Knowledge-Based Systems 22 (2009) 279–286

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A blended E-learning experience in a course of object oriented programming fundamentals

Jaime Gálvez *, Eduardo Guzmán, Ricardo Conejo

Dpt. of Lenguajes y Ciencias de la Computación, Bulevar Louis Pasteur, 35, Campus de Teatinos, 29071, Málaga, Spain

ARTICLE INFO

Article history:
Available online 29 January 2009

Keywords:
Blended learning
Assessment for learning
Learning reinforcement
Self-assessment in complex domains
Constraint-based modeling

ABSTRACT

In this paper, we present a blended e-learning experience consisting of supplying an undergraduate student population (in addition to traditional on-site classes) with a learning tool called OOPS (Object Oriented Programming System) and a testing system called SIETTE. OOPS is a problem-solving environment in which students can resolve Object Oriented Programming exercises. The system applies an assessment for learning strategy where students are formatively assessed, i.e. OOPS diagnoses their knowledge level but also generates feedback and hints to help students to understand and overcome their misconceptions and to reinforce correctly learnt concepts. In conjunction with OOPS, we have used SIETTE, a web-based assessment system in which students can take tests and teachers can construct them Subsequently, we have explored whether or not the use of OOPS contributes to improve the students' knowledge about Object Oriented Programming.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Intelligent tutoring systems are software solutions which provide students with personalized and self-paced instruction. These types of systems use Artificial Intelligence techniques in conjunction with learning theories obtained from psychological studies and research done in the educational field. Experts agree that what constitutes intelligence in intelligent tutoring systems is "real-time cognitive diagnosis" and "adaptive remediation" [1]. The main goal of Intelligent Tutoring Systems is to improve the student learning process. These systems supply an instructional environment which is adapted to the student's capabilities and learning needs, promoting even more effective learning than the traditional student–teacher instruction [2].

Even though the most common learning strategy continues to be face-to-face lessons imparted orally by a teacher, the number of alternative learning systems has increased. The ideal learning process is one where students can receive classes, resolve exercises and obtain immediate feedback from the teacher. Unfortunately, overcrowding in the classroom makes this desirable situation not feasible. Nowadays, teachers have to provide instruction to dozens or even hundreds of students, making it difficult for students to correctly assimilate the concepts being taught. By adopting learning systems such as intelligent tutoring systems, teachers could address this overcrowding situation using *blended learning*. This is a learning strategy based on incorporating different modes of teaching and learning styles. The aim is to introduce multiple media to facilitate student–teacher dialogue [3]. Several systems such as Assistment [4] have been used successfully in blended learning experiences.

The student overcrowding scenario mentioned above has given rise to the approach described in this paper. Several teachers provide instruction to undergraduate students, specifically, studying advanced programming in the second semester of Telecommunication Engineering at the University of Malaga (Spain). Around 300 individuals study this course each year. Three teachers are in charge of introducing students to the concepts of Object Oriented Programming (OOP). Up to that point, they will have only taken a course on the basic concepts of imperative programming. Each teacher provides instruction to two groups of around 50 students and the course syllabus is very dense. For this reason very limited classroom time is available for resolving programming problems or for assisting the students to develop programs. Consequently, from the last course, we have decided to introduce a blended learning strategy to facilitate the student learning process. This strategy consists of supplying the students (in addition to the on-site classes) with a learning tool called OOPS (Object Oriented Programming System) and access to a testing system called SIETTE.

SIETTE [5] is a web-based assessment system in which students can take tests and teachers can construct them. In general, the SIETTE tests could be classified in two categories in terms of the assessment procedure they use, i.e. conventional tests where student performance is measured heuristically by means of well-known criteria such as the percentage of success or the points

* Corresponding author. Tel.: +34 952 132863; fax: +34 952 131397.
E-mail addresses: jgalvez@lcc.uma.es (J. Gálvez), guzman@lcc.uma.es (E. Guzmán), conejo@lcc.uma.es (R. Conejo).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2009.01.004

obtained by totaling the questions answered correctly and subtracting those answered wrongly; and IRT-based tests in which well-found diagnosis are obtained using a model inspired by the Item Response Theory [6]. Other test classification criteria depend on how questions are posed to the student. Two types of tests can be identified, i.e. adaptive or non-adaptive ones. In the first, questions are dynamically selected. The goal is to select the most suitable question to improve the student's knowledge diagnosis and therefore, learning, using the least number of questions.

OOPS is a problem-solving environment in which students can resolve OOP exercises. The system applies an assessment for learning strategy, where students are formatively assessed. That is, OOPS diagnoses their knowledge level but also generates feedback and hints to help the students to understand and overcome their misconceptions and to strengthen the concepts they learnt correctly.

This paper is structured as follows. The next section describes the notion of assessment of learning and our initial experience in applying this strategy. Section 3 is devoted to the related work in programming tutors and in constraint-based modeling. This last modeling technique is the one which we have used to construct our domain model. The OOPS system is tackled in detail in Section 4. Section 5 describes the experiment in which we have applied blended e-learning using OOPS and SIETTE. Finally Section 6 outlines the conclusions we have reached with this work and some future research lines.

2. Background

Conventionally, assessment is used in education to measure and quantify the evolution of a student's learning. This is called *assessment of learning* (or *summative assessment*). However, assessment itself can be used as a learning strategy. This is what is called *assessment for learning* (or *formative assessment*), a process in which assessment is used in the classroom to improve student performance. It is based on the idea that students progress better when they understand their current state, the objectives of their learning, and how they can best achieve these objectives [7]. In the literature, we can find several studies showing that the application of this learning strategy leads to improvements in students' results [8].

We have previous experience in the application of blended learning strategies based on assessment for learning procedures using our system SIETTE. In [9], we explore the use of hints and feedback in self-assessment tests for a course of Language Processors (the goal being that students learn the most important issues about compiler construction) in the first semester of the 2004/2005 academic year. Accordingly, 57 students were administered a self-assessment test where each question was supplied with hints, to assist in understanding its stem, and feedback, revealed once the correction was shown. This test was administered after some face-to-face lessons given by the course teacher. The results suggested that those students who voluntarily took a self-assessment test with hints and feedbacks improved their overall mark.

Additionally, in other experiment [10], a different kind of test was used, i.e. open tests where students could take a test during a given time period but only the score and not the correction was shown. Once the time period expired, the test correction was shown to the students. Our aim was to establish whether we could use a testing system in a similar way to a drill-and-practice approach. We tested this type of test with the undergraduate students of a course of Artificial Intelligence and Knowledge Engineering and student data from three consecutive academic years was analyzed, i.e. between 2003 and 2006. Only in the last year was the open test made available to students. We compared the results of this experience with samples from the previous years and the evidence suggested that the open test could help students to improve their performance in the final exam.

The two results described above suggest that the use of self-assessment tests could contribute to improving student proficiency and therefore enhance learning. However, for complex domains, the use of testing is more difficult since a testing session would require an extremely large number of questions. This makes this kind of assessment exhausting from the student's perspective and an arduous task for teachers who would have to elaborate a huge set of questions. This is one of the main reasons which have inspired us to develop OOPS. Our goal is to present the students with a few problems but enough to be used instead of a self-assessment test and with analogous pedagogical efficacy.

3. Related work

Nowadays the advantages offered by Web technologies such as ubiquity or platform independence make them a useful way of facilitating the dissemination and the use of educational systems. As pointed out in [11], there are many Web-based systems available for educational purposes, however, this section focuses only on those using Constraint-Based Modeling (CBM) or OO paradigm tutoring systems.

3.1. Constraint-based modeling

This type of student modeling is based on Ohlsson's theory of learning from errors [12]. The technique proposes that students can learn from the feedback generated as the result of an error. According to this technique, the domain includes some basic principles which should be supported by all the solutions. In other words, these principles are a series of constraints that cannot be violated by any solution. Therefore, the relevance of the CBM does not depend on the steps the student follows, but rather on the type of solution that he/she finally produces. Another advantage of CBM is its computational simplicity, since the student model is reduced to a pattern matching process that will be explained in more detail below.

Within the user modeling research community, the Intelligent Computer Tutoring Group (ICTG) of Tanya Mitrovic has led the research on CBM. This group has developed a wide variety of tutoring systems which must be used as an indispensable point of reference: SQL-tutor [13], which is centered on the database domain; KERMIT (Knowledge-based Entity Relationship Modeling Intelligent Tutor) [14], whose goal is to teach database conceptual design by means of the model. The group has also developed some other tutors such as Normit [15] or Capit [16], and two authoring tools to facilitate the construction of tutors, such as WETAS (Web-Enabled Tutor Authoring Shell) [17], which provides student modeling, administration and automatic interface generation; and ASPIRE (Authoring System for Developing Constraint-Based tutors) [18] which works with an ontology of the system, providing automatic constraint generation. Using the previously mentioned tutors, Mitrovic et al. [19] have demonstrated that CBM is a efficient approach for student modeling.

3.2. Programming tutors

There exist several tutors whose goal is to teach programming concepts in different languages [20]. Perhaps the most popular ones are the family of ACT tutors [21–23], which are intelligent tutors for learning Pascal, Lisp or Prolog programming languages. These tutors use an approach called *Model Tracing* and use expert rules like those in CBM. Nonetheless, this approach is centered on the steps the student performs in order to arrive at a certain solution, instead of using the final state of a solution.

Inside the OO paradigm, there is a reduced number of tutors. ViRPlay [24] is centered on interactions in programs written in Java. This system tries to teach the execution of a program, using role-play simulations in a virtual 3D environment. However, the system seems to have neither a student model nor adaptive strategies. SmallTutor [25,26] is situated in the domain of OO programs written in Smalltalk. This system adapts the instruction using AI-planning techniques, according to a series of concepts that students must learn and a students must learn and a prerequisites graph. In [27] used but in this case to describe the problem application domain. This work proposes a generic architecture for intelligent programming tutors and focuses on inducing solutions to object oriented programs using a genetic programming technique. Although these approaches focus on different aspects of the OO Paradigm, none of them use the CBM. In the literature, we have only found one CBM-based tutor for learning object oriented concepts: COLLECT-UML [28] which can issues using UML.

4. The OOPS tutor

OOPS (Object Oriented Programming System) is a tutor which separates the functionality into four outstanding parts: (a) *an interface*, through which the students can solve problems in an easy to use workspace and teachers can elicit problems; (b) *a rule-based domain model*, which contains the representation of knowledge to be taught; (c) the mastery of a student in the topic is collected in the *student model*, which is used to adapt the learning process to his/her needs; (d) finally, the *pedagogical module* establishes the learning strategies and carries out the pedagogical tasks.

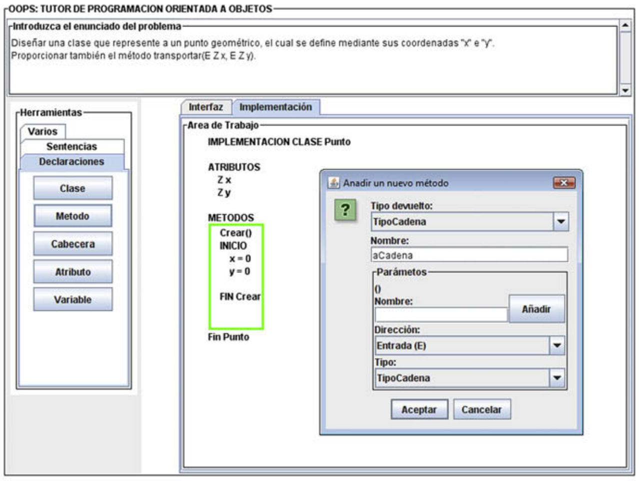
Our system allows teachers to add new problems using an authoring tool. For each problem, the teacher has to give its stem and should build what would be the correct solution to that problem in the same way as a student would do. In addition, using this authoring tool, problems and domain expert knowledge can be managed. From the students' perspective, the system has two working modes. In the first, hereafter referred to as the *guided mode*, students can resolve problems created by teachers for assessment purposes. OOPS sequences those problems adapting to every estimated user level. In the second mode, called *free mode*, students are allowed to use the system as a drill-and-practice environment; that is, solving their own exercises to check their knowledge or to resolve doubts.

4.1. The interface

This part of the system is the communication channel between the user and OOPS. Here, the system gets the representation of a solution and delivers appropriate problems, information and feedback. The students must apply their knowledge in order to create a solution for a given problem. This solution must be an OO program written using a pseudo-language, which has been created by the teachers of the programming courses of the Telecommunication Engineering degree at the University of Málaga.

Although OOPS only uses a subset of the whole language, it is enough to cover the main concepts of the OO paradigm. It should be observed that a program in this pseudo-language is formed by two modules: interface and implementation. The interface contains the public definition of the class without implementation code, and the implementation module contains the code of all these methods and other private content such as the attributes.

The interface allows OO programs to be built by means of a drag-and-drop mechanism, limiting the actions that can be done in the system and focusing users' thoughts on the solving process instead of writing source code. This strategy permits the system


OOPS: TUTOR DE PROGRAMACION ORIENTADA A OBJETOS

Introduzca el enunciado del problema

Diseñar una clase que represente a un punto geométrico, el cual se define mediante sus coordenadas "x" e "y".
Proporcionar también el método transportar(E Z x, E Z y).

Area de Trabajo

```
IMPLEMENTACION CLASE Punto

ATRIBUTOS
  Z x
  Z y

METODOS
  Crear()
  INICIO
    x = 0
    y = 0

  FIN Crear

Fin Punto
```

Anadir un nuevo método

Tipo devuelto: TipoCadena
Nombre: aCadena
Parámetros: ()
Nombre:
Dirección: Entrada (E)
Tipo: TipoCadena

Fig. 1. Interface look of OOPS working in guided mode.

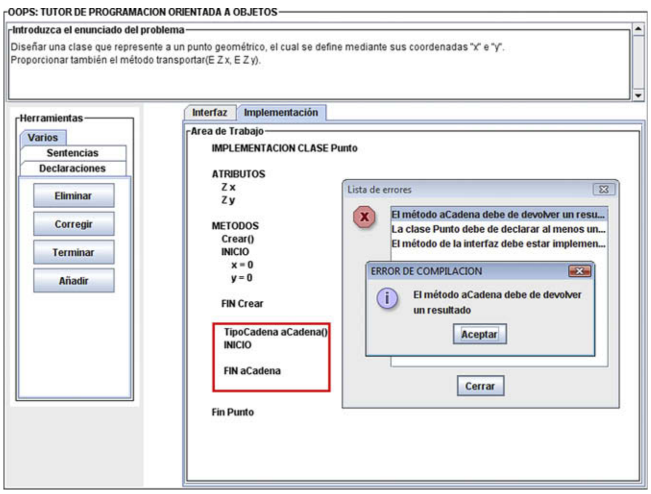
to trace all the actions performed by the students and accordingly facilitates the adaptation which can be provided by the system. As can be seen in Fig. 1, there are three main regions within the interface:

- The *stem*, situated at the top, is the problem description written by the teacher.
- A *toolbar*, on the left side (labeled "Herramientas"), organized into three tabs with every element that can be used to build a program, i.e. sentences, declarations and others. Declaration elements (the tab labeled "Declaraciones") are classes, attributes, methods, variables, etc. Sentences (the "Sentencias" tab) are expressions that allow defining, modifying and using objects such as assignations, arithmetic operators, method calls, etc. The remaining tab, i.e. the one labeled "Varios", contains actions that can be required to develop the solution. For example, to eliminate elements or compile programs. It is worth emphasizing that the two first tabs incorporate a button for each element and they can be used by dragging the button and dropping it into the workspace.
- The *workspace*, where the code of a solution program is constructed and edited. This area has two tabs labeled "Interfaz" and "Implementación" for both parts of a class, i.e. the interface and the implementation modules respectively. Each element the user wants to use is dragged from the tool bar into this workspace. When adding an element to the public or private part, some additional information such as names, types, parameters, etc. is requested.

At every moment during the problem-solving process, the whole set of expressions available in the domain model is available to the student. The use of some of these expressions will result in the wrong solution and the use of others will produce the right one, allowing the system to evaluate whether the student's actions are correct or not. The environment also acts as a compiler allowing users to know whether his/her solution is syntactically correct or not. When the compilation button is pressed, the system shows a list of all violated constraints (see Fig. 2) and related feedback, which will either be quite specific or more general depending on the estimated level of the student.

4.2. Rule-based domain model

The domain of OO Programming is very complex. There is no fixed sequence of actions that will leads to the solution, nor is there



Diseñar una clase que represente a un punto geométrico, el cual se define mediante sus coordenadas "x" e "y".
Proporcionar también el método transportar(E Z x, E Z y).

```
IMPLEMENTACION CLASE Punto

ATRIBUTOS
  Z x
  Z y

METODOS
  Crear()
  INICIO
    x = 0
    y = 0

  FIN Crear

  TipoCadena aCadena()
  INICIO

  FIN aCadena

Fin Punto
```

El método aCadena debe de devolver un resu...
La clase Punto debe de declarar al menos un...
El método de la interfaz debe estar implemen...

El método aCadena debe de devolver un resultado

Fig. 2. A set of feedbacks shown by OOPS.

only one solution for a given problem. Indeed, there are an infinite number of sequences/combinations which will lead the user to a valid solution. The use of CBM is useful for handling the wide space of solutions since it works with the final solution, separately of all possible steps to reach that solution.

According to CBM, what are important in the domain are those general principles that every OO program will support. These principles are represented by constraints that cannot be violated by any program. Ohlsson postulates that a constraint is a pair $\langle C_r, C_s \rangle$, where C_r is the relevance condition and C_s is the satisfaction condition. C_r is used to identify which will be the state of the problem to be satisfied, and C_s is what it is that the problem cannot violate. In our domain, an example would be the following: C_r = "*exist an assignation element*" and C_s = "types associated on both sides of the assignation must be equal". At the same time, constraints are encoded by rules of the form: *IF C_r is satisfied, THEN C_s should also be satisfied; otherwise a principle is being violated.* In summary, the domain model consists of a set of rules representing general principles that must not be broken.

We have modeled the domain of OOPS as CLIPS inference rules, where the antecedent of the rule is formed by a combination of the C_r and C_s condition, and the consequent is in charge of throwing an error with the information related to a violated rule. This error will be captured by the system, showing the corresponding feedback to the user. Inference rules have been written in collaboration with experts in the subject matter. A total amount of 86 rules have been identified and grouped into six different categories (see Table 1).

The rule-based domain is stored in a knowledge base, which facilitates their management by the teachers. Indeed, the system has a rule management module which allows rules to be modified and deleted easily. In addition to coding general principles in the domain, an appropriate representation of the program being developed is needed in order to check whether the goals are being satisfied or not, i.e. to evaluate the student mastery of the basic OOP concepts from the actions carried out. This representation is done in a declarative way and it consists of a series of facts that are associated with every element used in a solution. Hence, to determine the correctness of the student's final solution, the system matches the facts representing the student's solution with the facts representing the teacher's solution and applies a set of correspondence rules.

While the user is constructing a program, every action he/she carries out in the developing environment involves modifying the set of facts representing the solution. Every element in the code has an associated fact, which can be related to others. These relations are defined depending on the place where an element is situated in the code, forming a hierarchical structure. For example (see Fig. 3), knowing that the main element in the solution is a class composed of its interface and implementation modules, a reference should exist between the fact associated with the class and every fact corresponding to each element inside this class (methods, attributes and others). In the same way, every method will maintain a relation with every parameter or sentence declared inside it. In the figure we can also see an indirect relationship between the declaration of a method in the interface module and its corresponding implementation method. Summing up, a solution for a given problem is composed of a set of related facts that are added dynamically according to the actions carried out in the interface.

Facts used by the system are defined by CLIPS templates, which also define indirectly the corresponding elements in the code. That is, every element (class, method, attribute, sentence, etc.) will have a template associated. A template has a set of fields which determine the content necessary for every element. Some of these fields are common to all templates, such as an identifier, the template

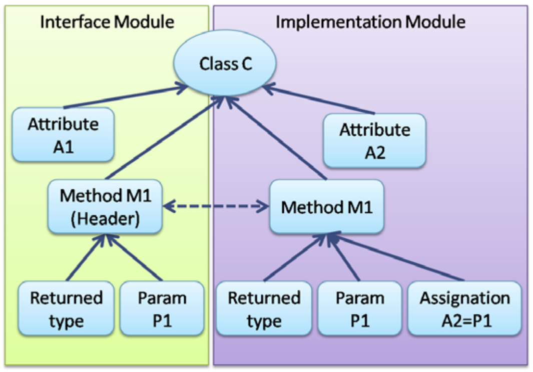


Fig. 3. Example of Hierarchical structure of OOPS generated facts.

Table 1
Rules categories of OOPS domain model.

Rule category	Description	Number of rules
Types	Try to search incoherencies among the types of expressions like method calls, arithmetic operators, etc.	10
Syntax	Related to syntactic errors and infraction of grammar rules	14
Modules correspondences	Check out the correspondence between elements in the interface (or public part) of a class and elements situated in the private part, such as implementation of a specific header defined in the interface	6
Visibility	Deal with errors derived from ignorance of the ambit rules of classes, methods, variables, parameters, etc.	15
Format of names	Principles related to the names of local variables, methods, etc. such as redefinition errors. Here are included redefining errors and code conventions such as uppercase at the beginning of a class name and other naming rules in order to aim the student using this well-practice	28
Missing references	Errors that may take place as consequence of code elimination. That happens, when a student removes the declaration of some variable, attribute or method which was used in the code and then, a reference remains to an element that no longer exists	13

name, a reference to the template where the element associated to this template is being used, and information about relationships with others elements in the code. We distinguish three different kinds of facts:

- Pre-defined facts, which are those associated with elements pre-defined in an O.O. program, such as basic types (Real, Integer, String, etc.) and pre-defined methods (for example, print).
- Facts given by the teacher, which are associated with elements that the teacher provides to students in order to facilitate the resolution of a specific problem. For instance, an algorithm which accomplishes a certain task.
- Facts corresponding to the solution being built, which are added, deleted or modified depending on the actions carried out through the interface.

4.3. The student model

OOPS keeps a model which estimates the student's knowledge level and also traces his/her actions. Using this model, the system can adapt the toolbar elements and the feedback shown thereby providing individualized instruction. The model is built while the student is working on solving problems. The system gathers relevant information, then stores, and updates the model using new evidence. According to CBM, the elaboration of a student model is an extremely simple task: it is composed of a list that contains every constraint violated by a student. In this way, the facts taken from every action done during the solving process are checked against the rules defined for the domain model. This is done by the pedagogical module and the pattern matching mechanism that is explained below.

In OOPS, the student model mainly consists of two elements, i.e. the list of violated constraints, and pairs of numerical values between 0 and 10, expressing the student's estimated level. These estimations are obtained and updated by the pedagogical module from the evidence generated by the student and the violated constraints. These values are used to adapt the instructional content presented to every student, which is done according to three intervals of equal length over the possible values where feedback and other presentation information are categorized. The information selected will depend on the category of the numerical value (*low*, *medium* or *high* level).

4.4. Pedagogical module

The pedagogical module is the core of the tutor since the whole system depends on it: material is presented and the evaluation is done here. Two outstanding pedagogical tasks, competence of pedagogical module, are: problem-solving assistance and curriculum sequencing.

4.4.1. Problem-solving assistance

Ohlsson [29] points out that learning from errors, the major principle of CBM, is a process consisting of two parts, error recognition and error correction. Psychological evidence indicates that feedback obtained immediately after an error is the most effective pedagogical action [13]. Feedback consists of pieces of knowledge that helps students to eliminate misconceptions or learn concepts previously unknown. There are two types of feedback [30]: (1) *negative feedback*, used to correct a wrong answer (e.g., a justification about why the answer is wrong) and (2) *positive feedback*, given to reinforce a correct answer. OOPS tries to assist users while they are practising their knowledge in order to detect misconceptions and to reinforce the concepts learnt correctly. Assistance occurs every time the user compiles the solution. The identification of errors is done using the domain rules and the facts representing the student's solution. For this purpose, a rule inference engine is needed. Specifically, JESS (Java Expert System Shell) [31] has been used in OOPS. It has a work memory where all rules and facts are loaded and it throws the corresponding errors when a rule is violated. The error thrown contains information for each of the three levels (low, medium, high). Only one of these feedbacks will be shown. It will be the one corresponding to the estimated level taken from the student's individual model. The lower the level, the more detailed the feedback shown will be, in order to allow the student to realize and correct their error.

4.4.2. Curriculum sequencing

The pedagogical module adapts the curriculum sequencing or instructional planning to provide a suitable level of difficulty to the students. It also uses the estimated level from the student model and additional information: the score for every rule and the estimated difficulty of a problem.

The score of a given rule represents a quantitative estimation of an error committed when that rule is violated. It is a value situated inside one of three intervals distributed between 0 and 10 (*non-relevant*, *medium*, and *severe*). This value will have been initially assigned by the teachers according to their expert criteria, and is updated with every compilation using a heuristic that compares the rule score with the average value for all rules. The score $S_r(t+1)$ of rule r after the $t+1$ compilation can be expressed as in Eq. (1):

$$S_r(t+1) = S_r(t) \cdot \frac{\delta}{\overline{S(t)}} \tag{1}$$

$$\overline{S(t)} = \frac{\sum_{i=1}^{N} S_i(t)}{N} \tag{2}$$

In this equation δ is the average value of the category where the rule is placed (i.e. 1.65 if the rule is *not important*, 5 if *medium*, and 8.35 if *severe*. $\overline{S(t)}$ is computed using Eq. (2), where N is the total number of rules.

The estimated difficulty of a problem is also represented with a value in the interval (0, 10] and it is updated when a compilation is done. It is worth noticing that repeated errors in successive compilations are not taken into account in order to avoid accumulation of errors. The heuristic formula (Eq. (3)) compares the average score of the problem for all students (Eq. (4)) with the mean (α) of all problems score (Eq. (5)). Notation meaning is as follows: p is the problem matter; D_p is the difficulty of the problem; S_{pi} is the score of the student i in the problem p; T is the number of students; and P is the number of problems in the system

$$D_p = \frac{\overline{P_p}}{\alpha} \cdot 5 \tag{3}$$

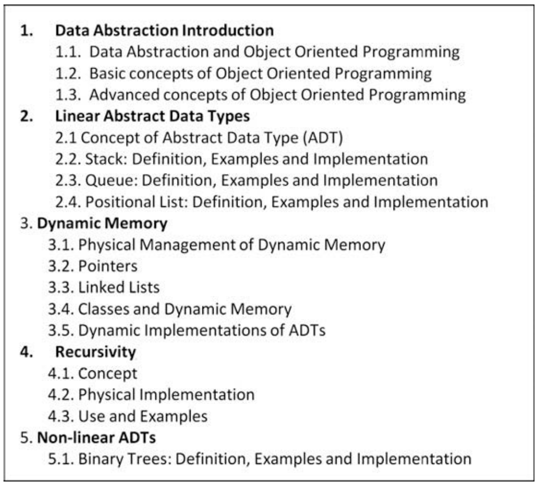
$$\overline{P_p} = \frac{\sum_{i=1}^{T} S_p i}{T} \tag{4}$$

$$\alpha = \frac{\sum_{j=1}^{P} Pj}{P} \tag{5}$$

Once these parameters are determined, it is necessary to adjust the choice of a problem to the student's estimated level and their specific weaknesses. The system tries to find problems with the same level the student has (low, medium, or high), ordered from lowest to highest (using D_p). If no problem has been found for this level, the problem with the subsequent nearest difficulty (belonging to other intervals) is chosen.

5. Experimental evaluation

This blended e-learning experience was incorporated into a course of advanced programming called *Programming Elements* taught during the second semester (approximately 14 weeks with



1. **Data Abstraction Introduction**
   1.1. Data Abstraction and Object Oriented Programming
   1.2. Basic concepts of Object Oriented Programming
   1.3. Advanced concepts of Object Oriented Programming
2. **Linear Abstract Data Types**
   2.1 Concept of Abstract Data Type (ADT)
   2.2. Stack: Definition, Examples and Implementation
   2.3. Queue: Definition, Examples and Implementation
   2.4. Positional List: Definition, Examples and Implementation
3. **Dynamic Memory**
   3.1. Physical Management of Dynamic Memory
   3.2. Pointers
   3.3. Linked Lists
   3.4. Classes and Dynamic Memory
   3.5. Dynamic Implementations of ADTs
4. **Recursivity**
   4.1. Concept
   4.2. Physical Implementation
   4.3. Use and Examples
5. **Non-linear ADTs**
   5.1. Binary Trees: Definition, Examples and Implementation

Fig. 4. Curriculum of the course.

only 2 h per week) in the Telecommunications Technical Engineering School of the University of Málaga (Spain). Fig. 4 illustrates the course curriculum. To pass the course, the students must assimilate correctly the concept of data abstraction and how it is implemented using an OOP language. These notions are introduced in topic 1, as can be seen in Fig. 4, and are vital for the students, since the rest of the course is based on them. The implementation of the Abstract Data Types (approached in topics 2, 3 and 5) is accomplished using OOP.

The pedagogical strategies applied in this course are based on face-to-face classes. These classes are theoretical, and teachers introduce the concepts the student should assimilate. At the end of each lesson, students are given a set of exercises which should be solved at home. In practical classes, the teachers solve those problems using the dashboard on the students' request. This last strategy has been proved quite ineffective, judging from the student failure rate. At the end of the semester, the students have to pass a test and a programming problem exam. The test assesses the theoretical concepts taught during the semester and represents 30% of the final score (3 points) where each student must obtain at least 1.5 points. Otherwise the exam is not corrected by the teacher and the student automatically fails. Results obtained at the end of each course, illustrate that this strategy is not adequate. The failure rate is considerably high and, according to the teachers, students have some important misconceptions about data abstraction from the beginning of the semester, probably making it very difficult for them to follow the course.

The ideal situation would be individual correction of students' solutions to the problem proposed during the semester in order to identify misconceptions and to apply appropriate feedback. Unfortunately, this is not feasible from the teacher's point of view due to the large number of students. To improve these negative results, we took a first step towards blended e-learning. At the end of the semester we applied the technique of making open tests available to the students. For a limited period they could try the open test but could not see the correction until the end of the time period allowed. The results of this academic year were encouraging: the number of students who took the final exam and passed was 74%, compared with 49% of the previous year. Taking all the students registered on this course, this percentage was 23% versus 14% of the previous year. However, the percentage of students who sat the final exam was only 31% whereas the previous year it had been 30%. As can be seen, despite this good result, the overall percentage of students who took the exam is still very small. The course teachers consider that this is probably due to the students' difficulties in assimilating the OO paradigm from the beginning of the semester.

5.1. Study design

To address this issue, we have decided this year to extend the blended learning-based strategy. We have developed OOPS to detect and correct misconceptions about OOP fundamentals. Problems in this system must be coded in a programming pseudo-language created by the course teachers. Nevertheless, any domain involving programming is complex from the point of view of monitoring the student actions. In the OOPS design, we have preferred not to use strategies such as scaffolding, since we consider that this limits the student actions while solving a problem. We have preferred instead to give the student freedom to do any kind of action and monitor them by means of a visual interface (based on drag-and-drop operations). Therefore the student does not code any instruction; he/she uses a tool bar to introduce sentences and other parts of the definition and implementation of a programming class.

After the lesson introducing OOP, i.e. about 6 h, we prepared a special session. This session took place in the labs of the school and was composed of three parts:

(1) It began with a pre-test administered through SIETTE. This test was composed of 15 multiple-choice questions where each question had three choices (only one of them was correct). Moreover, the students could leave any question blank. They also had a time limit of 15 min to complete the test. The test assessed all the concepts related to the data abstraction.
(2) Once they had finished the test, students had to solve problems using OOPS. These problems were similar to those included in the list of problems given by the teachers. Each problem consisted of constructing the public and private parts of a class with certain characteristics. The students had to determine and define the class attributes and the methods associated to it. The students had also to declare the public methods in the class interface. OOPS gave the student feedback on demand. Therefore, while the students were constructing their classes, they could push a button and the system indicated the number of errors they had and the reasons for those errors.
(3) After interacting with OOPS, students were administered a post-test similar to the pre-test, which was also administered using SIETTE. Our goal was to explore the improvement in learning experienced by the students after their interaction with OOPS. The format of the test was the same as in the pre-test, that is, 15 multiple-choice questions of three choices and a time limit of 15 min.

We also had a control group who only took the 6 h face-to-face classes but these neither worked with OOPS nor took the pre-test. This sample group was only administered the post-test. The criteria we used to divide the students into these two groups were established according to the academic groups each individual belonged to. Finally note that the score of both tests was expressed in terms of percentages, i.e. between 0 and 100. The cut off was 50%, that is, students who obtained a score equal to or greater than 50% passed the test.

5.2. Data analysis and results

A total of 47 students participated in this experiment. 29 individuals formed the experimental group, i.e. the group who accomplished the three steps of the experiment described above. The control group (of 18 students) was only administered the post-test.

Table 2
Comparison of the pre-test and post-test results between experimental and control groups.

		Number of students	Students who passed (%)	Mean	Std. Dev.
Experimental group	Pre-test	29	48%	43.79	21.94
	Post-test		81%	55.24	25.22
Control Group	Post-test	18	61%	44.32	26.68

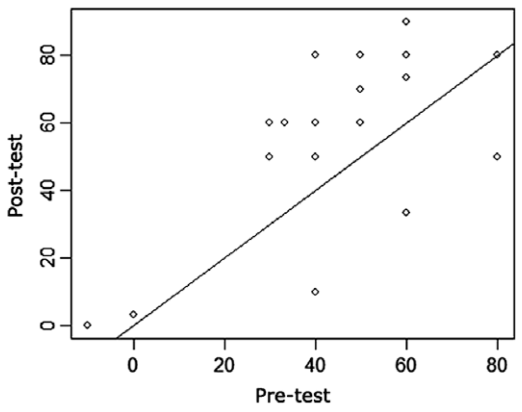


Fig. 5. Scatterplot comparing the performance of the experimental group individuals between the pre-test and the post-test.

Table 2 summarizes the results of both groups. The first two rows represent the data of the experimental group in the pre-test (first row) and in the post-test (second row). The last row contains the information regarding the control group. The third column shows the number of individuals who took each test and the fourth the percentage of them who passed the test (score equal to or greater than 50%). The last two columns contain the mean score of each sample and its standard deviation, respectively. As can be seen, the results suggest that even though the increase of the sample mean for the experimental group was not very high (12.70%), the percentage of students who passed each test increased from 48% to 81%.

We also performed a statistical analysis to see whether the results between the experimental pre-test group and the control group tests were similar or not. For this purpose, we used the classical statistical hypothesis test (i.e. the independent one-tailed *t*-test). The results suggest that we cannot reject the null hypothesis which states that the means of the two samples are different, ($p > 0.8223$) with 95% confidence. Furthermore, we made a pairwise comparison to determine whether the experimental group students increased their performance after working with OOPS or not. To this end we performed a paired *t*-test comparing the pre-test and the post-test performances of each individual. The goal was to find out if the difference between the performances of each student in both tests was statistically significant. The evidence suggests that we can reject the null hypothesis (which states that the mean of each individual before and after using OOPS is similar) with $p < 0.009115$ with 95% confidence. Fig. 5 illustrates the paired relationship using a scatterplot. It can be observed that most of the students improved their score between the pre- and the post-test.

6. Conclusions and further research

The main goal of this work was to try to improve the success rate of students enrolled on the Programming Elements course. To this end we have used strategies based on blended e-learning. In addition to the face-to-face classes (theoretical and practical classes) we have used two different e-learning systems: SIETTE, a web-based assessment system used to administer two similar tests; and OOPS, a problem-solving environment. The aim was to explore the contribution of OOPS for improving students' knowledge. To this end, OOPS presents the student with different problems and provides feedback during the process. Thus, the learning process is personalized and adapted to the student needs. This adaptation is done by means of a set of inference rules created by experts in the field and taking as input the student model and the performances of previous students who tackled the same problem in OOPS. Hints and feedback are adapted to the student's knowledge estimation. Thus this system is able to reinforce the student's weak points, thereby providing a very useful complement to the classes of Programming Elements. In this sense, the main strength of OOPS compared to the open tests we used in previous experiences is that with only a few exercises we can obtain more (or at least the same amount of) evidence about the student's knowledge as was obtained in those tests which contained many questions. Moreover, OOPS also provides tools for teachers to analyze the student's sessions, to add and modify new problems and to manage the rules, hints and feedback. The experiments we performed suggest an improved student performance after using OOPS. However, we still have to explore the influence of this improvement on the final score of the course which will take place in July 2008.

Despite already using this tool with real students, there is much work to do. OOPS only focuses on those concepts related to data abstraction. However, we would like to extend it to include new kinds of sentences, for instance, selection and/or iteration. For this purpose we should identify and add the constraints needed to model these sentences. The main problem is that this extension makes it much more difficult to determine whether or not the student's solution is correct. In addition, during the experiment, students pointed out several improvements from the usability point of view, which we will take into account in future versions of OOPS.

References

[1] V.J. Shute, J. Psotka, Intelligent tutoring systems: past, present, and future, in: David H. Jonassen (Ed.), Handbook of Research for Educational Communications and Technology, MacMillan, New York, 1996, pp. 570–600. Chapter 19.
[2] B.S. Bloom, The 2-sigma problem: the search for methods of group instruction as effective as one-to-one tutoring, Educational Researcher 13 (1984) 4–16.
[3] A. Heinze, C. Procter, Online communication and information technology education, Journal of Information Technology Education 5 (2006) 235–249.
[4] L. Razzaq, M. Feng, N. Heffernan, K. Koedinger, G. Nuzzo-Jones, B. Junker, M.A. Macasek, K.P. Rasmussen, T.E. Turner, J.A. Walonoski, Blending assessment and instructional assistance, in: Intelligent Educational Machines within the Intelligent Systems Engineering Book Series, Springer, Berlin, 2007, pp. 23–49.
[5] R. Conejo, E. Guzmán, E. Millán, M. Trella, J.L. Pérez de la Cruz, A. Ríos, SIETTE: a web-based tool for adaptive testing, Journal of Artificial Intelligence in Education 14 (1) (2004) 29–61.
[6] E. Guzmán, R. Conejo, J.L. Pérez-de-la-Cruz, Adaptive testing for hierarchical student models, User Modeling and User-Adapted Interaction 17 (1) (2007) 119–157.
[7] Assessment Reform Group, Assessment for learning: 10 principles – research-based principles to guide classroom practice, qualifications and curriculum authority, United Kingdom. <http://www.qca.org.uk/qca_4334.aspx>, 2002 (accessed January 2009).
[8] P. Black, C. Harrison, C. Lee, B. Marshall, D. William, Assessment for Learning: Putting It into Practice, Open University Press, United Kingdom, 2003.
[9] E. Guzmán, R. Conejo, Self-assessment in a feasible, adaptive web-based testing system, IEEE Transactions on Education 48 (4) (2005) 688–695.

[10] E. Guzmán, R. Conejo, J.L. Pérez-de-la-Cruz, Improving student performance using self-assessment tests, IEEE Intelligent Systems 22 (2007) 46–52.
[11] P. Brusilovsky, Adaptive and intelligent technologies for web-based education, KI – Kunstliche Intelligenz 13 (1999) 19–25.
[12] S. Ohlsson, Constraint-based Student Modeling, Student Modelling: the Key to Individualized Knowledge-based Instruction, Springer, Berlin, 1994, pp. 167–189.
[13] A. Mitrovic, B. Martin, M. Mayo, Using evaluation to shape its design: results and experiences with SQL-tutor, User Modeling and User-Adapted Interaction 12 (2002) 243–279.
[14] P. Suraweera, A. Mitrovic, Kermit: a constraint-based tutor for database modeling, Proceedings of Intelligent Tutoring Systems 2363 (2002) 377–387.
[15] A. Mitrovic, NORMIT: a Web-enabled tutor for database normalization, Proceedings of ICCE'02 International Conference on Computers in Education 2 (2002) 1276–1280.
[16] M. Mayo, A. Mitrovic, J. McKenzie, CAPIT: an intelligent tutoring system for capitalization and punctuation, in: Proceedings of IWALT'00, International Workshop on Advanced Learning Technologies, 2000, pp. 151–154.
[17] B. Martin, A. Mitrovic, Domain Modeling: Art or Science?, in: Proceedings of AIED'03, 11th International Conference on Artificial Intelligence in Education, 2003, pp. 183–190.
[18] A. Mitrovic, P. Suraweera, B. Martin, K. Zakharov, N. Milik, J. Holland, Authoring constraint-based tutors in ASPIRE, Proceedings of Intelligent Tutoring Systems, 4053 (2006) 41–50.
[19] A. Mitrovic, M. Mayo, P. Suraweera, B. Martin, Constraint-based Tutors: A Success Story, in: Proceedings of IEA/AIE'01, 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert systems, 2001, pp. 931–940.
[20] M. Gómez-Albarrán, The teaching and learning of programming: a survey of supporting software tools, The Computer Journal 48 (2) (2005) 130–144.
[21] A.T. Corbett, J.R. Anderson, Student modeling in an intelligent programming tutor, in: E. Lemut, B. du Boulay, G. Dettori (Eds.), Cognitive Models and Intelligent Environments for Learning Programming, Springer Verlag, New York, 1993, pp. 135–144.
[22] A.T. Corbett, J.R. Anderson, A.T. O'Brien, Student modeling in the ACT programming tutor, in: P. Nichols, S. Chipman, B. Brennan (Eds.), Cognitively Diagnostic Assessment, Erlbaum, New Jersey, 1995, pp. 19–41.
[23] A.T. Corbett, A. Bhatnagar, Student modeling in the act programming tutor: adjusting a procedural learning model with declarative knowledge, in: Proceedings of the Sixth International Conference on User Modeling, 1997, pp. 243–254.
[24] G. Jiménez-Díaz, M. Gómez-Albarrán, M.A. Gómez-Martín, P.A. González-Calero, ViRPlay: playing roles to understand dynamic behavior, in: Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, 2005.
[25] I. Morschel, SmallTutor – an intelligent tutoring system for object-oriented-programming, in: Proceedings of Möbus 1993, pp. 19–25.
[26] A. Zekl, I. Morschel, Embedding authoring support in an ITS for the learning of object-oriented programming, in: Proceedings of IEEE First International Conference on Multi-Media Engineering Education, Australia, 1994, pp. 59–64.
[27] N. Pillay, A generic architecture for the development of intelligent programming tutors, International Journal of Continuing Lifelong Learning 10 (2000) 275–285.
[28] N. Baghaei, A collaborative constraint-based adaptive system for learning object-oriented analysis and design using UML, in: Proceedings of Adaptive Hypermedia and Adaptive Web-Based Systems, Springer, Berlin, 2006, pp. 398–403.
[29] S. Ohlsson, Learning from performance errors, Psychological Review 103 (2) (1996) 241–262.
[30] R. Conejo, E. Guzmán, J.L. Pérez-de-la-Cruz, Towards a computational theory of learning in an adaptive testing environment, in: Proceedings of AIED'03 11th International Conference on Artificial Intelligence in Education, 2003, pp. 398–400.
[31] E.J. Friedman-Hill, JESS, The Java Expert System Shell, SAND-98-8206, 1997.