Revista Brasileira de Informática na Educação, Volume 21, Número 1, 2013

Sistema de apoio a atividades de laboratório de programação via Moodle com suporte ao balanceamento de carga e análise de similaridade de código

Title: System to support laboratory activities of programming via Moodle with support for load balancing and similarity analysis code

Allyson Bonetti França
Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática (DETI)
allysonbonetti@gmail.com

Danilo Leal Maciel
Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática (DETI)
daniloleal@alu.ufc.br

José Marques Soares
Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática (DETI)
marques@ufc.br

Resumo *Visando instrumentalizar o professor para o melhor acompanhamento de turmas numerosas de disciplinas de programação, este trabalho apresenta um ambiente que integra e adapta ferramentas Web para avaliar os programas implementados por alunos, bem como fazer inferências sobre a similaridade entre os códigos desenvolvidos. Com este sistema, alunos submetem e validam suas soluções e, adicionalmente, professores podem executar algoritmos de pré-processamento e comparação para dar suporte à análise de similaridade entre essas soluções. Todas as operações são realizadas por meio da interface do ambiente Moodle. Considerando que a compilação e execução concorrente de grande quantidade de programas podem exigir alta disponibilidade de recursos computacionais, o ambiente desenvolvido oferece suporte ao balanceamento de carga.*

Palavras-Chave: *Análise de similaridade, detecção de plágio, laboratório virtual, avaliação automática de código-fonte*

Abstract *Aiming to instrumentalize the teacher to make a better monitoring of large classrooms of programming disciplines, this work presents an environment that integrates and adapts Web tools to evaluate programs implemented by students, as well as making inferences about the similarity between the codes developed. With this system, students can submit and validate their solutions and, in addition, teachers can execute preprocessing and matching algorithms to support analysis of similarity between these solutions. All operations are performed through the interface of the Moodle environment. Whereas the compilation and concurrent execution of large number of programs may require high availability of computational resources, the developed environment supports load balancing.*

Keywords: *Similarity analysis, plagiarism detection, virtual laboratory, automatic assessment of source code*

Recebido: 14 de Maio de 2013 / Aceito: 25 Junho de 2013
DOI: 10.5753/RBIE.2013.21.01.91

1 Introdução

Disciplinas de técnicas de programação em cursos de computação e engenharia são, em geral, muito numerosas, exigindo bastante do professor e dos monitores que, muitas vezes, não conseguem realizar um acompanhamento individualizado de maneira eficiente. Isto pode provocar desestímulo, impelindo a turma, por vezes, à dispersão em aulas de laboratório, situação dificilmente controlável pelo professor. Conforme constata-se por Essi [1] e Tan [2], ainda é um desafio obter um bom rendimento em tais turmas, que, de modo geral, possuem altos índices de evasão e de reprovação.

Professores podem fazer uso de ferramentas de suporte para o seu trabalho. Aplicações Web, frequentemente, são usadas para disponibilização de notas de aula, proposição e submissão de trabalhos e registro de notas. Entretanto, embora o uso de tais ferramentas possa mitigar os problemas de natureza organizacional em práticas laboratoriais, não são suficientes para solucionar a dificuldade de acompanhamento e orientação aos alunos.

Para ilustrar situações comuns em laboratórios de programação, cita-se uma questão frequentemente colocada por alunos:

Professor, o meu programa está correto?

Para responder a esta questão, é necessário que o professor se desloque até o aluno no laboratório, observe a execução do programa, verifique o seu resultado e, eventualmente, analise o código desenvolvido pelo aluno. Em caso de erro, muitos alunos assumem posturas passivas e aguardam que o professor o descubra. Em uma turma de 60 alunos, por exemplo, essa atividade de simples verificação pode tornar o tempo de aula insuficiente.

Uma maneira de reduzir significativamente esse trabalho é permitir que o próprio aluno valide o resultado de seu programa em um procedimento semelhante ao realizado em maratonas de programação. Nesta perspectiva, visando contribuir com as condições de ensino e aprendizagem de cursos de programação, é apresentado neste trabalho um ambiente que permite a automatização de avaliações de programas propostos pelo professor para desenvolvimento nas linguagens de programação C, C++ e Java. O objetivo é, por um lado, fornecer ao professor uma ferramenta que permita o gerenciamento de seus recursos didáticos e que lhe dê apoio ao acompanhamento das práticas laboratoriais. Por outro lado, objetiva-se permitir ao aluno um *feedback* mais rápido, que o incentive a um comportamento mais autônomo.

Adicionalmente, é definido um modelo de integração de ferramentas, que é voltado especificamente para a avaliação de programas, aos chamados ambientes virtuais de aprendizagem (AVA). Os recursos e funcionalidades das ferramentas integradas são oferecidos aos usuários de forma complementar, através de uma interface única e coesa. Para a composição e avaliação do modelo de integração, adotou-se uma metodologia que se baseia no conceito de arquitetura orientada a serviços (*Service Oriented Architecture* – SOA) [3].

O ambiente desenvolvido para apoio a laboratórios de programação foi concebido como extensão do sistema BOCA [4]. Desenvolvido na Universidade de São Paulo (USP) para dar suporte ao julgamento de trabalhos desenvolvidos em maratonas de programação, permite submissão e avaliação automática de soluções para os problemas apresentados aos concorrentes, o BOCA precisou ser adaptado para atender a necessidades específicas ao trabalho em laboratórios de programação. Para viabilizar a sua integração aos demais recursos do ambiente de integração, foi desenvolvida uma camada de software para a exposição de suas funcionalidades em forma de serviços. Além disso, foi desenvolvido uma infraestrutura para dar suporte ao balanceamento de carga, visto que a solução para alguns problemas propostos podem apresentar carga computacional considerável para execução em um único servidor, levando-se em conta a complexidade da solução, o número de alunos e a quantidade de turmas com trabalhos concorrentes. O sistema estendido é denominado neste trabalho BOCA-LAB.

Com a finalidade de monitorar as submissões semelhantes, o sistema integrado oferece também ao professor uma ferramenta específica para oferecer suporte à análise de similaridade entre códigos de alunos, pois um problema não raramente encontrado em laboratórios de programação é a copia total ou parcial de soluções entre colegas, frequentemente com a mudança de nomes de variáveis ou com a inserção de comentários para tornar mais difícil a percepção da ação. Em cenários de turmas numerosas, a detecção deste tipo de conduta se torna bastante difícil.

Embora a detecção de plágio seja um aspecto importante na condução de uma turma de programação, a experiência trazida com a utilização da ferramenta de análise de similaridade revelou que os resultados da comparação entre códigos podem indicar outros aspectos que são de natureza irrepreensível. Dependendo da perspectiva e do contexto, a semelhança pode ser indicativa de parceria, de trabalho colaborativo, de referência a uma solução encontrada em livros ou exemplos fornecidos pelo professor, entre outros motivos. Por essa razão, ao invés de "detecção de plágio", adota-se neste trabalho a expressão "análise de similaridade".

A detecção de plágio (e a análise de similaridade) em código-fonte vem sendo estudada em diversos trabalhos [5] [6] [7] e algumas ferramentas foram disponibilizadas para este fim [8] [9]. No ambiente de integração proposto neste trabalho, avalia-se especialmente a sensibilidade do algoritmo Sherlock [10] quando os códigos são submetidos a técnicas de pré-processamento (denominadas "normalização") que ressaltam características especificas dos códigos antes da comparação. O objetivo é remover aspectos irrelevantes, como comentários, valores literais ou nomes de variáveis, destacando os aspectos estruturais do código. Os resultados apresentados com a aplicação das técnicas de normalização e uso do algoritmo Sherlock no ambiente integrado são confrontados com aqueles obtidos com o uso do JPlag [8] e Moss [9], ferramentas destinadas ao controle de plágio e amplamente discutidas [26].

O BOCA-LAB, nome dado ao ambiente integrado, foi implantado no Moodle [11], que forneceu a interface e o conjunto de funcionalidades necessárias à gestão e ao acompanhamento das atividades associadas ao laboratório de programação. A integração se apóia no uso de *Web Services* (WS), que se destacam como tecnologia para a implementação de SOA e vêm sendo utilizados em sistemas educacionais como o Sakai [12].

O texto está disposto da seguinte forma: a seção 2 aborda os trabalhos relacionados, apresentando soluções que buscam essa automatização na correção e avaliação de códigos fontes em sistemas de cunho educacional; a seção 3 mostra as características do Moodle, do BOCA [4] e do Sherlock [10], que são as ferramentas de base que compõem o ambiente de integração; na seção 4 é mostrada a arquitetura de integração. A interação entre os usuários e a arquitetura é explicada na seção 5. Na seção 6 discute-se a experiência de utilização do ambiente e apresenta-se os testes realizados, por último, são apresentadas as conclusões e perspectivas do trabalho.

2 Trabalhos Relacionados

2.1 Ambientes Virtuais

O uso de ambientes virtuais para dar suporte a atividades de programação vem sendo estudado há alguns anos.

Ng [13] investiga como a nova tecnologia pode auxiliar no processo de ensino e aprendizagem em disciplinas de programação, propondo um ambiente Web interativo para o ensino de linguagens de programação Java. O ambiente apresenta funcionalidades que permitem a compilação e o retorno de erros dos programas submetidos.

Wang [14], propõe um sistema Web para o ensino da linguagem de programação C. Esse sistema é desenvolvido em .NET e oferece funcionalidades que permitem a compilação e checagem de erros dos programas submetidos.

O Sistema Susy [15] é utilizado pelos alunos do Instituto de Computação, na Unicamp, para a submissão e o teste automático das atividades submetidas. Não apresenta integração com AVAs e possui código fechado.

Pode-se citar ainda, as ferramentas BOSS [16], *Programming Assignment aSsessment System* (PASS) [17] e Praktomat [18],que também são ferramentas para submissão e avaliação remota de código. As características e limitações específicas de cada ferramenta referenciada são detalhadas na Tabela 1.

Embora os trabalhos correlatos apresentem plataformas interativas para o ensino de linguagem de programação, eles não oferecem, em um ambiente integrado, outros recursos de apoio ao processo de ensino e aprendizagem, como ferramentas de discussão síncronas e assíncronas, suporte a gestão de conteúdo, entre outros recursos importantes, principalmente para disciplinas que possuam algum suporte a distância. Outro limite de algumas das plataformas citadas é restringir o suporte a apenas um tipo de linguagem de programação.

Em um contexto mais aproximado ao trabalho aqui apresentado, algumas iniciativas foram realizadas no sentido de integrar recursos de apoio a disciplinas de programação ao ambiente Moodle, como o VPL [19] e o Onlinejudge [20]. O VPL (*Virtual Programming Lab*) é uma ferramenta de código aberto que permite o desenvolvimento remoto de programas através de um módulo acoplado ao Moodle. A edição do código é feita através de um *applet* e a compilação e execução do código é realizada com segurança em um servidor Linux remoto. É possível efetuar a compilação em várias linguagens de programação, dentre elas C, C++, PHP, Java e Fortran. Para a correção e compilação de códigos fonte, este módulo necessita, a cada atividade cadastrada pelo professor, da configuração de como serão os processos de compilação de códigos fonte e de correção automática. A arquitetura utilizada pelo VPL não permite a adição de novas ferramentas ou o balanceamento de carga, visto que o servidor responsável pela compilação e execução do código submetido é único. A centralização deste aspecto pode se tornar um gargalo uma vez que podemos ter, em um mesmo servidor Moodle, várias turmas contendo dezenas de alunos submetendo soluções simultaneamente.

O Onlinejudge [20], também desenvolvido para gerenciar a submissão de códigos fontes adicionado ao Moodle, pode ser integrado com o uso de WS a uma aplicação denominada Ideone [21]. Essa aplicação permite escrever códigos fonte em aproximadamente 40 lin-

guagens de programação diferentes, sendo os mesmos executados diretamente a partir do navegador. O Onlinejudge também pode ser executado sem a integração com o Ideone, dando suporte, nesse caso, apenas às linguagens C e C++. Entretanto, como se trata de uma aplicação comercial e de código fechado, o modelo de integração permite a submissão de apenas 1000 códigos fonte por mês em uma conta gratuita e não aceita a submissão de vários códigos fonte por vez.

Apesar de todos os trabalhos citados conterem importantes contribuições para o apoio a práticas laboratoriais em turmas de programação, neste trabalho propõe-se um ambiente de auxílio a compilação e execução remota de programas que seja capaz de reunir as seguintes características: (i) ser integrado a um ambiente virtual de aprendizagem, permitindo o seu uso e o acompanhamento de resultados através da mesma interface de outras ferramentas disponíveis no ambiente virtual; (ii) dar suporte ao uso de diversas linguagens de programação; (iii) permitir a gestão de múltiplos servidores e executar o balanceamento de carga entre os servidores disponíveis; (iv) permitir análise de similaridade entre os códigos enviados, permitindo a relação entre as soluções de alunos ou, dependendo da situação, auxiliando na identificação de plágios.

Funcionalidade	BOCA-LAB	Sistema Susy	BOSS	PASS	Praktomat	VPL	Onlinejudge + Ideone
Submissão de Atividades	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Suporte para integração com AVAs	Sim	Não é possível determinar	Não	Não	Não	Sim	Sim
Detecção de Similaridade	Sim	Não é possível determinar	Sim	Sim	Não	Sim	Não
Escalabilidade (Balanceamento de Carga)	Sim	Não é possível determinar	Não	Não	Não	Não	Não
Código aberto	Sim	Não	Sim	Não	Sim	Sim	Não
Suporta ao menos 2 linguagens	Sim	Sim	Sim	Sim	Não	Sim	Sim
Análise estatística	Não	Não é possível determinar	Não	Sim	Não	Não	Não

Tabela 1: Comparação dos trabalhos relacionados

2.2 Detecção de Plágio

Diversas ferramentas podem ser utilizadas para realizar a análise de similaridade entre códigos-fonte, tais como SIM [22], YAP [23], JPlag [8], SID [24], Plaggie [25] e Moss [9]. Estudos comparativos sobre algumas destas ferramentas podem ser encontrados em [5]. De modo geral, ferramentas especificamente construídas para esta finalidade permitem encontrar semelhanças no caso de alterações de nome de variáveis, nome de funções, comentários ou, ainda, alterando-se a ordem de partes do código. Segundo Burrows [26], JPlag e MOSS são algumas das mais importantes e mais citadas para detecção de plágio, que são sucintamente descritos nos próximos parágrafos, seguidos pelo Sherlock [10], sobre o qual se apóiam as contribuições deste trabalho para a análise de similaridade.

O JPlag [8] é uma ferramenta desenvolvida em Java disponibilizada exclusivamente por meio de um WebService, possuindo código fechado. Ela requer o cadastramento e a requisição de autorização para o acesso ao serviço. O envio de arquivos para comparação é realizado através de um *applet* que retorna, em formato HTML, o resultado do processamento. A semelhança entre códigos é apresentada apenas para os com maior percentual de similaridade, não sendo, portanto, disponibilizada a visualização de pares com porcentagens pequenas.

O MOSS (*Measure of Software Similarity)* [9] também é acessado por meio de um WebService, disponibilizado na Universidade de Califórnia. Para usá-lo, é necessário realizar um cadastramento por *e-mail*. Os arquivos são enviados para o servidor por meio de um *script* fornecido pelo seu desenvolvedor. Ao final do envio, é gerada uma URL que identifica a página que apresenta o resultado da comparação. Segundo informações obtidas na página da aplicação, o desenvolvimento do MOSS iniciou-se em 1994, apresentando uma melhora significativa de outros algoritmos, que não são mencionados, para detecção de plágio. O MOSS, que também é de código fechado, é baseado no algoritmo Winnowing [27].

O Sherlock [10] representa uma alternativa às ferramentas apresentadas precedentemente por ser de código aberto, permitindo modificações, melhorias e integração a outros ambientes, permitindo que o BOCA-LAB tenha o controle total de seus recursos sem exigir conectividade a Web Services de terceiros. O Sherlock [10] realiza a análise de semelhança léxica entre documentos textuais, inclusive para códigos-fonte. Para encontrar trechos duplicados de cada documento, é gerada uma assinatura digital que calcula valores *hash* para palavras e sequência de palavras. Ao final, comparam-se as assinaturas geradas e identifica-se o percentual de semelhança.

Além das características citadas, o Sherlock [10] possui um bom desempenho por ser desenvolvido em C. Outra vantagem é ser de propósito geral, enquanto todas as outras ferramentas são configuradas para a comparação em linguagens de programação específicas.

3 Ferramentas de Base: Moodle, BOCA e Sherlock

3.1 O Moodle como interface do ambiente de integração

O Moodle (*Modular Object Oriented Distance Learning*) é um sistema de código aberto baseado na Pedagogia Social Construcionista [28]. Rico em recursos educacionais, oferece alta flexibilidade para configuração e uso. Seu desenvolvimento modular permite a fácil inclusão de novos recursos que podem melhor adaptá-lo às necessidades da instituição que o utiliza. Por ser um ambiente extensível e completo em termos de recursos para gerenciamento de atividades educacionais, o Moodle apresenta-se como ambiente propício para integrar ferramentas que dêem suporte ao processo de ensino e aprendizagem em disciplinas de programação.

3.2 O BOCA como recurso para Compilação e Verificação de resultados para Problemas de Programação

O BOCA [4] é um sistema de apoio a competições de programação desenvolvido para uso em maratonas promovidas pela Sociedade Brasileira de Computação. Oferece suporte *online* durante a competição, gerenciando times de alunos e juízes, permitindo a proposição de problemas de programação bem como a submissão e avaliação automática de soluções. Sendo um sistema de código aberto, o BOCA pode ser adaptado ao contexto de laboratórios de programação e integrado a um AVA, como o ambiente Moodle. As características de principal interesse para a integração do BOCA ao Moodle são apresentadas nos próximos parágrafos.

Para cada problema cadastrado no BOCA, são necessários um arquivo contendo um conjunto de entradas e outro contendo as respectivas saídas. Os arquivos de entrada e saída são obtidos pelo professor, através de um programa executável elaborado pelo mesmo como solução ao problema, onde as entradas enviadas para o programa e as saídas geradas são armazenadas em arquivos distintos.

Ao receber o código fonte submetido por um time, o sistema o compila. Caso não ocorra nenhum erro, é realizada a sua execução. O teste do programa é realizado com o processamento da entrada cadastrada para o problema. Em seguida, o sistema efetua a comparação da saída gerada pela solução do time com aquela cadastrada para o problema. Ao final das etapas de compilação e comparação, é enviado um *feedback* para o time, contendo eventuais erros encontrados no processo de compilação ou na comparação da saída.

Para dar suporte à integração das funcionalidades dos dois ambientes, o sistema de armazenamento de dados, a submissão de arquivos e a compilação realizada pelo BOCA precisaram ser adaptados.

Em sua concepção original, o sistema BOCA [4] só permite o envio de um único arquivo por problema computacional proposto.

O envio de mais de um programa fonte pode ser facilmente resolvido através da compactação do conjunto de arquivos usando ferramentas como ARJ ou ZIP. Entretanto, essa operação resolve apenas parcialmente o problema, tendo em vista que é necessário o servidor identificar o arquivo compactado, executar a descompactação, a compilação dos programas fontes e o armazenamento de maneira adequada dos mesmos.

Para a aplicação visada neste trabalho, os problemas devem ser propostos de forma individual, sendo necessário, portanto, adaptar o BOCA para armazenar informações de forma a identificar o aluno no Moodle, rastrear as atividades do mesmo e fornecer *feedbacks*.

Para a gestão do cadastro de alunos, registro de atividades e notas, entre outros aspectos administrativos, o ambiente Moodle oferece os recursos necessários. Assim, verifica-se a complementaridade entre os ambientes a serem integrados neste trabalho, valorizando o conjunto de competências peculiares a cada um. Além das alterações propostas para o BOCA [4], um módulo de extensão deve ser criado no Moodle de maneira a permitir a integração entre os ambientes. Este módulo de extensão deve: (i) permitir o acesso à funcionalidades disponibilizadas pelo BOCA [4]; (ii) usar estruturas específicas para regis-

tro dos dados relativos aos problemas propostos; (iii) apresentar interfaces para submissão de soluções ao BOCA e para apresentação dos resultados, ambos a partir da interface do Moodle.

3.2 O Sherlock para análise de similaridade

O Sherlock [10] encontra semelhanças entre documentos textuais, através de assinaturas digitais, e apresenta os resultados das análises em tempo real. O mesmo apresenta dois modos de operação. No primeiro ele pode descobrir plágio em tarefas de linguagem natural e, no outro, ele pode descobrir plágio em tarefas de código fonte.

Para verificar a semelhança, o software Sherlock [10] analisa certa quantidade de palavras para cada linha do texto e gera uma assinatura digital que identifica essas palavras. Nesse processo, as linhas e espaços múltiplos em branco ou comentários de código são ignorados. Contudo, alguns espaços em brancos são fundamentais para determinar o início e o fim de uma palavra. Por exemplo, a expressão "`for(`" é avaliada diferentemente da expressão "`for (`", com espaço, apesar de essa diferença não ser significativa para a compilação.

O procedimento de geração da assinatura digital é repetido até o final de cada documento comparado. Ao terminar essa etapa, o Sherlock [10] possuirá as assinaturas digitais que identificam todo o texto. Finalmente, para determinar a semelhança entre os dois arquivos, compara-se as assinaturas digitais dos textos e retorna a porcentagem de semelhança entre eles.

Outra função importante sobre o funcionamento do Sherlock [10] está na quantidade de comparações a serem realizadas para certa quantidade de textos. Uma vez que o *software* compara os arquivos em pares e todos os arquivos devem ser comparados entre si, a quantidade de comparações a ser realizada será dada pela equação 1, em que *m* é a quantidade de códigos a serem comparados. Desta forma, nota-se que é indesejável comparar um arquivo A com um B se B já foi comparado com A.

$$C\binom{m}{2} = \frac{m!}{2!(m-2)!} \qquad \textbf{(1)}$$

O percentual de semelhança é, na integração com o BOCA-LAB, calculado em função da técnicas de normalização utilizada. Essas técnicas são apresentadas na próxima seção, juntamente com a arquitetura de integração.

4 Arquitetura de Integração

A arquitetura da integração é composta por três módulos que se comunicam através do protocolo SOAP usando mensagens criptografadas no formato XML, e um módulo responsável pela análise de similaridade.

A Figura 1 ilustra a estrutura de comunicação destes módulos, ressaltando a coexistência de múltiplos servidores que dão suporte ao balanceamento de carga. Os módulos são detalhados nas próximas subseções.

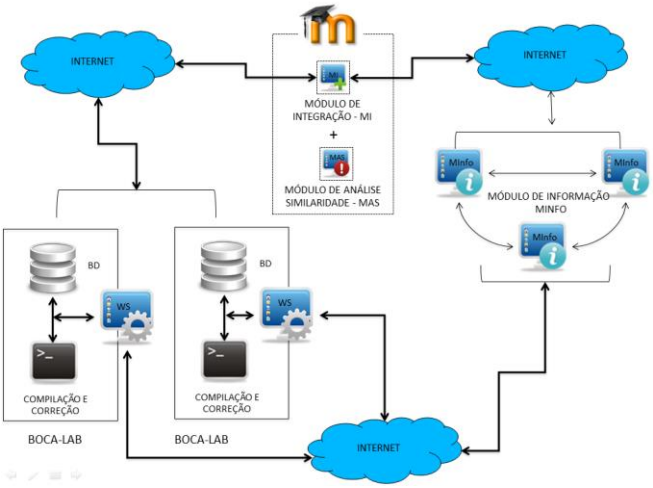

Figura 1: Arquitetura da integração

4.1 Módulo de Integração (MI)

O MI é responsável pelo acesso ao serviço de busca de servidores, registro dos dados necessários aos problemas computacionais e pelo envio e recuperação de *feedback* dos códigos fonte submetidos aos servidores MAB.

4.2 Módulo de Informação (MInfo)

O MInfo é o módulo responsável pela disponibilização dos serviços de localização e registro do estado dos servidores MAB. O armazenamento de informações sobre o estado dos servidores MAB permite efetuar o balanceamento de carga.

4.2.1 Controle e Balanceamento de Carga

O balanceamento de carga na arquitetura é realizado pelo MInfo, a cada requisição feita pelo MI, o módulo verifica dentre os servidores MAB aquele que retém menor número de submissões, visando minimizar o tempo de resposta e evitar sobrecarga.

Além de reduzir o impacto da concorrência por recursos computacionais no mesmo servidor para compilação e execução de problemas de programação, o balanceamento de carga agiliza o processo de *feedback* para o aluno, evitando que um processo permaneça tempo desnecessário nas filas de *jobs* em servidores sobrecarregados.

4.3 Módulo de Acoplamento BOCA-LAB (MAB)

O MAB é responsável pela disponibilização de serviços que permitem o recebimento e repasse ao BOCA-LAB dos dados relativos a problemas e códigos fonte. Além disso, realiza a recuperação de *feedback* e o controle secundário da carga de compilação no BOCA-LAB, evitando o recebimento de requisições caso o servidor esteja com sua carga máxima.

4.4 Módulo de Análise de Similaridade (MAS)

A análise de similaridade acontece em duas etapas: (i) pré-processamento (normalização) dos códigos; (ii) comparação pelo Sherlock [10]. A normalização atua principalmente na identificação e remoção de trechos de código sem relevância, além de padronizar os códigos de modo a otimizar os resultados obtidos com o Sherlock [10].

Assim, os critérios de normalização foram adotados para eliminar a falta de correspondência irrelevante entre os códigos, incluindo eventuais modificações inseridas para encobrir plágios. Essas modificações, em geral, não são complexas, sendo limitadas a mudanças que não alteram o funcionamento do programa.

Logo, com a finalidade de encontrar a melhor técnica de normalização, adotaram-se quatro abordagens diferentes de normalização, com graus de intervenção crescentes. Assim, a versão do código que é gerado com o uso da primeira técnica é a que está mais próxima da versão enviada pelo aluno, enquanto aquele gerado pela última técnica realiza alterações mais invasivas. As quatro técnicas são descritas na Tabela 2.

As técnicas propostas são formadas por algumas dezenas de regras, especificadas a partir de análise lógica e empírica. Observando a técnica de normalização 1 aplicada ao código exemplo da Tabela 2, verificou-se, por exemplo, que palavras reservadas indicativas de funções parametrizadas e estruturas condicionais, como: "`printf (`", e "`if (`", são melhor comparadas quando se eliminam os espaços antes do parêntese, resultando em "`printf(`" e "`if(`".

Da mesma maneira, devido a natureza da comparação feita pelo Sherlock [10] em relação a separação de palavras por espaço em branco, é necessário que, na normalização 4, os caracteres especiais sejam unidos, como em "`(,);`", ao invés de "`( , );`", para se obter o melhor resultado.

Em alguns experimentos, observou-se empiricamente que, para expressões como "`i%3==1`", obtêm-se melhores resultados quando o valor numérico está junto ao operador do lado direito, e os elementos do lado esquerdo estão separados, com a seguinte configuração: "`i % 3 ==1`". Contudo, para declaração de variáveis, por exemplo, "`int i=1;`", a organização que resulta em melhores resultados é "`int i = 1;`".

As quatro técnicas de normalização podem auxiliar a análise de similaridade sob óticas distintas, dependendo dos objetivos de quem a realiza. Alguns resultados relacionados ao uso desta ferramenta são apresentados na seção 6.2. A Figura 2 ilustra o resultado da análise de similaridade.

Figura 2: Página com resultado da análise de similaridade

Código Original
<Sem modificações>
	`// Imprime resposta`
	`for (i=1;i<= n;i++){`
	
	`  if (i %3== 1){`
	`  printf("Resp %d", v[i]/x+z);`
	`  }`
	`}`

Normalização 1
Remoção de linhas e espaços vazios;
Remoção de todos os comentários;
Remoção das referências aos arquivos externos (bibliotecas);
Inclusão (ou remoção) de espaços em branco entre expressões, declaração de variáveis e outras estruturas.
	`for( i=1; i<=n; i++ ) {`
	`if( i % 3 ==1 ) {`
	`printf ( " Resp %d ", v[ i ] / x +z );`
	`}`
	`}`

Normalização 2
Aplicação da normalização 1;
Remoção de todos os caracteres situados entre aspas.
	`for( i=1; i<=n; i++ ) {`
	`if( i % 3 ==1 ) {`
	`printf( , v[ i ] / x +z );`
	`}`
	`}`

Normalização 3
Aplicação da normalização 2;
Remoção de todos os valores literais e variáveis.
	`for( = ; <= ; ++ ) {`
	`if( % == ) {`
	`printf( , [] / +);`
	`}`
	`}`

Normalização 4
Aplicação da normalização 3;
Remoção de todas as palavras reservadas.
	`( =; <=; ++ ) {`
	`( %== ) {`
	`( , [] / +);`
	`}`
	`}`

Tabela 2: Descrição das técnicas de normalização

5 O Módulo de Integração implantado no Moodle

A aplicação desenvolvida nesse trabalho foi integrada ao Moodle, permitindo o seu uso por meio das mesmas semântica e interface definidas para os demais recursos do ambiente virtual. Por exemplo, em seu curso, o professor deve adicionar uma atividade denominada "Envio de arquivos para compilação" que foi agregada ao Moodle para a administração da submissão de problemas, códigos fonte e recuperação de *feedbacks* por parte dos alunos. A

atividade implementada constitui-se, assim, em uma interface entre o usuário (professor/aluno) e o MI. Configuradas as informações dessa atividade, o professor deve cadastrar seu problema através do formulário ilustrado na Figura 3.

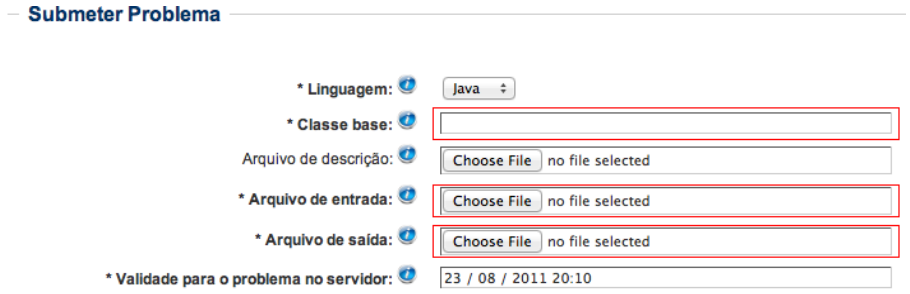

Figura 3: Formulário de cadastro de um problema

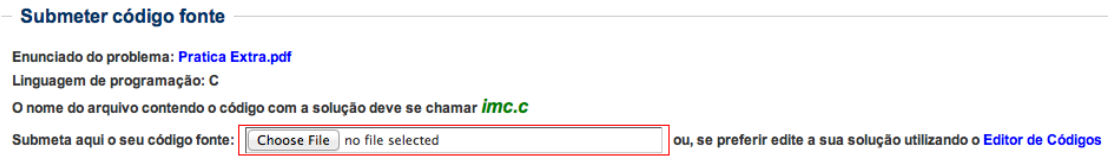

Figura 4: Formulário de envio de código fonte

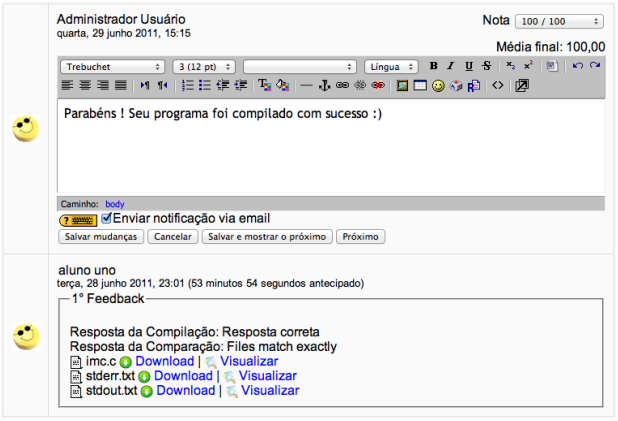

Figura 5: Interface de atribuição de notas

Ao submeter o formulário, o MI envia os dados para o MInfo que busca e retorna o endereço do MAB que melhor se adéqua aos requisitos do problema. Enviado o problema ao MAB, um formulário (Figura 4) é então disponibilizado na interface do aluno para a submissão de seu código fonte. Uma vez submetido, o código fonte é enviado ao MAB pelo MI.

A cada código fonte recebido ou processado, o servidor MAB atualiza a informação sobre o seu estado nos servidores MInfo, permitindo, assim, uma melhor distribuição de carga pelo mesmo. Essa atualização, só é feita caso a carga máxima do servidor esteja próxima de atingir o limite configurado.

Após o envio do código fonte, a interface do aluno fica bloqueada para novas submissões para o mesmo problema até ser disponibilizado o *feedback* para a última submissão. As informações retornadas ao aluno pelo BOCA-LAB são compostas por uma resposta do compilador, um arquivo contendo os erros da compilação, caso ocorram, e um outro contendo a saída gerada pelo programa. Os tipos de resposta retornados pelo compilador são mostrados pela Tabela 3.

Reposta	Descrição
YES	Programa foi aceito sem erros.
NO: Incorrect Output	Saída dos testes incorreta.
NO: Time-limit Exceeded	O programa excedeu o tempo estipulado.
NO: Runtime Error	Durante o teste ocorreu um erro de execução.
NO: Compilation Error	Programa possui erros de sintaxe.

Tabela 3: Tipo de *feedbacks* retornados pelo BOCA-LAB

Os resultados de todas as submissões são armazenados pelo sistema e apresentados na interface do professor, como apresentado na Figura 5, permitindo ao mesmo analisar o desempenho do aluno, facilitando assim, a atribuição da nota.

A nota atribuída às atividades de programação figuram junto ao conjunto de notas de atividades regulares de um curso Moodle, como Fóruns, Chats e outras atividades, compondo, assim, a nota final do aluno.

6 Discussões e Testes

A avaliação e os testes do ambiente de integração são apresentados nas próximas subseções seguintes.

6.1 Discussões do BOCA-LAB

O ambiente de integração desenvolvido foi, inicialmente, implantado em um servidor do Departamento de Engenharia de Teleinformática (DETI) da Universidade Federal do Ceará (UFC). Experimentações iniciais foram realizadas no ano de 2011 em turmas da disciplina de Técnicas de Programação para Engenharia I, do curso de Engenharia de Teleinformática, e de Fundamentos de Programação, do Departamento de Computação. Foi avaliada a percepção dos alunos sobre o ambiente e os resultados podem ser vistos em [29].

O ambiente encontra-se atualmente instalado em um servidor mantido em colaboração entre o DETI e a UFC Virtual, e vem sendo usado, com acompanhamento da equipe responsável por esta pesquisa, em turmas numerosas de primeiro ano do curso de Engenharia de Teleinformática desde 2012. As experiências de alunos e professores com a ferramenta tem permitindo o amadurecimento de aspectos relacionados à usabilidade e à interface. Essa experiência é importante para realizar os ajustes necessários e garantir a confiabilidade antes que a ferramenta seja disponibilizada para a comunidade em vários formatos: integração com Moodle, versão *standalone* do BOCA-LAB e módulo independente para a Análise de Similaridade.

Uma dificuldade que se apresentou para o uso sistemático do BOCA-LAB nas práticas de laboratório semanal é a necessidade de elaboração de exercícios com as restrições impostas pelo sistema BOCA, em que um programa deve ser avaliado para uma entrada e uma saída padronizadas. Isso requer que o professor não só elabore o enunciado, mas prepare os arquivos de entrada e de saída que serão usados como insumo para a validação das soluções dos alunos. Além disso, para testar diversas condições de um problema, é preciso preparar diversas pares de arquivos de entrada e saída. A fim de certificar-se que as entradas e as saídas estão absolutamente corretas, faz-se necessário, nessa abordagem, que o professor desenvolva também a solução para o problema. Essa situação não ocorre usualmente em abordagens tradicionais com a proposição de listas de problemas em turmas de programação, em que o professor, pela sua própria experiência, precisa inferir apenas o nível de dificuldade de uma questão antes de apresentá-la aos alunos.

Pelo fato de a validação da saída ser efetuada por comparação de *strings* (comando *diff*), outra dificuldade se revela: o sistema é muito sensível a pequenas alterações de formatação e apresentação de saídas (ex.: uso de diferentes textos como *prompt* de entrada, uso de caracteres especiais como separadores de valores de saída, etc.). Isso requer uma elaboração de enunciado muito detalhada e precisa por parte do professor, e atenção especial por parte do aluno no desenvolvimento de sua solução. Essas restrições acrescentam dificuldades que não são diretamente associadas à natureza do problema. Em uma maratona de programação, esse tipo de restrição não representa obstáculos aos usuários, em geral, programadores mais experientes. Entretanto, para um público de iniciantes, restrições dessa natureza exigem um aculturamento que apenas se constrói em múltiplas práticas.

Do ponto de vista do professor, as limitações discutidas, entretanto, podem ser mitigadas com o desenvolvimento de algumas rotinas de apoio e melhorias na interface. Requisitos associados a este tipo de limitação vem sendo elencados, discutidos e estão servindo de insumo para a melhoria da ferramenta. O uso repetido da ferramenta vem permitindo, também, que se forme uma base de exercícios e enunciados que atendem aos requisitos do ambiente desenvolvido, o que irá facilitar, sobremaneira, a utilização em diferentes turmas ao longo do tempo. É importante salientar que, se o BOCA-LAB, por um lado, requer um tempo maior para elaboração e preparação de atividades, por outro lado, vem demonstrando grande vantagem para validação de trabalhos propostos aos alunos em aulas de laboratório, otimizando o tempo de professores e monitores. Além disso, o sistema vem sendo utilizado com eficácia para a proposição de atividades em forma de prova, dispensando o tempo usualmente empregado para correção e atribuição de notas, situação crítica para o professor em turmas numerosas.

Pela perspectiva do aluno iniciante, uma dificuldade de cunho operacional se manifestou logo nas primeiras atividades. O desenvolvimento da prática de laboratório não é feita diretamente no BOCA-LAB. A metodologia empregada é a de utilização do compilador e testes na máquina local e submissão do programa fonte ao BOCA-LAB apenas após o desenvolvimento supostamente correto por parte do aluno. Percebeu-se que, sistematicamente, o aluno submetia o arquivo errado para validação remota. Esse problema foi mitigado com a submissão do código através de um editor embutido na atividade criada

no Moodle, em que o aluno pode "colar" o seu código-fonte e, em seguida, realizar a submissão. Outra dificuldade inicial é a interpretação das mensagens de erro apresentadas pelo sistema, principalmente quando a mensagem indica um problema de apresentação, o que requer as adequações de formatação já discutidas anteriormente. Apesar do tempo necessário para ambientação com esse tipo de ferramenta, percebe-se o desenvolvimento progressivo de maior autonomia do aluno para o desenvolvimento das soluções aos problemas propostos, em que ele se torna capaz de reagir às mensagens da ferramenta e corrigir os problemas sem a necessidade contínua de interação com professores e monitores.

A subseção seguinte apresenta os resultados de testes relativos ao balanceamento de carga.

6.2 Testes sobre o Controle e Balanceamento de Carga

Para avaliar a eficácia do balanceamento de carga, dois cenários de teste, ambos com 3 Servidores BOCA-LAB foram configurados. No cenário I, o Servidor I foi configurado com o compilador C e C++, o Servidor II foi configurado com os compiladores C e Java e o Servidor III foi configurado apenas com o compilador C++. Nesse cenário, todos os servidores foram limitados para o processamento de até 50 *jobs* simultâneos.

Para o primeiro teste, foram submetidos 80 códigos fonte divididos da seguinte maneira: 70 códigos em C e 10 códigos em Java.

Ao final do primeiro teste, a divisão dos códigos entre os servidores se deu da seguinte maneira: 33 *jobs* processados no Servidor I, 47 *jobs* processados no servidor II e 0 no servidor

No cenário de teste I, portanto, temos que: o servidor I recebeu 33 códigos da linguagem C; dos 47 códigos recebidos pelo servidor II, 37 foram da linguagem C e 10 da linguagem Java e; como esperado, o servidor III não recebeu nenhum código fonte, pois somente apresentava o compilador para a linguagem C++.

No segundo cenário, os servidores I e II foram configurados de maneira idêntica, contendo compiladores C e C++ e com capacidade de processar até 50 *jobs* simultâneos. O Servidor III foi configurado com apenas o compilador Java e com o limite de 25 *jobs* simultâneos.

Para esse segundo cenário, foram submetidos simultaneamente 90 códigos, divididos da seguinte maneira: 45 códigos em C, 25 códigos em Java e 20 em C++.

Conforme especificado, a distribuição dos códigos pelos servidores ocorreu da seguinte maneira: os Servidores I e II receberam todos os 65 *jobs* relativos aos programas em C e C++, sendo distribuídos em número de 30 para o primeiro e 35 para o segundo. Já o Servidor III, único que disponibiliza o compilador Java, recebeu todos os 25 *jobs* nessa linguagem.

Dos códigos recebidos pelo servidor I, 17 foram em linguagem C e 13 em C++. No servidor II, 28 códigos em linguagem C e 7 em C++ foram recebidos. No servidor III, único que possui compilador na linguagem Java, os 25 códigos nessa linguagem foram recebidos.

Através desses dois testes em cenários diferentes, foi possível verificar o funcionamento adequado da distribuição e do balanceamento de carga da arquitetura, em função da capacidade configurada e dos compiladores disponíveis em cada servidor, bem como a capacidade de endereçamento executada pelo módulo Minfo.

6.3 Avaliação do Módulo de Análise de Similaridade

A análise de similaridade foi avaliada em uma turma iniciante em programação, com a linguagem C, durante o 1º semestre de 2012. A turma é formada por 97 alunos divididos em 4 grupos. Cada grupo realizou 4 atividades contendo 2 problemas diferentes, perfazendo, ao todo, 32 problemas.

Para avaliar as quatro técnicas de normalização de códigos desenvolvidas, os resultados da análise de similaridade foram comparados entre si. Além disso, compararam-se, também, aos resultados apresentados pelas ferramentas JPlag e MOSS.

	Norm 1		Norm 2		Norm 3		Norm 4	
	> 20%	> 70%	> 20%	> 70%	> 20%	> 70%	> 20%	> 70%
Problema 1	13	2	33	1	52	0	10	1
Problema 2	5	0	22	0	19	1	4	0
Problema 3	5	0	23	1	32	4	12	10
Problema 4	4	0	12	0	30	0	30	3
Problema 5	15	0	45	4	38	3	72	10
Problema 6	13	0	59	5	57	3	101	22
Problema 7	12	3	42	4	65	4	15	6
Problema 8	16	0	120	55	71	1	10	5

Tabela 4: Comparação entre as 4 técnicas de normalização

A Tabela 4 relaciona a quantidade de códigos que apresentam índice de similaridade maior do que 20% e 70% para cada tipo de normalização. A análise da Tabela mostra que, em geral, a sensibilidade à identificação de similaridade apresenta um comportamento crescente da técnica 1 para a técnica 4. Entretanto, pode-se observar que, para alguns problemas, como o de número 8, essa característica não se confirma em relação às técnicas 2, 3

e 4, sendo, na verdade, invertidas. Isso mostra que as técnicas podem revelar semelhanças que vão além da organização estrutural do código, fornecendo elementos de análise diferenciados para o professor.

As Figuras 6, 7, 8 e 9 apresentam os gráficos comparativos do número de ocorrência de similaridades para os grupos A, B, E e D da turma de programação, levando-se em consideração apenas as técnicas de normalização 1 e 4, além do JPlag e do MOSS. Pela baixa incidência de similaridade apontada pela técnica de normalização 1 em relação às demais, infere-se que a técnica 1 produz os resultados menos significativos. Além disso, o JPlag, praticamente em todos os problemas, indica o maior número de casos de similaridade, e, por outro lado, o MOSS sempre aponta o menor número de similaridades registradas. Isso demonstra que a sensibilidade utilizada pelo MOSS é a mais conservadora para indicação de similaridade. A técnica de normalização 4, com o uso do Sherlock, apresenta resultados intermediários entre os do MOSS e do JPlag em 62% dos problemas para os registros de similaridades acima de 70%. demonstrando, assim, constituir-se em uma promissora ferramenta de análise. Um aspecto importante da técnica 4 associada ao Sherlock é a redução expressiva do tamanho do código a ser comparado devido à eliminação de grande parte do código, o que, na maioria das vezes, permitirá o processamento mais rápido da comparação.

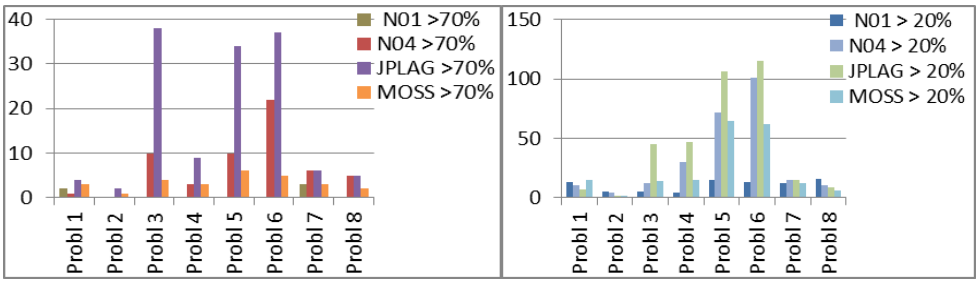

Figura 6: Grupo A: relação da técnica 1 e 4 com JPlag e MOSS

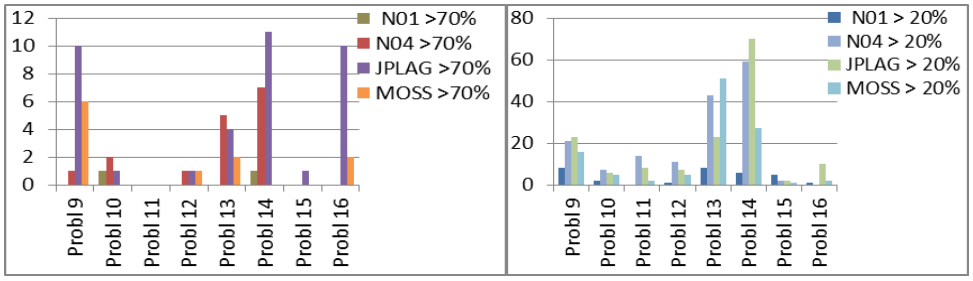

Figura 7: Grupo B: relação da técnica 1 e 4 com JPlag e MOSS

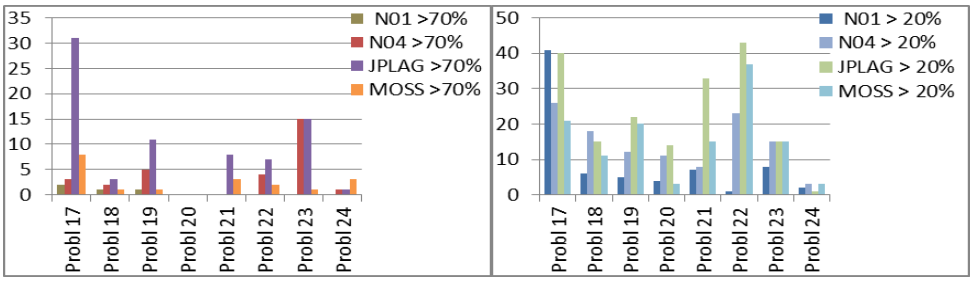

Figura 8: Grupo C: relação da técnica 1 e 4 com JPlag e MOSS

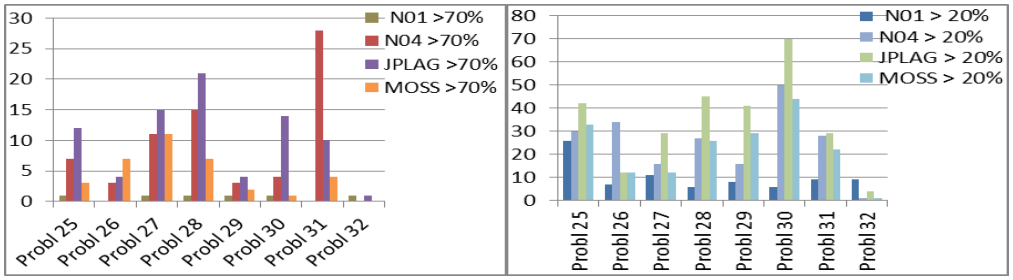

Figura 9: Grupo D: relação da técnica 1 e 4 com JPlag e MOSS

O uso do Sherlock com a técnica de normalização 4, em algumas situações, pode apresentar maior ocorrência de similaridade do que o JPlag, como pode ser observado pelo problema 31, que propõe a impressão por extenso do valor de um algarismo inteiro. A análise visual dos códigos dos alunos mostrou que quase todos desenvolveram uma solução com o uso do *swicth*, com código bastante semelhante, o que foi feito por orientação do professor (e poderia também ter sido conseqüência da uso de algum exemplo encontrado em fontes de referência). Na perspectiva da similaridade, os resultados não são, portanto, considerados falsos positivos para essa questão. Assim, verifica-se que os resultados do Sherlock com a normalização 4 foram mais significativos do que os apresentados pelo JPlag e MOSS.

6 Conclusões e Perspectivas

Este trabalho representa uma iniciativa em termos de instrumentalização para o professor de disciplinas de programação de computador com vistas a mitigar um problema que ocorre com frequência: um grande número de alunos em turmas de laboratório, situação recorrente em disciplinas iniciais em alguns cursos universitários.

Propõe-se, assim, uma tecnologia capaz de auxiliar o professor a melhor gerenciar os seus esforços para o acompanhamento das práticas de programação dos alunos. Sendo uma atividade que exige, com frequência, atenção individualizada, a dificuldade de atendimento a turmas numerosas pode comprometer um aprendizado de qualidade. O auxílio se constitui na automatização de algumas das etapas inerentes ao acompanhamento, sem prescindir da importante e insubstituível ação do professor, buscando também o comportamento mais autônomo do aluno.

O ambiente BOCA-LAB oferece recursos em forma de serviços para dar suporte a estas atividades práticas. Integrado ao ambiente Moodle, as funcionalidades de compilação e correção automática de códigos fonte são disponibilizadas como tarefas do próprio ambiente virtual. A arquitetura proposta pode ser estendida de maneira a permitir a integração do BOCA-LAB a outros ambientes, bastando, para isso, que seja construído um módulo de integração (MI) específico à plataforma.

A integração do BOCA-LAB a um AVA visa, assim, diminuir a sobrecarga de trabalho do professor no que concerne ao acompanhamento e à avaliação dos resultados das atividades propostas. Como consequência, prospecta-se a melhoria na qualidade do aprendizado de programação, pois se, por um lado, o tempo do professor com atividades de administração e de gestão de recursos pode ser reduzido, por outro lado, sua disponibilidade em termos de atenção aos alunos pode ser maior.

Tendo em perspectiva a administração de recursos técnicos institucionalmente compartilhados entre múltiplos cursos e usuários, atribuindo à solução a características de escalabilidade necessária a esse tipo de situação o balanceamento de carga se mostrou funcional no testes realizados. Na versão atual, a distribuição dos programas se baseia somente na quantidade de códigos fonte ainda não processados e armazenados nos servidores. Entretanto, o modelo pode ser adaptado facilmente para técnicas de balanceamento que levem em consideração outros parâmetros, como a complexidade dos códigos enviados ou as características físicas dos servidores, como quantidade de memória livre e uso de CPU, entre outros fatores.

Expandindo as perspectivas de análise e acompanhamento do professor, o ambiente desenvolvido conta com a análise de similaridade entre códigos-fonte. Foram desenvolvidas e avaliadas técnicas de normalização que, de acordo com os resultados apresentados, são comparáveis as duas principais ferramentas de detecção de plágio referenciadas na literatura. A possibilidade de configuração de uma das quatro técnicas de normalização propostas permite alterar a sensibilidade da comparação, ressaltando características diferenciadas em contextos particulares. Apesar de a técnica 4 ser a mais sensível em termos estruturais, é possível que seja importante para o professor, por exemplo, considerar os nomes de funções ou de palavras reservadas na comparação entre códigos, o que o remete para as outras técnicas. Além disso, é importante frisar que a ferramenta pode ser utilizada para finalidades diferentes da detecção de plágio. É possível que a similaridade seja até mesmo desejável. Para exemplificar, o professor pode usar a comparação para verificar se as

soluções apresentadas foram baseadas em um exemplo fornecido ou em uma orientação dada aos alunos, ou, ainda, se foram usadas lógicas e estruturas diferentes daquelas que foram propostas.

Além das melhorias no ambiente BOCA-LAB que estão sendo continuamente desenvolvidas em consequência das observações feitas por professores e alunos, paralelamente, estão sendo realizadas pesquisas mais aprofundadas sobre a análise de similaridade. Outros algoritmos de comparação foram integrados ao ambiente e estão sendo estudados de maneira a buscar maior qualidade em termos de identificação de semelhanças entre códigos-fonte. Pesquisa-se, ainda, abordagens de análise integrando grupos de alunos, não só com a perspectiva de identificação de possíveis situações de plágio, mas também para identificar grupos de alunos que trabalham em parceria, erros comuns ou comportamentos diversos que possam auxiliar na compreensão de como a aprendizagem vem se desenvolvendo e como se pode reagir a eventuais dificuldades identificadas.

Referências

[1] Essi L, Kirsti Ala-Mutka, Hannu-Matti J. (2005). "A study of the difficulties of novice programmers". In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05).ACM, New York, NY, USA, 14-18.

[2] Tan, Phit-Huan; Ting, Choo-Yee; Ling, Siew-Woei. (2009). "Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception," Computer Technology and Development, 2009. ICCTD '09. International Conference on Computer Technology and Development, vol.1, no., pp.42-46, 13-15 Nov. 2009.

[3] Papazoglou, Mike P.; Heuvel, Willem-Jan van den. ."(2007) "Service Oriented Architectures: approaches, technologies and research issues. The VLDB Journal, v. 16, p. 389-415, 2007.

[4] De Campos, C. P. ; Ferreira, C. E. (2004). "BOCA: Um Sistema de Apoio para Competições de Programação." Workshop de Educação em Computação, 2004, Salvador. Anais do Congresso da SBC, 2004.

[5] Hage, J.; Rademaker, P.; Vugt, N. V. A comparison of plagiarism detection tools. Utrecht University. Utrecht, The Netherlands, p. 28. 2010.

[6] A. S. Bin-Habtoor and M. A. Zaher, "A Survey on Plagiarism Detection Systems," International Journal of Computer Theory and Engineering vol. 4, no. 2, pp. 185-188, 2012.

[7] Đurić, Z., & Gašević, D. (2012). A Source Code Similarity System for Plagiarism Detection . The Computer Journal . doi:10.1093/comjnl/bxs018

[8] Prechelt, L.; Malpohl, G. & Phlippsen, M. (2002). "Finding plagiarisms among a set of programs with JPlag". J. UCS – Journal of Universal Computer Science.

[9] "A System for Detecting Software Plagiarism". Univ. California, Berkeley. Disponível em "http://theory.stanford.edu/~aiken/moss/". Acesso em: 01 de outubro de 2012.

[10] The Sherlock Plagiarism Detector. Disponível em:"http://sydney.edu.au/engineering/it/~scilect/sherlock/" Acesso em: 01 de outubro de 2012.

[11] Moodle – "A Free, Open Source Course Management System for Online Learning." Disponível em http://moodle.org/.

[12] Sakai: Collaborative and Learning Environment for Education. Disponível em https://confluence.sakaiproject.org/display/WEBSVCS/Home. Acesso em: 01 de outubro de 2012.

[13] Ng, S.C., Choy, S.O., Kwan, R., Chan, S.F.: A Web-Based Environment to Improve Teaching and Learning of Computer Programming in Distance Education. In: Lau, R., Li, Q., Cheung, R., Liu, W. (eds.) ICWL 2005. LNCS, vol. 3583, pp. 279–290. Springer, Heidelberg (2005).

[14] Wang, J., Chen, L., Zhou, W.: Design and Implementation of an Internet-Based Platform for C Language Learning. In ICWL(2008) 187-195.

[15] Sistema Susy. Disponível em: www.ic.unicamp.br/~susy/ Acesso em: 01 de outubro de 2012.

[16] Mike Joy, Nathan Griffiths, and Russell Boyatt. 2005. The boss online submission and assessment system. J. Educ. Resour. Comput. 5, 3, Article 2 (September 2005). DOI=10.1145/1163405.1163407 http://doi.acm.org/10.1145/1163405.1163407

[17] Gareth Thorburn, Glenn Rowe, PASS: An automated system for program assessment, Computers & Education, Volume 29, Issue 4, December 1997, Pages 195-206, ISSN 0360-1315, 10.1016/S0360-1315(97)00021-3.

[18] A. Zeller, "Making Students Read and Review Code", SIGCSE Bull., ACM, New York, NY,

USA, 32(2000)3, pp. 89-92

[19] VPL – "Virtual Programming Lab Disponível" em: "http://vpl.dis.ulpgc.es/ ". Acesso em 01 de outubro de 2012.

[20] Onlinejudge. Disponível em: https://github.com/hit-moodle/onlinejudge. Acessado em 1 de outubro de 2012.

[21] Sphere Research Labs – IDE ONE Disponível em http://ideone.com/. Acesso em 1 de outubro de 2012.

[22] Grune, D. and Vakgroep, M. 1989. Detecting copied submissions in computer science workshops. Tech. rep., Informatica Faculteit Wiskunde & Informatica, Vrije Universiteit.

[23] Lancaster, Thomas; Culwin, Fintan. (2001). "Towards an error free plagarism detection process". In Proceedings of the 6th annual conference on Innovation and technology in computer science education (ITiCSE '01). ACM, New York, NY, USA, 57-60.

[24] Chen, Xin; Francia, Brent; Li, Ming; Mckinnon, Brian; Seker, Amit; (2004). "Shared information and program plagiarism detection," IEEE Transactions on Information Theory, vol. 50, no. 7, pp. 1545–1551, 2004.

[25] Ahtiainen, Aleksi; Surakka, Sami; Rahikainen, Mikko. (2006). "Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises". In Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research, pages 141–142, New York, NY, USA, 2006. ACM.

[26] Burrows, S.; Tahaghoghi, S.M.M.; Zobel, J. (2007). "Efficient and effective plagiarism detection for large code repositories". Software-Practice & Experience, 37(2), 151-175.

[27] Schleimer, S., Wiljerson, D. S., & Aiken, A. (2003) "Winnowing: local algorithms for document fingerprinting". In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03). ACM, New York, NY, USA.

[28] Alves, L. and Brito, M. (2005) "O Ambiente Moodle como Apoio ao Ensino Presencial." Disponível em: www.abed.org.br/congresso2005/por/pdf/085tcc3.pdf. Acesso em: 01 de outubro de 2012.

[29] FRANÇA, A. B; SOARES, J. M. Sistema de apoio a atividades de laboratório de programação via Moodle com suporte ao balanceamento de carga. In: Simpósio Brasileiro de Informática na Educação, 22., 2011, Aracaju. Anais do 22º SBIE. Aracaju, 2011.