Abstract

In the process of the implementation of the eBologna program of the European states and the recent change of the German university system from the Diploma to the Bachelor/Master system, studies at German universities have been redesigned; courses have been condensed and learning content has been re-structured into granular “modules”, each of which requires an evaluation at the end of the semester. Simultaneously, the skills required for working as an engineer changed as well; handling of computer software, knowledge of mathematical or numerical algorithms and programming skills play an increasingly important role in the daily job routine of the working engineer. To support the learning by practical exercises, engineering faculties, mathematics and physics, and the Computing Center of the University of Stuttgart setup a project for implementing an online programming lab for teaching the required skills. The focus of this project is to provide easy access to the necessary software tools, avoid the overhead of installation and maintenance, and seamlessly integrate these tools into the eLearning infrastructure of the university. This paper describes the motivation and backgrounds, the software infrastructure and early results of this project.

1 Background and Introduction

Due to the change of the Diploma to the Bachelor/Master system at the German universities, courses at German universities had been redesigned: While the Master degree provides the necessary skills for working as a scientist in research, the Bachelor degree is supposed to teach students all necessary skills to work successfully in industry and applied fields. As such, knowledge of computer algebra systems, numerical methods and usage of software tools is of ever increasing importance. Lectures in the Bachelor studies thus include material on both the underlying mathematics as well as the usage and handling of such tools for quite a while. However, it turned out that simply providing exercises, skeleton code, installation media and the necessary software licences for students is not sufficient for a successful lecture. In our experience, a lot of time and effort has been spend on installing the software on the students’ computer systems, maintain the installations, down-load the homework and get the individual computer systems to work. Often as much as half a semester is lost by simple maintenance tasks on the students’ machines, wasting valuable lecture time. Furthermore, as students had to send homework solutions by email, they had to be checked by the teaching staff manually, and required careful adjustment of the installations to ensure interoperability between the students’ and the university machines.

To improve this situation, the University of Stuttgart setup the Virtual Programming Lab or short ViPLab project; the goal of this project is to implement an online access to all software tools required for the Bachelor study and integrate these tools into the eLearning infrastructure of the university. The students should profit from tools that aid in programming while not having to deal with the nuisances. Namely the browser-based interface should be appealing, and help focus on the task at hand: numerics, or programming in general. At the same time it should provide an integrated programming environment in the sense that compil-
ing, sending in solutions to homework or visualizing simulations reduces to mouse clicks.

Installation of software components on the students’ machines should thus be limited to an absolute minimum, namely to keep the maintenance overhead low; the goal was that a browser with a Java plug-in should be sufficient to run the whole system. Furthermore, access to homework assignments and hand-over of students’ solutions should all be possible within the Learning Management System (LMS) of the university. That is, homework assignments should be found consistently within the same system that also offers access to scripts, lesson schedules etc.

The necessary software infrastructure was implemented by the Computer Center (RUS) of the university which also installed and maintains the numerics cluster that runs the solutions handed in by the students. Users of the system are currently the Institute of Aerodynamics and Gas-dynamics, Aerospace Engineering and Geodesy (IAG), the Institute of Hydraulic Engineering (IWS), and the Institute for Applied Analysis and Numerical Simulation (IANS). Needless to say, the project is challenged by the various needs of its diverse user group and had to be designed to supply access to manifold mathematical tools, such as MATLAB, C or C++ compilers or specialized numerical simulation frameworks. The project was entirely funded by students tuition and promoted by the students of the university and is available under an Open Source Licence.

The ViPLab is of course not the first of its kind: It is in a long tradition of virtual laboratories that were designed to provide simulations to complement traditional hands-on courses for students in the engineering studies, see for example [3] and [4, 1, 2]. As far as computer science goes, the VPL module for Moodle [8] provides a similar plug-in for one popular Learning Management System. Quite unlike VPL, the ViPLab infrastructure is generic and works on any SCORM [5] compliant LMS infrastructure, and it is built around a very scalable and flexible load balancing architecture that, according to our tests, can handle several hundred students at once — see also section 4. As far as assessment software for mathematics is concerned, similar commercial products are available as well. One of them is the MapleTA [6], which, however, is restricted to one particular back-end and thus not quite as flexible as ViPLab. Despite that, it does not allow free programming in the same degree as ViPLab does.

This article is structured as follows: In the next section, the technical infrastructure of the project is described which is followed by some examples. Following that, insight into experiences with the system is given and student voices are presented. In the last section, we provide an outlook into future developments of the ViPLab.

Figure 1. The architecture of the ViPLab. For both student and teacher the infrastructure is solely operated through a browser. The actual computation is run on a backend which may support one or many numerical software packages, compilers or simulation software. Communication between the components is entirely based on HTTP/REST.

2 Software Architecture

Conceptionally, the ViPLab architecture, see Fig. 1 consists of three modules: A front-end Java applet that runs on the students’ machine and provides an editor for the source code and several visualization toolkits for rendering plots and graphics; a load balancing middleware, called the eLearning Community Server[11] (ECS), which collects requests from the front-end and passes them over to the number-crunching back-end. The ECS also provides a simple database that keeps all homework assignments in the form of code-templates and back-end configurations. The third component is a back-end which runs the actual numerical software, let it be a C or C++ compiler, a mathematical or numerical software like GNU Octave or MATLAB or a simulation framework like DuMu? [9, 10].

The communication between them is based on REST [12], a stateless protocol based on traditional HTTP or HTTPS using a trivial text encoding called JSON [14]. Quite unlike the heavy-weight XML encoding SOAP [13] works with, JSON is trivial to parse and easily extensible. That is, every homework “exercise” consists of a single database entry at the ECS middleware encoded in JSON and transmitted to the corresponding student front-end, see Fig. 2. A homework exercise is basically a template consisting of editable or non-editable (constant) code segments which has to be completed by the student to form a running program.

The data-base entry to request is part of the SCORM module configuration, and thus encoded within the module loaded from the Learning Management System. Similarly,
Figure 2. The student front-end, here on a simple linear algebra exercise. The student solution is on white on the left-hand side, the template in grey on the left, the output of the system on the right.

Each solution the student hands in is encoded in JSON by the front-end and posted to the ECS. One or several back-ends poll the ECS for posted solutions, merge the solution with the template, perform a syntax and validity check on the code and then compile and/or execute the students’ solutions. The output of the back-end, let it be text or graphics, is again encoded in JSON and posted to the ECS at which now the front-end polls. This result is then polled by the front-end and displayed or rendered there.

As already stated, the front-end is a Java applet running in the students’ browser; it is delivered by the Learning Management System of the university as a SCORM module — basically HTML code plus JavaScript for the communication between the LMS and the applet — that contains both the applet code and the configuration of the applet. This configuration states both the URL of the ECS middleware and a database ID identifying the code template to load into the front-end.

In addition to the student applet, an enhanced “teacher” front-end is also supplied which is used to create the code-templates, i.e. homework assignments, see Fig. 3. While both teacher and student front-end work from the same code-base, the teacher applet provides some additional functionalities and is only accessible by university staff and protected by HTTPS authorization. Thus, a password is required for its access. Unlike its student counterpart, the teacher applet includes the necessary credentials to gain write-access to the ECS code template, i.e. the homework database, and interaction between teacher client and ECS middleware is secured by HTTPS.

While possibly several hundred instances of the front-end and several back-ends run simultaneously, the middleware is a single central architectural component mediating between the two other components. Originally, the ECS middleware had been developed to share eLearning content between universities by connecting their Learning Management Systems with each other, though with some minor enhancements, the same software has been utilized by the ViPLab project to facilitate the communications between front and back-end.

As all interfaces are completely based on REST, all communication uses the HTTP or HTTPS protocol, and each item or end-point for such communications is a URL. That is, a homework assignment, i.e. code template, is available under a URL given in the applet configuration, the student solution is posted to a first-in first-out queue at the ECS which is polled by the back-end; transmission of computational results is similarly based on FiFos accessible at specific URLs encoded in the front- and back-end.

Back-ends are typically instances of virtual machines running in a large computer cluster, as currently employed by the Computing Center of the University of Stuttgart. These back-ends poll student solutions from the ECS and, after preprocessing, execute the resulting code. Output generated by this code is again encoded in JSON and posted back in one of the results queues of the ECS.

Preprocessing of student solutions is performed for two reasons: First, because the student solution needs to be merged with the homework template from the ECS. This merging process may provide a main program or additional

Figure 3. The teacher front-end showing the design of a simulation exercise. To the bottom right an interactive 3D plot rendering the result. The same visualization is also available in the student applet.
service routines required for running the exercise; for example, in case of MATLAB, the output of any visualization command — e.g. the `plot` command — must not go to the back-end, but to the front-end, and thus such commands must be replaced on the back-end side. The second reason for pre-processing is security: Needless to say, executing user-defined code on a cluster is a potentially dangerous operation as it may compromise the cluster. The preprocessing thus analyzes the student code and detects any functions that may damage the integrity of the server. That is, any function or method call that interacts with the server file system or executes a system command is identified, execution of such code is prohibited and a corresponding error message, if any, is generated on the back-end and forwarded to the front-end.

This function filtering may also be deployed to disable functions students are not allowed to use: If, for example, the task is to implement a numerical approximation of a transcendental function — say $\sin$ — then clearly students must not call the $\sin$ function but rather implement something on their own. The list of the function filter is also part of the code template and configured through a corresponding entry in the JSON format encoding the homework.

To enhance security further, the user code is neither run directly on the numerical cluster, but in a so-called chroot cage\(^1\) within which the code has only read-only access to a limited number of system components. Modification of any system resources from user programs is thus prohibited by a second line of defense. Since each back-end process is executed in a chroot cage, solutions posted by different users run in differing environments and are thus also isolated between each other.

### 3 Integration of the Architecture

Integration of the overall architecture into the eLearning infrastructure of the University of Stuttgart was an important goal to begin with. ViPLab should be easy to use for the teacher and the student, and should use and depend on the tools both user groups already know. Many commercial systems like MapleTA [6, 7] are stand-alone, but as such also separate from the remaining infrastructure and cause additional work as soon as they need to be linked to an existing infrastructure.

A second challenge we had to address is that of visualization; since the numerical software is running on the back-end and not on the students’ machines, any graphical output generated by such software would — without further change — appear needlessly on the back-end and would remain invisible for the student.

\(^1\)A chroot cage makes most file system components invisible to an executing process by remounting the file system root to an isolated directory

For integrating our software into the eLearning infrastructure, we depend on the SCORM standard [5]: it defines re-usable eLearning components — SCOs — that can be re-used and plugged into any compliant eLearning system. Homework exercises are prepared on a (access protected) web-page rendering the teacher client, see Fig. 3, and can be tested and maintained there; to this end, it provides also access to a database of all homework exercises. To make a homework accessible to the students, the teacher clicks on the SCORM button in the teacher front end. This will generate a ZIP file containing a SCORM object on the teachers’ desktop, which can be directly uploaded into the Learning Management System.

While SCORM is quite complex and rarely implemented completely, we only need a small subset of the standard that is supported by most eLearning systems, for example by Moodle [15] or by the Ilias [16] system deployed in Stuttgart. SCOs — reusable eLearning components — are represented by a ZIP archive containing basically a HTML page to render, plus meta-information on the package. That is, SCOs are basically “canned” web-pages. Communication between the eLearning content and the Learning Management System is possible, for example to give credits for exercises or to store the solution of the homework assignment found by the student. This communication is entirely based on JavaScript, and as Java offers also a bridge to JavaScript, data from the applet can flow back into the Learning Management System.

The second integration task concerns visualization of computational results; while numbers provide in principle all the information, a visualization is clearly considerably easier to understand and, from a pedagogical point of view, very desirable. Visualization and usability were identified as mission critical milestones of the whole project. Visualization of of data is the point where the students are typically confronted with an unnecessary overhead in the form of getting to know different kinds of visualization programs. However, with learning programming being the main goal, a seamless integration of visualization helps focusing the learning experience of the students.

Unfortunately, visualization toolkits offered by the deployed numerical software are of little use as they would be only visible on the back-end server. Thus, we developed plug-ins for MATLAB which overload the build-in visualization commands, most notably the `plot` command, which write the data points to a file instead of rendering it to the screen. This data file is then transmitted as part of the result via the ECS to the front-end, encoded in JSON similar to the rest of the result. Visualization tools in the client code then replace the corresponding back-end code. The currently available tools, namely a line (2D) plot, a grid (contour) plot and a 3D plot cover most applications we have in the
current lectures, though more visualization tools are likely merged into the front-end in the future.

4 Experiences

The system described above has been implemented by the Computing Center starting in 2009 and, after prototypical deployment in 2010, is now in production use by several institutes of the university, namely the IAG, IWS and IANS. A relatively new user is the University of Linköping in Sweden where students use ViPLab to program in Open-Modelica [17], a language to develop simulation models. In this section, our experiences in using the ViPLab shall be presented.

Before starting the production server, the computing center ran a stress test to estimate the performance of the overall system design. According to this test, a simple computer pool consisting of standard PCs was able to handle up to about 200 students simultaneously, proofing the soundness of the overall design. The current installation uses more powerful professional server class hardware instead.

After the successful test and a prototypical testing by volunteering students, the system was rolled out to the university; by far the largest user group is formed by the students of the undergraduate Numerical Mathematics course offered by the IAG, approximately 150 students currently participate here. A smaller number of students use ViPLab at the IWS and the IANS.

Not quite unexpected, we found a couple of issues during the first days of deployment: Initially, access to the system was protected by the border-router of the university. Students that were trying to access their homeworks from home had to setup a Virtual Private Network (VPN). The main reason for this restriction had been licensing issues for the numerical software running on the back-end. Unfortunately, students had many problems configuring the VPN correctly and then had no access to the ViPLab. This issue has been resolved now: As ViPLab is only accessible through the Learning Management System which controls membership in one of the university courses, we could open access to the system.

Some students had problems with installing the Java plugin, mostly users of Apple hardware. Unfortunately, Apple supplies only an outdated version of Java for its older machines and does not provide updates. To resolve this issue, we lowered the requirements of the front-end and re-implemented all of its functionality to work on a Java 1.5 virtual machine. A second class of users had problems accessing ViPLab from Linux machines where, by default, an incomplete and incompatible virtual machine is installed. Luckily, Linux users are usually more experienced than other users and could resolve the problems manually after we announced that the Sun (now Oracle) version of Java had to be installed.

A third group of problems was caused by the network access, namely that ViPLab requires connection to the ECS to load the code templates and to post solutions. We found that not all lecture halls in our university offer sufficient WLAN access and the network connection is, in some of them, slow or unreliable. Most of the time was spend in loading the actual ViPLab front-end as the communications overhead to post solutions and retrieve exercise results is rather small. To address this problem, student groups had been relocated to other rooms providing better wireless access; we also rewrote parts of the applet to shorten it. It is currently only one third of its original size deployed at the start of the semester.

Performing a reasonable user study turned unfortunately out to be nearly impossible because the lectures within which ViPLab is applied were redesigned; before the introduction of the virtual lab, students had to learn FORTRAN as a programming language, and not Matlab as now, and as such, exercises and content differ significantly. We were thus neither able to setup two groups of students and perform a study on a control group. Student satisfaction of “Numerical Mathematics” remains unfortunately still low, and it is one of the “tough” lectures students have to master. A virtual programming laboratory eases access to the technology, but does not change other factors such as the condensed Bachelor-Master program, neither does it significantly improve the performance of students under such constrained conditions. However, after solving initial technical problems, students now accept the system as self-evident and use it either in the computer pools on the campus or at home, and access it as a regular part of their courses. We also performed interviews with student lecturers participating in both courses, the FORTRAN based course, and the new Matlab course. According to their voices, the ease of use and the accessibility of the technology improved significantly.

5 Outlook

While the current ViPLab is already in productive use and addresses most needs of our users, we still have a couple of important goals on our agenda. While the architecture is currently deployed and accepted for homeworks, we plan to also implement electronic exams in the future. The challenges here are, however, mostly non-technical but rather organizational. For example, legal requirements for such exams need to be clarified, and we must ensure sufficient availability of computer hardware for this application; interestingly, availability of back-end capacity is not the prob-
lem here, but rather sufficient network bandwidth in the
lecture halls, and sufficient availability of computer hard-
ware at the students’ side. Unfortunately, the University
of Stuttgart currently does not offer a central computer pool
large enough to allow such scenarios, and we can legally not
enforce students to buy a system for assessment purposes.

Clearly, additional back-end systems are desirable. De-
spite the already existing C/C++, Dumu\textsuperscript{4}, MATLAB and
Octave configurations, we are looking into offering the sta-
tistical package \texttt{R} and Java in the future. Furthermore, the
student-side code editor is planned to be enhanced by addi-
tional features such as a bracket-checker and syntax high-
lighting. The overall goal of the project is to allow students
to focus on the key learning goals, namely programming,
which should be as comfortable as in MATLAB and as open
as C.

6 Acknowledgements

The authors want to thank the Computing Center for par-
tially sponsoring the project and the hardware.

7 Bibliography

References

ture For Virtual Laboratory Experimentation”, Proc. of

S. Gao: “A Virtual Laboratory On Fluid Mechanics”,

Remote and Virtual Experiments in eLearning in Statis-
tical Mechanics and Thermodynamics.” J. of Software

chitecture of Virtual Laboratories on Mathematics and
Natural Sciences.” Proc. of 3rd Intl. Conf. on Multi-
media and ICTs in Education, June 7-10, 2005, Cac-
eres/Spain, Badajoz/Spain, June 2005. FORMATEX.

[5] Advanced Distributed Learning: “SCORM Documen-
tation Suite” (SCORM 2004 4th Ed.), available online
at http://www.adlnet.gov/Technologies/

Assessment”, available online at http://www.
maplesoft.com/products/mapleta/ (2011)

tation of Test Items”, Technical report, available
online at http://www.adeptsscience.co.
.uk/products/mathsim/mapleta/MapleTA_
whitepaper.pdf (2011)

[8] J.C. Carlos Rodriguez-del-Pino: “VPL — Virtual Pro-
gramming Lab for Moodle”, available online at http:
//vpl.dis.ulpgc.es/ (2011)

toolbox for flow and transport processes in porous me-
dia.” In: A. Ibrahimbegovic, and F. Dias (Eds.): ECCO-
MAS Thematic Conference on Multi-scale Compu-
tational Methods for Solids and Fluids (28. - 30. No-

[10] B. Flemisch, M. Darcis, K. Erbertseder, B. Faigle,
A. Lauser, K. Mosthaf, S. Müthing, P. Nuske, P.,
A. Tatomir, M. Wolff, R. Helmig: DUMUX: DUNE
for Multi-\{-Phase, Component, Scale, Physics, ...\} Flow
and Transport in Porous Media. Advances in Water Re-
sources (2010). submitted

able online at http://freeit.de/en/ecsa/
ecs.html,(2011)

of Network-based Software Architectures”, Disserta-
tion of Univ. of California, Irvine, available online
at http://www.ics.uci.edu/~fielding/

Framework (Second Edition)”, W3C Recommendation
27, available online at http://www.w3.org/TR/
soap12-part1/#intro (2007)

[14] D. Crockford: “The application/json Media Type for
JavaScript Object Notation (JSON)”, IETF RFC 4627,
available online at http://www.ietf.org/rfc/
rfc4627.txt?number=4627 (2006)

for learning”, available online at http://moodle.
org/ (2011)

[16] ILIAS open source e-Learning e.V.: “ILIAS Open
Source e-Learning”, available online at http://
www.ilias.de/docu/ (2011)

Specification, Version 3.2”, available online at
http://www.modelica.org/documents/
ModelicaSpec32.pdf (2011)