An Environment for Teaching Object-Oriented Programming: ObjectKarel

Maya Satratzemi[*], Stelios Xinogalos[*], Vassilios Dagdilelis[**]
[(*)]*Department of Applied Informatics,* [(**)]*Department of Educational and Social Policy*
University of Macedonia, Thessaloniki, Greece
{maya, stelios, dagdil}@uom.gr

Abstract

In this paper we propose a new integrated programming environment, objectKarel, for teaching the object-oriented programming paradigm. Its main features are: a series of e-lessons, a special kind of structure editor, run-time error detection, program animation and recordability of students' actions. Finally, we describe the results of a pilot use of objectKarel to teach object-oriented programming to undergraduate students.

1. Introduction

An introduction to programming consists of a difficult mental task for students and for this reason intense research activity [2, 3, 4] has been carried out which aims to explore the relevant problems. These research studies show that often novices meet difficulties which can be independent of the programming paradigm used (such as procedural or object oriented programming), whereas other difficulties are associated more with the particular characteristics of each paradigm. One method which is often used in dealing with these difficulties is the development of programming environments created in such a way as to help novices overcome these difficulties. In this paper, we present one such environment, the *objectKarel*, for the support of an introduction to object-oriented programming through a microworld based on Karel++ [1]. We also present our first results from the lessons taught on an introduction to OOP which were organized with the aid of this environment.

2. Overview of the Programming Environment

o*bjectKarel* use the metaphor of a world of robots. The actor of the microworld is one or more robots (object) that is assigned various tasks in a world that consists of: crisscrossing horizontal streets and vertical avenues forming one block intervals, wall sections between adjacent streets or avenues used to represent obstacles (hurdles, mountains, mazes etc), and beepers - small plastic cones that emit a low beeping noise - placed on street corners. Students write programs that instruct robots how to perform their tasks.

The main teaching characteristics of objectKarel, are the following:

A special kind of Structure Editor. In order to avoid common syntactic errors that are both time *and effort* consuming, and mostly to prevent the students from focusing on syntactic details rather than the structure of the program, we developed a special kind of editor, which supports, in multiple ways, the writing of a program. In the *objectKarel* environment, writing a program is accomplished by: 1) choosing the appropriate action (class/method declaration, construction of object) or choosing a message to send to an object from a single menu. This menu is automatically updated whenever the user deletes or edits a class/method or declares a new class/method. 2) interacting with the system through dialog boxes.

Executing Programs - Program Animation. In order to help the students to understand the way in which the commands of their program are connected with the actions of the robots, we created an advanced system of animation and visualization. The user has three choices of executing it: Running the program: *objectKarel* presents the user with the final result of the program's execution. Tracing through the program: The program is executed in slow motion and each command is highlighted while the execution's result on the situation of the microworld and the actor *are* visible on the screen. Executing the program step by step: The user takes an active role and decides when the next command will be executed. Each command is executed in the same way as in tracing. The user can terminate the execution of the program at any time. When the students use the last two choices of executing programs, they are also presented with explanatory messages about the semantics of the command being executed. The explanatory messages use physical language and are presented in the pane that stretches across the bottom of the window. This feature is known as *explanatory visualization*.

Run-time Error Detection - Understandable Error Messages. The most difficult and frustrating phase of debugging is finding the logic errors of a program.

Proceedings of the The 3rd IEEE International Conference on Advanced Learning Technologies (ICALT'03)
0-7695-1967-9/03 $17.00 © 2003 **IEEE**

Unfortunately commercial compilers are designed for experts rather than novices and so they do not really help students in debugging their programs. Our programming environment intends to reduce this burden by detecting logic errors and reporting understandable and highly informative error messages: i) The line number reported is the actual line of the mistake; ii) Messages report not only what is wrong but also explain why it is wrong; iii) The error messages use physical language and not codes.

e-Lessons. *objectKarel* incorporates a series of e-lessons for supporting students in understanding the basic principles of object-oriented programming and the most common control structures. The main purpose of these lessons is to minimize the possibility of developing misconceptions and acquiring wrong or insufficient knowledge. For this purpose each lesson contains parts of: Theory intended to teach a specific concept of object-oriented programming or a control structure. The most important aspects of the OOP paradigm are disclosed before the control structures and are emphasized throughout the lessons. Students manipulate objects (robots) from the beginning and write their own classes almost immediately. An activity that lies *at the heart of the concept to be taught*. For example, the student can click on the buttons, labeled with the messages that the available object understands, and watch (i) the robot execute the message, (ii) the syntax of the command in the programming language.

3. Empirical Study

The software we developed was tested and evaluated by 20 undergraduate students from the department of Applied Informatics in Greece. Six lessons were carried out. The lessons took place weekly, and their duration was for 2 hours. We used the system's ability to automatically save a student's program and their errors/warnings each time the program was compiled. This possibility was incorporated in order to assist teachers in studying students' problem-solving techniques and also errors and misconceptions when they are introduced to the object-oriented programming paradigm.

In each of the first 5 classes we used the theory of the lessons. Next, students used the activities that are incorporated in the programming environment, and in some cases existing programs, so as to assimilate the concepts and become familiar with the features of the programming environment. Finally, the pairs developed one or two programs without help from the teacher. The 6th lesson was devoted to the assessment of students' knowledge and the evaluation of the programming environment and the series of lessons from the students. The choice of exercises was made in such a way as to be able to check the extent to which the specific environment as well as the programming course can reduce or even obliterate the difficulties related to programming and especially with the object-oriented program.

4. Results – Conclusions

The most important conclusions that were drawn from the analysis of the data are:

Students quickly became familiar with the programming environment and found the proposed problems interesting to solve. The analysis of the questionnaires showed that students evaluated positively the whole programming environment and the series of lessons.

The study of their programs revealed that in general, they did not encounter particular difficulties with the concepts taught. However, many students encountered difficulties in being able to differentiate between the concepts of polymorphism and overriding.

The syntax errors detected were mainly related to the scope of the *then* and *else* parts of the conditional structures. The study of the students' programs made clear that without the assistance of the programming environment - "good" error messages, step-by-step execution, explanatory visualization - most of the groups would have had great difficulty in solving the *proposed problems*.

Students have also difficulties with repetitive structures. A significant number of students finds it difficult to (i) distinguish the structures `if` and `while`, and (ii) develop programs that need both `if` and `while`.

Finally, after studying the students' programs we concluded that: understanding and debugging an existing program is quite difficult and time-consuming for some students, and editing a program, with our syntax editor, is easy. In general, the pilot use of the programming environment *objectKarel* showed that it is possible to teach object-oriented programming in a short period of time with specially designed educational environments, whereas this is more difficult with the usual professional programming environments.

5. References

[1] Bergin, J., Stehlik, M., Roberts, J. and Pattis, R., "Karel++-A Gentle Introduction to the Art of Object-Oriented Programming", 2nd edn., Wiley, New York, 1997.

[2] Du Boulay, "Some Difficulties Of Learning To Program". In Studying The Novice Programmer, Soloway, E., Sprohrer, J. (Eds.) Lawrence Erlbaum Associates, pp. 283-300, 1989.

[3] Holland, S. Griffiths, R. & Woodman, M., "Avoiding object misconceptions", Proceedings of the 28th SIGCSE, 131-134, 1997.

[4] Milne, I. & Rowe, G., "Difficulties in Learning and Teaching Programming - Views of Students and Tutors", Education and Information Technologies, 7:1, Kluwer Academic Publishers, 55-66, 2002.