Role-based specification of the behaviour of an agent for the interactive resolution of mathematical problems

Miguel A. Mora, Roberto Moriyón and Francisco Saiz

E.P.S, Universidad Autónoma de Madrid, Cantoblanco, 28049 Madrid, Spain
{Miguel.Mora, Roberto.Moriyon, Francisco.Saiz}@uam.es

Abstract. In this paper we describe how a computer system, which includes an authoring tool for teachers and an execution tool for students, is able to generate interactive dialogs in a Mathematics teaching application. This application is named ConsMath, and allows students to learn how to solve problems of Mathematics that involve symbolic calculations. This is done by means of an agent that is in charge of delivering the dialogs to students. The design process entails the dynamic generation of the user interface intended for learners with the insertion of decision points. During the execution, a tracking agent matches students' actions with the ones previously associated to decision points by the designer, thereby activating dynamic modifications in the interface by using a hierarchy of production rules. Students can use ConsMath both at their own pace or in a collaborative setting.

1 Introduction

There are nowadays many computer systems that can be used when learning Mathematics like Computer Algebra Systems (CASs), [3], [11], [12], that can be used as cognitive tools in a learning environment, but that lack the interactivity necessary for a more direct participation of teachers and students in the learning process. Some learning environments, like [2] [6] [1], present a variety of learning materials that include motivations, concepts, elaborations, etc, and have a bigger level of interactivity. Additionally, demonstrating alternative solution paths to problems, e. g. the behavior recorder mechanism used in [6], provides tutors with specification methods for some kind of interactivity. MathEdu, [5], provides a rich interaction capacity built on a CAS like Mathematica. Still, there exists a need of more intelligent systems with bigger capacity of interaction with the student.

The final interactivity of many teaching applications consists of dialogs between learners and applications where they have to answer to questions from an application. In this context, it turns out that the design of user interfaces is a complex process and it usually requires the creation of code. However, teachers are not usually prepared for this. Authoring tools are therefore very appropriate in order to simplify this process for tutors. Besides, it would be desirable to have WYSIWYG authoring and execution tools where students and teachers use similar environments.

Authoring tools for building learning applications allow tutors to get involved in the generation of the software to be delivered to students. For instance, it is usual to

find a scenario where teachers are able to add their own controls (buttons, list boxes, etc) that will form the final application, and even to specify the behavior of such controls when students interact with them. Nonetheless, those authoring tools do not usually give support to specify the feedback to be given to students depending on their actions.

As a consequence of that, models of authoring tools that allow the design of tutoring applications that interact more completely with the students, performing a dialog with them, are desirable. A dialog between a student and a tutoring application involves the existence of moments where the student has to make choices or give information to the system. It can be modeled by means of a tree structure that represents the different paths the dialog can follow, where the nodes represent abstract decision points, which can be used to control the dialogs that take place when solving different problems. This structure is simple enough as to allow teachers to create it interactively, without the need to use any programming language, and it is still powerful enough to represent interesting dialogs from the didactic point of view.

In this paper we present a role-based mechanism of specification of the model for the interaction with the student that is part of the ConsMath computer system, [7], [8], which allows the construction of interactive applications with which students can learn how to solve problems of Mathematics that involve symbolic computations.

ConsMath includes both a WYSIWYG authoring tool and an execution tool written in Java. Teachers design interactive tutoring applications for the resolution of problems using the authoring tool in an intuitive and simple way, since the development environment looks exactly like the students' working environment, except for the availability of some additional functionality, and at each instant the designer has in front of him the same contents and dialog components the student will have at a specific point during the resolution of the problem with the execution tool. The design process is possible in this simple setting thanks to the use of techniques of programming by demonstration, [4], an active research area within the field of Human-Computer Interaction.

ConsMath supports a methodology by which an interactive application for the resolution of sets of problems can be built in a simple way starting from a static document that shows a resolution of a particular problem, and adding to it different layers of interactivity. The initial document can be created by the designer using an editor of scientific texts or it can come from a different source, like Internet.

ConsMath includes a tracking agent that deals with the application being executed by students and matches their operations with the model created by the teacher. Thus, the agent owns all the information necessary for determining the exact state of the interaction. ConsMath has been built using a collaborative framework, [9], so it can also be used in a collaborative setting. For example, students can learn collaboratively, and the tutor can interactively monitor their progress, on the basis of the dialog model previously created.

The rest of the paper is structured as follows: in the next section we shall describe ConsMath from the perspective of a user. After that, we shall describe the mechanisms related to the tracking agent, together with the recursive uses of the model in case the resolution of a problem is reduced to the resolution of one or more simpler subproblems. Finally, we will explain the main conclusions of our work.

2 Interactive Design and Resolution of Exercises of Mathematics

As we have explained in the previous section, ConsMath exercises can be designed by means of a design tool, and they can be solved by means of an execution tool. Both processes are done in a WYSIWYG user interface, without the need of any programming. The development environment looks exactly like the students' working environment, except for the availability of some additional functionality, and at each instant the designer has in front of him the same contents and dialog components the student will have at a corresponding point during the resolution of the problem.

In order to design a problem, the designer specifies a sort of movie that includes a description of how the problem can be solved interactively, together with other movies that describe possible ways that do not lead to the resolution of the exercise, including the appropriate feedback for the student. Just like movie players act by behaving as the persons they are representing are supposed to do, ConsMath designers accomplish their task by imitating alternatively the behaviour of the students when solving the exercises posed to them, including actions that correspond to conceptual or procedural mistakes, and the behaviour of the system in response to their actions.

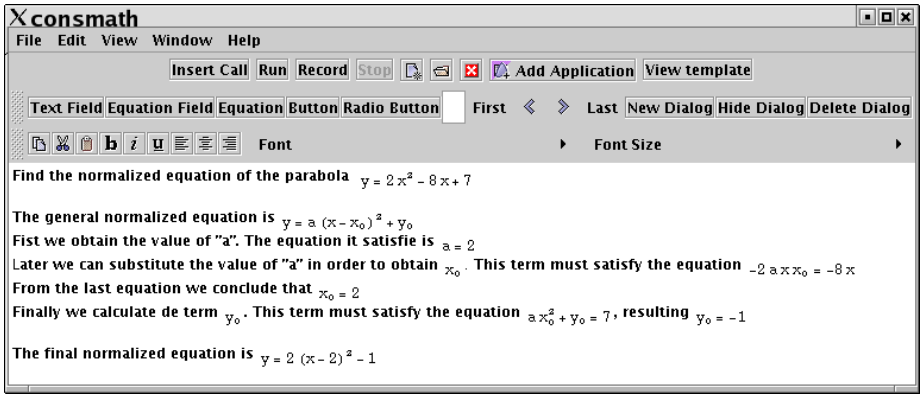

Fig 1. Initial document

Figs. 1, 2 and 3 show three steps during the design of a problem. The designer starts from a document, like in Fig. 1, which shows an editor of mathematical documents that contains a resolution of the problem in the way it would be described in a standard textbook. The document can be imported or it can be built using the ConsMath editor. In this case, the problem asks the student to normalize the equation of a parabola, putting it under the form (1),

$$y = \alpha (x - x_0)^2 + y_0 \qquad (1)$$

in terms of its degree of aperture α and the coordinates of its vertex (x0 , y0). After this, in a first step, the designer generalizes the problem statement and its resolution by introducing generic variables in the formulae that appear in the statement, and defining constraints among the formulae that appear in the resolution of the problem, in a spreadsheet style. For example, the designer can give names A, B and C to the

coefficients in the equation of the parabola, and he can also specify the different formulae that appear in the resolution of the problem in terms of these coefficients. These steps give rise to an interactive document that allows a student to change the equation to be normalized, and the document is updated automatically.

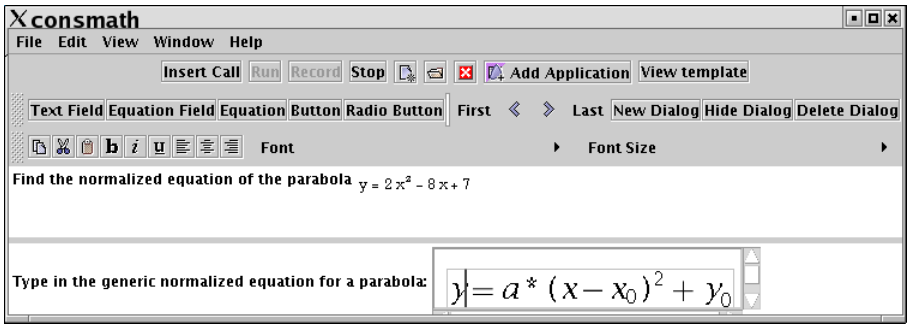

Fig 2. Designer entering the correct answer

Once a generalized interactive document has been specified, the designer describes the dialog between the system and the student. During this dialog, the student makes choices and gives information to the system, and the system gives the student feedback and asks for new decisions to be taken for the resolution of the problem. The teacher specifies the dialog by means of the ConsMath editor of mathematical documents, using some additional functionality that will be described next. At some points the designer switches the role he is playing between system and student. The change of role is done under the shadows by ConsMath when it is needed as we shall explain in the next section. During the steps described previously the designer has played the role of the system. Before the instant considered in Fig. 2, he also plays the role of the system, hiding first the resolution of the problem and typing a question to be posed to the student, where he is asked for the general normalized second degree equation. After this, he enters a field where the student is supposed to type his answer. At the instant considered in Fig. 2 the designer is playing the role of the student when answering the question. He does it by typing the correct formula. After that the designer starts playing again the role of the system. First he gives a name to the formula introduced by the student, then he erases the part of the editing panel where the last question and the answer appear, and finally he poses a new question to the student asking which of the coefficients in the general normalized second degree equation will be calculated first. This is shown in Fig. 3.

In order to create the interactive dialogs, the designer can write text using the WYSIWYG editor, and can insert ConsMath components, like text fields, equation fields, simple equations, buttons, etc. Also, other Java components can be used in the dialogs, importing these components to the ConsMath palette. Each component has a name and a set of properties. The designer can specify the value of a property using a mathematical expression that can contain mathematical and ConsMath functions. These functions allow us to define variables, to evaluate expressions and to create constraints between variables or components. It is important to notice that when the

designer erases parts of the document, although they disappear from the editor pane, they are not deleted, since formulae can still reference variables defined in them.

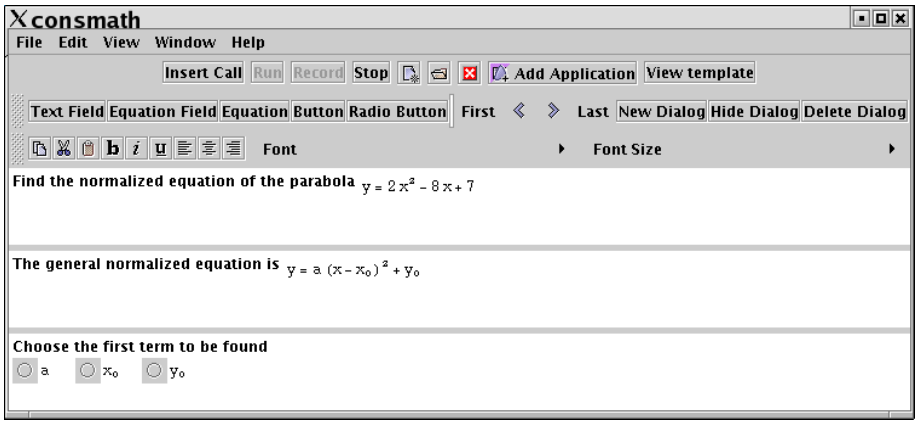

Fig 3. Designer specifying a multiple-choice question

At any time the designer can return to any of the previous states where he is playing the role of the student, and start working again as before. This can be done by means of buttons included in the user interface that allow him to go back and forth. When he is back at one of these points, the designer can continue working as before, and ConsMath interprets automatically that he is specifying a different path to be followed in case the answer of the student doesn't fit the one specified before. In this way, a tree of alternatives that depend on the students actions is specified. The rest of the design process follows a similar pattern.

Once the design is finished, it can be saved. After this, the execution process can start at any moment. There are two ways in which a student can start solving a problem: either the statement is completely specified or the system is supposed to generate a random version of a given class of problem to be solved. The first situation can arise either because the tutor or the tutoring system that controls the knowledge of the student decides the specific problem the student has to solve, or because the student is asked to introduce the specific formulae that appear in the statement. There is a third possibility that takes place when a problem is solved as part of the resolution of another one. During the resolution of a problem, the parts of the movie created by the designer where he has played the role of the system are played automatically, while the ones where the designer plays the role of the student are used as patterns to be matched against his actions during the interactive resolution of the problem. Each decision of the student directs the performance of the movie towards one of the alternative continuations. Hence, for example, if the general normalized equation typed by the student in the first step is incorrect, the system will show him a description of the type of equation that is needed.

The above paragraphs are a description of the way ConsMath interacts with designers and students. In order to achieve this interactivity, the design and resolution tools must be able to represent interaction activities in a persistent way, with the possibility to execute them and undo them at any moment from scratch. Moreover, the

following operations are possible: activities can bifurcate, in the sense that at any point of any activity an alternative can be available, and the actions accomplished by the users determines which of the existing alternatives is executed at each moment. Besides these functional requirements, an editor of mathematical documents that include constraints among parts of formulae is needed, as well as editing functionality that allows the dynamic change of the structure of the document, hiding parts of it that are kept in the background. The way these requirements are satisfied is described in the next section. Now we shall discuss some consequences of the satisfaction of them.

As a consequence of being able to store interaction histories in a persistent way, including alternatives to them, which can be replayed later, ConsMath has some interesting properties from the didactic point of view. The first and most obvious one is that teachers can review the work done by students. They can do it asynchronously by just replaying the work history, but they can also do it synchronously if they connect to a server that sends an event each time the student does some action that is stored. In case the server supports it, the teacher and the student can even collaborate in reviewing the work done and analyzing possible alternatives to it. Students can also review the alternatives proposed by teachers in an asynchronous way, by moving themselves along the tree of proposed alternatives using an interface similar to the one available to course designers.

3 Description of the Tracking Agent

In this section we describe the mechanisms used to implement the behaviour of ConsMath described in the previous section. The main concepts involved are the following ones: a design tree, where the necessary information is stored, a set of production rules contained in the designed tree, which are used in order to decide the actions to be taken by the system at each moment, and a tracking agent, which creates and interprets dynamically the information included in the design tree and activates the rules in order to help the student solve problems interactively.

The tracking agent is the component in charge of the high level interaction aspects in ConsMath. This agent has two main functions: firstly, to interpret the tutor intentionality when the tutor is designing the interaction that takes place during the resolution of the problem being designed using programming by example techniques, and, secondly, to interpret the student actions, comparing these actions with the ones previously exemplified by the tutor or course designer, and replaying the scenario designed by the tutor in response to those student actions. The tracking agent can be controlled be the designer by using a control panel to refine the interactive rules that are being designed, but when the agent is used by the students to execute a previously designed course, the agent interprets the student's actions using the information stored during the design phase, reacting in a proactive way.

The actions exemplified by the tutor are modeled by a design tree with two different interlaced zones, see fig. 4, namely decision zones and performance zones. The design tree represents the conceptual tree of alternative paths that the resolution of a problem can follow depending on the students' actions, described in the previous section. For example, the different answers that can be produced by a student in the

example of the fig.3 can be modeled by a decision zone with two alternatives, one for the correct answer and other for the incorrect one. Each alternative is connected with a performance sequence of actions, one performance sequence to show the student an error dialog, and the other to continue with the problem resolution.

Decision zones are subtrees of the design tree that are formed by one node, which marks the starting point of the decision zone, and its children. The tracking agent stores in them the information to discriminate the different situations that can be produced by the student's actions. More specifically, each descendant node represents one of these alternatives and is linked to one performance zone, forming a decision rule. A decision rule is fired when it is enabled and its head, which is the descendant node of the corresponding decision zone, matches an action from the student. Students' actions give rise to events produced by the user interface. These events form the descendant nodes in decision zones.

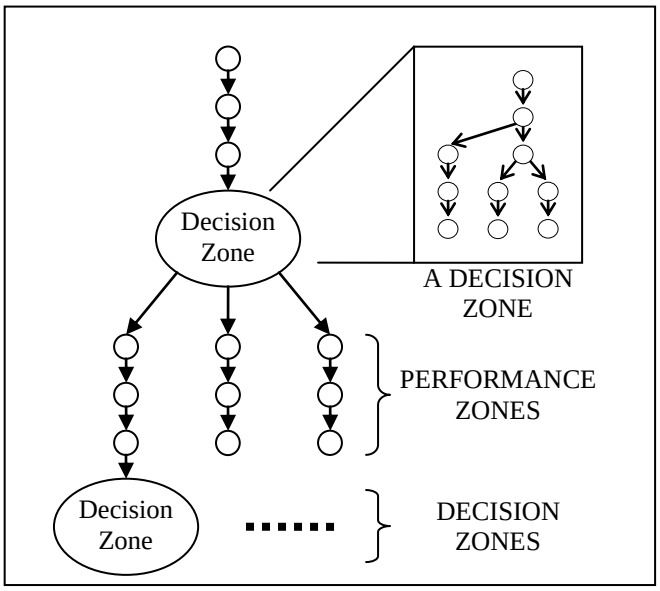

Fig 4. Structure of a design tree

The designer specifies these events interactively emulating the students' actions. In case these events are produced within a performance zone, the tracking agent automatically starts a decision tree. The specification of these events is accomplished as follows:

- Button and multiple-choice events are generated directly by clicking on these components after they are created.
- Conditional events are produced by the evaluation of a condition that corresponds to a formula, like a comparison between two mathematical expressions. When the designer creates a condition, he types its corresponding formula in a field, including dependencies with respect to previous formulae that appear in the document. The designer simulates the action of the students that generates the

event by entering a value in one of the input controls on which the condition depends. The tracking agent enters this elaborated event in the decision tree.

- Matching events are produced by the evaluation of a pattern matching condition between a formula typed by the student and a pattern specified by the designer, like a pattern of trigonometric polynomials. Similarly to the previous case, the designer has to create a pattern by entering the expression that specifies it, and he must simulate the action of the students that generates the corresponding event by entering a value in the input control associated to this pattern.

Performance zones are sequences of execution steps, previously designed by the tutor or designer, which manipulate dynamically the document and can pose the student a new question. The steps that form performance zones can be of the following types: insert text, create random generator, insert or change formula, insert input component, etc. The creation and modification of formulae involves also the creation and modification of constraints among them, as described in the previous section. There are also higher order steps that consist of the creation of subdocuments, which are formed by several simpler components of the types described before. Performance zones that pose questions to the student are followed by another decision zone, forming the tree of decision-performance zones.

The design tree starts with a performance zone that contains a problem pattern. Problem patterns are generalized problems statements whose formulae are not completely specified, like a problem that asks for the normalization of the equation of an arbitrary parabola. Mathematical formulae appearing in problem patterns are formulae patterns. Each part of a formula in a problem pattern that is not completely specified has an associated random generator, which is a program that is called when a student starts solving a problem that must be generated randomly.

As the student progresses in the resolution of the problem, the tracking agent keeps a pointer to a point in the tree model that represents the current stage in the resolution process. If the pointer is in a performance zone, then all the actions previously stored by the designer are reproduced by ConsMath to recreate the designed dialog, stopping when the agent finds the beginning of a decision tree. As the resolution goes ahead, new subdocuments are created dynamically that include constraints that depend on formulae that are already present, and they are updated on the fly.

When the tracking agent is in a decision tree, it enables the corresponding decision rules, and waits until the user generates an action that fires one of them. Then, it enters the corresponding performance zone. This iterative process ends when the agent arrives to the end of a branch in the tree model. When this happens, in case a subproblem is being solved, the resolution of the original problem continues as we will see in the next subsection.

4 Using calls to sub-models

The models created with the tracking agent can be stored in the server, creating a library of reusable problems. This is done at design time using a button to insert a call to a subproblem. At run time, when the agent arrives to the call, it pushes the new model in the execution stack, and begins the execution of the new model until its end.

The whole execution ends when the execution stack is empty once the agent arrives to the end of the first model in the stack.

For example, if we want to create a model to teach how to compute limits using L'Hôpital rule, we can create a problem pattern that can be used to compute (2),

$$\lim_{x->c} \frac{f(x)}{g(x)} \tag{2}$$

where "f", "c" and "g" are the input variables of the problem pattern. In our example we can create a first dialog showing to the student the problem to solve and asking him which method he is going to use to solve that problem. For example we can ask him to choose among several methods for the computation of limits, including L'Hôpital rule and the direct computation of the limit.

Each time the student chooses one of the options, the system has to check that his decision is correct. In case it is not, the designer must have specified how the system will respond. Each time the student chooses L'Hôpital rule the system makes a recursive call to the same subproblem with new values for the initial variables "f" and "g". Finally, when the student chooses to give directly the solution the recursion ends.

5. Evaluation

We have tested how ConsMath can be used for the design of interactive sets of problems. These tests have been performed by two math teachers. A collection of problems from the textbook [10] on ordinary differential equations has been designed. The teachers found that the resulting interactive problems are useful from the didactic point of view, the use of the tool is intuitive and simple, and they could not have developed anything similar without ConsMath. The teachers have also warned us that before using the system on a larger scale with less advanced users like students, the behaviour of the editor of math formulae should be refined. Since this editor is a third-party component, we are planning to replace it by our own equation editor in a future release.

Also, we have done an initial evaluation of previously designed problems in a collaborative setting, where two experts try to collaborate in order to solve a problem and another one, using the teacher role, supervises and collaborates with them. In these tests the experts with the role of students were collaborating synchronously, while the teacher was mainly in an asynchronous collaborative session, joining the synchronous session just to help the students. The first results helped us to improve some minor usability problems that we plan to fix in the next months in order to shortly carry out tests with the students enrolled in a course.

6 Conclusions

We have described a mechanism to design the interaction between students and a computer system in a learning environment using Programming by Example

techniques which allow the designer to create highly interactive applications without any programming knowledge. This mechanism includes the specification of rules that define the actions students have to make during the resolution of problems. Teachers define these rules by means of a role-based process where they act based on the assumption that sometimes they play the role of instructors and other times they act as real students. ConsMath allows the design of collections of problems related to different subjects in Mathematics like elementary Algebra and Calculus.

7. Acknowledgements

The work described in this paper is part of the Ensenada and Arcadia projects, funded by the National Plan of Research and Development of Spain, projects TIC 2001-0685-C02-01 and TIC2002-01948 respectively.

References

1. Beeson, M.: “Mathpert, a computerized learning environment for Algebra, Trigonometry and Calculus”, Journal of Artificial Intelligence in Education, pp. 65-76, 1990.
2. Büdenbender, J., Frischauf, A., Goguadze, G., Melis, E., Libbrecht, P., Ullrich, C.: “Using Computer Algebra Systems as Cognitive Tools”, pp. 802-810, 6th International Conference, ITS 2002, LNCS 2363, Springer 2002, ISBN 3-540-43750-9.
3. Char, B.W., Fee, G.J., Geddes, K.O., Gonnet, G.H., Monagan, M.B.”A tutorial introduction to MAPLE”. Journal of Symbolic Computation, 2(2):179–200, 1986.
4. Cypher, A.: “Watch what I do. Programming by Demonstration”, ed. MIT Press (Cambridge, MA), 1993.
5. Diez, F., Moriyon, R.: “Solving Mathematical Exercises that Involve Symbolic Computations”; in “Computing in Science and Engineering, pp. 81-84, vol. 6, n. 1, 2004.
6. Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M. A.: “Intelligent tutoring goes to school in the big city”. Int. Journal of Artificial Intelligence in Education, 8, 1997.
7. Mora, M., A., Moriyón, R., Saiz, F.: “Mathematics Problem-based Learning through Spreadsheet-like Documents”, proc. International Conference on the Teaching of Mathematics, Crete, Greece, 2002, http://www.math.uoc.gr/~ictm2/
8. Mora, M., A., Moriyón, R., Saiz, F.: "Building Mathematics Learning Applications by Means of ConsMath " in Proceedings of IEEE Frontiers in Education Conference, pp. F3F1-F3F6, November 2003, Boulder, CO.
9. Mora, M., A., Moriyón, R., Saiz, F.: “Developing applications with a framework for the analysis of the learning process and collaborative tutoring”. International Journal Cont. Engineering Education and Lifelong Learning, Vol. 13, Nos. 3/4, 2003268-279, pp. 268-279, 2003, USA
10. Simmons, G. F.: “Differential equations: with applications and historical notes”, ed. McGraw-Hill, 1981.
11. Sorgatz, A., Hillebrand, R.: “MuPAD”. Linux Magazin, (12/95), 1995.
12. Wolfram, S.: “The Mathematica Book”, ed. Cambridge University Press (fourth edition), 1999.