Journal of Universal Computer Science, vol. 11, no. 9 (2005), 1505-1517
submitted: 15/2/05, accepted: 30/6/05, appeared: 28/9/05 © J.UCS

Integrating Educational Tools for Collaborative Computer Programming Learning

Crescencio Bravo
(Escuela Superior de Informática, Universidad de Castilla – La Mancha, Spain
Crescencio.Bravo@uclm.es)

Maria Jose Marcelino
(Centro de Informática e Sistemas da Universidade de Coimbra, Portugal
zemar@dei.uc.pt)

Anabela Gomes
(Instituto Superior de Engenharia de Coimbra, Portugal
anabela@isec.pt)

Micaela Esteves
(Escola Superior de Tecnologia e Gestão de Leiria, Portugal
micaela@estg.ipleiria.pt)

Antonio Jose Mendes
(Centro de Informática e Sistemas da Universidade de Coimbra, Portugal
toze@dei.uc.pt)

Abstract: Computer Programming learning is a difficult process. Experience has demonstrated that many students find it difficult to use programming languages to write programs that solve problems. In this paper we describe several educational computer tools used successfully to support Programming learning and we present a global environment which integrates them, allowing a broader approach to Programming teaching and learning. This environment uses program animation and the Computer-Supported Collaborative Learning (CSCL) paradigm.

Keywords: Collaborative Programming, Computer Programming Teaching and Learning, Program Animation and Simulation
Categories: K.3.2

1 Introduction

Computer Programming learning is a difficult process. To become a good programmer, a student must develop several skills that go well beyond knowing the syntax of a programming language. Experience has shown that most difficulties arise from students' low capacity to develop algorithms that solve problems effectively. This is mostly due to the lack of a suitable mental model. These difficulties are independent from the programming language or programming paradigm used. With the objective of improving Programming teaching and learning we have developed several educational tools that take advantage of program animation and visualization and also collaboration between students during program development [Bravo, 2004;

Esteves, 2004; Gomes, 2001; Redondo, 2003].

The dynamic nature of programs suggests that their operations and interactions are, in general, better described by means of dynamic visual representations. A program animation tool allows the visualization of dynamic graphical representations of program execution. Some authors [Levy, 2003] confirm the effectiveness of program animation when it is integrated in long term teaching experiences. The underlying idea is to facilitate students' work, allowing them to interact in the first learning stages with visual representations of algorithms instead of with C or Java code (or any other programming language).

The materialization of Collaborative Learning using computer environments results in the CSCL paradigm [Koschmann, 1996a]. In collaborative environments, the dialogue among users during their activity, the joint work and the resulting product are the elements that sustain, promote and cause learning. In particular, Real Time Collaborative Programming allows geographically distributed students to work concurrently and collaboratively in the same programming task in order to design, code, debug, test and document [Shen, 2000]. Previous studies [Nosek, 1998; Williams, 2000] indicate that Collaborative Programming not only accelerates problem resolution processes, but substantially improves the quality of the software products that are built.

In this paper we describe how different educational tools used independently for Programming learning have been integrated from the technological point of view. They are based on program simulation and animation, planning, and on collaboration among the users. Each tool operates at a different level and has interest and utility by itself. In this integration, technologies such as XML[1], JMS[2] and JSDT[3] are used. XML offers the possibility to exchange structured data among applications. JMS is a messenger API for Java that allows components of applications to create, send, receive and manage messages. JSDT is a toolkit especially suitable for the development of distributed synchronous collaborative applications.

This paper is organized as follows: section 2 describes the tools we have developed to support different stages of Programming learning; section 3 outlines a global methodology for Programming learning based on the integration of those tools previously described; and finally, the conclusions and implications of this work are presented.

2 Software Tools for Programming Teaching and Learning

Software tools that allow the interactive development and testing of programs are a useful complement to the theoretical contents in Programming teaching and learning. In the following sections we describe some tools we have developed in the framework of our research in this area.

---

[1] Extensible Markup Language: http://www.w3.org/XML/

[2] Java Message Service: http://java.sun.com/products/jms/

[3] Java Shared Data Toolkit: http://java.sun.com/products/java-media/jsdt/

2.1 SICAS

SICAS (Interactive System for Algorithm Development and Simulation) [Gomes, 2001] is a system designed to support learning of the basic concepts of procedural programming. Its main objective is the development of problem solving skills, namely in the utilization of control structures and subprograms to solve problems. This system includes support to two different activities:

- Design/Edition of solutions (algorithms) to teacher proposed problems. The algorithm is specified using a visual representation, where graphical symbols that represent the algorithms building blocks (selections, repetitions, procedures…) are used. This representation is independent of any programming language that may be used in the course, allowing students to focus on the algorithm design and not on any specific language syntax. It is also useful to convey the idea that a well designed algorithm can be implemented in several programming languages without a significant effort.
- Execution/Simulation of solutions. The student can verify how his/her algorithm works. SICAS simulates the algorithm and shows its results using animation. Students can analyze how the algorithm behaves in detail and at their own pace, identifying and correcting eventual errors.

SICAS does not include theoretical contents, but it consists of an environment of experimentation and discovery, which enables the detection of errors, their correction and the learning based on these activities. In our opinion, these activities improve problem solving skills resulting in the ability to build programs. In several tests we confirmed that the students who used SICAS built better algorithms and made them faster than those who did not use SICAS [Rebelo, 2005].

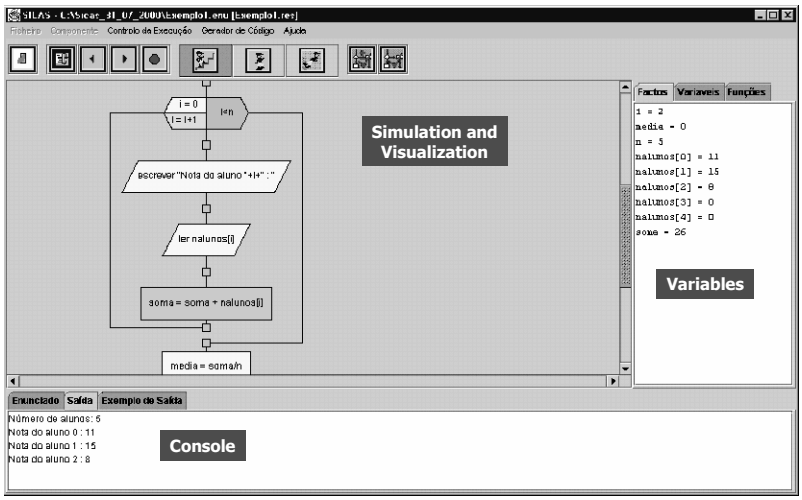

Figure 1: A SICAS animation session

Once a valid solution has been designed, SICAS allows the automatic generation of code in the form of pseudo-code, C language or Java language. Fig. 1 is a screenshot of SICAS. The main area is the visualization and simulation area (center), which contains the algorithm representation. The variable list (with their values) can be seen on the right and the output console at the bottom.

2.2 PlanEdit

DomoSim-TPC is a collaborative environment for the distance learning of domotical design by means of problem solving activities. This environment incorporates a tool called PlanEdit [Redondo, 2002], which is used to build an abstract solution to a problem by means of the planning of its design.

This tool is also being used to support Programming learning. We consider a program as a set of instructions (or blocks of instructions) organized according to the program control flow. From this point of view, a similar approach to the one used in DomoSim-TPC can be followed to build an abstract solution to a programming problem, considering the sentences of a program as the objects that PlanEdit [Redondo, 2003] manipulates. It is necessary to have an intermediate representation language that allows the planning of a program with a high level of abstraction and that facilitates reflection on the decisions made. That is to say, we need a language that encourages discussion about the sentences (instructions) that should be part of a program and about how they relate to the rest of the program. This language must represent the different types of instructions: assignment, loop, selection and call to procedures.

During the Planning of Program Design with PlanEdit three workspaces are used:

- The plan editor (individual workspace) is used to build design plans individually, so that users can outline the solution to a programming problem using a representation language developed for this purpose. Different representations can be used to visualize the plans of program design: sequence of actions followed to build the activity diagram, flow diagrams (like in SICAS), pseudo-code equivalent to the diagram planned or an equivalent to the pseudo-code in some programming language.
- The messaging and representation of the group process (discussion and justification workspace) utility organizes and presents all the dialogue contributions that the group has generated during the activity, including those that the teacher or the system itself can generate. Some of these contributions are design proposals previously elaborated with the plan editor, others are comments, questions, etc. In the user interface (Fig. 2), the contributions are shown in a tree structure, and different buttons allow the users to issue contributions.
- The table of contents (results workspace) allows users to organize and present the final solution elaborated and agreed upon by the group. In this workspace only the contents generated by the group are shown, separated from the process followed to obtain them. A hierarchical structure is used to represent the results of the activity in the form of a table of contents.

Figure 2: PlanEdit's workspace for group discussion

We can identify a clear equivalence in objectives between PlanEdit and SICAS, although they use a different representation language. SICAS approaches learning from an individual perspective, whilst PlanEdit promotes a collaborative process of construction and discussion of solutions.

2.3 COLLEGE

The laboratory (practical classroom) is the natural space for practical programming tasks. However, students do not always finish them in the time defined, so that an extension of this space –and time– is necessary. With COLLEGE [Bravo, 2004] distance work is allowed and encouraged. Thus, the students can work at home, at free-use laboratories, or they can use laptops and the wireless networks of the campuses in order to collaboratively solve programming problems.

COLLEGE (COLLaborative Edition, compilinG and Execution of programs) facilitates the collaborative learning of Programming. The collaboration between students, besides offering cognitive benefits, is a motivating aspect since the students use tools that are familiar to them, such as the chat or the electronic mail.

This system materializes the application of the structuring model and of the synchronous collaboration support mechanisms of the DomoSim-TPC system [Bravo, 2002b] to the Collaborative Programming systems. Thus, a distributed collaborative system that can be used from the labs as well as from home has been developed. Its user interface is shown in Fig. 3. According to the structuring model, in the process of programming we distinguish various stages: (1) edition/revision of the source code, (2) compilation of the source code, and (3) execution of the object programs. These stages correspond with three shared workspaces. The edition of the source code is carried out by a single user, which follows the *Driver-Observer* model characteristic of Pair Programming [Williams, 2001]. The edition floor is agreed on by the students,

who, using the coordination support of COLLEGE, also democratically decide when to compile and to execute.

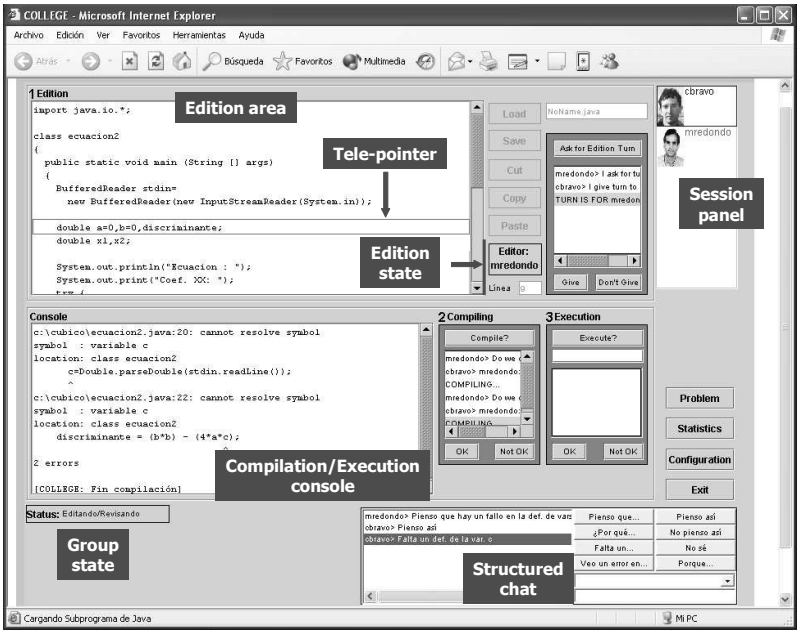

Figure 3: COLLEGE interface

Besides the tasks of the Programming domain, the system offers a collaborative support that consists mainly of an instant messaging tool (structured chat) and a decision-making tool. In addition, the system offers awareness functionalities to facilitate the perception and carrying out of group work (session panel, tele-pointers and other techniques [Bravo, 2002a]).

This system has been implemented using the Collaborative Systems Synchronization Infrastructure (CSSI) developed by the CHICO Group [Bravo, 2004]. This infrastructure, with a centralized architecture, is based on JSDT, and allows developers to turn a mono-user application into a collaborative one, making use of a session management tool and using abstractions such as *client*, *message* and *channel*.

2.4 OOP-Anim

OOP-Anim [Esteves, 2004] is an environment for the learning of basic concepts of Object-Oriented Programming (OOP) and to use them to solve problems. In essence it is a tool for visualization and animation of object-oriented programs created by students. With this tool the students build solutions to problems, simulate the execution of these solutions, detect errors and hopefully try to correct them. Fig. 4 shows a screenshot of the system. It is divided into four parts: program listing, animation area, output area and control panel for managing the animation.

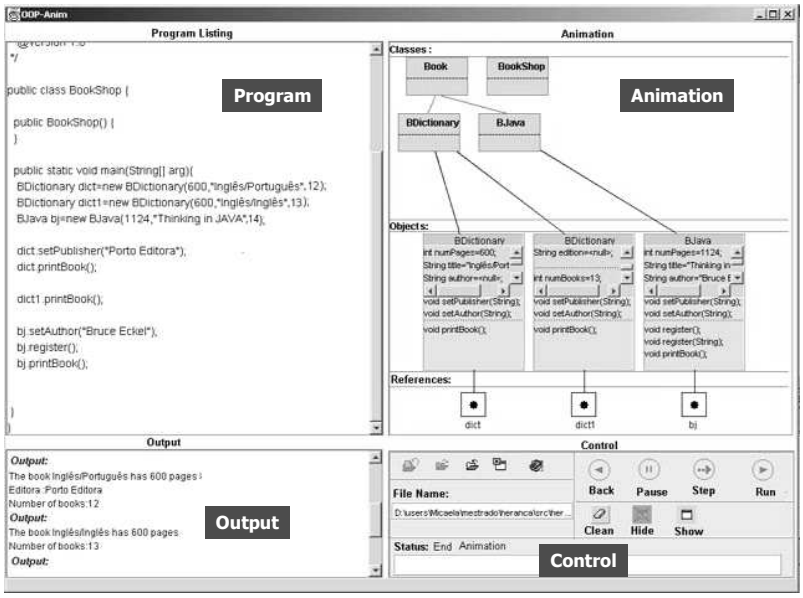

Figure 4: OOP-Anim interface

At an initial stage, this environment can be useful for the analysis of example programs presented by the teacher to the students. The student becomes familiar with the concepts of class and object and their relations (common concepts of OOP). However, although most students can understand programs previously written, creating their own programs is not so easy. In this phase, OOP-Anim can assist the student showing how his/her program works, helping him/her to locate, understand and correct errors. As we mentioned when describing SICAS we believe that self-detection and correction of errors is a very rich learning activity for programming students.

3 Tools Integration

We think that the environments described above are useful independently of each other. However, if they are integrated in a wider environment, allowing easy communication between the different tools, the resulting environment can be even more useful to students. For example, this communication will allow the animation in OOP-Anim of a code written collaboratively in College by simply pressing a button.

The integration of formats and media is important for the management of knowledge through different representations and tools [Hoppe, 2002]. With this in mind, our objective is to integrate and coordinate the previous tools to obtain a productive synergy for Programming teaching and learning. Initially, each of the tools was designed for a particular purpose and has proved to be effective for it. SICAS improves the students' skills for algorithm construction, PlanEdit aids students in making explicit and discussing strategies for programming problem solving, OOP-

Anim helps the students to analyze and understand object-oriented programs, and COLLEGE supports collaborative programming. We are seeing these benefits in some of the experiments we are carrying out, firstly with each separate system , and then with the complete environment (see the Conclusions section). Before approaching this integration, we shall describe the educational methodology that justifies it and situates the use of this synergy.

3.1 Educational Methodology

Object-Oriented Programming (OOP) has been gaining an increasing importance in recent years. Consequently, many universities have adopted OOP languages (Java for example) in their initial programming courses. Our own institutions have followed the same path. However, one of the objectives of first year programming courses is to prepare students for other courses that appear in the following years. Some of them need the students to dominate C and procedural programming. This leads to the decision to use procedural programming in the first semester and object-oriented programming in the second semester. Java is used in both courses.

We follow a common educational methodology in which students first approach easy problems, and then progressively progress to more complex ones. The *problem* abstraction arises from adopting PBL (Problem-Based Learning) [Koschmann, 1996b] as a learning method. Students learn as result of collaboratively solving real or simulated programming problems with well-defined objectives. When the teacher defines a problem, along with other information given he/she indicates the tools students should use to solve it and the order in which they should be used.

In our opinion, visual representation of algorithms can facilitate students' work in the initial stages of Programming learning. Students can start using SICAS individually, and then in groups together with PlanEdit and the asynchronous collaborative infrastructure of DomoSim-TPC. The next step consists in students coding their algorithms in the chosen programming language. These programs will be created individually or in groups using COLLEGE, but it should be possible to automatically express them through flow diagrams, so that they can be visualized in SICAS. When students progress to object-oriented programming, programs developed with COLLEGE should be easily transferred to OOP-Anim, so that students can see their animation, facilitating comprehension and the detection and correction of errors.

In CSCL scenarios shared workspaces with specific visual representations are used to facilitate and enrich the communication and synchronous collaboration [Hoppe, 2002]. In accordance with this, we have considered it interesting to include synchronous collaborative support in OOP-Anim. This system can be started autonomously or directly from COLLEGE. We should point out that programs animated with OOP-Anim should not be very big or too complex, since the number of objects (attributes, methods, references…) that can be visualized with this tool in an effective way is limited.

Fig. 5 shows the aforementioned methodological approach. As the student progresses in learning, he/she makes use of the most suitable tool for the difficulty and type of problem. Thus, at the initial stages, in which the work is typically on easy problems and the students build algorithms instead of programs, they use SICAS and PlanEdit, taking advantage of algorithm simulation and asynchronous collaboration. In more advanced stages OOP-Anim and COLLEGE are used, allowing users to deal

directly with the source code. However, the global system is flexible and allows the students to move to the tool they consider more suitable for their task.

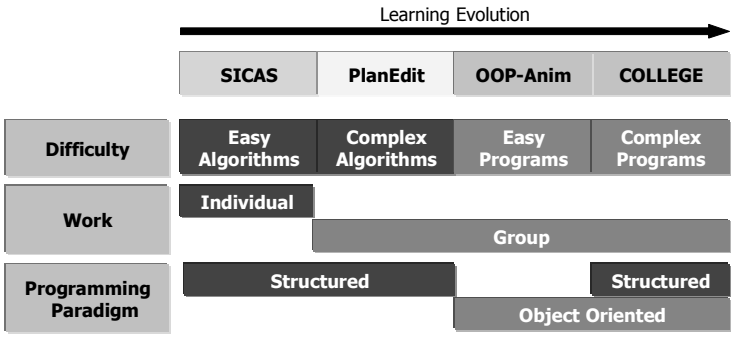

Figure 5: Educational methodology for Computer Programming learning

At the moment, SICAS is a mono-user environment, while the other tools support group work. SICAS and PlanEdit are based on structured programming and are independent of the programming language used, although the automatic code generation currently available only supports C and Java. OOP-Anim only supports Java, and COLLEGE allows any language that can be compiled and executed using external programs.

To illustrate the joint use of the tools, we present an example. A teacher proposes a problem consisting of developing an easy Java program to solve a second degree equation. The teacher has previously organized the students in groups and proposes to them that they use SICAS, PlanEdit and COLLEGE in this order (it is a structured programming problem). First, the students use SICAS to develop an algorithm expressed using a flow diagram to solve the problem. They approach this task individually. Then, they discuss their solution asynchronously with the other group members using PlanEdit, considering the flow diagram as well as the process carried out to obtain it (plan). This discussion will result in a group agreed solution. This solution can then be used to generate code in the target programming language. The code can then be transferred to COLLEGE where students will synchronously collaborate to complete, compile and test the solution.

3.2 Technological Architecture

To support the above described educational methodology, it is necessary to develop some transformation tools that allow direct and inverse engineering between the models used in each of the four applications (see Fig. 6):

- H.FD>Pl: It turns a flow diagram (FD) of SICAS into a plan of PlanEdit. A FD is a visual model of related objects that is transformed into a sequence of high level instructions. This transformation is based on the correspondence between the visual objects (assignment, condition, flow…) and the types of instructions (assignment, selection, loop…).

- H.Pl>FD: It consists in the inverse operation, converting a plan into a FD using the same rules.
- H.FD>Pr: It turns an FD into a program expressed in a programming language (C and Java are currently supported). This transformation is based on patterns that define a relationship between a sequence of instructions and each FD structure.
- H.Pr>FD: It is the inverse transformation, turning a program into its corresponding FD.

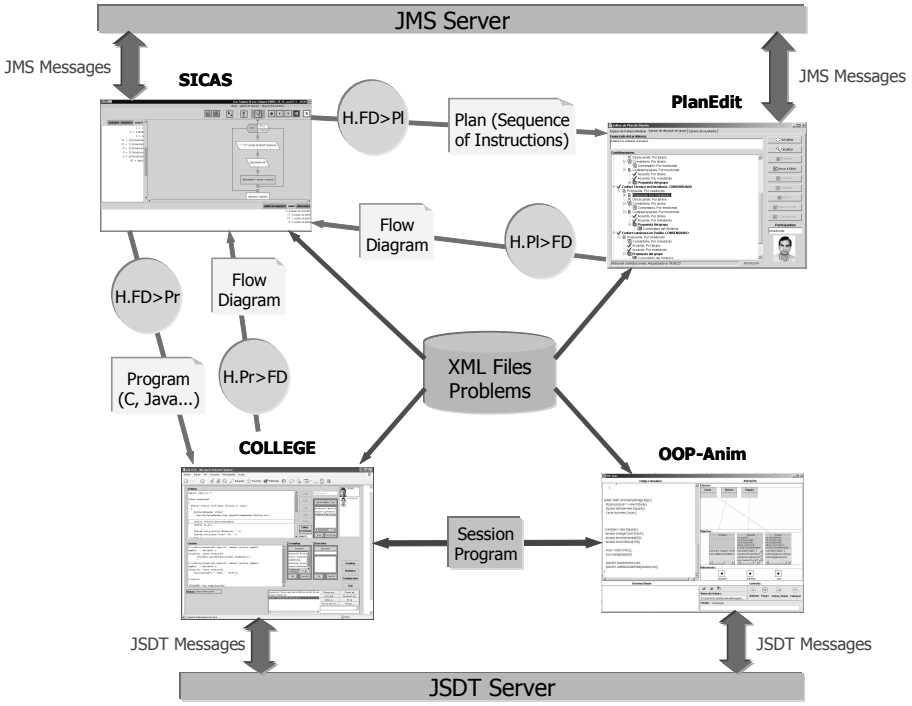

Figure 6: Technologies used to integrate SICAS, PlanEdit, COLLEGE and OOP-Anim

These four transformations are expressed by XML documents, which are the link between the tools, since they maintain their original independence. Thanks to this notation, computational representations are built, and they can be manipulated by other software tools. The information contained in these specifications consists not only of the manipulated models (algorithms and programs), but also includes information about the users that have built the models, the problems solved, the programming language used, etc. These transformation tools are currently included as components of the corresponding educational tools.

Each tool is independent and has its own storage services. However the global system has a library of problems expressed by XML documents. We took advantage of problem structuring in DomoSim-TPC [Bravo, 2002b] to describe a programming

problem. Each problem has an identification, a formulation, a complexity level, a help level –the help the tools offer–, a set of constraints and a set of requirements that make up the objectives. In the same way, the solutions built are stored in XML documents with the same structure used in the transformations.

There is a more direct connection between SICAS and PlanEdit. A notification system based on JMS has been used to keep the models built by both tools synchronized. When users work in synchrony, any change made in a PlanEdit plan is notified to SICAS that updates its FD. In the opposite direction a similar notification is made. Although the messages are available to be processed immediately after being sent, usually these tools are not used simultaneously. In this case the messages are picked up by the JMS server that will deliver them when the destination tool requires.

The same notification technique could be used to connect SICAS and COLLEGE directly. When the users work with COLLEGE and OOP-Anim, this tool behaves as a workspace of COLLEGE. When the students invoke the animation function in COLLEGE, the program in its editor is provided to OOP-Anim and a synchronous session is started in which the group members access the program animation. This requires the use of the session started with COLLEGE, which contains information about the group of users. This synchronous connection between the two tools requires a JSDT server that distributes the interactions that took place during the users' work.

Fig. 6 shows the communication architecture devised to connect the four tools.

4 Conclusions

In this paper we have presented some educational tools to support Computer Programming teaching and learning. These tools, by themselves, have proved to be effective for their particular aim and at specific moments and learning stages. The new objective we have outlined is to integrate them and, consequently, to create a new and more powerful environment. The new environment can be used throughout the learning process, from initial stages, in which easy programming problems are solved, to more advanced stages, in which more complex tasks are approached.

Currently, this integration is being used for Programming learning, in laboratories and at distance, in individual experiences and in group experiences with students from the University of Coimbra in Portugal and the University of Castilla – La Mancha in Spain. We expect to obtain a very significant set of results that allow us to confirm the hypothesis that its utilization leads to improvement in the whole learning process. It will also provide us with a valuable set of information that will allow us to improve the tools, so that we can provide our students with better support. To carry out these experiences we are proceeding in the following way:

- Three sub-sets of students from the total number available and a library of problems with increasing complexity have been defined.
- The first sub-group solves the problems in the library individually in a traditional way, that is to say, using the editors, compilers and tools that are usually used in educational centers.
- The second sub-set solves the same problems individually or in groups, but using only one of the four tools presented.

- The third sub-set solves the same problems following the proposed methodology and using the integrated system we have developed.
- The solutions developed by the three sub-sets, as well as the behavior and work of the students, will be compared using statistical techniques.

A future objective consists in adapting the integration mechanisms to common educational standards. To do this, we are analyzing the state of the different standard proposals, such as IMS-LD[4], LOM[5], SCORM[6] and LTSA[7], in order to consider their suitability for our requirements. It is necessary to point out that these specifications can be expressed by means of XML, which would facilitate the task indicated thanks to XSLT transformations[8].

References

[Bravo, 2002a] C. Bravo, M.A. Redondo, M. Ortega, M.F. Verdejo, Collaborative Discovery Learning of Model Design, In S.A. Cerri, G. Gourdères, F. Paraguaçu (eds.), Intelligent Tutoring Systems, Springer Verlag, Lecture Notes in Computer Science, Berlin, 2002, 671-680

[Bravo, 2002b] C. Bravo, Un Sistema de Soporte al Aprendizaje Colaborativo del Diseño Domótico Mediante Herramientas de Modelado y Simulación, Doctoral Thesis, Computer Science Department, University of Castilla - La Mancha, ProQuest Information and Learning (Current Research), 2002, http://wwwlib.umi.com/cr/uclm/fullcit?p3081805

[Bravo, 2004] C. Bravo, M.A. Redondo, M. Ortega, Aprendizaje en grupo de la programación mediante técnicas de colaboración distribuida en tiempo real, In Proceedings of V Congreso Interacción Persona Ordenador, Lleida, Spain, 2004, 351-357

[Esteves, 2004] M. Esteves, A.J. Mendes, A simulation tool to help learning of object oriented programming basics, In Proceedings of 34th ASEE/IEEE Frontiers in Education Conference, Savannah, Georgia, USA, 2004, F4C7-F4C12

[Gomes, 2001] A. Gomes, A.J. Mendes, SICAS: Interactive system for algorithm development and simulation, In M. Ortega, J. Bravo (eds.), Computers and Education in an Interconnected Society, Kluwer Academic Publishers, 2001, 159-166

[Hoppe, 2002] H.U. Hoppe, K. Gabner, Integrating Collaborative Mapping Tools with Group Memory and Retrieval Functions, In Proceedings of CSCL'2002, Boulder, Colorado, USA, 2002, 716-725

[Koschmann, 1996a] T. Koschmann (ed.), CSCL: Theory and practice of an emerging paradigm, Lawrence Erlbaum Associates, 1996

[Koschmann, 1996b] T. Koschmann, A.C. Kelson, P.J. Feltovich, H. Barrows, Computer-Supported Problem-Based Learning: A Principled Approach to the Use of Computers in Collaborative Learning, In T. Koschmann (ed.), CSCL: Theory and practice of an emerging

---

[4] IMS Learning Design: http://www.imsglobal.org/learningdesign/index.cfm

[5] Learning Object Metadata: http://ltsc.ieee.org/wg12/index.html

[6] Sharable Content Object Reference Model: http://www.adlnet.org/index.cfm?fuseaction=scormabt

[7] Learning Technology Systems Architecture: http://edutool.com/ltsa/

[8] Extensible Stylesheet Language Transformation

paradigm, Lawrence Erlbaum, 1996, 83-124

[Levy, 2003] R. Levy, M. Ben-Ari, P.A. Uronen, The Jeliot 2000 program animation system, In Computers & Education, 40, 2003, 1-15

[Nosek, 1998] J.T. Nosek, The Case for Collaborative Programming, In Communications of the ACM, 41 (3), 1998, 105-108

[Rebelo, 2005] B. Rebelo, M.J.Marcelino, A.J.Mendes, Evaluation and utilization of SICAS – a system to support algorithm learning, In Proceedings of CATE05 – Computers and Advanced Technology in Education, Oranjestad, Aruba, August, 2005 (accepted for publication)

[Redondo, 2002] M.A. Redondo, C. Bravo, M. Ortega, M.F. Verdejo, PlanEdit: An adaptive tool for design learning by problem solving, In P. de Bra, P. Brusilovsky, R. Conejo (eds.), Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Verlag, Lecture Notes in Computer Science, Berlin, 2002, 29-31

[Redondo, 2003] M.A. Redondo, A.J. Mendes, M.J. Marcelino, C. Bravo, M. Ortega, Planificación colaborativa del diseño para el aprendizaje de la Programación, In Proceedings of VIII Taller Internacional de Software Educativo (TISE'03), Santiago de Chile, 2003

[Shen, 2000] H. Shen, C. Sun, RECIPE: a prototype for Internet-based real-time collaborative programming, In Proceedings of the 2nd International Workshop on Collaborative Editing Systems in conjunction with ACM CSCW Conference, Philadelphia, Pennsylvania, USA, 2000, 3-4

[Williams, 2000] L.A. Williams, R.R. Kessler, All I really need to know about pair programming learned in kindergarten, In Communications of the ACM, 43 (5), 2000, 108-114

[Williams, 2001] L.A. Williams, R.L. Upchurch, In Support of Student Pair-Programming, In Proceedings of 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC, USA, 2001, 327-331