An Experimental Environment for Teaching Java Security

Anna Riccioni
DEIS - University of Bologna
v.le Risorgimento 2
I-40136 Bologna, Italy
anna.riccioni@unibo.it

Enrico Denti
DEIS - University of Bologna
v.le Risorgimento 2
I-40136 Bologna, Italy
enrico.denti@unibo.it

Roberto Laschi
DEIS - University of Bologna
v.le Risorgimento 2
I-40136 Bologna, Italy
roberto.laschi@unibo.it

ABSTRACT

In many fields of Computer Engineering education it is crucial that students gain both conceptual understanding and practical skills. To this end, an effective teaching approach relies on a blended-learning strategy that combines face-to-face lessons with students' practice on some suitable educational tool. Such a tool should support students through the whole learning process, and help them to improve their modelling ability along with their programming expertise. In this paper we describe S-vLab, a virtual laboratory for supporting teaching and learning in several applicative domains; in particular, we analyse S-vLab applied to the Information Security field. In this context, one of its main goals is to help students in dealing with the Java Security Platform, supporting different learning styles and allowing multiple formative paths. Using S-vLab students can, in various steps, model a secure system, simulate its behaviour and evaluate its performance, and finally build and test a software prototype by leveraging numerous facilities. These include the provision of extracts of sample Java code, the availability of supporting resources and a continuous and immediate feedback, aimed at pointing out possible errors and suggesting solutions. After describing more in detail the features provided by S-vLab, as well as how the depicted functionalities can help in achieving the settled learning goals, we refer on the results obtained in an experimentation of the virtual lab during a course on Information Security with about 150 students.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer Science Education; D.3.3 [**Language Constructs and Features**].

General Terms

Experimentation, Languages, Performance, Security.

Keywords

Virtual Laboratory, Computer Engineering Education, Java Cryptography Extension.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PPPJ 2008, September 9–11, 2008, Modena, Italy.
Copyright 2008 ACM 978-1-60558-223-8/08/0009...$5.00

1. INTRODUCTION

Java is deserving increasing attention in Computer Engineering curricula [26]. In fact, introductory Computer Engineering courses, aimed at teaching the fundamentals of programming, often choose Java as the reference language. Such courses provide learners with a broad view of the Java language, platform and technology, while advanced courses focus on specific facets. Among the various possible advanced courses, nowadays those focussed on security-related aspects play a strategic role. E-government, e-business and e-health are a few samples of widespread fields which strongly rely on security issues; thus, knowing how to build and manage a secure system becomes a key ability for computer engineers. According to the modern idea of education as a combination of conceptual understanding and practical skills [22, 6], Information Security courses should focus on teaching both theoretical principles and their practical applications. To this end, supporting both students' learning of theoretical security basis and practice on the Java Security Platform is of primary relevance.

However, in the typical higher education context several factors concur to make this a challenging task. One first aspect is students' heterogeneity: especially in advanced, optional courses, attendees often have different backgrounds and starting competence – for instance, in case of students attending a Computer Engineering curricula who take a course together with others enrolled to Telecommunications or Electronics specializations.

Furthermore, students with different expertise are likely to achieve the course learning goals to different extents in terms of in-depth insight into the covered topics and complexity of the practical experimentations.

Finally, it is widely acknowledged that each person follows his/her own learning style and approach [10, 2]. Examples of learning styles include the *learning-by-doing*, *learning-by-example* and *learning-by-exploring* strategies. The *learning-by-doing* method is rooted in the constructivism cognitive theory. Its basic assumption is that learning is more effective when a student is actively involved in the construction of knowledge rather than passively receiving it. However, the effectiveness of active instructional techniques has been recently questioned [21]. Several authors document that autonomous activities can be motivating for learners who have sufficient background competence and expertise to successfully complete the given tasks, while novices can find un-

guided exercises too difficult. In these cases, the *learning-by-example* strategy, where learners follow step-by-step demonstrations of how to perform a task or how to solve a problem, can be a more adequate approach [28]. Furthermore, several authors in the literature recognize the importance of *learning-by-exploring*, where learners examine the supporting materials for elements addressing their specific interests and needs [6].

To cope with these issues, students could be provided with an experimental environment suitable for supporting their investigations [14]. More in detail, referring to the Computer Engineering education context, the integration of theoretical knowledge with technical skills can be effectively supported through a learning tool that allows students to:

- design a model of the system they are going to study;
- evaluate the system behaviour and performance;
- build a software prototype.

To accomplish the first task, such an educational tool should include an interactive editor and a set of basic components for building a simplified yet representative version of the considered protocol or system. A simulator and a set of data gathering and analyzing tools could help to perform the second step. Finally, autonomous practice within an Integrated Development Environment is an invaluable training for learning how to implement the studied system. In addition, the educational tool could support users through the whole exercitation session for instance by providing hints, help examples for specific tasks, as well as an immediate feedback on the outcomes yielded by their actions. The considered environment should also be suitable both to novices and advanced students, and be adaptable to different learning styles. To this end, it should allow users to experiment at different levels, taking basic to complex exercises, and providing several tools and features for achieving the settled learning goals through different paths, freely forged by the students, as well as guided by given suggestions.

Following the depicted guidelines, we developed an open source experimental environment aimed at supporting students' learning and practicing in Java-based security exercises [1]. There, students' proficiency in Java is mainly connected to their knowledge of the Java Cryptography Architecture (JCA) and of the major classes and methods in the Java Cryptography Extension (JCE). Since cryptographic operations often involve heavy computations, tools and strategies for evaluating the performance of systems assume a strategic importance: in fact, the implementation of a software prototype should always be based on a preliminary deep and careful examination of possible critical aspects in terms of computational resources.

Despite its application to the security context, such an experimental environment is indeed designed according to general principles, so as to be easily exportable in several other Computer Engineering courses.

2. THE EXPERIMENTAL ENVIRONMENT

In the context of Computer Engineering education, the learning objectives include the ability to model a system, to evaluate its performance and robustness, and to build it, possibly in a prototypal version. This learning process can be represented as an incremental cycle, composed of four distinct steps. The first is a *modelling* stage, where students are called to design the secure system they are going to study and implement. Then, the *evaluation* step focusses on the system model and configuration: students analyse the system behaviour and test how its performance responds to variations of the characteristic parameters. The software *implementation* stage, where learners build their prototypes, is followed by the *test and evaluation* phase. The last step aims to identify architectural optimizations and possibly more suitable deployment solutions.

Different paths are practicable through the depicted stages, each with its own strengths and motivations. For instance, a possible learning path could start with the system modelling phase and proceed towards through the model evaluation, the prototype implementation and to test phase, and then start a new cycle, in an spiral-shaped incremental process. An alternative approach could instead move from the implementation phase, and then cycle between consequent software refining and testing sessions: this kind of learning experience is more likely selected by students with a quite high programming expertise and a sufficient knowledge of the theoretical basis of the considered subject.

2.1 S-vLab: goals and architecture

In the context of Information Security, the learning process can be effectively supported through an interactive virtual laboratory where students can arrange and perform their experiments by modelling, analysing, implementing and testing a secure system. To this end, the virtual lab should provide a graphical editor and a set of building blocks suitable for designing a simplified yet representative version of a system or protocol. Thus, the available basic components should include the parties involved in cryptographic protocols as well as the security mechanisms and services needed to recreate a wide variety of scenarios. Then, to test and evaluate the effectiveness and the robustness of the considered solution from an operational point of view, students should be able to simulate attacks. By playing the attacker role students gain a greater insight into how to analyse systems for weaknesses, how to devise designs avoiding these flaws, and how to deploy defenses. The data analysis tools included within the virtual lab also support students in system performance evaluation. Referring to the security case study, tools for measuring the operation execution time or for verifying that the yielded data satisfy some desired statistical properties are two possible strategic features. Furthermore, learners can improve their programming skills by directly implementing and testing their code.

Such features effectively support a *learning-by-doing* strategy, since settling and performing experiments requires students to play an active role in the learning experience. This methodology also stresses the learners' self-determination, as it usually does not provide fixed and detailed instructions to perform the exercitations, but simply declares the goal that students should achieve.

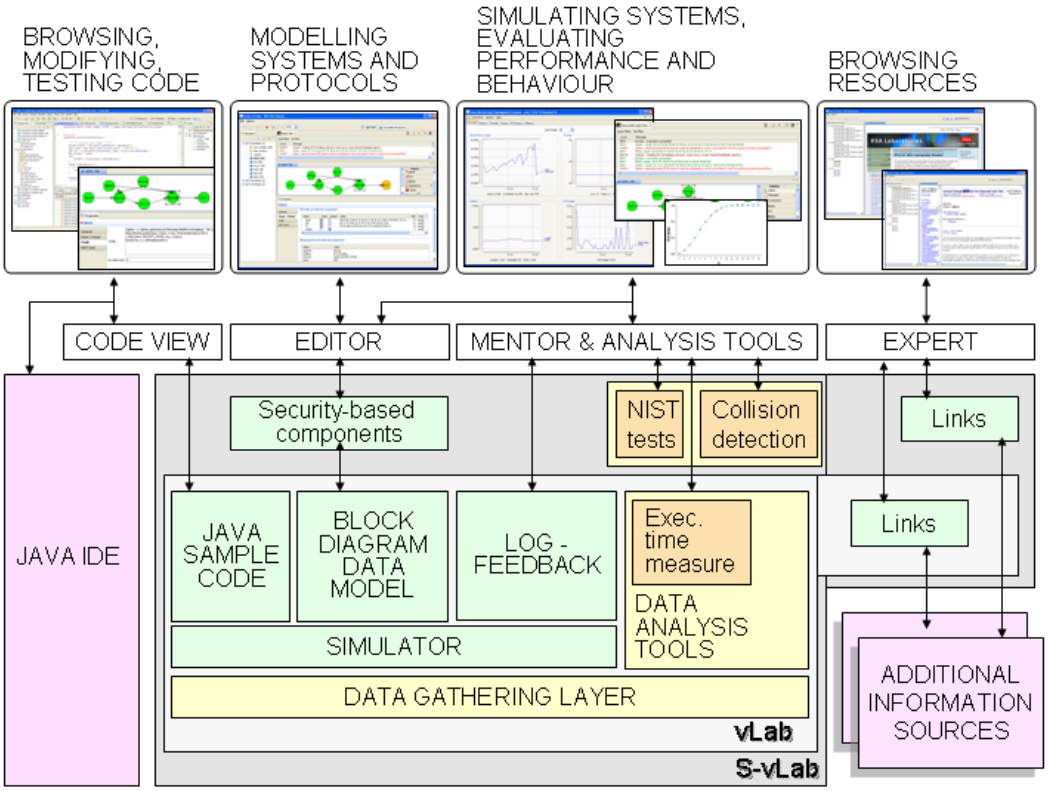

Figure 1: S-vLab Architecture

At the same time, a *learning-by-example* approach is supported via structured, guided paths, as well as wizards to simplify the modelling step. Moreover, in order to support both novices and advanced students, the laboratory should include a feature to fill the gap from modelling to implementation, such as providing students with snapshots of sample code to facilitate the prototype implementation through a *learning-by-example* strategy. In fact this approach, by enlightening the transition from a graphical model to a sample and partial code, helps novices in understanding the responsibility of each code extract better than a tutorial. Yet advanced students could exploit the generated sample code as a starting point to be refined, integrated, re-engineered and structured in a more complex software architecture.

Of course, students should also be enabled to quickly access informative resources, such as reference standards, Java technical documentation, RSA standards, RFC documents, Java Cryptography Extension API, etc: in fact, providing such materials is strategic for supporting the *learning-by-exploring* approach as well as the other learning styles.

The resulting architecture of S-vLab is shown in Figure 1. The workbench includes an editor and a palette of domain-specific components, like actors and operations. Palette items are organized in groups, such as Actors, Cryptographic primitives and Mathematical operations: in fact, S-vLab allows students to perform experiments at various detail levels, since it provides both the cryptographic primitives and the mathematical operations needed to build them. The editor and the underlying simulator are based on the block diagram abstraction: students can design the structure of the considered system in terms of interconnected blocks, by dragging and dropping the desired elements from the palette. Then, they can simulate the system behaviour by executing each item, being careful to observe possible precedence constraints due to the data flow among blocks and connections.

S-vLab includes a set of functions for evaluating the randomness of a byte sequence and detecting collisions, allowing students to perform interesting experiments within a secure system. Moreover, during the simulation the virtual lab tracks the execution time and the memory occupation for each block of the system. Thus, after a few experiments, students gather a set of data suitable for comparing several possible configurations of the studied system.

In order to make the learning experience as effective as possible, it is crucial to provide students with a continuous and immediate feedback on what is going on within the exercitation. To this end, the virtual lab includes a help tool called *mentor*, whose main purpose is to supply various message typologies, ranging from purely informative details to warning and errors: in this case, if possible, it suggests how to fix the occurred problem. In the context of Information Security, a typical error message provided by the *mentor* can notify students trying to build a cryptographic key that the requested size in bits is not available, due to the chosen algorithm or to the Java cryptography restricted policies.

Browsing informative resources such as the cited standards or technical documentation is another key issue for learners.

So, S-vLab provides a further supporting tool called the *expert*, whose goal is to allow navigation through the available external resources.

Furthermore, the *judge* is intended to guide the student through the possible learning tasks according to a declared learning goal and to the student's attested starting competence and preferred learning style. Its responsibilities include the evaluation of the student's performance within the exercitations, providing feedback and updating his/her profile.

Finally, in order to help students to become familiar with the Java Security Platform, S-vLab can provide extracts of sample code for each component in the palette. When a system model is drawn, S-vLab can automatically generate the corresponding sample Java code to be completed by the student. This is why S-vLab is strictly integrated with an IDE where students can freely modify and extend the basic sample code and autonomously write new programs starting from scratch.

2.2 Deployment

In order to address the above requirements, a plug-in based architecture seemed to be an effective solution for designing and implementing the virtual lab. Thus, the Eclipse platform was a natural candidate: in fact, building S-vLab on top of Eclipse simplified the creation of a close link between the modelling editor and the Java IDE, as well as the virtual laboratory implementation, given the wide availability of projects, facilities and third-party plug-ins that helped in developing several features. As a further benefit, Eclipse is a well-known reality both in the academic and in the professional scope, and often computer engineers start learning how to deal with it during their first years at University. So, students are supposed to be quite familiar with S-vLab user interface, as it relies on the same abstractions used by Eclipse: the major ones are views, editors and perspectives. In Eclipse glossary, a *perspective* defines the visual arrangement of the workbench parts, which can be *views* or *editors*. One of the main differences between these is the permitted number of active instances at the same time: there is generally only one instance of a given view per workbench page, but there can be several instances of the same type of editor. Furthermore, editors can be associated with a file name or an extension and can be in a "dirty state", meaning that their contents are unsaved and will be lost if the editor is closed without saving, while modifications made in a view are saved immediately. Finally, editors can appear in only one region of the page, whereas views can be moved to any part of the page and minimized as fast views.

S-vLab is distributed in two releases: plug-in and stand-alone. The former should be installed within an Eclipse platform. However, since it is widely acknowledged that the most desirable characteristic in an educational tool is simplicity, for a complex tool could discourage students, we provided also a stand-alone release of S-vLab [15, 31]. This second version is implemented as a Rich Client Application, a rich client built on top of the Eclipse Rich Client Platform [23]. Such approach allowed us to leverage the Eclipse architecture to design a flexible and extensible application reusing numerous existing functionality and patterns. The availability of two alternative releases is a good compromise between the tool simplicity and powerful potentialities: in fact, the RCP version simplifies the installation and management issues for users who prefer to avoid updating their Eclipse instance, and at the same time preserves most of the virtual lab characteristic features. On the other hand, the plug-in release is more appealing for advanced students who prefer to experiment more complex tasks and, possibly, to extend the virtual lab functionalities by relying on additional Eclipse features.

2.3 S-vLab and Eclipse: perspectives

As previously discussed, S-vLab, in both its versions, relies on a graphical editor and on a set of related views. It also provides several perspectives, each suitable for a specific stage of the learning process. For instance, the *Exercitation perspective* is indicated for modelling and simulating the system, in order to perform some first, simple evaluation on its behavior. Its default configuration, shown in Figure 2, includes the graphical editor along with the *mentor*, the navigator and a tabbed property view.

The navigator view, like in Eclipse, is used to display and navigate through the workspace. In fact, students can organize their exercitations in projects, and the navigator allows users to visualize and modify their structure, as well as to select a file and open it in a proper editor. Figure 2 represents a typical schema where Alice and Bob exchange ciphered messages through a non-secure channel where an eavesdropper (Eve) could be listening. The graphical model includes both the involved actors and the symbols representing cryptographic operations, and each component has one or more pins for inputs and/or outputs that can be used to connect blocks. Each item in the model has a specific function that, in order to be completed, must be properly initialized and then executed.

To this end, the tabbed property view assumes a strategic importance.

2.3.1 The Input/Output tab

The Input/Output tab helps users in understanding the relation occurring between the input and the output of the considered component by means of two interactive tables. The first is dedicated to the input and output data managed by the considered block. In fact, each pin is described through several information, like its name, its direction (input or output), the type of the associated data, its value codified in hexadecimal format, its size in terms of number of bytes. The second table presents a list of the component parameters: by varying them, users can specify in detail how the inputs will be transformed into outputs. In fact, each component represents a general cryptographic operation that can be specified through a set of parameters. As an example, the Cipher in Figure 2 symbolizes the operation of encrypting or decrypting a message: the specific function needed in a given case study can be obtained by setting the Cipher characteristic parameters. Among these, there are the operating direction (encrypting or decrypting messages), the algorithm we want to be used, the operational mode and finally the padding scheme to be added at the end of the input. The parameter values, in most cases, can vary in a finite range of feasible options, so users are usu-

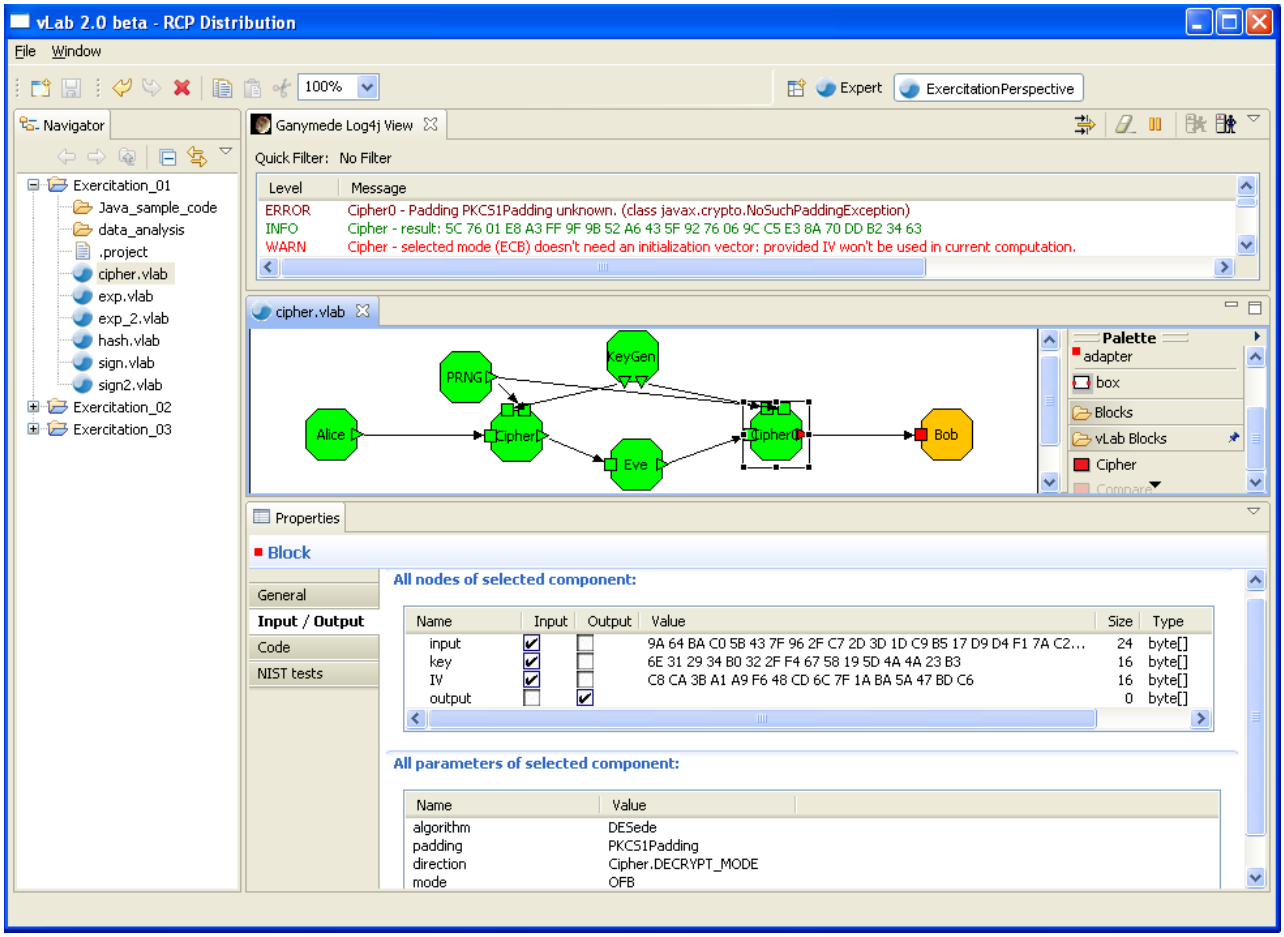

Figure 2: The Exercitation Perspective within S-vLab

ally requested to choose within a set provided by the virtual laboratory. S-vLab, in fact, relies on lists to show the feasible values for parameters when they are part of a finite and small set.

At this stage, S-vLab helps users to orient themselves in the wide choice of Java classes, methods and setting parameters. In fact, the selection from the palette, the configuration and the execution of a graphical component representing an operation correspond to instantiate and initialize an object of the proper Java class and then invoke one or more of its methods, in order to perform the desired task. The availability of a list containing the component parameters helps students to map conceptual decisions to the practical ways for defining them. Moreover, the Java Cryptography Architecture is based on providers for the implementation and possibly the extension of the classes exposed through the Java Cryptography Extension. S-vLab allows students, when they are configuring a specific component, to request the implementation of a desired operation supplied by a specific cryptographic provider. The desired option can be selected through a list containing all the available providers in the local machine.

Finally, both the *mentor* and the simulator help students to understand which parameters are mandatory, which are optional (and will not be used), which configurations are feasible and which contain errors that make the operation not executable. In fact, the simulator prevents the execution of not fully configured components, while the *mentor* provide possible feedback on inconsistent, incomplete or redundant configurations.

2.3.2 *The Code tab*

While designing, configuring and executing a system model, the Input/Output tab is precious to provide students with a broad view of the operation-specific options. However, the strongest bind between the modelling and the implementation step is represented by the Code tab (Figure 3).

The Code tab is part of the property view, and can be displayed as an alternative to the Input/Output tab. Its goal is to provide students with a sample code corresponding to the selected block in the graphical editor. Such an extract is strictly dependent from the current component configuration: in fact, it is gradually built as the corresponding block is being configured.

The block-related code samples are a very useful tool both for novices and advanced students. In fact, they help students in orientating themselves in the widespread set of Java classes and methods for cryptographic operations. Each

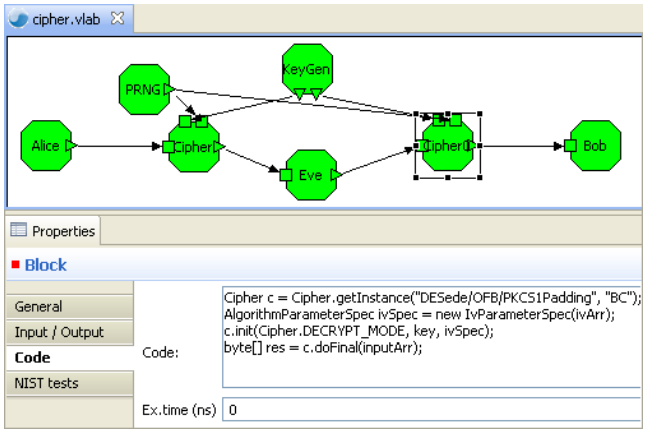

Figure 3: S-vLab Code tab

code sample presents one possible Java class suitable for performing the desired task, symbolized by the corresponding graphical block. It also shows how to instantiate the desired objects, helping students in learning the factory pattern typical of the Java Cryptography Architecture, and how to initialize them. Furthermore, by observing the code extract, students learn which methods can be invoked to perform several operations, and how to specify the method arguments. They can also see the data types the methods deal with, and so understand where additional data elaborations or conversions are required in order to meet the proposed interface. Thus, the combination of the two Input/Output and Code tabs provides a complete view of the main classes, along with their methods and feasible parameters, that are part of the JCE.

Once that a model has been built, or that several blocks have been interconnected, it is possible to switch from a component-based code view to a more complex didactic code extract. In fact, S-vLab can provide an automatically generated version of sample code corresponding to a whole diagram or sub-diagram drawn in the graphical editor. Such a complete code is shown in the Code tab when the considered part of the schema is selected through a selection tool. It is also written on a Java source file, and stored into a dedicated folder within the project. The file extension, which is usually associated with the default Java editor, allows students who use the S-vLab plug-in version to open it within the Java editor of the Eclipse IDE just double clicking on the file name given in the Navigator view. There, students can freely modify, complete and test the starting sample code, either leveraging Eclipse tips for resolving problems or by doing it on their own. In fact, since the provided samples have only an indicative function, these yield a partial code that, in order to be compiled and executed without errors, should be adapted and completed. One of the main issues to be considered is, for instance, the exception handling. In this context, the *mentor* becomes a strategic tool, because, during the simulation of the designed system the involved operations are actually executed, not just simulated. This makes the *mentor* able to trace and report about the exceptions that possibly occur at this stage: in particular, it notifies the user of the exception name and the associated message. Students can then infer the cause of the exception by performing several experiments with different configurations in terms of parameters and input data. This also helps them in figuring out where the exception handling block should be inserted within the code sample, as well as what type of errors do they need to intercept and how should such situations be treated in order to avoid errors in the final program.

Then, after modifying or writing a Java program, students can test it through the Eclipse Java IDE and its facilities, and refine it according to several software engineering considerations, in order to re-organize the code, optimize it, and build distributed systems.

2.4 Analyzing performance

When dealing with security issues, algorithm performance is crucial. For this reason, S-vLab provides students with support tools for analysing the performance of the considered system within an exercitation. Execution time is a common index often used to estimate it. However, since algorithms are platform-independent and can be implemented in any programming language, there are significant drawbacks to using an empirical approach to compare the performance of a given set of algorithms, as well as of different implementations of the same algorithm. Especially within the Java platform, several factors concur in making execution time measurement a difficult task: some of them are the garbage collector, the just-in-time compiler and the HotSpot optimizer. In the context of security-related applications, the JCA and JCE implementation based on the factory pattern is a further aspect.

S-vLab faces the depicted issues by providing students with various tools for analysing performance, described in detail in the following section. The considered instruments differ in various aspects, and sometimes provide complementary information. Such tools consists of well-known, external software: in fact, since our aim is to teach students how to cope with the performance analysis, relying on existing, standard and reusable instruments provides a more formative experience rather than leaning on ad-hoc features implemented within the virtual laboratory.

2.4.1 Tool comparison

In order to conduct an accurate performance analysis, we considered three possible solutions for integration within S-vLab. All the examined tools aim to measure execution time, as well as other indicators, and to provide users a certain control over the possible interferences. The first, *JConsole*, is a graphical tool for monitoring a Java virtual machine, and is shipped with the JDK since J2SE 5.0. The second option is represented by direct calls of some of the APIs exposed by the *Java Virtual Machine Tool Interface* (JVMTI), allowing users both to inspect the state and to control the execution of applications running in the Java virtual machine. Finally, we tried the *Test & Performance Tools Platform* (TPTP), an Eclipse project for building test and performance tools on top of an open platform which offers basic services for monitoring, profiling and debugging. Table 1 summarizes the comparison between the cited solutions on the basis of the main features provided and usability.

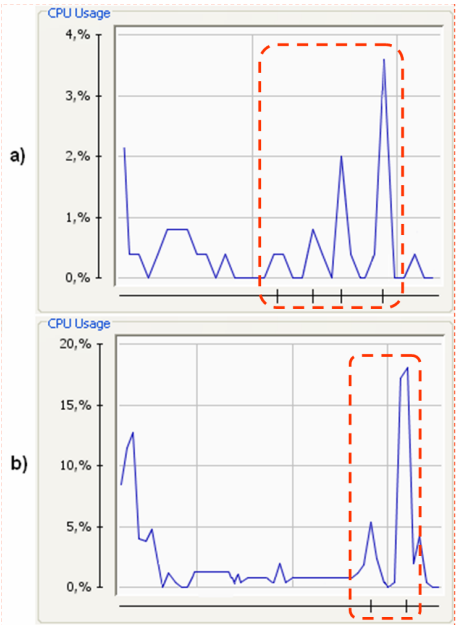

Figure 4: The CPU usage while benchmarking the SHA-1 algorithm (a) and RSA decryption (b)

JConsole is the simplest tool to use, as it offers just a basic set of features, integrates the commands for configuring the Java virtual machine optimizations and renders through diagrams the gathered data. Furthermore, being thread-oriented, it provides a clear and quite precise gauge of several parameters.

As benchmarks to test and compare the provided tools, we considered the execution of the SHA-1 algorithm for hashing inputs of different length and the decryption of a ciphered text with RSA, tracking the performance obtained with and without the optimization based on the Chinese Remainder Theorem. In the former example, our aim was to leverage the provided analysing tools for demonstrating the correlation between the input size, with respect to the SHA-1 operation block size, and the algorithm execution time. However, monitoring the same test in terms of CPU time was a difficult task in all the three proposed solutions. In fact, the resolution at which the time is updated within the Java virtual machine is not specified, and the methods providing timing information declare nanosecond precision, but do not grant a nanosecond accuracy. Figure 4a shows the CPU usage percentage tracked by *JConsole* while benchmarking the SHA-1 algorithm in four test sessions, each using a double-sized input message with respect to the previous. After an initial noise, the diagram indicates that the CPU usage doubles in each test. The second benchmark compares the RSA decryption based on the Chinese Remainder Theorem and the standard operation: Figure 4b remarks that the considered optimization reduces by four times the CPU usage. Yet, in both cases the required CPU time does not present the same variation rate in neither the data tacked by *JConsole*, by *JVMTI* or by *TPTP*. That confirms the importance of training students to use several tools and methods instead of relying on a unique option.

TPTP is the most complex and complete tool among the three we considered: it includes advanced tools for studying memory occupation and load balancing, is primarily aimed at high-level analysis mostly oriented towards software modules rather than threads and methods, and provides structured reports by leveraging BIRT (Business Intelligence and Reporting Tools), the most used reporting system for Eclipse [25]. So, *TPTP* seems more indicated as tool for monitoring and profiling software prototypes instead of algorithms: this makes it suitable for supporting students in the learning step aimed at refining the software architecture and deployment of their programs.

2.4.2 *Evaluating behaviour*

Studying performance also involves the analysis of the envisioned system robustness and behaviour. To this end, since the virtual lab allows students to simulate the modelled system by executing each component, it is possible to integrate within S-vLab several tools for analysing data gathered through various experimentations. This allows students to study, filter, aggregate and elaborate data in order to infer some information about the system robustness against various attacks. Such an analysis is strictly connected to the specific system, to the conditions of test and to the type of simulated attack: so, it is difficult to provide tools suitable for every kind of possible test and investigation. To overcome this issue, we integrated in S-vLab a data gathering system based on a database, and then let students free to elaborate and study the data stored during the work sessions using BIRT. In fact, BIRT offers basic functions for creating reports to visualize data in tables and diagrams. Furthermore, it includes a set of features for simple data elaborations. Being based on the plug-in mechanism, the set of provided features can easily be extended: by now, S-vLab includes a simple BIRT extension for finding collisions, but students can yield further extensions addressing different issues and then share their contributes in order to extend the virtual laboratory.

For instance, the collision detection tool can be used to prove the statistical properties of SHA-1 in random number generation. In fact, students can iteratively hash different input messages and sequence the obtained digests in order to build a longer bit string. Then, they can analyse the gathered data for finding collisions among bit blocks. Figure 5 shows one of the diagrams yielded by the collision detection tool: it traces the cumulative distribution for the number of four-bit chunks, called *nibbles*, required to find a colli-

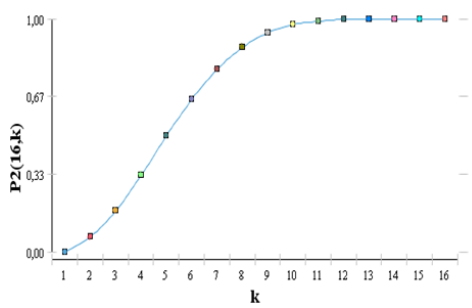

Figure 5: Collision detection

Table 1: Tools used for analysing performance

Tool	Main performance indexes	Run-time control	Output data presentation	Usability
JConsole	CPU usage; CPU time per thread; memory management	Perform garbage collection on demand; turn off or configure HotSpot optimization	Real-time data summarized in tables and diagrams	Very simple, intuitive, user-friendly
JVMTI	Thread management; memory management	Perform garbage collection on demand	Raw data returned by methods	Easy to use for users with some programming expertise
TPTP	Thread management; memory management	Perform garbage collection on demand	Real-time data summarized in tables and diagrams; gathered data can be exported to BIRT for further elaborations	Complex to use; previous knowledge of Eclipse projects helps in integrating it with other plug-ins

sion in a sequence of 20000 bits generated through SHA-1. In this proof, students verify that the probability to find a collision peaks after considering 11 nibbles. Furthermore, 5 nibbles suffices to have a probability greater than 0.5. Since these result perfectly match the expected values, calculated through the theoretical formulas presented during the face-to-face lectures, such an experiment helps students to verify that the SHA-1 algorithm can be used as a random number generator.

3. STUDENTS' RESPONSE

S-vLab, in its prototypal version, has been tested within the 2008 edition of the Information Security course held at the School of Computer Engineering of the University of Bologna. About 150 students attending the fourth year of a five-year curriculum took the course.

The course was structured according to a blended-learning approach: during face-to-face lessons, students were introduced to algorithms and protocols for assuring information integrity, confidentiality and authenticity, and learnt the basic principles for designing secure systems. Then, three exercitations have been scheduled to allow students to directly verify and perform experiments on the studied concepts. Each exercitation was introduced through a face-to-face work session in laboratory, where students could use S-vLab to discuss and perform the proposed tasks in real time. In order to allow students who did not attend the lectures to perform the exercises on their own, we published on the course web site the given assignments as well as a user guide and tips on how the virtual lab should be used. At the end of the exercitations, learners were requested to provide a written report on their work. Although both the attendance in laboratory and the autonomous accomplishment of the given assignments were voluntary, we noticed a significant participation.

At the end of the course, students who provided evidence of using S-vLab were asked to fill a survey to express their opinions about the virtual lab and its utilization. The survey was anonymous, and was given to students immediately after they had completed the final exam, in order to gather not conditioned responses. At the present time, when only one examination session has been carried out, we collected 52 filled surveys, which is about the 87% of students who took the exam in the first session, and about one third of the estimated total course attendees.

The survey includes both closed and open-ended questions: the first allow students to judge the relevance of the main features offered by the virtual lab through a five value scale, and to evaluate the envisioned strategy for using S-vLab within the next course edition; the second ask students to detail their opinions, provide feedback and suggestions.

Such results indicate that the virtual lab can effectively support learning. One of the most appreciated features is the chance to verify the studied concepts through simulation. The *mentor*, that provides feedback and notifies possible errors, is the second preferred functionality: 90% of students signaled the possibility to perform experiments at different detail level as one of S-vLab's main strengths. All rated as positive the blended-learning experience: in particular, 14% of students said the exercitations with the virtual lab should be kept optional, while 37% expressed the idea that should be mandatory as part of the course standard lectures; the remaining 49% would make the exercises compulsory, but recognized as extra activities.

4. RELATED WORKS

The design and adoption within a course of an educational tool for supporting learning and practicing on secure systems in Java is related to several research issues.

The first concerns the most effective way to devise and manage such an educational tool. There, a longstanding debate compares hands-on, remote and virtual laboratories [11]. Hands-on and remote laboratories involve students, provide

them with real data and realistic situations to face. Experimentations on such systems reported very positive results both in terms of students' satisfaction and perceived learning [18, 12, 19, 29]. One further benefit is that students acquire proficiency in dealing with open source tools actually used in real contexts [8]. However, hands-on and remote approaches require many resources in terms of equipment, ongoing monitoring and handling possible contingencies. Thus, many authors preferred to design modelling frameworks or simulators suitable for education. S-vLab, according to this approach, is implemented as a virtual laboratory. Yet, in order to help students to acquire skills useful for their professional future, it supports practice on open source, well known and widely adopted tools.

In the context of Information Security, several educational tools have been developed to foster students' ability to model and simulate secure systems and protocols. A few examples of such tools consist of statically-defined simulations: this is the case of some Java applets that act as an autonomous animation of a protocol or mechanism or service for security, but does not allow users to configure the depicted scenario nor to interact with the animation [5]. Other systems, that can be traced to the active learning theory, enable the learners to actively interact with the system in order to modify the desired experiment, although the possible investigations are limited to predefined, guided paths [32, 9]. S-vLab differs from these approaches as it is strongly based on a constructivist teaching method. In fact, it provides an interactive environment where learners can freely settle and modify their experiments, by starting from scratch as well as by remodelling given exercitation schemes.

The task of facilitating the switch from the modelling to the implementation step has been explored in the past. This issue has typically been faced through the adoption of Computer Assisted Systems Engineering tools [24]. In our case study, several works focus on guidelines [3] and CASEs for modelling and testing secure systems and protocols [20, 4]. Such contributes suggest the adoption of UMLsec, an UML extension for security, for the modelling step, and the CASEs rely on automatic theorem provers for assessing the security of the designed system. These strategies, although efficient and effective in a professional context, seem not to be the best choices in an educational scope, since the evaluation of the system robustness is statically determined, and students can not leverage the simulation benefits such as the detailed analysis of data provided in each step. Furthermore, UML could disorient novices with a low programming expertise. This is the main reason for S-vLab to rely on a very intuitive modelling language, easy to use tool despite of the students' knowledge of UML.

Finally, several systems have been developed to support students' practicing with the Java language. Some of them help searching the increasingly large Java API libraries [30], provide code samples illustrating the use of Java APIs [16, 27], and support learners by allowing knowledge sharing among peers [31]; others can automatically collect and assess Java program fragments [7, 17] and offer a pedagogical support tool to be added to commonly used software development frameworks [13]. All of these can effectively facilitate learning and gaining programming expertise both for novices and advanced students. However, such tools are focussed on the implementation of Java programs and neglect system modelling and evaluation. S-vLab aims to integrate these directions into a unique comprehensive tool, so as support students through the whole learning process.

5. CONCLUSION AND FUTURE WORK

Nowadays defending information security is a strategic issue: thus, training computer engineers to build and deal with secure systems is crucial. To this end, modern education can rely on tools such as virtual labs to help students to gain greater insight into subjects, as well as to acquire practical and long-lasting proficiency. According to this approach, we designed and implemented S-vLab, a virtual laboratory for fostering teaching and learning of security-related aspects, with a special attention towards the Java Security Platform. Since S-vLab is built on top of a general framework, it can be easily suited to different teaching contexts.

S-vLab is suitable both for novices and advanced students, and can effectively support several learning styles. It provides learners with numerous features for modelling and simulating a system, for evaluating its performance and implementing a software prototype, by supporting students through the whole learning process. In order to fill the gap between modelling and implementation, S-vLab provides students with Java code samples related to the studied system, yields continuous feedback notifying possible Java exceptions, allows navigation through technical documentation and reference standards, and permits practice on a Java Integrated Development Environment.

The experimentation of S-vLab within a course on Information Security was highly appreciated by the students. In fact, the great majority of attendees performed the optional exercitations within the virtual lab. Furthermore, the results of an anonymous survey issued at the end of the course indicate that students believe the virtual lab can effectively support learning. Students also suggested some improvements for the future versions: the most desired features include additional tools for analysing simulation results, as well as minor software refinements for improving the virtual lab usability.

Finally, an interesting research issue concerns the users' activities evaluation, which is strictly related to the presentation of customized feedback and the suggestion of new tasks based on users' profiles, formative objectives and activity history. Thus, our future efforts will address both improving S-vLab adding new advanced features, and dealing with such an open and challenging issue.

6. REFERENCES

[1] Website of the course on Information Security, University of Bologna: http://lia.deis.unibo.it/Courses/TecnologieSicurezzaAK/S-vLab.html.

[2] O. Akdemir and T. A. Koszalka. Investigating the relationships among instructional strategies and learning styles in online environments. *Comput. Educ.*, 50(4):1451–1461, 2008.

[3] A. Apvrille and M. Pourzandi. Secure software development by example. *IEEE Security and Privacy*,

3(4):10–17, 2005.

[4] B. Best, J. Jurjens, and B. Nuseibeh. Model-based security engineering of distributed information systems using UMLsec. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 581–590, Washington, DC, USA, 2007. IEEE Computer Society.

[5] D. Bishop. *Introduction to Cryptography with Java Applets*. Jones and Bartlett Publishers, 2003.

[6] J. B. Black and R. O. McClintock. An interpretation construction approach to constructivist design. *Constructivist Learning Environment: Case Studies in Instructional Design*, 1996.

[7] P. Brusilovsky and C. Higgins. Preface to the special issue on automated assessment of programming assignments. *Journal of Educational Resources in Computing*, 5(3), 2005.

[8] J. C. Brustoloni. Laboratory experiments for network security instruction. *J. Educat. Res. Comput.*, 6(4):1–18, 2006.

[9] G. Cattaneo, A. D. Santis, and U. F. Petrillo. Visualization of cryptographic protocols with GRACE. *J. Vis. Lang. Comput.*, 19(2):258–290, 2008.

[10] C. S. Claxton and P. H. Murrell. Learning styles: implications for improving educational practices. *ASHE-ERIC Higher Education Report No. 4*, 1987.

[11] J. E. Corter, J. V. Nickerson, S. K. Esche, C. Chassapis, S. Im, and J. Ma. Constructing reality: A study of remote, hands-on, and simulated laboratories. *ACM Trans. Comput.-Hum. Interact.*, 14(2):7, 2007.

[12] E. Crowley. Experiential learning and security lab design. In *CITC5 '04: Proceedings of the 5th conference on Information technology education*, pages 169–176, New York, NY, USA, 2004. ACM.

[13] L. N. de Barros, A. P. dos Santos Mota, K. V. Delgado, and P. M. Matsumoto. A tool for programming learning with pedagogical patterns. In *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 125–129, New York, NY, USA, 2005. ACM.

[14] P. J. Denning. Great principles of computing. *Commun. ACM*, 46(11):15–20, 2003.

[15] W. Du and R. Wang. SEED: A suite of instructional laboratories for computer SEcurity EDucation. *J. Educ. Resour. Comput.*, 8(1):1–24, 2008.

[16] D. M. et al. Jungloid mining: Helping to navigate the API jungle. In *Proceedings of 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 48–61, 2005.

[17] O. Gotel, C. Scharff, and A. Wildenberg. Extending and contributing to an open source web-based system for the assessment of programming problems. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 3–12, New York, NY, USA, 2007. ACM.

[18] J. M. D. Hill, J. Curtis A. Carver, J. W. Humphries, and U. W. Pooch. Using an isolated network laboratory to teach advanced networks and security. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 36–40, New York, NY, USA, 2001. ACM.

[19] J. Hu, C. Meinel, and M. Schmitt. Tele-lab IT security: an architecture for interactive lessons for security education. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 412–416, New York, NY, USA, 2004. ACM.

[20] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 322–331, New York, NY, USA, 2005. ACM.

[21] P. A. Kirschner, J. Sweller, and R. E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

[22] D. Kolb. *Experiential learning: experience as the source of learning and development*. Prentice-Hall Englewood Cliffs, New Jersey, 1984.

[23] J. McAffer and J.-M. Lemieux. *Eclipse Rich Client Platform: designing, coding, and packaging Java applications*. Addison Wesley Professional, October 2005.

[24] S. H. Moritz and G. D. Blank. A design-first curriculum for teaching java in a cs1 course. *SIGCSE Bull.*, 37(2):89–93, 2005.

[25] D. Peh, A. Hannemann, and N. Hague. *BIRT: a field guide to reporting*. Addison-Wesley, October 2006.

[26] E. Roberts. Resources to support the use of Java in introductory computer science. *SIGCSE Bull.*, 36(1):233–234, 2004.

[27] N. Sahavechaphan and K. Claypool. XSnippet: Mining for sample code. In *Proceedings of the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 413–430, Portland, OR, 2006.

[28] S. Schworm and A. Renkl. Computer-supported example-based learning: when instructional explanations reduce self-explanations. *Comput. Educ.*, 46(4):426–445, 2006.

[29] T. A. Yang, K.-B. Yue, M. Liaw, G. Collins, J. T. Venkatraman, S. Achar, K. Sadasivam, and P. Chen. Design of a distributed computer security lab. *J. Comput. Small Coll.*, 20(1):332–346, 2004.

[30] Y. Ye and G. Fischer. Supporting reuse by delivering task-relevant and personalized information. In *Proc. of 2002 Int. Conf. on Software Engineering (ICSE'02)*, pages 513–523, Orlando, FL, 2002.

[31] Y. Ye, Y. Yamamoto, K. Nakakoji, Y. Nishinaka, and M. Asada. Searching the library and asking the peers: learning to use Java APIs on demand. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 41–50, New York, NY, USA, 2007. ACM.

[32] J. Zaitseva. TECP - tutorial environment for cryptographic protocols. Master's thesis, University of Tartu, 2003.