2. Workshop „Automatische Bewertung von Programmieraufgaben“
(ABP’2015), Wolfenbüttel 2015

Automatische Aufgabenkorrektur mit ViPLab

Thomas Richter[1]

Abstract: ViPLab ist ein ILIAS-Plugin zur Durchführung von Programmieraufgaben mit C, C++, Java, Matlab und DuMux [4] im Browser. Dieser Artikel beschreibt eine neue Komponente der ViPLab 2.0 Architektur, die eine automatische Bewertung von studentischen Lösungen erlaubt und so insbesondere bei der Durchführung von Grundlagenkursen eine erhebliche Erleichterung für das Lehrpersonal ermöglicht.

Keywords: Automatische Korrektur, ILIAS, ILIAS-Plugin, Programmierübungen

1 Einführung

Das Ziel des ViPLab-Projektes der Universität Stuttgart [2] ist die Durchführung von Programmierübungen und elektronischer Prüfungen im ILIAS Lernmanagement-System der Universität Stuttgart. Historisch entstand das Projekt aus der Notwendigkeit heraus, Studierende in den Bachelor-Studiengängen so rasch wie möglich an Programmiersprachen und numerische Software wie Matlab heran zu führen, wofür traditionelle Umgebungen wie Eclipse oder auch die graphische Oberfläche von Matlab nur bedingt geeignet sind. Einerseits entstehen hier zusätzliche Barrieren durch die Installation von zusätzlicher Software auf den Rechnern der Studierenden, andererseits entfällt ein nicht unerheblicher Zeitanteil des Semesters auf die Einarbeitung in die Bedienung der Oberfläche. Zugleich sind die in den ersten Semestern zu erstellenden Programme derart einfach, dass ein Großteil der Komplexität von Entwicklungsoberflächen nicht notwendig ist und von der zu lösenden Aufgabe lediglich ablenkt.

Aus diesem Grunde starteten Jahr 2009 drei Institute — das Institut für Wasser- und Umweltmodellierung (IWS), das Institut für Aero- und Gasdynamik (IAG) und das Institut für Angewandte Mathematik — und das Rechenzentrum der Universität (heute TIK) ein Projekt zur Einbettung einer einfachen Programmierumgebung in das Lern-Management-System der Universität. In seiner ersten, im Jahr 2011 abgeschlossenen Implementierung stellte das Projekt einen Code-Editor als ein aus dem ILIAS-System heraus gestarteten Java-Applets zur Verfügung. MatLab oder Eclipse brauchten nicht installiert werden. Studentische Lösungen werden dabei vom Applet über eine Middleware an die Server des Rechenzentrums geschickt, dort berechnet und die Lösung im Browser der Studierenden dargestellt. Weitere Details zur Architektur sind in Abschnitt 2 erläutert.

Im Jahr 2011 startete ein Nachfolgeprojekt mit dem Ziel, über ViPLab sowohl elektronische Prüfungen durchführen zu können, als auch den Korrekturaufwand von abgegebenen Hausaufgaben zu senken. Hierbei sind Studierendenzahlen von über 400 Teilnehmern pro Kurs in den Grundlagenkursen nicht ungewöhnlich, wobei ein Großteil der von den

[1] TIK Universität Stuttgart, 70550 Stuttgart, richter@tik.uni-stuttgart.de

Studierenden zu implementierenden Algorithmen einfach genug sind, als dass sie sich einer automatischen Auswertung nicht verschließen. Im Zuge dieses Projektes wurde auch die Nutzerschnittstelle von ViPLab neu in Javascript implementiert und enger mit dem Lernmanagement-System verzahnt, so dass auch die automatische Übertragung von Rechenergebnissen und Punktzahlen in das ILIAS-System hinein problemlos möglich wurde.

Während ViPLab 1.0 noch den Versand von studentischen Lösungen per Mail an den Dozenten vorsah, erleichtert die überarbeitete Architektur den Schritt der Aufgabenbewertung erheblich. Sie sieht zwei Möglichkeiten zur Korrektur und Bewertung der Lösungen vor: Erstens können die Lösungen im ILIAS-System eingesehen werden und von dort kann der Bewertungsalgorithmus manuell gestartet werden. Die Punktzahl wird automatisch in die ILIAS-Eingabemaske eingetragen, der Dozent kann aber diese Bepunktung jederzeit überschreiben sollte der Algorithmus zu keiner geeigneten Bewertung kommen. Für größere Erstsemesterveranstaltungen ist dieser Mechanismus aber noch zu aufwändig, bedarf es doch pro Abgabe einiger Mausklicks. Hierfür entsteht momentan ein Korrekturserver, der diese Abläufe durch entsprechende Interfaces zum ILIAS automatisiert. Der Bewertungsalgorithmus ist ansonsten jedoch prinzipiell identisch zur manuell gestarteten Korrektur. Details hierzu in Kapitel 3.

Dieser Aufsatz gliedert sind wie folgt: Im nächsten Kapitel wird knapp die Softwarearchitektur von ViPLab beschrieben, gefolgt von einem Kapitel über den Aufbau von Aufgaben insbesondere bei der Verwendung der automatischen Korrekturfunktion. Darauf folgend beschreibe ich unsere Erfahrungen mit dem System (siehe auch [6]). Das letzte Kapitel gibt einen Ausblick über die noch geplanten Arbeiten am System.

2 Architektur

Die ViPLab-Architektur gliedert sich grob in drei Schichten, siehe auch Abb. 1: Zuoberst eine graphische Nutzeroberfläche, die zusammen mit den ILIAS-Webseiten ausgeliefert wird. Diese Nutzeroberfläche bietet einen Editor mit Syntax-Highlighting nebst Textausgabe sowie einen interaktiven Funktionsplotter. In der Version 1.0 von ViPLab war die Oberfläche in Java implementiert und erforderte deshalb von den Nutzern, das Java-Plugin zu installieren. Mittlerweile kommt mit GWT [1] zu Javascript compilierter Java-Code zum Einsatz; der compilierte Code kann somit nativ von allen Browsern ausgeführt werden, ohne dass weitere Software installiert werden muss. ILIAS liefert hierbei Daten an das Frontend über Javascript-Aufrufe, wohingegen das Front-End über Javascript unsichtbare Eingabemasken mit Datensätzen belegt und diese über das http-Protokoll verschickt werden.

Die vom Studierenden erstellten Lösungen werden in das JSON-Format [3] codiert und vom Frontend an eine Middleware, den ECS-Community-Server verschickt. Diese Middleware übernimmt die Lastverteilung an diverse Backend-Server und diente bei ViPLab 1.0 auch zur Speicherung der Programmrümpfe der Aufgabenstellungen. Die Speicherung von Aufgaben übernimmt in der 2.0-Architektur komplett das ILIAS-System. Der Aufbau von ViPLab-Aufgaben wird genauer in Kapitel 3 beschrieben.

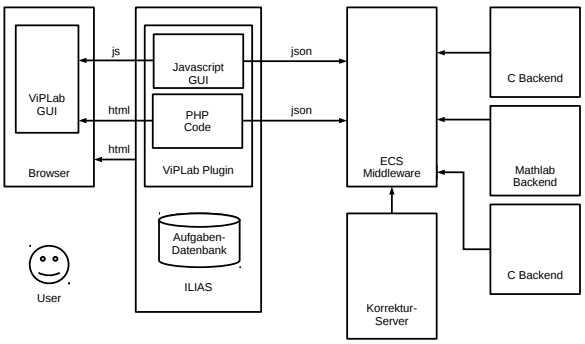

Abb. 1: Die ViPLab-Architektur, bestehend aus dem ILIAS-System, dem Plugin nebst dem Javascript-Frontend, der ECS Middleware, dem Korrekturserver und den numerischen Backends. Zwecks Lastverteilung können identische Backends auch mehrmals vorhanden sein.

Die Compilierung bzw. der Ablauf von studentischem Code erfolgt auf den Servern des Rechenzentrums der Universität. Um eine Kompromittierung der Server auszuschließen, werden die Compiler und der compilierte Code in einem *chroot*-Käfig gestartet, innerhalb dessen ein ablaufendes Programm nur sehr eingeschränkten Zugriff auf Systemresourcen hat. Zusätzlich findet während der Compilierung eine syntaktische Vorfilterung des abgegebenen Codes statt, so dass etwa die Verwendung von Bibliotheksfunktionen aus didaktischen oder technischen Gründen eingeschränkt werden kann. Ein typischer Einsatzzweck für eine derartige Filterung bestünde etwa bei einer Aufgabe zur Erstellung einer numerischen Approximation einer transzendenten Funktion wie *sin* ohne die Verwendung der in der C-Bibliothek vorhandenen Implementierung.

Ergebnisse einer Berechnung werden vom Backend an den ECS (Middleware) geschickt, an welchem das Frontend nach Nachrichten über Rechenergebnisse pollt. Diese beschränken sich hierbei nicht auf Text, sondern können auch vom Backend in einer GNUplot-ähnlichen Syntax erzeugt werden, die dann vom Frontend geplottet wird (siehe Abb. 2). Dabei werden entsprechende Bibliotheksfunktionen zum Plotten in Matlab überladen, so dass aus studentischer Sicht die gleiche Syntax wie bei einer lokalen MatLab-Installation verwendet wird. Die Plot-Funktionen beschränken sich dabei nicht auf zweidimensionale Graphen, auch dreidimensionale Flächen und Objekte können vom Frontend gerendert werden.

Der Ablauf im Korrekturbetrieb unterscheidet sich aus technischer Sicht nicht grundsätzlich vom Übungsbetrieb, wobei zwei verschiedene Abläufe der Korrektur möglich sind: Manuelles Starten des Korrekturalgorithmus mit der Möglichkeit der Nachkorrektur des Ergebnisses, oder vollautomatische Korrektur aller Teilnehmer. In beiden Fällen werden Teile des Aufgabencodes durch einen Korrekturcode ersetzt und dieser Gesamtcode über den ECS an das Backend genauso wie oben verschickt, siehe Abb. 3. Der Aufbau und Zweck dieses Korrekturcodes wird genauer in Abschnitt 3 beschrieben. Die Rückmeldung des Servers enthält dann zusätzlich zur textuellen oder graphischen Ausgabe die vom Korrekturcode ermittelte Punktzahl, welche vom ILIAS-System übernommen und in der Datenbank hinterlegt wird.

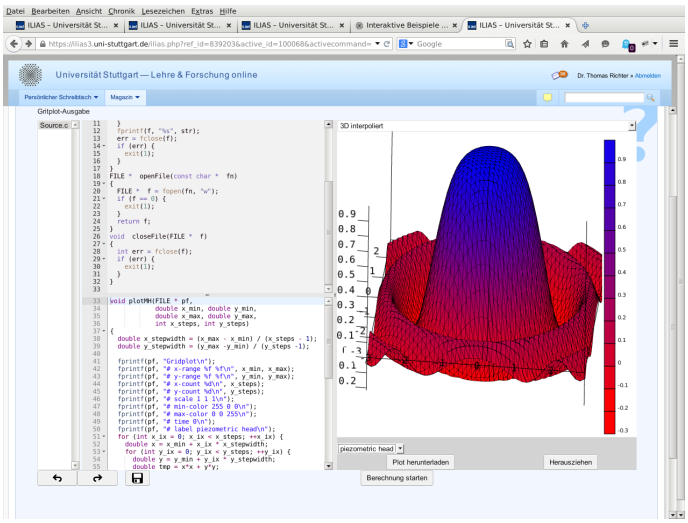

Abb. 2: Eine ViPLab-Aufgabe aus Sicht der Studierenden. Ganz links die Auswahlleiste der Übersetzungseinheit, daneben invarianter Code (grau) und darunter veränderbarer Code (weiß). Rechts daneben eine graphische Ausgabe des Codes.

Die manuelle Korrektur erfolgt über eine separate Eingabemaske des ILIAS-Systems, siehe Abb. 5: Nach Durchführung der Klausur muss hier der Dozent für jeden Teilnehmer einzeln den abgegebenen Code einsehen. Durch eine Schaltfläche auf dieser Eingabemaske wird dann der Korrekturalgorithmus angeworfen und eine Punktzahl ermittelt. Diese Punktzahl wird dann von der Eingabemaske übernommen, wobei hier der Dozent die Möglichkeit hat, die Punktzahl zu verändern und somit die Bewertung des Korrekturalgorithmus zu überstimmen. Dies ist bei komplexen Aufgaben dann sinnvoll, wenn etwa durch triviale Flüchtigkeitsfehler der Korrekturalgorithmus zu einer unstimmigen Bewertung kommen sollte.

Die vollautomatische Korrektur erfolgt im Batch-Betrieb nach der Durchführung der Klausur und kann in der Konfiguration der Aufgabe selbst angefordert werden. Hierbei werden, für den Dozenten unsichtbar, alle studentischen Abgaben vom ILIAS-System über den ECS an den separaten Korrekturserver verschickt, die den im Abschnitt 3 beschriebene Zusammenfügung von Aufgabe, studentischer Abgabe und Korrekturcode übernimmt und selbst den resultierenden Code an die Backends verschickt. Er extrahiert dann aus den Ausgaben des Backends die Punktzahl und liefert diese an das ILIAS-System zurück, welches die Punktzahl dann in seine Datenbank einträgt. Auch hier ist nachträglich eine Änderung der Bewertung möglich, jedoch wird anders als bei der manuellen Bewertung der Dozent nicht dazu gedrängt, die studentische Lösung auch wirklich anzusehen.

3 Aufbau von ViPLab Aufgaben

Eine ViPLab-Aufgabe besteht ähnlich wie ein C-Programm aus mehreren Übersetzungseinheiten, die aus Header-Dateien, Quellcode-Dateien oder reinen Rohdaten (etwa Daten-

sätzen zur numerischen Auswertung) bestehen können. Eine Übersetzungseinheit kann grob mit einer einzelnen Quell-Datei auf dem numerischen Server identifiziert werden.

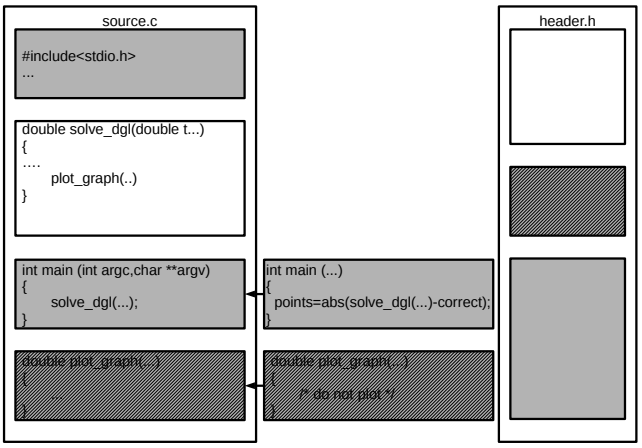

Abb. 3: Eine ViPLab-Aufgabe besteht aus mehreren Übersetzungseinheiten, von denen jede aus mehreren Codeblöcken besteht. Einige der Blöcke (grau) sind invariant, von denen wieder einige unsichtbar sein können (schraffiert). Im Korrekturmodus können invariante Codeteile durch Korrekturalgorithmen ersetzt werden.

Jede Übersetzungseinheit besteht aus mehreren Blöcken von Zeilen des Quellcodes oder der Quelldaten. Ein Block kann entweder durch die Studierenden geändert werden oder invarianten Code, etwa ein vorgegebenes Hauptprogramm, enthalten. Invarianter Code kann angezeigt werden, kann dann aber nicht bearbeitet werden, oder kann vollständig verborgen werden, siehe Abb. 4. Invarianter sichtbarer Code dient oft zur Vorgabe von Rumpfcode durch den Dozenten, etwa zur Definition eines festen Hauptprogrammes, zur Fixierung der erlaubten Include-Dateien für C usw., wohingegen unsichtbarer Code Bibliotheksfunktionen enthält, die für die Didaktik der Aufgabe ohne Belang, für die eigentliche Funktion und den Betrieb aber notwendig ist. Mit unsichtbarem Code werden etwa die Matlab-Funktionen zur graphischen Ausgabe überladen, so dass diese Ausgabe statt auf dem Server letztendlich im ILIAS-System am studentischen Rechner angezeigt wird.

ViPLab-Aufgaben sind dabei ILIAS-Entitäten wie alle anderen Aufgaben auch, d.h. können mit den bereits verfügbaren Mechanismen des LMS zwischen Dozenten ausgetauscht und geteilt werden.

Zwecks automatischer Korrektur können invariante Codeteile weiterhin beim Korrekturvorgang durch einen Korrekturalgorithmus ersetzt werden, siehe Abb. 5: Steuert etwa im Übungsbetrieb das invariante Hauptprogramm die Ausgabe des studentischen Codes auf die Konsole oder plottet diese, so kann das Hauptprogramm im Korrekturbetrieb den studentischen Algorithmus gezielt mit Testdaten versorgen und die Korrektheit der Lösung anhand der zurückgelieferten Resultate evaluieren. Die Korrektur besteht also einem vom Dozenten zu erstellenden Algorithmus, der als invariante Code-Teile in den studentischen Code eingefügt wird und der Daten oder Programmteile des Übungsbetriebes ersetzt.

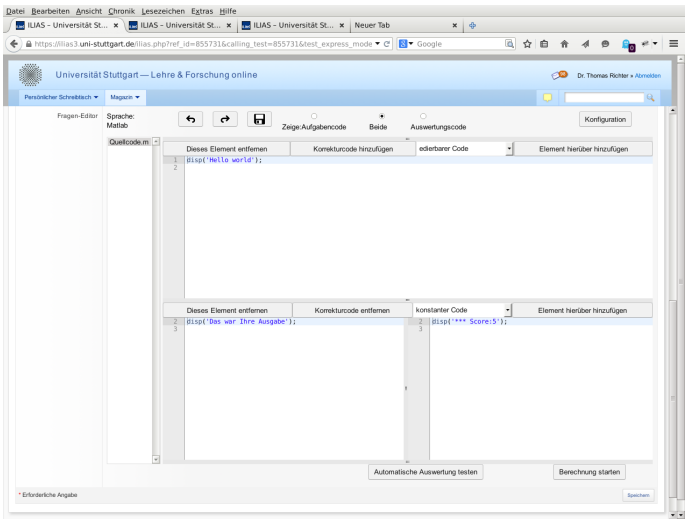

Abb. 4: Der ViPLab-Editor aus Dozentensicht. Oben ein veränderbarer Codeteil, darunter ein invarianter Codeteil mit einem Ersatzcode für die automatische Korrektur

Der Korrekturcode stellt somit keine Musterlösung dar, sondern füttert das studentisches Hauptprogramm mit Testdaten und evaluiert die Korrektheit der zurückgelieferten Resultate. Besteht etwa die Klausuraufgabe in der Implementierung einer numerischen Lösung einer gegebenen parametrisierten Differentialgleichung, so würde im Übungsbetrieb das vom Dozenten erstellte invariante Hauptprogramm die Anfangsbedingungen liefern und das Resultat des studentischen Codes ausdrucken oder plotten. Bei der automatischen Korrektur wird dieser invariante Code durch einen Korrekturcode ersetzt, der den studentischen Code etwa mit randomisierten Ausgangsdaten ansteuert und die Resultate mit der korrekten Lösung vergleicht. Im einfachsten Falle ergibt sich eine Punktzahl aus der Abweichung der vom studentischen Code gelieferten Lösung mit der korrekten Lösung.

Eine anderweitige algorithmische Bewertung des Codes selbst, etwa durch Codemetriken oder durch Beurteilung der Implementientierungsqualität findet momentan nicht statt, wäre aber prinzipiell durch den Korrekturserver möglich. Wir haben momentan vom Einsatz komplexerer Verfahren abgesehen, da für die in den ersten Semestern behandelten Probleme einfachere Auswertungsverfahren momentan noch ausreichend sind.

4 Erfahrungen mit ViPLab

Nach sechs Jahren im Einsatz ist ViPLab für die Studierenden der Universität Stuttgart zu einer Selbstverständlichkeit geworden, es ist ein Dienst des Rechenzentrums, von dem man erwartet, dass er stabil funktioniert. In [7] beschreiben wir eine quantitative Studie über die Nützlichkeit von ViPLab im Übungsbetrieb. Einer der Resultate der dort ausgeführten Studie ist, dass ViPLab im Vergleich zu einer Kontrollgruppe mit lokaler Softwareinstallation genauso gut angenommen wird, wobei wir eine signifikante negative Korrelation zwischen

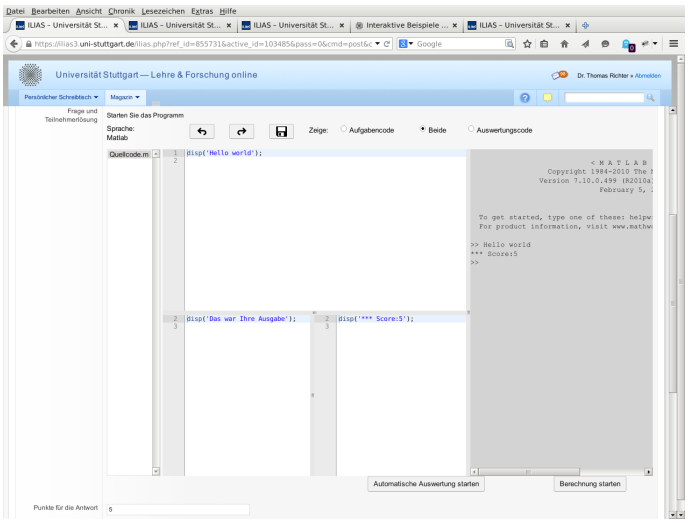

Abb. 5: Die manuell gestartete Aufgabenkorrektur aus Dozentensicht. Rechts die Ausgabe der Bewertung, links unten die im ILIAS eingetragene Punktzahl, die vom Dozenten überschrieben werden kann.

der Erfahrung mit dem Windows-Betriebssystem und der Zufriedenheit mit ViPLab festgestellt haben, d.h. Nutzer konnten mit ViPLab um so besser arbeiten, je weniger sie mit der Windows-Oberfläche vertraut waren. Für andere Betriebssysteme zeigte sich diese Korrelation nicht. Die Zufriedenheit und der Lernerfolg mit dem System insgesamt ist höher als die einer lokalen Installation, die Unterschiede sind aber statistisch nicht signifikant. Details finden sich in [7].

Technische Anregungen der Nutzer wurden dabei im Rahmen der Weiterentwicklung des ViPLab-Systems realisiert: Basierten die erste Version noch auf Java, so zeigte sich im Betrieb schnell, dass die Notwendigkeit der Installation und Aktualisierung des Java-Plugins eine Hürde für die Nutzer darstellte. Ebenso musste in der Anfangsphase ein Virtual Network Client (VPN) installiert werden, um den Dienst von außerhalb des Universitätsnetzes nutzen zu können; auch dessen Installation erzeugte unnötige Barrieren. Aufgrund dieser Rückmeldungen basiert die Version 2.0 auf Javascript und funktioniert auch aus öffentlichen Netzen heraus. Die Installation eines Web-Browsers ist für die Bedienung ausreichend, wobei ViPLab mit Firefox, Chrome, dem Internet-Explorer, Opera und Safari problemlos zusammenarbeitet.

Auf Wunsch der Dozenten ergab sich rasch die Notwendigkeit einer engeren Integration in das ILIAS-System: Die erste Version von ViPLab bestand aus einem SCORM-kompatiblen [5] Plugin und war somit zwar aus dem ILIAS-System heraus startbar, die Abgabe der Lösungen musste aber aufgrund von Beschränkungen des SCORM-Standards noch manuell über email erfolgen. ViPLab 2.0 ist stattdessen als ein ILIAS-Plugin realisiert, welches enger mit dem LMS verzahnt ist und Lösungen direkt in der ILIAS-Datenbank ablegen kann. Ebenso folgt die vollautomatische Evaluation durch den Korrek-

turserver dem Wunsch der Dozenten, nicht die Auswertung manuell für größere Zahlen von Studierenden anwerfen zu müssen, obwohl der zugrunde liegende Algorithmus als solches identisch ist.

Die Einfachheit der automatischen Bewertung, d.h. eine Bewertung durch den Austausch des Hauptprogrammes durch ein vom Dozenten zu erstellendes Bewertungsprogramm hat sich für die Anwendung im ersten Semester nicht als hinderlich erwiesen. Für größere Programmierprojekte ist ViPLab nicht entwickelt worden, und wir raten in diesem Falle Dozenten, stattdessen eine vollständige IDE einzusetzen. Der typische Einsatzzweck von ViPLab besteht in der Durchführung elementarer Aufgaben aus der numerischen Mathematik, die sich häufig auf wenige Zeilen Quellcode beschränken und die auf das Wesentliche reduziert sind.

5 Ausblick

Mit dem Aufbau des Korrekturservers hat sich die Möglichkeit ergeben, zur Bewertung auch komplexere Code-Metriken verwenden zu können, die dann auf dem Korrekturserver ablaufen. Auf diese Weise lässt sich auch eine Rückmeldung über den Programmierstil einer studentischen Lösung geben. In unseren Einsatzszenario ist dies allerdings meist von untergeordneter Bedeutung. Eine Plagiatserkennung findet momentan nicht statt, wäre aber im Rahmen des Korrekturservers durchaus möglich, indem dieser identische oder sehr ähnliche Abgaben identifiziert. Durch Randomisierung der Testdaten kann der Korrekturcode in begrenztem Umfang zumindest das Raten von Lösungen erkennen und verhindern.

Literaturverzeichnis

[1] GWT : Productivity for developers, performance for users. `http://www.gwtproject.org/`.

[2] ViPLab website. `http://www.uni-stuttgart.de/viplab/`.

[3] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. `https://tools.ietf.org/html/rfc7159`. RFC 7159, accessed 2015.

[4] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle, A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, A. Tatomir, M. Wolff, , and R. Helmig. Dumu^x: Dune for multi-{Phase, Component, Scale, Physics, ...} flow and transport in porous media. *Advances in Water Resources*, 34(9):1102–1112, 2011.

[5] Advanced Distributed Learning. SCORM Documentation Suite. `http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Documentation.aspx`.

[6] Th Richter, Stephan Rudlof, B. Adjibadji, Heiko Bernlöhr, Christoph Grüninger, Claus-Dieter Munz, Andreas Stock, Christian Rohde, and Rainer Helmig. Viplab – a virtual programming laboratory for mathematics and engineering. *Interact. Techn. Smart Edu.*, 9(4), 2012.

[7] J. Vanvinkenroye, Ch. Grüninger, C-J. Heine, and Th. Richter. A quantitative analysis of a virtual programming lab. *Proc. of Multimedia (ISM), 2013 Intl. Symposium on*, pages 457–461, 2013.