opcional. Indica se será testada uma função ou o programa (ver *inputMethod*). Default: false;

3.4.5.13 functionName

Opcional. Nome da função a ser testada. Se for omitido, o teste será sobre entrada e saída do programa (ver *inputMethod*)

3.4.5.14 inputMethod

Indica o método que o corretor irá passar os dados de entrada para o programa. Aplicável apenas se *functionTest* for *false*. Valores permitidos: *stdin*, *args*. No caso de *stdin*, a entrada será passada como se uma passoa estivesse digitando os valores após o programa ser executado. No vaso de *args*, a entrada é passada como parâmetros na hora de executar o programa.

3.4.5.15 displayOutput

Booleano. Indica se a saída gerada pelo programa será exibida no relatório.

3.4.5.16 displayExpectedOutput

Booleano. Indica se a saída esperada pelo teste será exibida no relatório. Os valores de entrada do teste serão exibidos se *displayOutput* ou *displayExpectedOutput* forem *true*.

3.4.5.17 displayResult

Booleano. Indica se o resultado (correto/incorreto) será exibido no relatório.

3.5 Desenvolvimentos futuros

A seguir citamos algumas características que poderiam ser incorporadas no futuro.

3.5.1 Avaliação automatizada da mantenibilidade

A aplicação de métricas sobre o código para estimativa da qualidade da mantenibilidade visa avaliar a qualidade dos comentários, adequação dos nomes das variáveis, endentação e modularidade, que são itens difíceis de serem avaliados automaticamente. Para

cada métrica pode ser atribuído um peso, e a nota resultante contribuir para modificar a nota final, após a revisão de um responsável.

As métricas dependem do paradigma, mas, como já citado no primeiro capítulo, algumas que podem ser avaliadas são: número de caracteres por linha, proporção de linhas de comentários, proporção de endentação, proporção de linhas em branco, espaços por linha, tamanho médio do módulo, número de palavras reservadas, tamanho dos identificadores, uso de GOTO, número de arquivos *include*, proporção de *define*, entre outros [Jackson 2000].

Como já citado, considerar estes fatores na correção pode fazer com que alguns alunos dediquem tempo excessivo aprimorando o aspecto estético de um código já funcional, enquanto poderiam estar resolvendo outros problemas [Cheang et al. 2003]. Assim, para evitar este cenário, é importante deixar claro a finalidade desta avaliação, e sua influência na nota final.

3.5.2 Detecção de plágio

Para manter os casos de plágio sob controle, é importante submeter os arquivos enviados pelos alunos para um detector automático. É sempre possível fazer isto manualmente, obtendo o arquivo com todas as submissões, organizando-as, e enviando para um dos serviços disponíveis, como o Moss ou o Jplag. Porém, automatizar esta tarefa economizaria tempo do avaliador, além de facilitar e incentivar sua realização.

Existem três plugins para o Moodle que podem desempenhar essa função. O CROT[29] é um analisador de plágio autônomo, portanto não precisa de conexão com um mecanismo externo, e pode ser usado para qualquer tipo de atividade. O Urkund[30] é um mecanismo bastante avançado, porém não é gratuito. Por fim, o Anti-Plagiarism Plugin[31] é específico para exercícios de programação. É baseado no Moss[32], e automatiza o envio de exercícios para o serviço.

---

[29] http://moodle.org/plugins/view.php?plugin=plagiarism_crot

[30] http://www.urkund.com

[31] https://github.com/hit-moodle/anti_plagiarism/wiki

[32] http://moss.stanford.edu/

Assim, enquanto o corretor não oferece suporte nativo a um detector de plágio, é possível utilizar um destes plugins ou realizar o controle manualmente.

3.5.3 Editor especializado de atividades

É sempre responsabilidade do professor escolher um exercício e escrever seu enunciado da melhor maneira possível. Porém, é possível dividir o enunciado em campos específicos para incentivar seu preenchimento, enriquecendo o conteúdo. É desejável que um enunciado bem escrito tenha seções de objetivos, pré-requisitos, exemplos e dicas. Portanto seria melhor que, ao adicionar uma atividade, estes campos não fossem opcionais e implícitos em um grande campo "descrição".

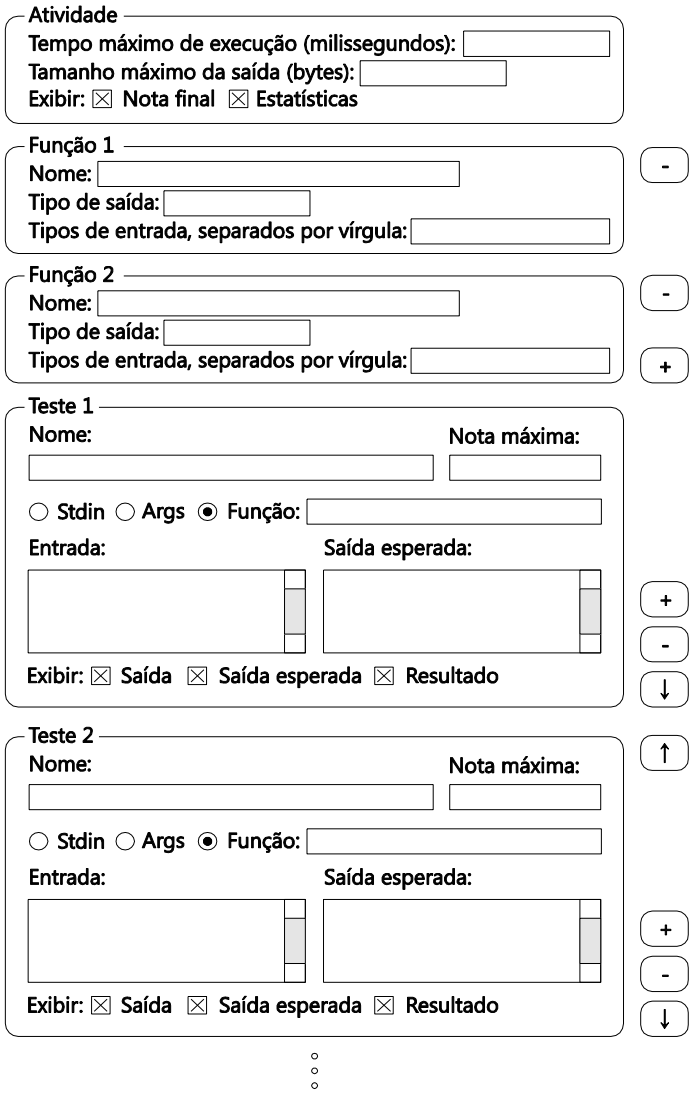

Figura 13 - Maquete do editor de configuração

Outra melhoria possível é a da construção de um editor específico para os casos de teste, hoje embutidos em um arquivo XML. A fim de manter a compatibilidade com o módulo Sakai, seria importante a possibilidade de obter o arquivo XML posteriormente, gerado automaticamente dos campos. Uma maquete deste editor pode ser vista na figura 13.

3.5.4 Suporte a múltiplos arquivos

Atualmente o programa enviado pelo aluno deve estar contido integralmente em um único arquivo. Como a aplicação inicial da ferramenta é para cursos básicos, esta limitação não é tão significativa. Porém, estar preparado para trabalhar com múltiplos arquivos é um pré-requisito importante para cursos mais avançados.

3.5.5 Repositório de atividades

Para evitar que a cada semestre todas as atividades tenham de ser cadastradas manualmente, pode existir um repositório de atividades previamente cadastradas, ou ser possível recuperar uma atividade cadastrada em outro curso, mesmo já encerrado. Poderia ser possível também ao aluno escolher uma atividade para praticar de um repositório, sem que o professor tenha que explicitamente cadastrá-la no curso.

3.5.6 Gravação automática e Ajax

Para melhorar a usabilidade do plugin Moodle, seria útil se o código sendo editado online fosse gravado de tempos em tempos, para evitar casos onde, por algum imprevisto, a janela do navegador se fechasse e o documento fosse perdido. Também é possível utilizar chamadas assíncronas para atualizar o relatório, sem a necessidade de recarregar a página.

4 Testes e resultados

4.1 Exercícios

O corretor pode ser usado para diferentes tipos de exercícios de programação. Com o uso da ferramenta, percebemos que é importante que o enunciado, além da descrição do problema, enumere os pré-requisitos para a resolução do exercício, como capítulos de apostilas ou livros, forneça alguns exemplos de entrada e saída, principalmente exemplos de como não deve ser feito, e algumas dicas para a resolução, como pseudocódigo e erros de compilação comumente encontrados.

Para serem adequados, os exercícios devem ser elaborados com testes que cubram o maior número de situações possível. Alguns dos testes devem ter entrada e saída escondidas, para evitar programas que tentem burlar o corretor com artifícios, como saídas condicionadas. É importante que o professor acompanhe os exercícios sendo entregues, inclusive antes do prazo final, para identificar falhas nos testes ou situações não previstas. E ao final do prazo verificar manualmente todos os exercícios, procurando técnicas não permitidas e avaliando e comentando aspectos não corrigidos automaticamente, como a mantenibilidade do código.

Iremos enumerar a seguir alguns tópicos que podem ser aplicados a um curso de introdução à programação, conforme os cursos ministrados pelo IME-USP.

4.1.1 Condições e Repetições

Os exercícios mais simples envolvem condições (*if/case*) e repetições (*for/while*). Um exemplo de exercício que introduz estes conceitos é o de *Cálculo de potência.* Nele, o aluno deve ler a base e o expoente e retornar a potência. O cálculo é feito com a repetição da multiplicação da base, e o contador pode ser crescente ou decrescente. O aluno deve avaliar corretamente os valores iniciais e a condição de parada.

Outro exemplo é o exercício para determinar o *Máximo de uma sequência.* O programa deve ler os números através de repetições e, por meio de expressões condicionais, verificar qual o maior número. Na entrada, o primeiro número indica a quantidade de números, seguido dos números propriamente ditos. A saída esperada é o maior número da sequência.

Estes dois exercícios podem ser feitos alternativamente utilizando recursão. Neste caso o programa corretor irá considerar o resultado válido, pois apenas entrada e saída são observadas. Cabe ao professor analisar o código e avaliar se deve aceitar ou não a solução.

4.1.2 Números reais

No enunciado deve estar claro que qualquer número real deve ser impresso com pelo menos quatro dígitos de precisão. O corretor reconhece números e ignora diferenças menores de quatro casas decimais.

Um exercício possível é para cálculo da *Média.* Uma sequência de números é lida e o programa retorna o valor médio calculado, com quatro casas de precisão. Outro exercício é o cálculo do *Arco tangente,* realizado utilizando uma série convergente infinita. O cálculo para quando for atingida a precisão de quatro casas decimais.

Mais uma vez, como o corretor analisa apenas entrada e saída, o aluno poderia usar uma função de uma biblioteca para calcular o arco tangente e ainda assim seria atribuída a nota máxima. Para evitar esse tipo de situação, o professor deve ficar atento aos códigos enviados. Outra saída é pedir para ser calculada uma variação da série convergente, cujo cálculo não exista nas bibliotecas.

4.1.3 Ponteiros e funções

Para avaliar o conhecimento de ponteiros é interessante introduzir funções e passagem de parâmetros por valor ou referência. É possível fazer variações dos problemas anteriores, evidenciando que o resultado deve vir de uma função que receba parâmetros por valor ou referência. Nesse caso, o código precisa ser revisado manualmente para certificar que o aluno utilizou o método solicitado.

Outra possibilidade é usar o avaliador de funções do corretor, onde a função precisa possuir uma assinatura especificada no enunciado, com nome, número de parâmetros, tipos dos parâmetros e tipo de retorno.

4.1.4 Caracteres e strings

Depois de introduzido o conceito de ponteiro, já é possível falar de cadeias de caracteres. Dentre os exercícios que podem ser solicitados estão *Inversão de cadeia*, onde o resultado deve ser a entrada de trás para frente, *Inversão de palavras*, onde deve-se inverter uma sequência de palavras separadas por espaço, entre outros. Ao redigir um exercício deve-se atentar ao fato que o corretor despreza a diferença entre maiúsculas e minúsculas, assim como espaços extras. Portanto um exercício que considere estas diferenças pode ter testes bem sucedidos mesmo em programas incorretos.

4.1.5 Vetores

Uma maneira de entrar um vetor no programa é especificando que o primeiro número lido denomina quantos valores serão lidos, seguido dos valores em si. Por exemplo, o vetor [1, 12, 4] será ingressado como "3 1 12 4". Diversos exercícios podem ser feitos, como por exemplo, *Eliminar repetições* em um vetor não ordenado lido. A saída deve ser o vetor com apenas a primeira ocorrência de cada número.

4.1.6 Matrizes

Já que uma string é um vetor de caracteres, um vetor de strings é uma matriz. Assim, para trabalhar com matrizes podemos utilizar programas que trabalham com vetores de strings. Para a entrada, é lido um número que especifica o total de strings a serem lidas, e depois os strings, separados por espaço. Um exercício possível é o de *Ordenação de palavras*, onde uma lista de palavras deve ser impressa ordenada na saída.

4.2 Testes

4.2.1 Primeiro ciclo

O primeiro ciclo de testes, utilizando a versão modificada da ferramenta de atividades para o Sakai, foi aplicado em uma turma de 46 alunos da disciplina de introdução à programação ministrada pelo IME-USP. Ao todo foram disponibilizadas 28 atividades, divididas em 13 etapas. A primeira era apenas para o aluno se familiarizar com a

ferramenta, e a segunda era a de *Cálculo de potência*. Esta foi entregue por 31 alunos, e 28 obtiveram nota máxima. A assiduidade foi diminuindo ao longo do curso para 23 submissões na quinta etapa, e apenas 15 na última. Este número já era esperado, por se tratar de um reoferecimento da disciplina oferecida para alunos repetentes. Para acompanhar estes números, observe o gráfico nas figuras 14 e 15.

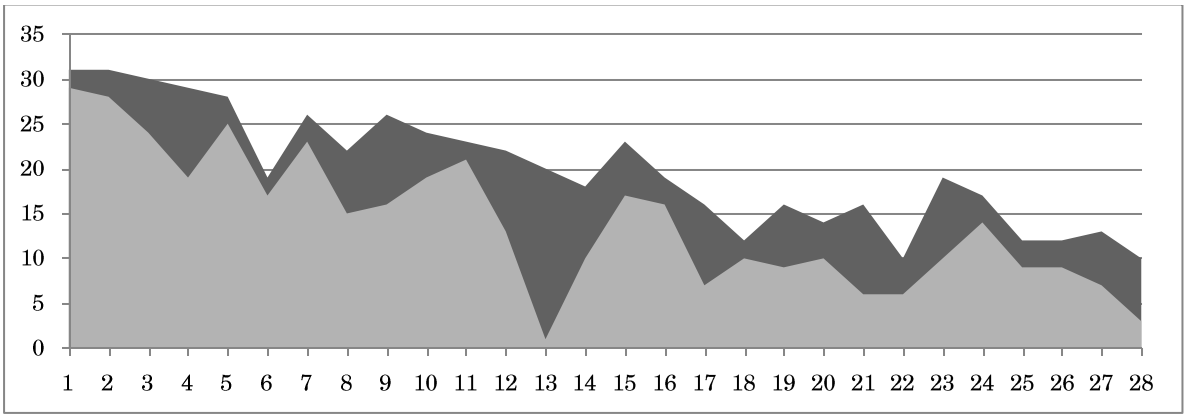

Figura 14 - Total de submissões (cinza escuro) e submissões com nota máxima (cinza claro) para cada exercício

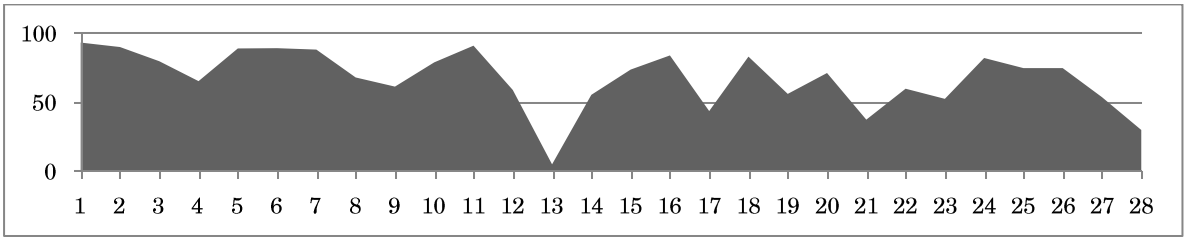

Figura 15 - Percentagem de submissões com a nota máxima para cada exercício

Como podemos ver, apesar do número de alunos participantes ao longo do curso ter diminuído, a proporção de atividades resolvidas com sucesso foi razoavelmente constante, o que mostra que boa parte dos alunos conseguiram entregar seus trabalhos com sucesso.

4.2.2 Segundo ciclo

Com o plugin moodle desenvolvido, será possível aplicar o mesmo teste a uma turma semelhante. A principal diferença será na agilidade, uma vez que o processo de resolução do exercício está bem mais rápido. O tempo de espera pela resposta do corretor foi cortado imensamente, e a possibilidade de editar o código no próprio navegador, com suporte a realce de sintaxe, deve se mostrar de grande utilidade.

5 Conclusão

Com a ferramenta desenvolvida para o Moodle, foi possível utilizar a ampla aceitação deste software, unido a seu contínuo desenvolvimento, qualidade e segurança, para atingir um grande número de potenciais usuários do corretor. Assim como o Moodle, nossa ferramenta é gratuita, de fácil instalação e utilização, e estará disponível para qualquer pessoa com acesso a internet.

Para o aluno, uma interface unificada reúne o enunciado, o editor com realce de sintaxe, o relatório e a nota em uma única página. Isso deixa mais rápida e confortável a resolução do exercício. O relatório do corretor é organizado, retorna quase instantaneamente e possui mensagens de compilação traduzidas e comentadas. Isso auxilia o aluno a focar na resolução do problema, com menos distrações, aumentando sua autoconfiança.

Para o professor, a tela de acompanhamento de exercícios traz uma visão geral de como os alunos estão utilizando a ferramenta em uma interface única. Aí é possível comentar, revisar e alterar as notas, mesmo antes do término do prazo, facilitando a revisão dos exercícios corrigidos automaticamente. Esta revisão é importante para recompensar trabalhos que não passam nos testes, mas foram feitos parcialmente. E para punir aqueles que passam nos testes, mas não seguem o enunciado, usam técnicas não aceitas ou possuem características não desejadas.

O método de comparação de resultados usado pelo corretor permite grande flexibilidade na definição de soluções, tolerando variações como diferença de precisão numérica e textos explicativos. E a possibilidade de testar ou a saída do programa como um todo, ou o retorno de funções específicas, permite uma grande versatilidade de tipo de teste que

pode ser criado. A criação de exercícios através da configuração de um arquivo XML possibilita transpô-lo facilmente para o Sakai.

A segurança é maior com o corretor executando os exercícios em uma máquina virtual isolada, acessível apenas por uma porta HTTP. Assim, os recursos do servidor onde o LMS é executado não podem ser acessados pelos programas enviados pelos alunos. A utilização de tecnologias de código aberto ativamente desenvolvidas também contribui para a segurança geral do ambiente.

Com sua construção modular, é fácil estender a ferramenta para outras necessidades, e atender um número maior de disciplinas e linguagens de programação.

A aplicação da ferramenta em cursos introdutórios de computação no IME-USP conseguiu atender todos os tópicos do curso, com a maioria dos alunos obtendo nota máxima em todos os exercícios. Isso mostra que a ferramenta está pronta para trazer os benefícios da realização frequente de exercícios, da correção automática e da integração com o Moodle para qualquer pessoa, curso ou instituição.

6 Referências Bibliográficas

Aiken, A. (1998). *Moss (measure of software similarity) plagiarism detection system*.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Berry, R. (1966). *Grader programs*. The Computer Journal, v. 9, n. 3, p. 252.

Botev, C., Chao, H., Chao, T., Cheng, Y. & Doyle, R. (2005). *Supporting workflow in a course management system*. ACM SIGCSE, v. 37, n. 1, p. 262.

Braden, R. T. & Perlis, A. J. (1965). *An introductory course in computer programming*.

Carrington, D., Robinson, K. & Whale, G. (1984). *Give: a system for collecting and testing student assignments*. Em Proceedings of 7th Australian Computer Science Conference, p. 21–1.

Cheang, B., Kurnia, A. & Lim, A. (2001). *Online judge*. Computers & Education, v. 36, n. 4, p. 299–315.

Cheang, B., Kurnia, A., Lim, Andrew & Oon, W. C. (2003). *On automated grading of programming assignments in an academic institution*. Computers & Education, v. 41, n. 2, p. 121-131.

Cheung, K. (2007). *A Comparison of WebCT, Blackboard and Moodle for the teaching and learning of continuing education courses*. Enhancing learning through technology, p. 219–228.

Computer-history.info (2011). *Computer History*. http://computer-history.info, [acessado em 12 Dez 2011].

Daly, C. (1999). *RoboProf and an introductory computer programming course*. Em Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, p. 155–158.

Daly, C. & Waldron, J. (2004). *Assessing the assessment of programming ability*. Em Proceedings of the 35th SIGCSE technical symposium on Computer science education, p. 210–213.

Darbhamulla, R. & Lawhead, P. (2004). *Paving the way towards an efficient Learning Management System*. Em Proceedings of the 42nd annual Southeast regional conference, p. 428–433.

Deursen, A. V., Mesbah, A. & Cornelissen, B. (2010). *Adinda: a knowledgeable, browser-based IDE*. Proceedings of the 32nd ACMIEEE International Conference on Software Engineering, v. 2, p. 203-206.

Douce, C., Livingstone, D. & Orwell, J. (2005). *Automatic test-based assessment of programming: A review*. Journal on Educational Resources in Computing (JERIC), v. 5, n. 3, p. 4.

English, J. (2004). *Automated assessment of GUI programs using JEWL*. Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, v. 36, n. 3, p. 137–141.

Forsythe, G. E. & Wirth, N. (1965). *Automatic grading programs*. Communications of the ACM, v. 8, n. 5, p. 275–278.

Foubister, S. P., Michaelson, G. J. & Tomes, N. (1997). *Automatic assessment of elementary Standard ML programs using Ceilidh*. Journal of Computer Assisted Learning, v. 13, n. 2, p. 99-108.

Fuks, H. & Cunha, L. M. (2003). *Participação e Avaliação no Ambiente Virtual AulaNet da PUC-Rio.*

Georgouli, K. & Guerreiro, P. (2011). *Integrating an Automatic Judge into an Open Source LMS*. International Journal on E-Learning, v. 10, n. 1, p. 27-42.

Ginns, P. & Ellis, R. (2007). *Quality in blended learning: Exploring the relationships between on-line and face-to-face teaching and learning*. The Internet and Higher Education, v. 10, n. 1, p. 53-64.

Goldman, M. & Little, G. (2011). *Real-time collaborative coding in a web IDE*. Proceedings of the 24th annual ACM,

Hext, J. B. & Winings, J. W. (1969). *An automatic grading scheme for simple programming exercises*. Communications of the ACM, v. 12, n. 5, p. 272–275.

Higgins, C., Hegazy, T., Symeonidis, P. & Tsintsifas, A. (2003). *The CourseMarker CBA System : Improvements over Ceilidh*. Education and Information Technologies, v. 8, n. 3, p. 287-304.

Hollingsworth, J. (1960). *Automatic graders for programming classes*. Communications of the ACM, v. 3, n. 10, p. 528-529.

Ilias.de (2011). *Ilias Booklet*. http://ilias.de/docu/repository.php?ref_id=1854, [acessado em 12 Dez 2011].

Isaacson, P. C. & Scott, T. A. (1989). *Automating the execution of student programs*. ACM SIGCSE Bulletin, v. 21, n. 2, p. 15–22.

Jackson, D. (2000). *A semi-automated approach to online assessment*. Em Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education, p. 164–167.

Jackson, D. & Usher, M. (1997). *Grading student programs using ASSYST*. ACM SIGCSE Bulletin, v. 29, n. 1, p. 335–339.

Janković, I. & Gledec, G. (2010). *WebIDE - online tool for remote teaching and programming*. Proceedings of the 33rd MIPRO International Convention on Computers in Education, p. 385-388.

Jianxia, C., Qianqian, L., Lin, C. Y. ., Huapeng, C. & Chunzhi, W. (2011). *Application of innovative technologies on the e-learning system*. Em 2011 6th International Conference on Computer Science & Education (ICCSE), p. 1033-1036.

Joy, M. & Luck, M. (1998). *Effective electronic marking for on-line assessment, Effective electronic marking for on-line assessment*. ACM SIGCSE Bulletin, v. 30, n. 3, p. 134-138.

Korhonen, A. & Malmi, L. (2000). *Algorithm simulation with automatic assessment*. Em ACM SIGCSE Bulletin, v. 32, p. 160–163.

Kosowski, A., Malafiejski, M. & Noinski, T. (2008). *Application of an online judge & contester system in academic tuition*. Advances in Web Based Learning–ICWL 2007, p. 343–354.

Kosowski, A., Małafiejski, M., Noiński, T. & Pomykalski, P. (2005). *An integrated system for the automated assessment of solutions to algorithmic problem s applie in university tuition: Sphere Online Judge*. Zeszyty Naukowe, v. 7, p. 523–528.

Kumar, S., Gankotiya, A. K. & Dutta, K. (2011). *A comparative study of moodle with other e-learning systems*. Em Electronics Computer Technology (ICECT), v. 5, p. 414–418.

Lukashenko, R., Graudina, V. & Grundspenkis, J. (2007). *Computer-based plagiarism detection methods and tools*. Proceedings of the 2007 international conference on Computer systems and technologies - CompSysTech '07, p. 1.

Malikowski, S., Thompson, M. & Theis, J. (2007). *A Model for Research into Course Management Systems: Bridging Technology and Learning Theory*. Journal of Educational Computing Research, v. 36, n. 2, p. 149-173.

Malinowski, A. & Wilamowski, B. M. (2000). *Web-based C++ Compiler*. Em Proceedings of the ASEE 2000 Annual Conference, 18th–21st June, St. Louis, MO, USA, p. 1–7.

Matt, U. Von (1994). *Kassandra: the automatic grading system*. ACM SIGCUE Outlook, v. 22, n. 1, p. 26-40.

Moodle.org (2011). *Moodle.org: Moodle Statistics*. http://moodle.org/stats/, [acessado em 6 Dez 2011].

Morris, D. S. (2003). *Automatic grading of student's programming assignments: an interactive process and suite of programs*. Em Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference, v. 3, p. S3F-1-6.

Naur, P. (1964). *Automatic grading of students' ALGOL programming*. BIT 4, p. 177-188.

Palvai, T. (2010). *Web-Based IDE for Interfacing View Controller*.

Pardoe, J. & Vickers, P. (1994). *Using a prototype program assessment tool*. Em 2nd All-Ireland Conference of Teaching of Computing, p. 232–236.

Pino, J. C. D., Royo, E. R. & Figueroa, Z. J. H. (2010). *VPL : Laboratorio Virtual de Programación para Moodle*.

Prechelt, L., Malpohl, G. & Philippsen, M. (2002). *Finding Plagiarisms among a Set of Programs with JPlag*. Journal of Universal Computer Science, v. 8, n. 11, p. 1016-1038.

Rahman, K. A., Ahmad, S. & Nordin, M. J. (2007). *The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique*. Em National Conference on Software Engineering and Computer Systems, at University Malaysia Pahang,

Reek, K. A. (1989). *The TRY system -or- how to avoid testing student programs*. SIGCSE Bull., v. 21, n. 1, p. 112–116.

Revilla, M., Manzoor, S. & Liu, R. (2008). *Competitive learning in informatics: The UVa online judge experience*. Olympiads in Informatics, v. 2, p. 131–148.

Saikkonen, R., Malmi, L. & Korhonen, A. (2001). *Fully automatic assessment of programming exercises*. Em ACM SIGCSE Bulletin, v. 33, n. 3, p. 133–136.

Stallman, R. M. & Pesch, R. H. (1991). *Using GDB: A guide to the GNU source-level debugger*. Free software foundation.

Temperly, J. & Smith, B. W. (1968). *A grading procedure for PL/1 student exercises*. The Computer Journal, v. 10, n. 4, p. 368.

Trent, S., Tatsubori, M., Suzumura, T., et al. (2008). *Performance Comparison of PHP and JSP as Server-Side Scripting Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg. v. 5346p. 164-182

Truong, N., Roe, P. & Bancroft, P. (2005). *Automated feedback for fill in the gap programming exercises*. Em Proceedings of the 7th Australasian conference on Computing education-Volume 42, p. 117–126.

Wainwright, K., Osterman, M., Finnerman, C. & Hill, B. (2007). *Traversing the LMS terrain*. Em Proceedings of the 35th annual ACM SIGUCCS fall conference, p. 355–359.

Wang, T., Su, X., Ma, P., Wang, Y. & Wang, K. (2011). *Ability-training-oriented automated assessment in introductory programming course*. Computers & Education, v. 56, n. 1, p. 220-226.

Webber, A. B. (1996). *The Pascal Trainer*. Em Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, p. 261–265.

Wise, M. J. (1996). *YAP3: Improved detection of similarities in computer program and other texts*. Em SIGCSE '96 Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, v. 28, p. 130–134.

Yueh, H. (2008). *Designing a learning management system to support instruction*. Communications of the ACM, v. 51, n. 4, p. 59-63.

Zamin, N., Mustapha, E. E., Sugathan, S. K. & Mehat, M. (2006). *Development Of A Web-Based Automated Grading System For Programming Assignments Using Static Analysis Approach*. Em International Conference on Electrical and Informatics, Bandung, Indonesia,