Journal of Universal Computer Science, vol. 15, no. 7 (2009), 1472-1485
submitted:28/11/08, accepted: 28/3/09, appeared: 1/4/09 © J.UCS

Learning to Program with COALA, a Distributed Computer Assisted Environment

Francisco Jurado, Ana I. Molina, Miguel A. Redondo, Manuel Ortega
(University of Castilla-La Mancha, Computer Science and Engineering Faculty
Paseo de la Universidad 4 13071 Ciudad Real, Spain
{Francisco.Jurado, AnaIsabel.Molina, Miguel.Redondo, Manuel.Ortega}@uclm.es)

Adam Giemza, Lars Bollen, H. Ulrich Hoppe
(Universität Duisburg-Essen, COLLIDE Research Group
Dept. of Computational and Applied Cognitive Sciences
Lotharstr. 63/65, 47057 Duisburg, Germany
{giemza, bollen, hoppe}@collide.info)

Abstract: Learning to program is an important subject for students of Computer Science. Mentoring these students is a time-consuming and complex task. In this paper, we present a learning and tutoring environment that integrates task/solution delivery, assessment support and tutor's annotations, by extending Eclipse to a "Real World Integrated Development Environment". We will present a distributed system that uses Tuple Space architecture to integrate Eclipse with an evaluation module and a hand-writing annotation feature.

Keywords: Learning Programming, Intelligent tutoring system
Categories: D.2.4, D.2.5, D.2.6, D.2.7, D.2.8., I.2.4., L.0.0, L.2.0

1 Introduction

Learning programming is an essential part of Computer Science studies. However, it is not well suited for (only) being taught in lectures. It is widely accepted that learning to program requires a "learning by doing" approach [Kumar, 03] [Scholemeyer, 96], typically in the form of programming labs that provide an active learning [McConnell, 96]. In this paper, we focus on early stages of programming learning in which the design and encoding of basic algorithms constitute the central individual activity (as opposed to larger group programming projects). Our focus in this paper is, primarily, on enriching and supporting the interaction between the learner-programmer and a teacher or tutor. In this target situation, students must overcome some typical difficulties for acquiring programming related knowledge [du Boulay, 89], [Brusilovsky, 98], [Gomes, 07]. Our overall goal is to build learning support systems that allow teachers and students to overcome the known difficulties of programming courses.

Several systems and approaches have been proposed in order to support the acquisition of programming knowledge and skills [Kelleher, 05] [Garner, 03]. Among the approaches, we will centre our attention on those systems that assist the students with any kind of adaption and feedback allowing constructivist learning experiences [Ben-Ari, 2001].

Thus, it is worth highlighting the student adaption provided by electronic books such as ELM-ART [Brusilovsky, 96] for learning LISP and KBS-Hyperbook [Nejdl, 99] for learning Java. In general, these systems offer a guided navigation through given didactic material, trying to adapt this navigation to each student's individual profile. In this sense, they are adaptive electronic books. The knowledge acquired by the students is assessed by questionnaires, quizzes or tests from programs developed by the students as solutions to assignments. In this way, ELM-ART shows examples to students allowing them to modify, to debug and to execute the LISP programs in its evaluator web interface. This evaluator provides assessment and explanation for mistakes at runtime. However, the kind of programming environment provided through this web interface is limited and allows students to work only with simple code.

Another online problem-solving approach is the work presented in [Kumar, 04] [Fernandes, 05]. The work presents *problets*, a set of tutors that provides visualisation and animation on several programming C++ topics such as expression evaluation, loops, encapsulation, pointers, parameters passing and scope concepts. Authors assert that *problets* provide detailed feedback to students, so they can be used as a supplement to classroom instruction. There is a *problet* for each topic, and each *problet* is a particular simulation environment.

In order to allow students to write more complex code, the work presented in [Pérez, 06] provides a web interface for a Java development environment. The work is focused on the analysis of code written by the students. It allows detecting, removing and preventing mistakes by using language processing techniques. To do so, the system logs, stores and analyses the errors. Thus, programmers can learn from their own mistakes and can avoid making the same mistakes in the future. Although the web-oriented embedment allows a system-independent execution that requires only a web browser, it is not a distributed application that would facilitate reusability and interoperability with other learning tools.

On the other hand, apart from feedback provided by the system, it can also come directly from the teacher or from other students, by following the Computer Supported Collaborative Learning (CSCL) paradigm [Koschman, 96]. Therefore, in [Redondo, 04], the authors show the integration of several tools for learning to program in a distributed way in specific domain contexts by applying CSCL. Instead of a development environment, this work uses visual tools for learning both structured programming and object oriented programming, and it introduces collaborative planning [Redondo, 02] as a way to support CSCL.

Also using the CSCL paradigm, in [Duque, 08] we can see a Java development environment that allows students to write, compile and execute complex Java code in a synchronous way. The tool uses a structured chat, context awareness, and coordination mechanisms in a Java application launched through a web browser. The tool provides support to allow experts to analyse the collaborative work productivity, and some quality measures about the code the students have written. However, this environment leaves the actual assessment of the code to the evaluators (teachers), and does not provide an automatic analysis of the code written by the students.

Searching for a system that provides feedback to the students who are learning programming, we have seen that not all the proposals use development environments that allow working with complex code. Furthermore, we have shown that there are approaches that allow for the integration of several tools in a distributed way, and other approaches that propose the analysis of solutions developed by the students. However, there is not one single tool that would support all these things together. So, the suggestion presented in this paper is focused on developing a distributed environment and a set of software facilities/tools for learning initial algorithmic programming with specific support for the teacher-learner interaction and the ability to analyse the solutions delivered by students. With this, we do not expect to create a system that substitutes the teacher, but rather to implement an approximation that provides support in a traditional classroom [Schofield, 94]. The suggested architecture is based on a central blackboard for data sharing, and the use of agents interacting with this blackboard to provide intelligent analysis and supervision support.

The rest of the paper is structured as follows: Firstly, we will specify a computer-enhanced scenario for learning algorithmic programming (section 2); then, we will describe the communication architecture and other implementation issues of the proposed environment (section 3); after this, our scenario in action will be explained (section 4) to show the details and possibilities of the system; and finally, some concluding remarks will be extracted and future perspectives will be discussed (section 5).

2 Our Computer Assisted Environment for Learning Algorithms

We are now going to describe a typical "programming lab" scenario in academic education. We propose to enrich this scenario by means of a distributed computing support [Molina, 05] [Paredes, 08]. The sequence of steps to be carried out is the following: At first, the teacher specifies programming assignments for students and sends these to a server. Next, the students download the assignment from the server and work it out individually. This environment could include an intelligent module for evaluation of the students' proposals. So, during their programming, the students can ask this system for an automatic evaluation to check their solution. After completing the solution, the students send their results to the server. All the time, the teacher will be notified about the students' actions and can see the code sent by the students on his/her computer.

Furthermore, if the teacher has a device that allows pen-based input, such as a tabletPC or an electronic whiteboard, he/she can use the handwriting feature to annotate the code. This will allow for more natural interaction during the evaluation process. Furthermore, if the teacher considers showing concrete code to the students in class, these annotations can also be very useful when using an electronic whiteboard.

In short, the described scenario is a distributed environment that merges communication and notification capabilities, auto-assessment provided by the system, and free handwriting annotations features. We will discuss all these issues in detail in next section.

3 Implementation Issues

In this section, we will explain the two main implementation issues taken into account in our system. Firstly, the communication architecture that will allow us to create a heterogeneous distributed system by means of the *ad-hoc* interconnection of several services according to the growth system. Secondly, we will show the environment we have chosen where the students will perform the learning activities.

3.1 Blackboard Architecture Using SQLSpaces

In the design and implementation of distributed and possibly collaborative learning applications, one crucial issue is the choice of communication and synchronisation architecture. The principle distinction is between sharing data (e.g. through a central database) and synchronising processes (e.g. using remote method call mechanisms, as described in [Jurado, 07a]), as a basic starting point.

Our target scenario involves the distribution of programming tasks or assignments to groups of students, the flexible (asynchronous) downloading of such assignments, local elaboration, upload, correction, annotation and feedback from the teacher/tutor to the student. That is, the basic activities are typically asynchronous, but having a shared pool of data and notifications would be highly desirable. Indeed, both requirements are met by a blackboard architecture based on Tuple Spaces. Although the original idea is already quite old (see below), Tuple Spaces have recently been used in several implementations of collaborative distributed environments, such as the Group Scribbles classroom environment [Brecht, 06] or the Amenities project focusing on group coordination [Garrido, 06].

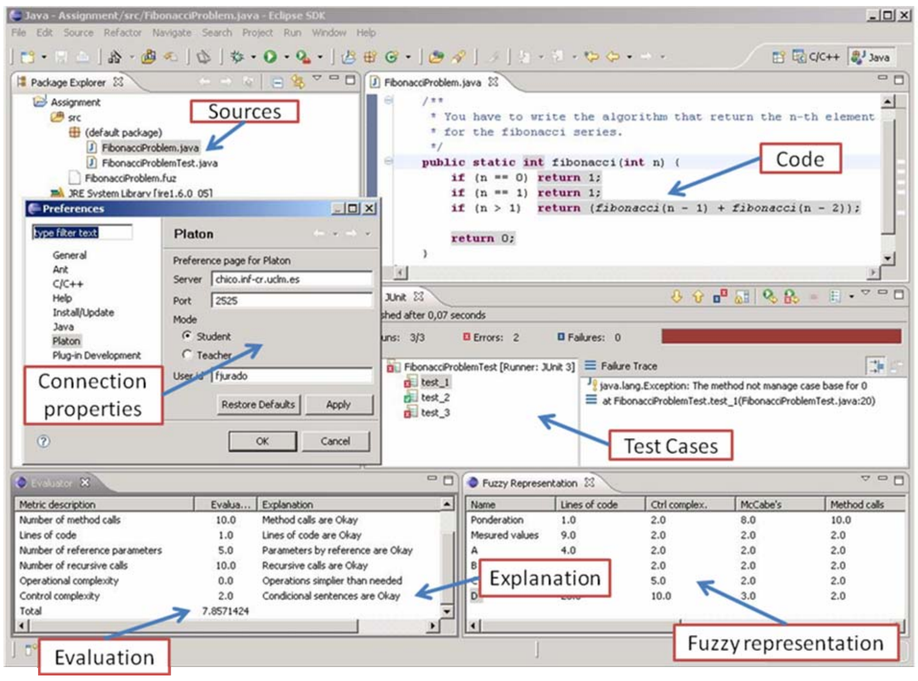
The Tuple Space approach as an implementation of the blackboard architecture introduced, together with the coordination language Linda, by Gelernter [Gelernter, 85] in the 1980s. It is based on a central server, which holds all messages. The clients exchange messages solely with the server and do not have any direct client-to-client connections. So, the server can be seen as a tuple exchange place or shared working memory. Clients communicate indirectly by writing and reading (or “taking”) tuples to/from the blackboard. A widely available recent implementation of Tuple Spaces came as part of the Jini framework under the name of JavaSpaces (Sun Microsystems, 1998). JavaSpaces also provides “leases” to manage the lifetime of Tuple Space entries, and an event mechanism that can actively notify clients. A JavaSpaces server is not just one tuple container, but rather consists of several disjoint spaces, which can be addressed by different names. This is particularly useful if different types of agents working on different levels are to be supported. Almost simultaneously, another Java-based Tuple Space implementation called TSpaces [Lehman, 99] has been developed and distributed by IBM’s Almaden Research Center.

For our project, we have used a Tuple Space implementation called SQLSpaces developed at the University of Duisburg-Essen [Giemza, 07]. SQLSpaces support all essential features including notification and lifetime management. An outstanding feature of the SQLSpaces is the support for multiple programming languages. SQLSpaces comes with predefined clients for Ruby, C#, PHP and Prolog in addition to the host language of the server; Java. Additionally, SQLSpaces provide Web Service access that makes them usable with any kind of client language with support for Web Services. Thus, SQLSpaces can also be seen as a “language switch board” which enables communication in heterogeneous programming language environments.

3.2 Environment: Customizing Eclipse

To reach our aim of creating an environment suitable for learning programming, it is essential to use an environment that is not so different from the one that students will find in their future work. That is, not to use virtual environments or simulation tools, but employ a real-world Integrated Development Environment (IDE).

Thus, to further develop our approach we have selected the widely available Eclipse platform [Eclipse]. Eclipse is an integrated development environment, which allows creating extensions by using its own API. Such extensions are implemented as plug-ins that can be optionally loaded by users. Eclipse is a full-fledged development environment that works with Java, C/C++ and other programming languages.



```
/**
 * You have to write the algorithm that return the n-th element
 * for the fibonacci series.
 */
public static int fibonacci(int n) {
    if (n == 0) return 1;
    if (n == 1) return 1;
    if (n > 1)  return (fibonacci(n - 1) + fibonacci(n - 2));

    return 0;
}
```

Platon

Preference page for Platon

Server: chico.inf-cr.uclm.es

Port: 2525

Mode: [✓ Student] Teacher

User id: fjurado

```
shed after 0,07 seconds
uns: 3/3      Errors: 2      Failures: 0

FibonacciProblemTest [Runner: JUnit 3]
    test_1
    test_2
    test_3

Failure Trace
java.lang.Exception: The method not manage case base for 0
at FibonacciProblemTest.test_1(FibonacciProblemTest.java:20)
```

Metric description	Evalua...	Explanation
Number of method calls	10.0	Method calls are Okay
Lines of code	1.0	Lines of code are Okay
Number of reference parameters	5.0	Parameters by reference are Okay
Number of recursive calls	10.0	Recursive calls are Okay
Operational complexity	0.0	Operations simplier than needed
Control complexity	2.0	Condicional sentences are Okay
Total	7.8571424	

Name	Lines of code	Ctrl complex.	McCabe's	Method calls
Ponderation	1.0	2.0	8.0	10.0
Mesured values	9.0	2.0	2.0	2.0
A	4.0	2.0	2.0	2.0
B	[illegible]	2.0	2.0	2.0
C	[illegible]	5.0	2.0	2.0
D	[illegible]	10.0	3.0	2.0

Figure 1: Customized Eclipse Environment

Figure 1 shows a screenshot of the developed plug-in, once integrated into Eclipse. The figure shows the appearance of the user interface that supports student activities. It displays the properties page where the server location, port, user role and user ID can be set. Furthermore, it shows the code written by the students, the test cases they can execute, and the evaluation and explanation the system can give about the algorithm they have written as a solution to an assignment. In section 5, we will show different parts of this figure in more detail.

As a basic means of communication, the plug-in for the Eclipse environment allows communication with the SQLSpaces server. In the same way, we have implemented capabilities to allow free hand annotations over code.

Details about the evaluation process and annotation facilities supported by the tool are presented in the following sections.

4 Assessment Using Fuzzy Logic and Test Cases

To overcome the first difficulties the students may encounter while developing their solution, we present an architecture that provides a first assessment of the students' solution. [Ben-Ari, 01] [Traynor, 06]. This allows students to ask the system what is wrong with the solution they are developing, without teacher intervention. With this, our aim is to create a system that assists the students in understanding what they are doing and help the teachers in their labour in the classroom.

There are a lot of approaches about how to assess programming learning activities and a good survey about static and dynamic assessment of computer programs can be found in [Ala-Mutka, 05].

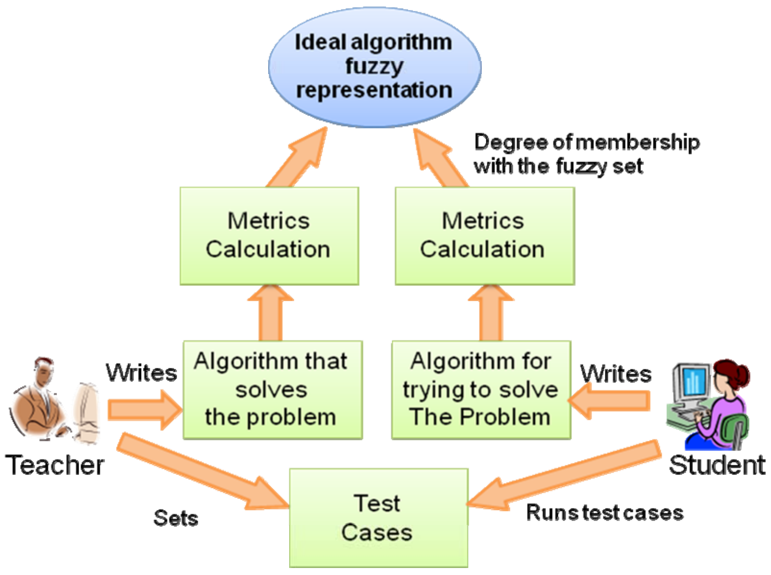

Ideal algorithm fuzzy representation
Degree of membership with the fuzzy set
Metrics Calculation
Metrics Calculation
Writes
Algorithm that solves the problem
Algorithm for trying to solve The Problem
Writes
Teacher
Student
Sets
Test Cases
Runs test cases

Figure 2: Evaluating the Student Algorithm.

We are going to briefly describe the evaluation process we have implemented and which is shown in depth in [Jurado, 07b]. Firstly, the teacher writes an implementation for the ideal approximate algorithm that solves a problem (at the bottom left of figure 2). Next, several software metrics that shape its structure will be calculated. Thus, we obtain an instance of the ideal approximated algorithm. Then, a fuzzy set for each metric will be established in the following way: initially, each fuzzy set will be a default trapezoidal function around the metric value from the approximate algorithm; the teacher can easily modify the fuzzy set indicating:

- The maximum value that the teacher considers low for the solution.
- The minimum value for correctness.
- The maximum value for correctness.
- The minimum value that the teacher considers high for the solution.

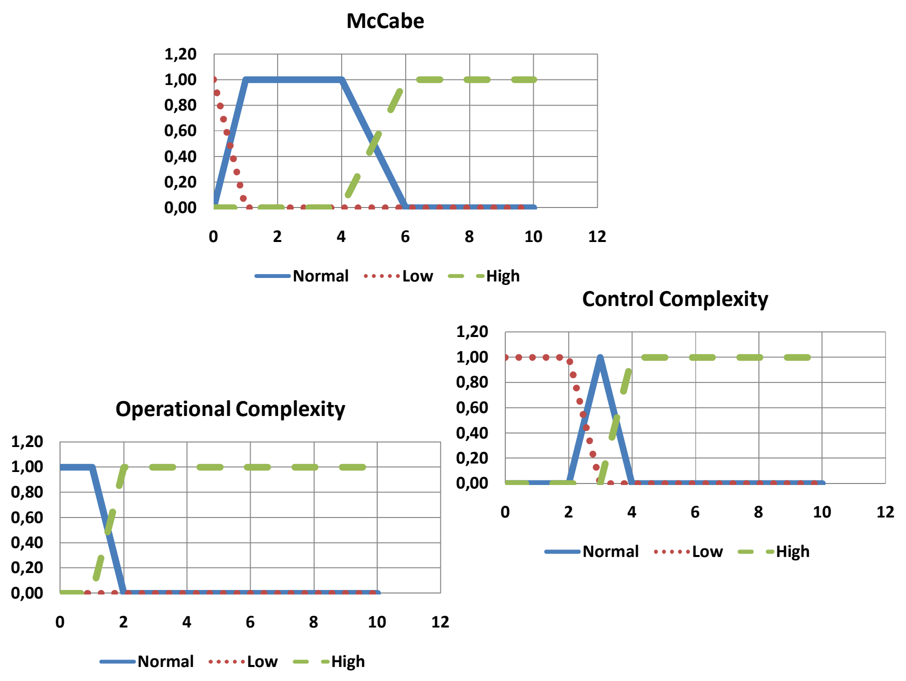


Figure 3: Fuzzy Sets for Some Calculated Software Metrics.

In this way, we obtain a collection of fuzzy sets that characterises the algorithm and allows us to know when a measured software metric extracted from an algorithm can be considered as normal, low or high and to what degree. In figure 3, we can see the “normal”, “low” and “high” fuzzy sets. For each graphic, the x axis represents the values we can obtain for a software metric and the y axis shows the membership degree for the measure of the software metric for each fuzzy set. For instance, if the measured value for the McCabe Cyclomatic Complexity (at the top of figure 3) is 4, we obtain that the membership value to the “normal” fuzzy set is 1, and 0 to the “high” and “low” fuzzy sets. So, in that case, we can say that the value for the McCabe Cyclomatic Complexity is what the teacher considers “normal”. As we increase the value for the metric, the membership value for the “normal” fuzzy set reduces and the membership value for the “high” fuzzy set increases. If we look at value 5 of the metric, we can see that the membership value for the “normal” and “high” fuzzy sets is equal to 0.5 in both cases, that is, the metric could be considered "normal" and "high" in the same degree, but not "low". In that case, we can say something like: the measured metric is "a bit high". Moreover, if the value for the metric continues growing, it will come to a state where the membership value for the "normal" and "low" fuzzy sets will be 0, and 1 for the "high" fuzzy set. So we can say that the measured value is "high" according to the teacher.

Thus, we get a fuzzy representation of that ideal approximated algorithm, that is, we obtain an ideal approximated algorithm fuzzy representation that solves a concrete problem (at the top of figure 2).

Algorithms that students have written (on the right of figure 2) will be correct if they are instances of that ideal algorithm fuzzy representation. Knowing the degree of membership for each software metric obtained from the algorithm written by students in the corresponding fuzzy set for the ideal approximated algorithm fuzzy representation, will give us an idea of the quality of the algorithm that the students have developed.

Furthermore, taking into account the fuzzy nature of the process, some fuzzy rules have been defined to provide messages as feedback to the students, related to their evaluations. These sentences are presented in natural language and provide the learners with useful information. So, for example, if the system detects a discrepancy because the McCabe's Cyclomatic Complexity is "high", then the system can give advice with the message "The algorithm has more bifurcations than needed".

Moreover, as we can see on the bottom of figure 2, the teacher writes some test cases that the students can run to test their algorithms. With the fuzzy evaluation process that allows analysing the code structure and the test cases that allow checking if the algorithm solves the problem, we have a complete tool that can help the students quite a lot in the initial stage of solving an assignment.

So, firstly, the implemented module receives the fuzzy representation the teacher has specified. Then, when the student sends the code that solves the assignment, the module analyses it and, finally, it provides an evaluation and an explanation about what is wrong in the code.

In our first studies of this technique [Jurado, 07c], we analysed the effectiveness of the proposal by contrasting the evaluation the teacher has carried out on some students' assignments with the automatic evaluation for the same assignments. In that study, we obtained the following results with evaluation between 0 and 5:

- in 52.17% of the cases, the evaluations were the same;
- in 34.78% of the cases, the evaluations differ by one point;
- in 13.04% of the cases, the evaluations differ by more than one point.

These results encourage us to work on that line, analysing the code by using fuzzy logic. Moreover, this technique can be extrapolated to other programming areas such as analysing assignments on Object Oriented Programming learning, by simply changing the software metrics to be used for the evaluation.

5 COALA in Action

As we have mentioned in section 2, we envisage a typical "programming lab" scenario in academic education. We have enhanced this scenario with distributed computer support, an automatic evaluation module and a set of plug-ins that provide an enriched IDE for supporting teachers' and students' activities. The resulting environment and architecture has been called COALA, the acronym of "COmputer Assisted Environment for Learning Algorithms". In this section, we will explain in detail the way in which this system works.

Following our explanation, figure 4 shows the different steps and messages (tuples) between the teacher, the students, the SQLSpaces server and other software modules such as the evaluator module.

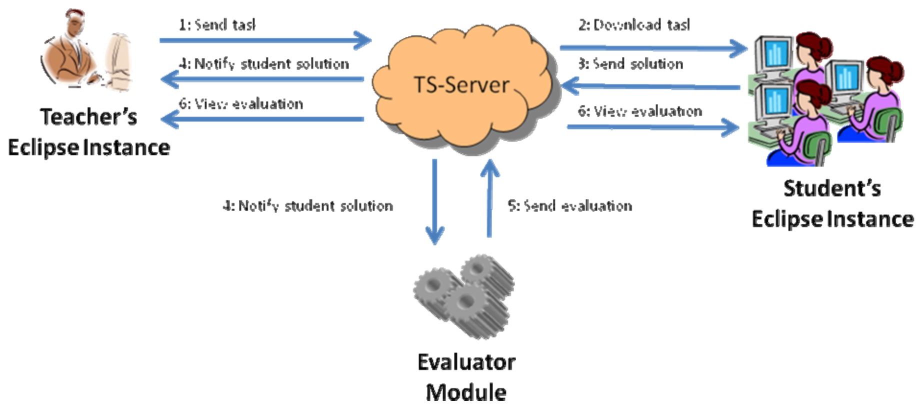
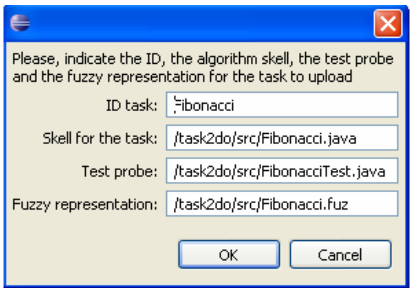


Figure 4: Message Passing among Components and the Tuple Space.

As we can see in figure 4, at the beginning, the teacher specifies an assignment using his/her Eclipse environment. In our case, an assignment consists of:

- A template for a Java class and methods to be implemented by the student containing the task description.
- A JUnit test class so that the students can test their solutions.
- A fuzzy representation of the algorithm that solves a problem. This fuzzy representation was the obtained by the teacher from his/her ideal solution and it is the one the system will use to provide an auto-evaluation to help the students understand what might go wrong.



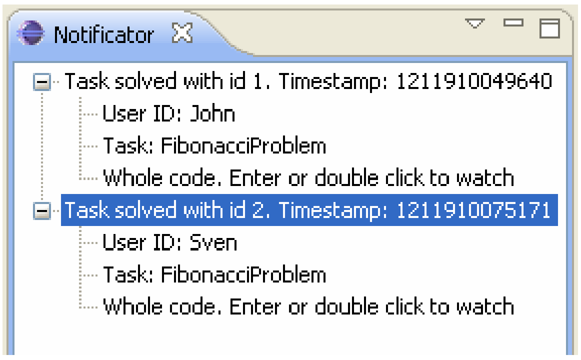
Please, indicate the ID, the algorithm skell, the test probe and the fuzzy representation for the task to upload

ID task:	Fibonacci
Skell for the task:	/task2do/src/Fibonacci.java
Test probe:	/task2do/src/FibonacciTest.java
Fuzzy representation:	/task2do/src/Fibonacci.fuz

Figure 5: Files to Upload for a Task.

After this, the teacher uploads the template, the test cases and the fuzzy representation to the server by sending a tuple with the form <task_id; template; test_cases; fuzzy_representation> (step 1) using the “Send Task to TS” action in the plug-in. At that moment, the task is available for all the students in the classroom. Figure 5 shows the dialogue for the teacher to select the template, the test cases and the fuzzy representation.

Now the students are able to download the assignment onto their workspace reading the tuple uploaded by the teacher (step 2), using their “Download Task from TS” action menu in the plug-in. Then, each student can work out the task by writing the code, compiling, etc.

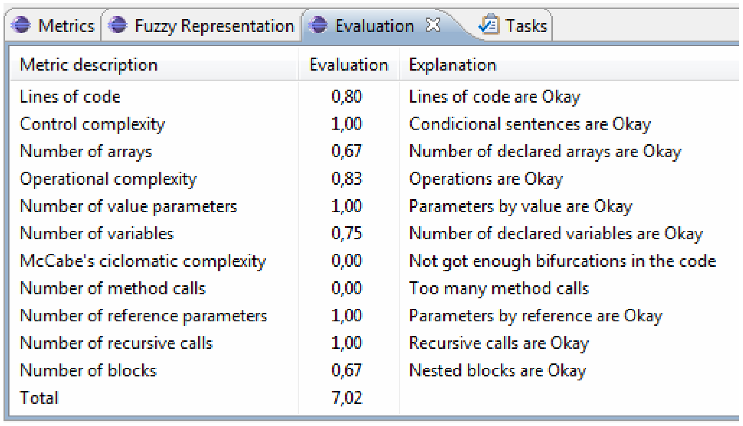


```
Task solved with id 1. Timestamp: 1211910049640
    User ID: John
    Task: FibonacciProblem
    Whole code. Enter or double click to watch
Task solved with id 2. Timestamp: 1211910075171
    User ID: Sven
    Task: FibonacciProblem
    Whole code. Enter or double click to watch
```

Figure 6: Notifications Received by the Teacher.

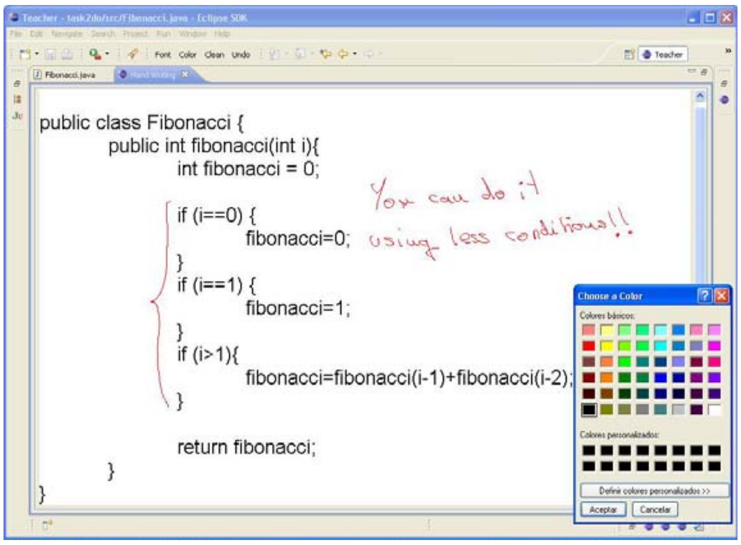
Once the students have finished the assignment, they can send their results to the server and, from there they can be downloaded and reviewed by the teacher. Thus, students upload the solution to the server sending a tuple with the following content: <user_id; task_id; solution_code> (step 3). The teacher will be notified about the task sent and can check the code written by the student on his/her computer, reading all the tuples with the form <user_id; task_id; solution_code> from the server (step 4). Figure 6 shows the view where the teacher can see the notifications received from the tuple spaces server. In the figure, we can see how the notifications are shown as a tree in which each branch is a notification that has the ID of the student who sent the task, the ID of the task, and a message that requests double clicking to see the whole code sent.

The architecture proposed allows other software to interact with the system by reading and writing tuples from/in the Tuple Space server and to use them for other purposes. So, we have implemented an evaluator module that reads the tuples the students have sent; that is, the same tuples the teacher reads (step 4), and process the code to obtain a set of metrics and an evaluation explanation (in the way presented in section 3.3). These calculated metrics are sent to the Tuple Space server in the form <task_id; user_id; metric1; metric2; ... metricN>. Also, an explanation associated with each metric is sent in a tuple in the following format: <task_id; user_id; explain_ metric1; explain _metric2; ... explain _metricN> (step 5). Then, both the teacher and the students can read the software metrics and the corresponding explanations from the server and analyse them (step 6). So during their programming, students can use the tests created by the teacher as well as asking the system for an automatic evaluation to check their solution. Figure 7 shows the evaluation (in the central column) and explanation (in the right column) of an algorithm developed by a student.



Metric description	Evaluation	Explanation
Lines of code	0,80	Lines of code are Okay
Control complexity	1,00	Condicional sentences are Okay
Number of arrays	0,67	Number of declared arrays are Okay
Operational complexity	0,83	Operations are Okay
Number of value parameters	1,00	Parameters by value are Okay
Number of variables	0,75	Number of declared variables are Okay
McCabe's ciclomatic complexity	0,00	Not got enough bifurcations in the code
Number of method calls	0,00	Too many method calls
Number of reference parameters	1,00	Parameters by reference are Okay
Number of recursive calls	1,00	Recursive calls are Okay
Number of blocks	0,67	Nested blocks are Okay
Total	7,02	

Figure 7: Assessment Loaded in within the Environment



```
public class Fibonacci {
        public int fibonacci(int i){
                int fibonacci = 0;

                if (i==0) {
                        fibonacci=0;
                }
                if (i==1) {
                        fibonacci=1;
                }
                if (i>1){
                        fibonacci=fibonacci(i-1)+fibonacci(i-2);
                }

                return fibonacci;
        }
}
```

You can do it using less conditionals!!

Figure 8: Free Handwriting Annotations over the Code.

As we have mentioned before, to allow the teacher to have more natural interaction in the evaluation process, we have implemented a handwriting feature. This feature lets the teacher annotate the code by using a digital tablet, a TabletPC or an electronic whiteboard. Thus, figure 8 shows a screenshot with the free handwriting annotations over the code that the teacher can do. This will allow the possibility of creating pedagogically interesting computer enhanced scenarios. We have added this feature, implementing a new Eclipse editor that allows capturing the corresponding pen-events that a teacher can make over the code to create annotations.

6 Concluding Remarks and Future Work

In this article, we have presented a distributed system that allows creating computer-enhanced scenarios for learning initial algorithmic programming. The system uses a particular implementation of Tuple Spaces (SQLSpaces) as an engine for communication and data sharing, and it includes handwriting features to allow for more natural and flexible interaction. Also, an automatic evaluation module based on the use of software metrics, test cases and algorithm fuzzy representation is included in the overall environment, called COALA.

Testing the system in the described scenarios will give us the necessary feedback to improve the application and create more complex scenarios that integrate other devices and software modules. Thus, it will allow for the improvement of the quality of the learning/teaching process in computer programming. The approach is likely to be transferable to other subjects, including formal exercises.

Acknowledgements

This research work has been partially supported by the Junta de Comunidades of Castilla-La Mancha, Spain through the projects AULA-T (PBI08-0069) and M-GUIDE (TC20080552) and Ministerio de Educación y Ciencia, Spain through the project E-Club-II (TIN-2005-08945-C06-04).

References

[Ala-Mutka, 05] Ala-Mutka, K.: A Survey of Automated Assessment Approaches for Programming Assignments, in Computer Science Education, vol. 15, num. 2, Routledge, part of the Taylor & Francis Group, pp. 83-102. 2005

[du Boulay, 89] du Boulay, B.: Studying The Novice Programmer, Lawrence Erlbaum Associates, Chapter: Some Difficulties Of Learning To Program, pp. 283-300, 1989

[Ben-Ari, 01] Ben-Ari, M.: Constructivism in Computer Science Education, *in* Journal of Computers in Mathematics and Science Teaching Vol. 20, Association for the Advancement of Computing in Education, pp. 45—73, 2001

[Brecht, 06] Brecht, J., DiGiano, C., Patton, C., Tatar, D., Chaudhury, S. R., Roschelle, J., Davis, K.: Coordinating Networked Learning Activities with a General-purpose Interface, *in* Proceedings of the 5th World Conference on Mobile Learning, Banff, Canada, 2006

[Brusilovsky, 96] Brusilovsky, P.; Schwarz, E.W., Weber, G. ELM-ART: An Intelligent Tutoring System on World Wide Web, *in* Intelligent Tutoring Systems, pp. 261-269, 1996

[Brusilovsky, 98] Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., Miller, P.: Mini-languages: A Way to Learn Programming Principles, *in* Education and Information Technologies, vol 2, 65 -83, 1998

[Duque, 08] Duque, R., Bravo, C.: Analyzing Work Productivity and Program Quality in Collaborative Programming, *in* 'ICSEA '08: Proceedings of the 2008 The Third International Conference on Software Engineering Advances', IEEE Computer Society, pp. 270-276, 2008

[Eclipse] Eclipse, online http://www.eclipse.org, last visited February 2009

[Fernandes, 05] Fernandes, E., Kumar, A.: A Tutor on Subprogram Implementation, *in* J. Comput. Small Coll. Vol. 20, Consortium for Computing Sciences in Colleges, pp. 36-46, 2005

[Garner, 03] Garner, S.: Learning Resources and Tools to Aid Novices Learn Programming, *in* Informing Science & Information Technology Education Joint Conference (INSITE), pp. 213-222, 2003

[Garrido, 06] Garrido, J.L., Noguera, M., Gonzalez, M., Gea, M., Hurtado, M.V.: Leveraging the Linda Coordination Model for a Groupware Architecture Implementation, *in* Proceedings of 12th International Workshop on Groupware. Springer LNCS 4154. pp. 286-301, 2006

[Gelernter, 85] Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1): 80-112, 1985

[Giemza, 07] Giemza, A., Weinbrenner, S., Engler, J., Hoppe, H.U.: Tuple Spaces as Flexible Integration Platform for Distributed Learning Environments, *in* Proceedings of ICCE 2007, Hiroshima (Japan), November 2007. pp. 313-320, 2007

[Gomes, 07] Gomes, A., Mendes, A. J.: Learning to Program - Difficulties and Solutions, *in* International Conference on Engineering Education – ICEE 2007, pp. 283-287, 2007

[Jurado, 07a] Jurado, F., Redondo, M. A., Ortega, M.: Enabling Distributed eLearning Environments Integrating ICE-based Services, *in* Proceeding of the International Technology, Education and Development Conference INTED2007, Valencia, Spain, pp. 375, 2007

[Jurado, 07b] Jurado, F., Redondo, M. A., Ortega, M.: Fuzzy Algorithm Representation for its Application in Intelligent Tutoring Systems for the Learning of Programming, *in* do Nascimento, R. P., Gerqia, A., Serendero, P. & Carrillo, E. (ed.): EuroAmerican Conference On Telematics and Information Systems, EATIS'07 ACM-DL Proceeding. Faro, (Portugal), pp. 1-8, 2007

[Jurado, 07c] Jurado, F.; Redondo, M. A., Ortega, M.: Applying Approximate Reasoning Techniques for the Assessment of Algorithms in Intelligent Tutoring Systems for Learning Programming (in Spanish), *in* Isabel Fernandez de Castro, (ed.): VII Simposio Nacional de Tecnologías de la Información y las Comunicaciones en la Educación (Sintice'07), Thomson, Zaragoza, Spain, pp. 145-153, 2007

[Kelleher, 05] Kelleher, C., Pausch, R.: Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, *in* ACM Comput. Suv. Vol. 37, ACM, pp. 83-137, 2005

[Koschmann, 96] Koschmann, T.: Paradigm Shifts and Instructional Technology: an Introduction, *in* Koschmann, (ed.), Hillsdale, NJ: Lawrence Erlbaum, pp. 1-24, 1996

[Kumar, 03] Kumar, A.N. Learning Programming by Solving Problems, *in* Informatics Curricula and Teaching Methods (ICTEM). L. Cassel and R.A. Reis ed. Kluwer Academic Publishers. pp. 29-39. Norwell MA, 2003.

[Kumar, 04] Kumar, A.: Using Online Tutors for Learning - What do Students Think?, *in* Proceedings of Frontiers in Education Conference (FIE 2004), IEEE, pp. 524-528, 2004

[Lehman, 99] Lehman, T.J., McLaughry S.W., Wycko P.: T Spaces: The Next Wave, *in* Proceedings of the 32nd Hawaiian International Conference on Computer Systems 1999, HICCS, Maui, Hawaii, pp. 8037, 1999

[McConnell, 96] McConnell, J. J.: Active Learning and its Use in Computer Science, *in* SIGCUE Outlook, vol. 24, ACM, pp. 52-54, 1996

[Molina, 05] Molina, A.; Redondo, M., Ortega, M.: A System to Support Asynchronous Collaborative Learning Tasks Using PDAs, *in* Journal of Universal Computer Science Vol. 11, pp. 1543-1554, 2005

[Nejdl, 99] Nejdl, W, Wolpers, M.: KBS Hyperbook—A Data Driven Information System on the Web, *in* 8th International World Wide Web conference, (WWW8), 1999

[Paredes, 08] Paredes, M.; Molina, A.; Redondo, M., Ortega, M.: Designing Collaborative User Interfaces for Ubiquitous Applications Using CIAM: The AULA Case Study, *in* Journal of Universal Computer Science Vol. 14, pp. 2680-2698, 2008

[Pérez, 06] Pérez, J. R. P.: Classification of Users Based on Detecting Errors Using Techniques Processors Language, PhD thesis, University of Oviedo, 2006

[Redondo, 02] Redondo, M.Á., Bravo, C., Ortega, M., Verdejo, M.F.: PlanEdit: An Adaptive Problem Solving Tool for Design, *in* De Bra, P.; Busilovsky, P. & Conejo, R. (eds.) Adaptive Hypermedia and Adaptive Web-Based Systems', Springer LNCS, Berlin, pp. 29-31, 2002

[Redondo, 04] Redondo, M.A., Bravo, C., Molina, A.; Marcelino, M., Mendes, A.: Tools for Programming Learning: An Approach to Provide a Social Perspective using Collaborative Planning of Design, *in* Proceedings of IADIS International Conference e-Society 2004, Avila (Spain), pp 315-322, 2004

[Sanders, 87] Sanders, D. & Hartman, J.: Assessing the Quality of Programs: A Topic for the CS2 Course, *in* SIGCSE '87: Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education', ACM, pp. 92-96, 1987

[Schofield, 94] Schofield, J. W.; Eurich-Fulcer, R., Britt, C. L.: Teachers, Computer Tutors, and Teaching: The Artificially Intelligent Tutor as an Agent for Classroom Change, *in* American Educational Research Journal vol. 31, pp. 579-607, 1994

[Schollmeyer, 1996] Schollmeyer, M.: Computer Programming in High school vs. College, *in* SIGCSE '96: Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education, ACM, pp. 378-382, 1996

[Traynor, 06] Traynor, D.; Bergin, S., Gibson, J. P.: Automated Assessment in CS1, *in* ACE '06: Proceedings of the 8th Australian Conference on Computing Education, Australian Computer Society, Inc., pp. 223-228, 2006