M-PLAT: Multi-Programming Language Adaptive tutor

Alberto Núñez, Javier Fernández and Jesús Carretero
Universidad Carlos III de Madrid
Spain

1. Introduction

The learning process is, most of the times, a difficult journey that one has to travel mainly on its own. A successful trip is usually determined by the indispensable help of a good teacher and the toolset at their disposal. The former is especially true when one tries to acquire programming skills. These new skills require a new way of thinking that can only be obtained with practice and experience. Subsequently, it can be a very frustrating and time-consuming process. Several works like (Lewis, 2002) and (K. Proulx, 2000) describe the difficulties that novice programmers have to face.

The main requirement to achieve good programming skills is to perform a lot of practices and exercises. The concepts that are under the hood have to be explained properly (here, the teacher's hand is essential), but the ability to use them correctly and smoothly can only be obtained by the practice and the experience.

The labour of the teacher not only includes the theoretical lessons but also the practical ones. The teacher's labour to instruct these practical lessons include things as: designing of practices and exercises, helping with doubts that could arise, checking the solutions proposed and warning about the mistakes committed. All these tasks consume most of the teacher's time and limit his ability to instruct a large number of students. That is the reason why most programming classes have a low number of students.

Up to now, there have been many efforts to improve the productivity of the teacher by automating these processes as much as possible. That is why tutoring systems came into play. The first tutoring systems were very simple. They were usually oriented to learn only one programming language and they only offered a set of multiple-choice questions to evaluate the student.

Nowadays, a new generation of ITS (Intelligent Tutoring Systems) and IPTs (Intelligent Programming Tutors) has emerged (See (Park, et al, 2001) for a good overview). There are several works related to programming tutoring systems for undergraduate students oriented to languages like Java, C, Lisp … The common aim of all those tutoring systems is to allow the students to familiarize themselves with the language basics (data structures, syntax, nomenclature, etc.).

An example of a more sophisticated ITS is (Aguilar & Kaijiri, 2002). This work is based on the evaluation of prior knowledge and abilities, learning styles, intelligence type and the

determination beliefs and attitudes. This system generates a tailored course based on the obtained results of a previous evaluation. Another example is (Aase & Kurfess, 2004). This work presents a Web-based learning environment to teach Artificial Intelligence concepts to college students.
At present there are several IPTs (Intelligent Programming Tutors) like RAPITS (Rapid Prototyping Adaptive Intelligent Tutoring System) (J. Woods & R. Warren, 1995), SIPLeS (Y. S. & S. Xu, 1997), SIPLeS II (Xu & Y. S. Chee, 1999), MoleHill (S.R. Alpert et al., 1995), PROUST (Lewis, 2002) or INTELLITUTOR (Ueno, 2000). Also, there are works (like (Pillay, 2003)) that proposes a generic architecture for the development of intelligent programming tutors.
One of the biggest problems that all tutoring systems have to face is that each student has its own way of thinking and learning style. The work (Valdés, 1999) describes the concept of learning style as "The set of cognitive, affective and physiological factors that will be used as stable indicators of how the learner perceives, how the learner interacts with his or her environment, or how he or she responds to the environment." Furthermore, it mentioned that the learner uses learning styles when he or she applies strategies in the learning process. The learning styles have their basis in the neurological structures and in the experiences acquired through the life of an individual. A good tutoring system has to bend to this reality and adapts its behaviour depending on the learning style the student shows.
In this work we present M-PLAT, an intelligent tutoring system aimed at learning programming languages. M-PLAT is focused on achieving a system that helps the student to gain practice and experience in learning a specific programming language. The design of M-Plat tries to cover those objectives that are crucial to achieve a good ITS. The basic objectives that M-PLAT intended to fulfil are the following: First, M-PLAT should allow the student to perform programming exercises that are adequate to his level of knowledge. Then, the system will check the solution proposed by the student. Finally, the student will receive a feedback about the quality of his solution and a summary of the existing faults and mistakes. Other objectives of the M-PLAT design are: support for on-line documentation, access to historical reports and evaluations, adapt the behaviour to each student characteristics, ensure authentication and non-refusal, present an usable and fool proof environment, etc.

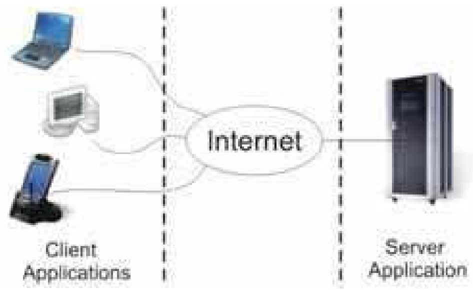


Fig. 1. Global System Schema

M-PLAT has been developed as a web application using a client-server architecture (figure 1). It has been developed using Web Services technology. Thus, the system is both platform and language independent. Client side application has been developed in Java. Basically it contains a GUI where students can study the corresponding topics and write their programs, which will be sent to server side application to be evaluated. Server side application has been written in C# and contains a database, lessons, exercises, compilers, an expert module that evaluate students, a pedagogical module that adapts the tutor behaviour, etc. The communication between those two applications is performed using the SOAP protocol. Due to our proposed system is platform independent, it can be used in a mobile device, in a laptop, or in a personal computer.

2. M-PLAT Design

M-PLAT has been designed as a teacher's tool that eases the learning process for the student. The objective is to automate most of the learning process related to perform practical exercises. As a good intelligent tutor system, M-PLAT will adapt the level of requirements to the learning pace of each student. M-PLAT will provide to the student a learning environment where the student can perform the exercises proposed by the system. The system will compile, execute and check the student solution and a report with the faults and mistakes founded is sent to the student. The system will also record the student history in order to improve the learning experience.

M-PLAT is designed to operate in one of two working modes: online mode and offline mode. Each mode has different objectives and requirements. The online mode is focused on acquiring the practical basis and fastens the individual concepts. The work of the student consists on several exercises to be performed individually and oriented to acquire a specific concept or knowledge. These exercises are classified on different levels of exigency. The student should open an online M-PLAT session. After that, the system proposes exercises to the student, one at a time. Once the exercise is finished, it is submitted to the system. M-PLAT will compile, execute and check the solution. If it is OK the system proposes a new exercise. If not, the system sends the faults and mistakes founded and waits for the student to fix the solution.

The offline mode is focused on learning how to mix the individual concepts and to perform larger exercises. The online behavior cannot be used here because these exercises take days or weeks to be finished, and it also common to propose the exercise as teamwork. These proposed exercises are intended to be developed using state-of-the-art programming environments (not M-PLAT). MPLAT is in charge of collecting the solutions and, after compiling, executing and checking, it sends to the authors a report with the results as soon as possible (normally within a day). But the tools to compile are external and maybe any of the existing into the market.

M-PLAT has been implemented as a web application. This design lets the client application to run on every platform and to update it everywhere instantly. The client is merely a GUI that offers the M-PLAT functionality that is implemented in the server. The communication between client and server is performed using Web services. The server offers a generic service called Programming Languages Learning Method (PLLM). This service is independent of the programming language used. The service is implemented as a bunch of modules, each one in charge of an specific task. These modules are:

- The communication module: Used as interface between client and server.
- The expert module: Used for compiling and checking the online exercises.

- The online documentation module: Used for offering documentation to the user.
- The pedagogical module: Used to adapt the exercise level to the student behaviour.
- The offline exercises module: Used to collect, compile and check the offline exercises.
- The database module: Used to keep the students profiles and other useful information.

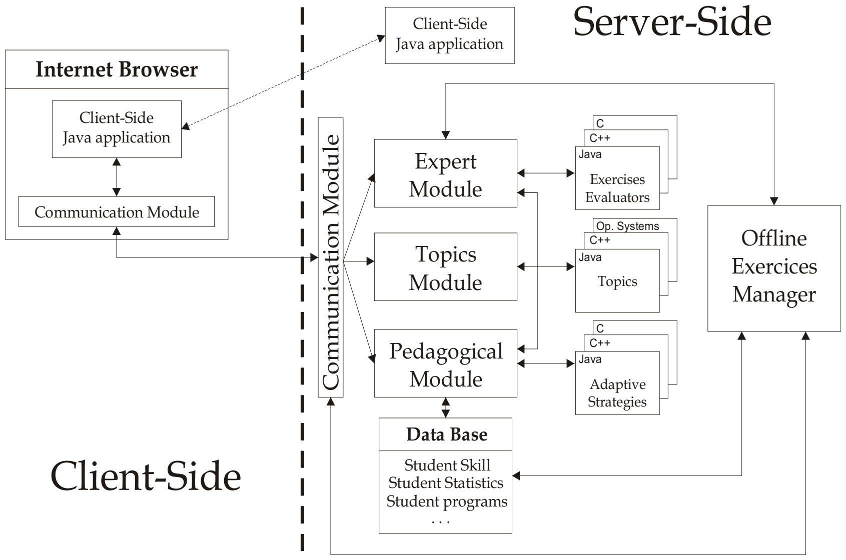


Fig. 2. M-PLAT Architecture

2.1 M-PLAT working modes

The main aim of M-PLAT is to ease students the way they learn programming languages. Mainly, M-PLAT is focused on monitoring the progression of students that are coursing certain subjects, like Operating Systems, Distributed Systems, Advanced Programming, etc. Thus, the M-PLAT system provides two different working modes for students.

First mode, called online mode, is used for performing exams and exercises that can be developed in relatively short time. In this mode, a set of exercises is presented to the student, where the student has to solve an exercise before trying to solve next one. The student develops his solutions on the M-PLAT environment. When a student has finished its program, the source code is sent to the server to be compiled and evaluated. Then, a response with the results is sent back to client-side application. After that, the student will receive one of the following messages:

- Compilation errors have been found.
- There are no compilation errors, but the provided program is not correct.
- The program is correct.

If the program is not correct, then the student has to modify its program and send it again to the server. The maximum number of tries that a student has to solve a corresponding exercise is configurable in M-PLAT.

The most common way to use M-PLAT online is that the student performs exercises in order to get practice and experience. Thus, the student has to open a session on the M-PLAT environment. Then, the system will propose exercise after exercise trying to adjust the difficulty with the needs of the student. One of the most relevant features of MPLA-T is that it can adapt itself dynamically to each student's behaviour. To do so, M-PLAT records the sessions and grab data about the behaviour of the student when solving the exercises and use this information to decide which exercise should be proposed next. For example, if M-PLAT detects that a student solves exercises fast with few errors, the next exercise will be more difficult. By the contrary, if a student has a considerable number of errors and spends more time than estimated to solve exercises, then next exercises will be easier. The main objective is to avoid the student to get stalled.

Another popular use of the online mode of M-PLAT is to perform evaluation test and exams to obtain the level of the students. In this case the exercises are previously defined (normally by the teacher). Another variation is that the system output when a exercise is submitted can be restricted. For example, the system will notify the compilation errors, but not the execution errors; or it can notify if there are execution errors, but not which ones. Also the maximum number of submissions can be configured as well as a maximum time to perform the exam.

Second mode, called offline mode, is used to evaluate exercises asynchronously. As said before the online behaviour cannot be used here because these exercises take days or weeks to be finished, and it also common to propose the exercises as a teamwork. This kind of exercises is commonly used to evaluate the level of the students. Thus, it is normal that the exercises proposed have a submission deadline. In this case M-PLAT is only in charge of the submission, correction and returning back the correction report. These practices are supposed to be performed in a real programming environment.

The offline mode requires a fool-proof, secure and non-refutable submission environment That includes things like: using secure channels, authenticating using username and password, restricting the submission only for the correct files by checking the name and the type (using the extension and MIME types), returning the student an electronic-signed copy of his submission that can be used as a proof for him.

The corrections are done off-line periodically (for example, once a day) and the resulting reports are sent to the students by e-mail. The policy about the correction reports depends on the learning objectives. The reports can be brief or detailed depending on the level of requirements to the students. The number of submission can also be limited to force the students to use they own correction techniques.

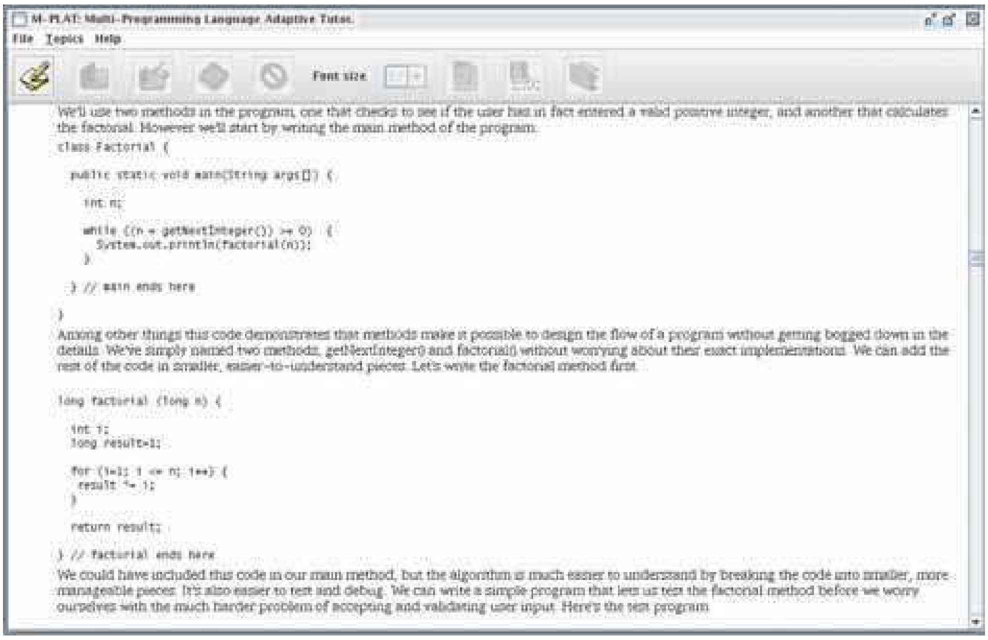
2.2 M-PLAT Modules

In this work we have commented that the M-PLAT system follows a client-server architecture. The entire system consists of a set of modules; where each one has been designed to perform a concrete task. In this section, the functionality and the purpose of all those modules will be described with detail.

2.2.1 Client-Side Java application

This module contains a friendly and easy-to-use graphical interface. This GUI contains two sections. First one (Figure 3) is used to access the online documentation of a concrete programming language. Those lessons consist of HTML pages that are hosted and downloaded from the server. Those pages contain corresponding topics related to theoretical lessons, like loops, functions, modules, data structures …

The second mode (evaluation mode) is used to evaluate students and check their skills about specific topics of the corresponding programming language. In evaluation mode the screen is split in two areas. The top area is the exercise description area, and the bottom area is the program area. In the program area students can write their programs (see Figure 4).
Also, a student can obtain help while are writing a program. Thus, in the exercise description area is shown the current topic related to the corresponding exercise.
This application is initially placed at the server side (a binary .jar file). Students have to use an Internet Browser to establish connection with server. Then, using the Web Start application, the client-side Java application will be downloaded to the client computer and then executed using the browser. The advantage of this system is two-folds. First, there is only one client-side application copy, which is stored in the server. Then, if we have to modify the application, we only have to change and compile one code. Thus, each time a student establish connection with server-side, the student will download the last version of the application. Second, we can enable access for certain versions of the client application. Thus, if we detect a security failure in the client-side application code, we can generate a new version that solves the problem and denying the access to the server to all previous versions that contain the failure.



We'll use two methods in the program, one that checks to see if the user has in fact entered a valid positive integer, and another that calculates the factorial. However we'll start by writing the main method of the program.

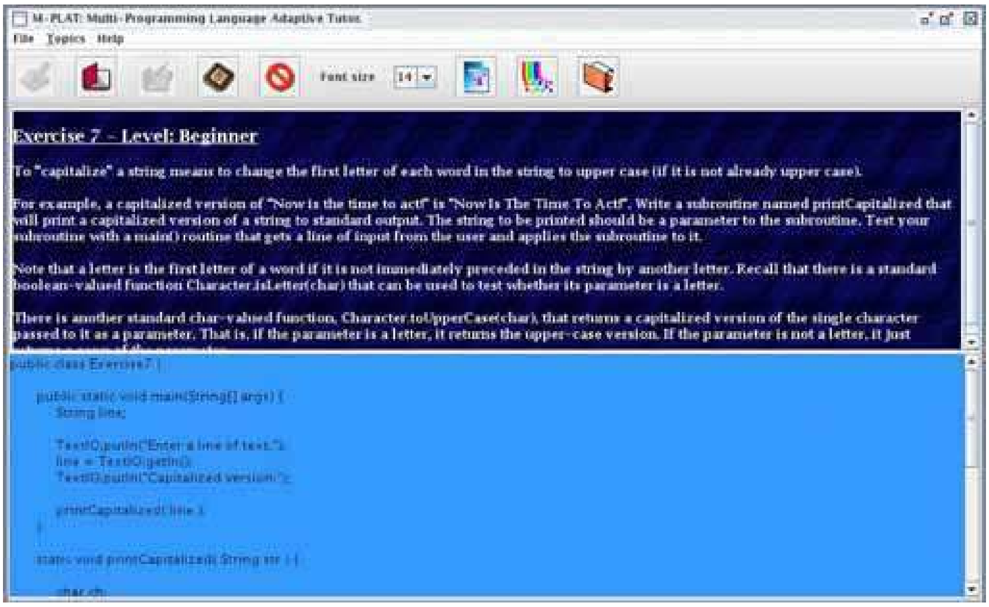
```
class Factorial {

  public static void main(String args[]) {

    int n;

    while ((n = getNextInteger()) >= 0)  {
      System.out.println(factorial(n));
    }

  } // main ends here

}
```

Among other things this code demonstrates that methods make it possible to design the flow of a program without getting bogged down in the details. We've simply named two methods, getNextInteger() and factorial() without worrying about their exact implementations. We can add the rest of the code in smaller, easier-to-understand pieces. Let's write the factorial method first.

```
long factorial (long n) {

  int i;
  long result=1;

  for (i=1; i <= n; i++) {
   result *= i;
  }

  return result;

} // factorial ends here
```

We could have included this code in our main method, but the algorithm is much easier to understand by breaking the code into smaller, more manageable pieces. It's also easier to test and debug. We can write a simple program that lets us test the factorial method before we worry ourselves with the much harder problem of accepting and validating user input. Here's the test program.

Fig. 3. M-PLAT GUI - Topic section



Exercise 7 – Level: Beginner

To "capitalize" a string means to change the first letter of each word in the string to upper case (if it is not already upper case).

For example, a capitalized version of "Now is the time to act!" is "Now Is The Time To Act!". Write a subroutine named printCapitalized that will print a capitalized version of a string to standard output. The string to be printed should be a parameter to the subroutine. Test your subroutine with a main() routine that gets a line of input from the user and applies the subroutine to it.

Note that a letter is the first letter of a word if it is not immediately preceded in the string by another letter. Recall that there is a standard boolean-valued function Character.isLetter(char) that can be used to test whether its parameter is a letter.

There is another standard char-valued function, Character.toUpperCase(char), that returns a capitalized version of the single character passed to it as a parameter. That is, if the parameter is a letter, it returns the upper-case version. If the parameter is not a letter, it just

```
public class Exercise7 {

    public static void main(String[] args) {
        String line;

        TextIO.putln("Enter a line of text.");
        line = TextIO.getln();
        TextIO.putln("Capitalized version:");

        printCapitalized( line );
    }

    static void printCapitalized( String str ) {

        char ch;
```

Fig. 4. M-PLAT GUI - Evaluation section

2.2.2 Communication Module

Communication between client and server applications is performed by using Web Services. Web services can be seen as an application programming interfaces (API) exported across Internet. It lets clients to execute code on the servers, no matter how is developed each part (client or server).

Web services allow different applications from different sources to communicate with each other without time-consuming custom coding, and because all communication is in XML, Web services are not tied to any operating system or programming language. Thus, software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks. For example, a client application written in Java running on Windows can communicate with a server application written in C# running on Windows.

The characteristics that define web services are:

- Extensible Markup Language (XML)
- The HTTP standard is allowing more systems to communicate with one another. Moreover, web services can work through many common firewall security measures without requiring changes to the firewall filtering rules.
- SOAP (built on XML) standardizes the messaging capability on different systems.
- UDDI standardizes the publishing and finding of Web services.
- WSDL standardizes the description of Web services so providers and requesters are speaking the same language.

Web Services are the core of M-PLAT communications. All the interfaces that define the services M-PLAT provides are in the communication module. This module solves any subject related with intercommunications issues, leaving the core business to the rest of the modules.

2.2.3 Expert Module

The expert module is in charge of evaluating the programs written by students during the online sessions. This is done executing a set of tests for each exercise in order to evaluate the student solution. The aim of those tests is to check the correctness of a program implementation.
Software testing, which is the process of analyzing the software to detect the differences between the real behavior and the required behavior can be one using two techniques: white box technique and black box technique.
Black box testing ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. With *black box testing*, the software tester does not have access to the source code itself. The code is considered to be a "big black box" to the tester who can't see inside the box. The tester knows only that information can be input into to the black box, and the black box will send something back out. Based on the requirements knowledge, the tester knows what to expect the black box to send out and tests to make sure the black box sends out what it's supposed to send out.
White box testing takes into account the internal mechanism of a system or component. *White box testing* focuses on the internal structure of the software code. The white box tester knows what the code looks like and writes test cases by executing methods with certain parameters.
M-PLAT System uses both White box and Black box testing methods. For example, in Java we use the Reflection API. With this API we are able to inspect the structure of a class at run-time, thus a method signature can be checked dynamically to check if a program satisfies the corresponding requirements or not. This is white box testing.
When a program is evaluated, it is not enough to compile the source code and check the method's signatures, a set of tests must been checked. All tests are executed using the programs written by the students and the results are compared with the correct ones. If the student program passes a corresponding percentage of tests, then the program is considered correct. Those tests are black box testing.
This module contains all tests corresponding to each programming language supported by M-PLAT, and also the configuration files to customize the number of tests of each problem.

2.2.4 Online Documentation

This module contains all lessons related to a concrete programming language or a certain subject (like Operating Systems). Those lessons are hosted in the server machine in HTML format. Thus, client applications can download the corresponding page to be visualized in the client machine. Those pages are visualized in the client application, no additional browser is needed.

2.2.5 Pedagogical Module

This module contains the strategy to adapt the exercises level to each student programming skills. Thus, an advanced student is supposed to write complex exercises while a novel student should start solving very basic programs. The major goal of this module is to reduce the frustration that novice students fell when they try to solve complex programs without the necessary knowledge. Thus, a novice programmer will not solve complex programs until (s)he acquires the required level solving basic programs.

We think that the best feature characteristic of M-PLAT is the possibility to dynamically adapt itself to the learning capacity of each student. In order to perform this adaptation successfully, our IPT has to learn how the students assimilate the concepts that the teacher explains in the classroom. One of the main tasks of this module is to calculate which is the actual level of knowledge of the student and decides the category of next exercises, using information obtained from the client application.

We analyzed which aspects are relevant in order to study the corresponding learning capacities. We considered that the three following parameters are the most relevant: Time, memory, and number of errors. Next we briefly describe how these attributes are treated as well as their influence in the total computation.

Time: The time that a student spends to solve an assignment is indeed very important. If a student does it very fast then we may conclude that this student understood the concepts underlying the assignment. Moreover, the student knows a good way to solve it. In contrast, some other students might take a longer time to solve the very same assignment. This is mainly the case if this student has written few programs before. Once a student starts to solve assignments relatively faster, this student is ready to advance to the next level. Thus, time represents the speed and fluency that the student has to solve problems at his/her current level of knowledge.

Memory: As we said in the previous section, a student can obtain help from our system while (s)he is solving an assignment. In particular, that student can have a look at the corresponding lecture. If a student accesses this help system, we may deduce that that student did not completely understand the concepts. Thus, the student needs to study them more thoroughly. If a student solves an assignment without asking M-PLAT for help then we deduce that (s)he is learning well. Intuitively, this parameter represents the capacity of the student to retain the concepts presented in the classroom.

Number of errors: First, we have to remark that by this parameter we do not mean the number of errors that contains a program when a student compiles it. The number of errors represents the times that a student clicks the checking button until the program is correct. Let us remind that, as we explained before, the answer to an assignment can be incorrect either because the program has compilation errors or because it does not preserve the statement of the assignment. As we might expect, a student with some doubts will increase the number of tries while the number of errors will decrease when the student improves his command on the topic. Intuitively, this parameter represents the clarity with which a student controls the involved concepts.

Let us remark that these three parameters are complementary to each other, that is, in order to compute the capacity of the student we should not use one of these parameters alone. Each parameter has a different weight, but the values of these weights can vary when we change the environment. For example, teachers usually appreciate that students have the smallest number of errors as possible. In particular, this means that students understand and learn the programming language that (s)he is teaching. On the contrary, the director of a company values with more priority the time parameter. This is so because it is not relevant that the employee has a lot of errors in previous versions of the program as long as the task is finished soon, and the program works.

2.2.6 Offline exercises Module

The main purpose of this module is to manage all the issues related to the offline exercises. This includes submitting, testing, and sending the results. The submission is performed using a dynamic web page hosted in the M-PLAT server that uses HTPPS secure protocol. The solutions submitted are stored on the server. Periodically, this module takes the latest submissions and checks them with the test battery for this exercise. The results are then mailed to the students.
This module is able to manage:
Students: The system must contain lists of students that have access to the system. Those students are stored in the database, and need a username and a password to obtain access. Once a student is registered in the system, the corresponding username and password will be sent by e-mail to the student's account. System administrator will link each student with the corresponding subjects in current academic year.
Subjects: The system can manage subjects for current academic year. Each subject has associated a list of students and a list of exercises.
Exercises: Each subject can contain one or several exercises to be delivered along the academic year. The system can configure a list of exercises, each one with the corresponding testing modules and the corresponding deadlines.
Those data is stored in a data base (see Figure 2). Figure 5 shows the Entity-Relation datagram that this module performs with the data base.
Once students are registered in the system, they can send the corresponding programs related to an exercise. By default, the system is configured to evaluate all solutions daily at the same time. The e-mail sent to each student shows the corresponding qualification and a list of tests that the corresponding solution passes and does not.

2.2.7 Data Base Module

Our proposed system keeps profiles for each student where the relative information about the programs written as well as the evaluation of them will be stored. We think that this is a very useful feature of our system. On one hand, teachers can obtain statistics and evolution of all students. On the other hand, any student can check its own evolution in the knowledge of the respective programming language or subject.
Moreover, M-PLAT system is able to load students in the Data Base using lists. Those lists must contain a set of students where each student is represented with two values: Name and ID.

Name: Contain the complete name and surname.
ID: Identity National Document number.

Our University (Universidad Carlos III de Madrid) provides a system to export list of students in two formats: plain text and excel format. Thus, M-PLAT system can import those lists using whatever of those two formats.

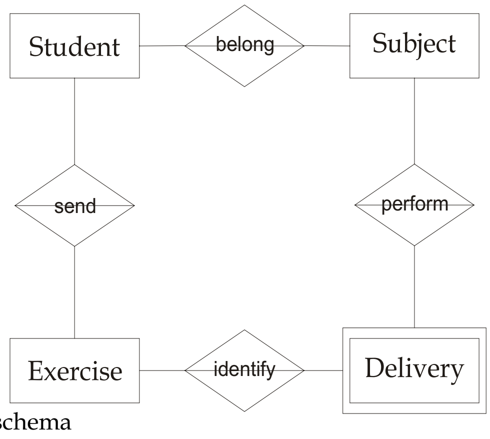


Fig. 5. Entity-Relation schema

3. Performance Evaluation

In order to check the performance of the proposed system, a performance test simulating the learning style of a group of students has been made. The main objective of this performance evaluation test is to obtain the maximum number of students that can use the system simultaneously, maintaining a reasonable performance of the system. When several students using the system send programs to the server side, the system will compile and evaluate all those programs simultaneously. The response time of each student logically will increase when the number of students grows. The features of each machine that contains the service (server) are:

- Intel Pentium 4 CPU 3.20 GHz.
- 2 GB of RAM. Cache size of 1024 KB.
- Seagate Disk (300 GB)

Those machines are connected to Internet through an Ethernet Gigabyte network. All programs sent by students are written in Java Language. The Java compiler version used is 1.5.0_07.

The performance test consists of several students making a M-PLAT-Java exam. An exam in M-PLAT consists of 10 problems where students have to write a program for each problem. Each one of those programs is sent to server side to be compiled and evaluated, and then the results are sent to client side. A student has 10 tries to write a correct program for each problem.

Because it is very difficult to get enough students to make a real test, we have decided to simulate the student's behavior. The students have been grouped, depending of the learning style, in three categories: novice programmers, intermediate programmers and advance programmers.

Figure 5 shows the simulated learning style of each category. The **x** axis shows the number of times that a student has sent a program to the server to be evaluated. The **y** axis shows the probability that the program sent will be correct. If a program is not correct, the simulated behavior of each student waits, depending of the student category, an amount of time (between 2 and 5 minutes) after send again the program to server.

Novice student behavior has been modeled with a normal distribution with parameters μ=6,5 and σ=2,5. This model reflects that the probability of a program written by a novice programmer to be correct is very limited the first times. This kind of students tends to send several times an incorrect program and debug it with the feedback provided by the server, where several error and advices messages are shown. This category of students needs to send a program 7 times to reach the 50% of probability of the sent program to be correct.
Intermediate student behavior has been modeled with a normal distribution with parameters μ=5 and σ=3. This kind of students corrects faster an incorrect program than novice programmers. Thus, those students send a smaller number of programs to the server. To reach the 50% of probability that the sent program is correct those users need to send a program 5 times.
Advanced student behavior has been modeled with a normal distribution with parameters μ=2 and σ=5. This kind of students learns fast. At the 2nd try, this kind of students reach the probability of 50% that the sent program is correct.
Figure 6 shows the results when several simulated students make a M-PLAT Java exam. Based on the qualifications of last year, the category student ratio is: 30% of novice students, 50% of intermediate students and 20% of advanced students.

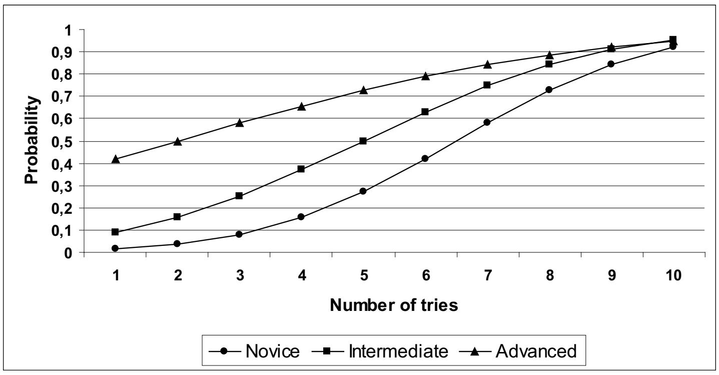


Fig. 6. Modeling of student learning style

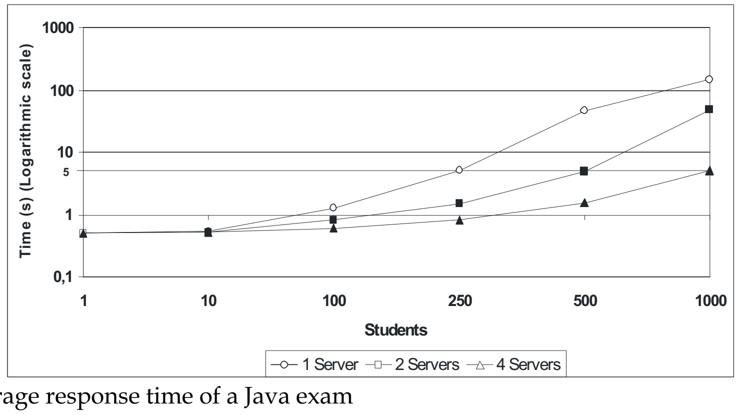


Fig. 7. Average response time of a Java exam

With only one student, the response time is 0,494 seconds. When we increase the number of students, we can observe that response time increases.
We want to obtain a response time not superior than 5 seconds. With only one server, we reach that limit with 250 students, and that limit is really exceeded with 500 students (46,7 seconds). When we use 2 servers, we can support 500 students obtaining a response time approximately of 5 seconds (4.88 seconds). Finally, using 4 servers, we can support up to 1000 students with a response time of 5,005 seconds. In general, analyzing those results we can conclude that our system has a great scalability. In fact we can assume that each one of the used server here can support up to 250 students simultaneously.

4. Conclusions and Future Work

In this paper we have presented an intelligent adaptive tutor for allowing students to overcome learning difficulties of programming languages. Moreover, the proposed system adapts itself dynamically to each student learning style. The tutor can be easily customized to deal not only with different programming languages but also with different approaches to teach them.
The evaluation includes a scalability and performance test that was made by simulating the learning style of several students while they were learning several programming languages using the intelligent tutor.
As future works we plan upgrade M-PLAT to support more programming languages like C, C++, Lisp, Pascal, etc. Finally a new feature that we have in mind is to provide a non interactive automatic correction system for a great number of students, which will be notified via e-mail with the corresponding qualification.

5. References

Aguilar, G & Kaijiri, K. (2002). Personalization Approach of a Web Based Java Programming Tutorial, Proceedings of Computer and Advanced Technology in Education, ISBN: 0-88986-332-6, Cancun (México), May, 2002.

Aase, Magnus & Kurfess, Franz. (2004). Utilizing Learning Styles for Interactive Tutorials, Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04), pp. 828-830, ISBN: 0-7695-2181-9, 2004, September.

J. Woods, Pamela & R. Warren, James. (1995). Rapid Prototyping of an Intelligent Tutorial System. http://www.ascilite.org.au/conference/melbourne95/smtu/papers/ woods.pdf. 1995.

K. Proulx, Viera. (2000). Programming Patterns and Design Patterns in the Introductory Computer Science Course, SIGCSE Bulletin: Conference Proceedings of the Thirty First SIGCSE Symposium on Computer Science Education, Vol. 32, No. 1, 2000, pp. 80-84, ISSN:0097-8418.

Lewis Johnson, William. (1990). Understanding and Debugging Novice Programs, Artificial Intelligence and Learning Environments, Vol. 42, No 1, 1990, pp. 51-97, ISSN 0004-3702

Park Woolf, Beverly; Beck, Joseph; Elliot, Christopher & Stern, Mia. (2001). Growth and maturity of intelligent tutoring systems: a status report, In: Smart Machines in Education, AAAI Press/The MIT Press, pp. 99-144, ISBN:0-262-56141-7.

Pillay, Nelishia. (2003). Developing Intelligent Programming Tutors for Novice Programmers, ACM SIGCSE Bulletin. Vol. 35 , No 2, June, 2003.

S. R, Alpert & M. K, Singley & J. M, Carroll. (1995). Multiple Multimodal Mentors, Delivering Computer-Based Instruction via Specialized Anthropomorphic Advisors, Behaviour and Information Technology, Vol. 14, No. 2, pp. 69-79, Taylor & Francis, Ltd, 1995.

Ueno H., A Generalized Knowledge-Based Approach to Comprehend Pascal and C Programs, IEICE Transactions on Information Systems, Vol. E81-D, No.12, pp.1323-1329, IOS Press, 2000.

Valdés Salmerón, Verónica. (1999). Estilos de Aprendizaje, Manual del ITESM, Campus Chiapas. 1999.

Xu S. & Y. S., Chee (1999). SIPLES-II: An Automatic Program Diagnoses System for Programming Learning Environments, Proceedings of AI-ED 99: 9th International Conference on Artificial Intelligence in Education, pp. 397 – 404, Le Mans, France, July 1999, IOS Press.

Y. S, Chee & S. Xu. (1997). SIPLeS: Supporting Intermediate Smalltalk Programming Through Goal-based Learning Scenarios, Proceedings of AI-ED 97: 8th World Conference on Artificial Intelligence in Education, pp. 95-102, Kobe, Japan, 1997.