Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 891–896
ISBN 978-83-60810-51-4

Robot simulator facilitating the education of concurrent programming

Łukasz Szweda Jakub Flotyński Daniel Wilusz Paweł Dąbrowski

Department of Information Technology, Poznań University of Economics
al. Niepodległości 10, 61-875 Poznań, Poland
Email: {szweda, flotynski, wilusz, dabrowski}@kti.ue.poznan.pl

Abstract — The paper presents an experiment of teaching Java-based concurrency using a robot simulator. The computer programming education is a challenging task, especially when non-computer science students are taught complex programming concepts. Recently, great deals of simplified programming languages, environments and simulating software have been proposed to support teaching and self-learning different programming techniques. But still, there is no solution facilitating effective teaching in the domain of concurrent programming in the Java language. In this paper we present our original concept of exercises using a robot simulator to teach Java-based concurrency. The simulator seems to be a good solution facilitating the education of concurrent programming, as actions performed in real-time by the simulator allow students to quickly identify their mistakes.

I. Introduction

The courses introducing computer programming aspects have been gaining difficulty in terms of both the scope of required knowledge and complexity of programming environments. This phenomenon is the consequence of the progress made in computer science. Students require new approach to teaching, different from the one practiced by previous generation [1]. In the era of the Internet, PlayStation and tablet PCs, the purely theoretical exercises, such as finding the mean of a set of numbers, are no longer perceived by students as an exciting challenge. Ubiquitous presence of computer devices causes the emergence of new expectations; for instance such issues as GUIs or even dispersed systems are often introduced during the first years of studies. These new concepts introduced in the courses require more time and effort from students, making the course more difficult to pass. According to the Computer Curricula 2001 Report, the increase of the number and difficulty level of subjects in computer science, which are taught, is proportional to the level of the problems' complexity, which students have to deal with [2].

It looks like the major problems, the computer science education has to face in order to preserve high quality are related with software complexity and the lack of stability. The complexity is steadily increasing, as the number of different aspects of software development has risen over past years. In the meanwhile, the programing languages, libraries, frameworks and programming environments are evolving rapidly [3]. These two problems lead to a contradiction. On the one hand, increasing the difficulty of computer science subjects lengthens development of effective educational strategies. On the other hand, the increasing velocity of computer science expansion causes new teaching materials becoming soon out of date.

Development of a method reducing the complexity of computer science issues and easing the impact of instant changes in this field would significantly benefit the education concerning computer science and, in particular, software development. In order to improve educational process, the factors impacting the computer science have to be identified. The most significant factor, causing the complexity and instability is related the lifetime of computer science field. Since the popularity of the Pascal language twenty five years ago, the number of computer scientists has increased significantly. At present, a lot of freelancers and enterprises are involved in development of computer-based products and services. When a new prospective technology appears, such as Java, people and businesses invest their time and resources in fostering the development of this technology. From their point of view, the rapid changes to Java and its API can be perceived as indicators of the success. The fact that Java has been developed faster than other programming languages comes from the wide reception from the IT industry, which has accelerated its continuous evolution.

The suitability of choosing Java for software development seems to be out of question. The competitors of Java are either obsolete or, as in the case of C#, they represent similar approaches and concepts. Moreover, as Java was intended to be used by embedded systems, this language may be used to develop wide range of software applications, from web services, by desktop programs, to the microwave oven control systems. These arguments show that there is no reason to replace Java by any other solution. However, the issues concerning the complexity and instability still need to be solved.

The concept of concurrency is an example of one of the difficult issues which students need to face at early stage of computer programming education. Topics concerning threads management and synchronization, resource sharing, locks and deadlocks cause many misunderstandings and unidentified bugs in the source code. These problems need

an effective method of teaching concurrency related aspects, which would visualize the anomalies of incorrectly implemented threads.

In this paper we present the robot simulator facilitating the education in field of Java-based concurrency. The remaining of the paper is organized as follows. Section II describes the state of the art in computer programming educational platforms, e.g., MiniJava, Karel, Scratch and Alice projects. Section III presents and evaluates a simulator-based exercise, which aims at facilitating teaching the non-computer science students concepts of concurrency. Finally, Section IV concludes the paper.

II. PLATFORMS SUPPORTING EDUCATION OF COMPUTER PROGRAMMING

During the introduction of advanced aspects of programming, requiring proficiency in the application of Java libraries is not a good idea. Instead, students should be introduced to computer programming by means of a more limited environment, which is closely related to Java but simultaneously lacks the complexity [9]. As Java has been developed for professional use, not for teaching programming concepts, there are plenty of technicalities which make understanding Java more difficult for non-computer science students with no programming background. These are obstacles for successful teaching advanced aspects of Java programming such as the concurrency. In conclusion, there is a strong need for a simplified environment having all the Java syntax and semantics, but simultaneously allowing to present complex issues in a simplified way.

A. MiniJava

Beginning programmers often have trouble translating their intentions into syntactically correct statements that the computer can understand [11]. Java has a number of weak points which limit its successful education on the introductory level [16, 17]. On the whole, the problem escalates as a result of forcing the students to get insight into conceptually elaborate aspects of its structure while developing even the simplest programs. Accordingly, beginners in Java have to become acquainted with a great deal of complicated details before they are able to understand crucial ideas. The most gifted students will handle this issue, however, the others will become frustrated and will leave the world of programming unaware of its power and usefulness. The main reasons why Java is so elaborate in the process of education include:

- size of the language and its libraries,
- evolution caused by the long lasting variation in Java platforms,
- standard input operations,
- handling exceptions,
- the graphical model.

Although MiniJava has been designed as a simpler form of Java, it is not its exact subset. Relation between Java and MiniJava has been presented in Fig. 1. The limited subset of relevant elements of the entire language, MiniJava core, is created starting from the full set of Java represented by the greatest ring in the top-left corner.

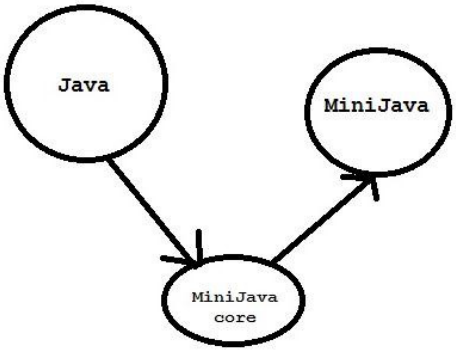

Fig. 1 The relation between Java and MiniJava

In the following step one can note a simplification to MiniJava of the elements of the new subset. That makes it easier for beginners to utilize the elements. Students start learning Java with carefully selected tools appearing in the more approachable form. As the students expand their programming skills they learn how the simplification was implemented and how to handle without it. Each restriction made while building MiniJava can be rationally explained. One of the restrictions is a limitation of the number of libraries. It is obvious for developers that understanding of Java packages is indispensable in the process of developing programs. At the same time they assume that it is sufficient for the students solving simple programming problems to use a significantly narrowed collection of packages. Finally, today, even for experts, programming to a very great extent consists of finding relevant structures within an extensive collection of available tools. Work with extensive libraries and the ability to ignore the fragments which are unnecessary for the application is a significant programming skill which is to be mastered while learning programming. Implementation of this approach can be very problematic for students learning sole programming. That is because in fact they do not know what they need and the list of available tools is overwhelming. Thanks to MiniJava it is possible to get students acquainted with more refined concepts such as multithreading. Although, it is possible to do otherwise, Minijava has been designed mainly to teach basic methodology elements: decomposition, encapsulation, etc.. The lack of visual perception of many threads in MiniJava causes that the process of learning of this concept is not much different than learning based on standard libraries.

B. Karel – the robot

Rich Pattis, graduate student of Stanford University, came to the conclusion that it will be easier to teach basics of programming if students have the possibility to learn these

concepts in a new and simple environment, instead of using an enterprise-driven approach. Inspired by success of LOGO project, Rich Pattis created a pilot programming environment, called Karel, in which students program a virtual robot to solve simple problems [12].

Karel is a virtual robot embedded in a very simple world. There are avenues running north and south numbered one to infinity and streets running east and west, also numbered one to infinity [13]. By giving a series of commands we can make it perform certain tasks. The process of defining those commands is called programming. Karel knows only few predefined commands, however, a crucial part of programming is teaching Karel new commands which expand its capabilities.

Predefined commands and syntactic rules define a Karel programming language. This language bears some resemblance to Java. Owing to this fact it is easy to switch between those two dialects. Karel and Java programs have a very similar structure and include the same basic elements. The most important difference between the two languages is that the scope of Karel programming language is narrow, i.e. it does not have many commands and rules. It is easy, for example, to learn the entire language within several hours. Then the student knows Karel's capabilities and is able to define its activities in the program. Details when compared to Java are easier to master. Despite this fact, one can easily find out that a solution to a problem may turn out to be a big challenge. Solving problems constitutes the essence of programming and the rules are only a minor obstacle.

Certainly, Karel is a step in the right direction. The environment emphasizes the visual aspect of programming with robots and capitalizes on the “robots are fun” mantra [14]. However, this technology has been simplified excessively. Although we can discuss basic aspects of programming methodology, such as: object-orientation, decomposition, encapsulation and hiding information, there is little time left for more advanced issues including concurrency. As a result, Karel is a good choice to facilitate education of Java basics; however, when it comes to more sophisticated Java issues, new stimulating environment is required.

C. Scratch – MIT Solution

Scratch is a project inspired by the fact how children play with Lego blocks, putting them together into certain structures. The authors - MIT research team, were convinced that such approach to education will have incredibly positive impact on students programming skills. In fact the scratch grammar is a set of graphic blocks used for coding. As in case of Lego blocks, the graphic blocks are characterized by proper projections and joints implying how the blocks fit to each other [10]. Programming is done by dragging and dropping these blocks to form scripts that control the animation of two dimensional sprites on a stage. [15] In comparison to MiniJava or Karel, there is absolutely no irritating syntax and implementation details which make the concept difficult to understand. Undoubtedly Scratch has an advantage over MiniJava such that it provides a lot of enjoyment and enables the programmer to mix graphics, animations, photos and sounds. Unlike in Karel’s world, Scratch users not only learn basic concepts but can also face advanced ones like multiple threads. The recipe is easy, several behavior elements can be used at once what enables the multiple threads to work simultaneously on a Scratch's graphical layer. Even though Scratch is a powerful tool, it’s far from being perfect for learning advanced aspects of programming. It has been already said that, unlike MiniJava, Scratch has no irritating syntax. The problem is that there is practically no programming syntax, whereas MiniJava uses it, and at the same time attempts to simplify it. As a result little correlation is provided with Java programming; advanced Scratch users do not necessary become experts in the Java language. All aspects of programming are purely conceptual. Simplification of the tool also eliminates crucial elements of multiple threads which we are unable to present, e.g., race condition.

D. Alice

Alice [4] is a tool designed for effective teaching of basic concepts in computer programming [5] by developing animations, storytelling [6], computer games [7], as well as more sophisticated applications like controlling behavior of real robots [8]. The primary entity of each animation and game is a scene. 3D scenes are composed of independent graphical components that may be dragged and dropped to a main application panel. During the modeling, the author may freely move in the virtual world. Components that may be things, animals and characters (people, fantasy, etc.) are individual objects with their own properties (e.g., color, size), variables, methods and interfaces (e.g., jump, rotate, say something). Obviously, it is possible for the user to define new methods implementing new behavior of objects. Alike in object-oriented programming, similar objects (with the same design) are of the same class. They may interact each other through interfaces as well as with the user by events in the graphical user interface, e.g., mouse click or key down. Methods of components are comprised of standard imperative programming instructions like loop, switch and conditional as well as operations on constants and variables.

However Alice is an excellent tool for beginners’ programming course that visualize basic elements of Java, it does not support teaching more advanced issues, e.g., abstract classes, polymorphism and inheritance. Although objects in the scene perform actions simultaneously, the actions cannot be presented by the application as individual threads, so Alice is not a convenient tool for teaching Java-based concurrency.

III. THE EDUCATIONAL STUDY

In order to facilitate teaching complex Java programming concepts, we have worked out a teaching study. The aim of the study was to teach non-computer science students the concept of concurrency in Java. The idea of the exercise was to enable the students to watch the execution of many parallel, independent threads on the example of behavior of simulation environment. Although the students had a basic Java knowledge, their knowledge and skills concerning concurrency were measured by the entrance and exit tests.

A. The exercise

The exercise has been performed by students working in pairs with Eclipse IDE. The aim of the exercise was to implement a robot behavior by use of concurrency. The application simulates a self-defending robot that switches its operation modes (guard, attack and retreat) depending on data received from users. The simulated robot works in a few modes. When operating in the guard mode, the robot simulator indicates its rotation, plays the patrol sound, displays the current state on the console, and checks the distance to the nearest object. After the distance bar reaches a critical position, robot switches into the attack mode – it assumes the defensive position, switches on the lights, plays attack sound, and starts shooting. After a few seconds, the robot switches into the retreat mode – it rotates and runs straight for a few seconds. Finally, it returns to the guard mode and the whole process can be repeated. At any time the robot may be switched off by pressing a button.

In more details, the students have been supposed to implement the thread to examine the close button all the time the robot is switched on. The exercise requires also using individual threads in the guard mode when the rotation, the distance check, and the patrol sound are performed in parallel. Moreover, the threads responsible for the defensive position, switching on the lights, playing the attack sound, and shooting have to be properly implemented, executed and managed during attack.

The application form incorporates four parts: the group of information labels, the scrollbar specifying the distance to the nearest object, the console, and the close button. Information labels represent all activities that may be performed by the robot. When the robot is performing an activity, the related label is red, otherwise it is black. As mentioned before, some activities may be performed in parallel, than all relevant labels are red. The scrollbar is used by the user to change the mode of the robot – when the value is small, the robot changes its state from guard to attack. The current mode is visible on the console all the time.

To allow students to focus only on concurrency aspects the GUI (Fig. 2) has been provided to them as a separated `Robot` class implementing the presented form with components. The interface of the class contained following methods.

- `beepUp()` – making the noise,
- `display(String text)` – displaying a given text on the console,
- `forward()` – making the robot to run straight,
- `getProximity()` – returning the distance to the nearest object,
- `isTouched()` – informing whether the close button has been pressed,
- `playTone()` – playing the tone,
- `rotateClockwise()` – turning the robot,
- `shoot(Integer shots)` – shooting given number of times,
- `stop()` – stopping the robot,
- `switchLights()` – switching on the lights,
- `turnOffLight()` – switching off the lights.

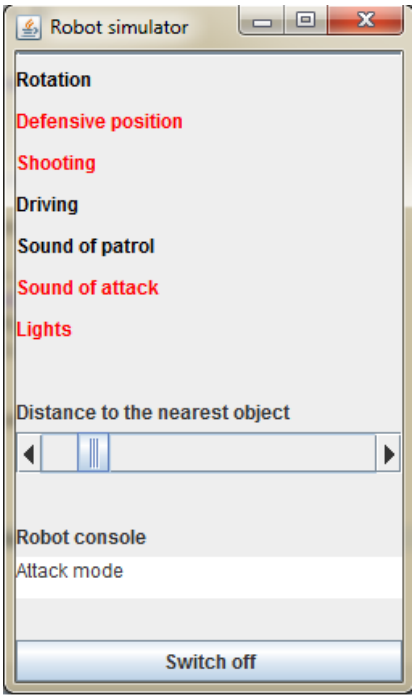

Fig. 2. GUI of the robot simulator

The task for students was to implement a `Defender` class that calls the above methods from within different threads when required.

The exercise has presented many concurrency aspects to the students. The race condition has been experienced while observing different messages on the console or when the robot stopped unexpectedly due to the incorrect implementation of the threads. In addition, the students could observe improper robot's behavior if some threads (e.g., playing the tone, rotating the robot) were forgotten to be interrupted. Moreover, the effects of not-handled exception thrown as a result of an interruption were visible as well and thus had to be properly handled.

The exercise requires from students to implement software carefully by revealing their mistakes during tests performed on the robot. The presented form of the exercise stimulated

the students to rethink their code in order to overcome the multithreading traps. Note that, due to the complexity and mutual dependence of the tasks performed by the robot, it was not possible to program the robot behavior without using threads and concurrency. Thus, to complete the exercise, students were forced to learn more deeply about Java threads and mutual communication among them. Note also, that it was not yet-another annoying theoretical exercise of producer-consumer problem, typical for a course on concurrent programming.

B. The educational results of the study

Forty five non-computer science students attending the Computer Programming classes at the Bachelor course concerning Econometrics and Business Informatics participated in the experiment. The students were required to fulfill quizzes at the beginning and at the end of the exercise. The quizzes consisted of theoretical as well as practical (code analysis) questions. The quizzes were prepared to verify students' knowledge in the area of following aspects:

- Java classes and methods for concurrency,
- sequence of instructions performed by concurrent threads,
- simultaneous access to shared variables by many threads,
- interrupting threads,
- executing threads.

The effects of the exercise measured by results of the quizzes are presented in Table I and described below.

The conducted study revealed that the students made the greatest progress (the increase by 38 percentage points) in the area concerning the familiarity of Java classes and methods. The big number of atomic task during the exercise caused a good acquisition of Java knowledge by students. Another progress concerned understanding of parallel execution of particular instructions by many threads. After performing the exercise the understanding of race condition by students increased from 24% to 44%. This result showed that graphical visualization of threads' behavior positively influences the understanding of race condition. Simultaneous access to shared variables by threads was another issue in which noticable progress was made. The entrance test revealed that about 76 % of students knew that threads may access the same variables and compete for them. However, after performing the exercise in simulating environment, the rate of students understanding the issue of shared variables increased to 91%.

According to the results of the quizzes, the simulation environment noticeably influenced the issue of thread interruption and exception handling (increase by 9 % points).

On the other hand, the issue of running threads indicated no simulation environment impact on the students' knowledge. Executing the threads is quite intuitive and the students had required theoretical knowledge. In result there was no potential of the exercise to increase students' skills and knowledge concerning starting threads.

The application of simulation environment allowed for visualizing the behavior of many threads and facilitating education process. The research revealed that utilization of simulating environment significantly impacted the students' progress in understanding such complex issues as race condition or handling the exceptions caused by interrupted threads.

IV. Conclusions and Future Works

Despite of the popularity of simplified programing languages, simulation environments and robots in facilitating computer programming education, the improvement of education in the domain of concurrent programming is still a challenging task. The lack of the possibility to depict the Java concurrency concept in existing educational environments causes that they are useless for education in this particular field.

The study described in this paper has utilized the robot simulator for teaching. The exercise revealed the positive impact on students' education. Multithreading and concurrency being quite complicated and difficult for analyzing can be better understandable when presented graphically. Students, observing the behavior of many threads are able to notice the anomalies in robots behavior in

TABLE I.
RATES OF RESULTS OF ENTRANCE AND EXIT TESTS, GROUPED BY QUESTION TYPE

Question type	The entrance test: percent of correct answers	The exit test: percent of correct answers	Progress
Java classes and methods for concurrency	40%	78%	38%
Sequence of instructions performed by concurrent threads	24%	44%	20%
Simultaneous access to shared variables	76%	91%	15%
Interrupting threads	64%	73%	9%
Executing threads	80%	76%	-4%
Average	**57.8%**	**72.4%**	**15.6%**

real-time. Moreover, students are stimulated to identify the bugs concerning threads' management and in result became more careful when implementing concurrency.

Concurrency is not the only challenging issue that may be clearly taught by the use of a robot simulator. Other concepts, like Service Oriented Architecture (SOA) and Event Driven Architecture (EDA) can also be presented using simulators. Moreover, simulators enable getting student involvement in learning process by making the classes more interesting. However, such tools are only a supporting part of classes, useful in effective introduction to computer programming. The facilitated programming concepts should be also presented in the standard manner, once the students gain necessary programming skills.

As for the future works, more exercises with simulators of different kind could be worked out to teach programming in such domains, as SOA services and their orchestration, swarm intelligence or pattern recognition. Obviously, the next step for teaching emerging topics in programming is using more sophisticated simulators (e.g., with 3D virtual objects) as well as real robots and devices. The results of utilizing them in education could be compared with benefits of simulators.

References

[1] Miller D., Nourbakhsh I, Siegwart R.: Springer Handbook of Roboics. Springer-Verlag Berlin Heidleberg (2008).

[2] Computing Curricula 2001 Computer Science, ACM, IEEE, http://www.acm.org/education/curric_vols/cc2001.pdf.

[3] Eric Roberts, The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education, In: ACM SIGCSE Bulletin Homepage Volume 36 Issue 1, March 2004.

[4] Alice, http://www.alice.org, retrieved May 11, 2012.

[5] P. Mullins, D. Whitfield, M. Conlon, Using Alice 2.0 as a first language, In: Journal of Computing Sciences in Colleges, vol. 24 issue 3, January 2009, pp. 136-143, ISSN 1937-4771.

[6] Caitlin Kelleher, Randy Pausch, Sara Kiesler, Storytelling alice motivates middle school girls to learn computer programming, In: Proceedings of the SIGCHI conference on Human factors in computing systems, San Jose, CA, USA – April 30 - May 03, 2007, pp. 1455-1464, ISBN 978-1-59593-593-9.

[7] R. H. Seidman, Alice first: 3D interactive game programming, In: ACM SIGCSE Bulletin, vol. 41 issue 3, September 2009, pp. 345-345, ISSN 0097-8418.

[8] B. Lowe Wellman, J. Davis, M. Anderson, Alice and robotics in introductory CS courses, In: The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations, Portland, USA – April 01 - 04, 2009, pp. 98-102, ISBN 978-1-60558-217-7.

[9] Eric Roberts, An Overview of MiniJava, In: ACM SIGCSE Bulletin Homepage Volume 33 Issue 1, March 2001, ISBN:1-58113-329-4.

[10] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, Yasmin Kafai, Scratch Programming for All, In: Magazine Communications of the ACM - Scratch Programming for All CACM Homepage archive Volume 52 Issue 11, November 2009.

[11] Caitlin Kelleher and Randy Pausch, Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers, In: ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 37 Issue 2, June 2005.

[12] Pattis R., Roberts J.: Karel The Robot: A Gentle Introduction to the Art of Programming, 2nd Edition. Wiley (1995).

[13] Byron Weber Becker, Teaching CS1 with karel the robot in Java, In: ACM SIGCSE Bulletin Homepage Volume 33 Issue 1, March 2001.

[14] Monica M. McGill, Learning to Program with Personal Robots: Influences on Student Motivation, In: ACM Transactions on Computing Education (TOCE) archive Volume 12 Issue 1, March 2012.

[15] Orni Meerbaum-Salant Michal Armoni Mordechai Ben-Ari, Habits of programming in scratch, In: ITiCSE '11 Proceedings of the 16th annual joint conference on Innovation and technology in computer science education ACM New York, NY, USA ©2011.

[16] Robert Biddle and Ewan Tempero, Learning Java:Promises and pitfalls. In: Proceedings of the 14th Uniforum NZ Conference, 1997.

[17] Ann Fleury, Student conceptions of object-oriented programming in Java, In: Journal of Computing in Small Colleges, 1999.