Dynamic Binding is the Name of the Game *

Marco A. Gómez-Martín,
Pedro P. Gómez-Martín and Pedro A. González-Calero

Dep. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain
email: {marcoa,pedrop}@fdi.ucm.es, {pedro}@sip.ucm.es

Abstract. This paper presents a tutoring system aimed at teaching how to compile Java into the language of the Java Virtual Machine, and, at the same time, promotes a better understanding of the underlying mechanisms of object-oriented programming. The interaction with the systems takes the form of a 3D videogame where the student must compete to provide the right machine instructions, collect resources needed by the instructions and use her knowledge about Java compilation to find the best strategy.

1 Introduction

In 1970, Carbonell publicizes its classical paper about the integration of artificial intelligence techniques in computer assisted instruction [2]. This application would be later known as *Intelligent Tutoring Systems* (ITSs) [7]. During the last decades, much effort has been made to enhance these systems using different pedagogical approaches. One particularly widespread is based in the constructivist approach [4], that put emphasis on the learner's active role while they acquire new concepts.

Learning-by-doing method [6] is based on these theories. The apprentice approach has been used for ages, and today has been extrapolated to ITSs. More and more systems provides an habitable 3D environment [1]. Learners are represented in that environment by an avatar that they guide through the virtual world. The student has to make their avatar perform actions to solve the current problem the system has posed.

However, those systems focus on the learning factor, and often are boring applications that the student uses because they have to. On the other hand, computer users are more and more engaged to entertainment software. Players are immersed in microworlds, becoming part of the environment. When the game design, story and appearance are good enough, users can spend a huge amount of time using them. Mixing up both areas, ITSs and games, results in what is known as *game-based learning* or *game-based teaching* [5].

Related to these kind of systems, over the last few years, we have been developing JV^{2}M, a system to teach how to compile Java code, in particular,

* Supported by the Spanish Ministry of Education & Science (TIN2005-09382-C02-01)

which kind of target instructions has to generate the compiler of Java [3]. The application presents a virtual world where the student controls an avatar that executes the action they order to resolve the exercise.

2 JV^{2}M, a system to teach to compile

Usually, compiler subjects are quite hard due to the strong formal knowledge required to understand how these program work. However, if both source and object languages are known, the *translation* is, in fact, quite intuitive.

For some years we have been developing JV^{2}M, an application where students are faced to Java programs that must be translated to JVM assembler code. Users have to make this translation by themselves, using their intuitive knowledge about the source code semantic and the JVM internals.

Being JVM a *software machine*, some of its instructions have a quite high abstraction level. In that sense, some of them are designed to implement very specific characteristics of the object oriented programming, like dynamic binding. In that sense, pupils not only learn the translation process, but also practise these object orientation concepts.

The aim is the user generating the object code for the exercise. This could be done just written it down. But in this way, the complete process would be quite boring. Instead of that, our system recreates a game atmosphere, showing a 3D virtual world where the action develops. The environments has a special room where the compilation takes place. But, before entering in there, user must go around all over the place collecting resources (such as operands) needed for this translation. Apart of the 3D environment, the program has more typical game ingredients like time limit, enemies, and other strategy aspects.

As said before, some JVM instructions are quite pedagogically interesting for object orientation. Instead of just *compile them* in the special room, student must *execute them* in order to practise the runtime aspects of the object–oriented programming. In this sense, the most significant JVM instruction is `invokevirtual` used to call a method using dynamic binding.

The environment is enriched with Javy, an avatar that helps the student about the compilation process. This character has information about each exercise and the general aspects of the translation in order to adapt his explanation to the user knowledge.

3 Sample exercise

As an example of execution of the system, we will describe an exercise, to show the kind of Java programs we are thinking of. We will also detail the kind of JVM instructions the user has to face.

The exercise is a typical example in object oriented programming. It has an abstract class `Figure`, with a single `public abstract` method, `getArea` that returns a `double`. Two concrete classes inherit from it: `Circle` and `Square`, with

```
public class Circle
        extends Figure {

    Circle(double radius) {
        _radius = radius;
    }

    public double getArea() {
        return _radius * PI;
    }

    protected double _radius;
    protected final double
        PI = 3.1415926535898;

} // class Circle
```

```
public class Exercise {

    public static void main(
            String params[]) {

        Figure f1;
        Figure f2;
        double total;

        f1 = new Circle(10);
        f2 = new Square(10);
        total = f1.getArea();
        total += f2.getArea();
    } // main

} // class Exercise
```

Fig. 1. Code example

their particular constructors: the `Square`'s one receives the edge's longitude and store it in an object's attribute, and the `Circle`'s one receives its radius. The exercise is completed with a `main` function in a fourth class, that creates two figures, and calculates the sum of both areas. Figure 1 shows the Java code of two classes: `Circle` and `Exercise`.

The exercise does not pursue a perfect object oriented design. For example, `Figure` class should be an interface instead of an abstract class. Besides, `PI` attribute should be static or, even better, `Circle` class should use `Math.PI`.

However, this example introduces a lot of interesting concepts that the learner has to practice. The content creator (tutor) has to decide, based on their experience, how many concepts include in each exercise for the user to face.

In this concrete code, the student has to practice arithmetical expressions (sum and multiplication), attribute access and object construction. But, above all, the main goal of the example is dynamic binding. During the exercise resolution, the student has to think about the *execution* of the JVM `invokevirtual` instruction. When the program counter reaches `f1.getArea()`, the student has to realize that, though Java code seems to say that the method `Figure.getArea` has to be called (due to the *static type* of the `f1` variable, they have to find out its *execution type* to call the correct method.

Figure 2 shows the environment with part of the compiled code for this exercise. All the operands in the instructions (for example `Figure.getArea` in instruction 23) are the *resources* the student must look for before entering in the compilation room. This enforces the user to think about the translation while she is playing the level in the rest of the spaceship. Instructions 23 and 29 in the `main` code are the `invokevirtual` mentioned previously. Although they are both the same instruction in the object code, their executions are different because the runtime type of the references they affect are distinct.

Fig. 2. Application snapshot with object code

4 Conclusions

After a few years developing JV^{2}M, a system to teach how Java code is executed in the Java Virtual Machine using game-based teaching strategies, we are now in a position of been able to face users to real exercises.

This paper shows a description of the system and describes an scenario of execution. In particular, we list the Java code of an exercise that face the student with concepts such as attribute access, object construction, inheritance and dynamic binding.

References

1. W. H. Bares, L. S. Zettlemoyer, and J. C. Lester. Habitable 3D learning environments for situated learning. In *ITS '98: Proceedings of the 4th International Conference on Intelligent Tutoring Systems*, pages 76–85, London, UK, 1998. Springer.
2. J. R. Carbonell. AI in CAI: an artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):190–202, 1970.
3. M. A. Gómez-Martín, P. P. Gómez-Martín, and P. A. González-Calero. Game-driven intelligent tutoring systems. In *Entertainment Computing - ICEC 2004, Third International Conference*, Lecture Notes in Computer Science, pages 108–113. Springer, 2004.
4. J. Piaget. *The Construction of Reality in the Child*. New York: Basic Books, 1955.
5. M. Prensky. *Digital Game-Based Learning*. McGraw-Hill, 2004.
6. R. Schank and C. Clearyv. *Engines for Education*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.
7. D. H. Sleeman and J. S. Brown, editors. *Intelligent Tutoring Systems*. Academic Press, London, 1982.