Modeling and Simulation in Inquiry Learning: Checking Solutions and Giving Intelligent Advice

Crescencio Bravo[a], Wouter R. van Joolingen[b], Ton de Jong[b]

[a]Department of Information Systems and Technologies, Computer Engineering School
University of Castilla - La Mancha
Paseo de la Universidad 4, 13071 Ciudad Real, Spain
[b]Department of Instructional Technology, Faculty of Behavioral Sciences
University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands

Abstract. Inquiry Learning is a didactic approach in which students acquire knowledge and skills through processes of theory building and experimentation. Computer Modeling and Simulation can play a prominent role within this approach. Students construct representations of physical systems using modeling. Using simulation, they execute these representations to study the phenomena or systems modeled. However, the modeling task is complex, and students can fail to create adequate models, which prevents effective learning. This necessitates supportive measures to scaffold the modeling processes. In this article, we address the issue of designing such support through the development of intelligent advice to be incorporated in modeling environments. The advice is based on the definition of a family of reference solutions for each modeling problem, on the comparison of the reference solutions with the students' solution, and on the use of an advice knowledge base. This advice guides the students to the construction of a better solution, helping them acquire the knowledge required for successful modeling and for the correction of modeling mistakes. In a collaborative session, having the advice encourages discussion between students about the advice and the best way of proceeding. Empirical validations of the advice approach are presented.

Keywords. Inquiry Learning, Intelligent solution analysis, System Dynamics modeling.

1. Introduction

According to Hansen [1], students retain 25% of what they listen to, 45% of what they listen to and see, and 70% when they manipulate, control and modify experiments, putting into practice what they are learning. Computer Modeling and Simulation (M&S) is an excellent instrument for configuring environments to allow students to carry out manipulation and experimentation, specifically in science domains. Using modeling, students can construct representations of physical systems and make their understandings of a domain visible, sharable, and executable by means of a model. Using simulation they can execute these representations to further explore the systems modeled.

This approach fits nicely with the more general didactic approach of Scientific Discovery Learning or Inquiry Learning. In Inquiry Learning [2, 3], a student performs experiments involving a phenomenon and is asked to infer domain knowledge from those experiments. Modeling facilities add students' capability of making an explicit model of their understanding of the domain. They can run this model to see if their ideas are compatible with other data they know and verify that the model they have made is actually a proper description of the system to be modeled. The assignment to create a model helps students to focus on a specific goal, and the model they create is sharable with other students.

In M&S students must use their inquiry skills to articulate their domain knowledge and then to combine this domain knowledge with modeling skills to solve a specific modeling problem. This is not an easy task, and they can fail due to a mixture of faulty inquiry skills, domain knowledge, and/or modeling skills. Scaffolds for inquiry learning are extensively discussed elsewhere (e.g., [4]). In the current article, we focus on scaffolds for the modeling process.

We address the issue of designing such scaffolds by the creation of an intelligent advice support to be incorporated in modeling environments.

Some systems that analyze solutions to modeling problems can be found in the Intelligent Tutoring Systems (ITS) area. These systems, besides indicating whether the solution is correct or not, can show what is wrong or incomplete and what knowledge deficiencies are responsible. Two prominent approaches have been used to realize such ITSs. Cognitive Tutors [5] feature a cognitive model of the targeted skills, expressed as production rules. Constraint-based Tutors [6] are based on Constraint-based Modeling (CBM) [7], which is a student modeling approach. In CBM, knowledge is modeled by a set of constraints that are used to identify errors in student solutions. These tutors use a more abstract representation of the domain compared to Cognitive Tutors [8].

A number of systems have been developed following these approaches. The argument and expert coaches of BELVEDERE [9] carry out a syntax and consistency analysis to inform students when their solution (inquiry diagram) differs from the expert's solution. SYPROS [10] is an ITS for synchronization of parallel processes with semaphores. It derives a hierarchical representation of the problem solving knowledge for a specific task to be matched to the student's solution. KERMIT [11] is capable of analyzing the student's solutions to database modeling problems using domain knowledge represented as a set of syntactic and semantic constraints. The semantic constraints compare the student's solution to the ideal one. DomoSim-TPC [12] is an environment for the teaching and learning of Domotics (House Automation) that incorporates facilities for evaluating the models built and generating feedback to students. However, this support is offered to the students in a different workspace than the one in which they perform the modeling, and it can only be accessed later in a session.

In this article, we propose a method of providing intelligent advice to students who carry out modeling tasks. Our advice approach is based on the definition of a family of reference solutions for each modeling problem in a library. The reference solutions represent the expert knowledge about the system that is being modeled. These solutions are compared to the student's solution to identify possible differences. Based on these differences, the method generates advice that is presented on-line to students in order to help and guide them towards building a better solution to the modeling problem. This advice is given to students when they require it and only under certain conditions, so as not to overwhelm students with too much advice.

This proposed advice method has been implemented in the Co-Lab environment. Co-Lab [13] is a scientific discovery learning environment in which students, organized in groups, can experiment with simulations and remote laboratories. In particular, this environment includes System Dynamics model editor in which the advice method has been integrated. System Dynamics appears to be an effective technique to represent the dynamic nature of physical systems and to allow the study of variables and their relationships over time by means of simulation. The next section describes the learning setting in which the advice approach is applied. Section 3 shows the models we have developed to represent solutions and modeling problems. The advice method is presented in Section 4: first, the procedure of solution matching; second, the process of advice generation; third, some details about the advice method configuration; and fourth, the integration of the method in Co-Lab. Section 5 describes evaluation results, and Section 6 draws some conclusions and discusses limitations and future lines of research suggested by the method.

2. The learning setting

In this study we used the Co-Lab[1] environment. Co-Lab [13] is an environment designed for synchronous collaborative inquiry learning. Students work in small groups, in a shared workspace, exploring "phenomena" in the form of simulations and/or remote laboratories, and they create their own models of these phenomena. For this, Co-Lab incorporates a Model Editor tool that allows students to construct System Dynamics models and to run simulations with them. It supports shared simulation and goal-oriented modeling according to the classification of CMSL (Collaborative Modeling and Simulation for Learning) systems proposed by Bravo et al. [14]. In this study we did not use Co-Lab's collaborative facilities; students worked on their own and received individual advice on their models.

System Dynamics (SD) [15, 16] is a powerful computer simulation modeling technique for framing, understanding, and discussing complex issues and problems. SD involves translating real life systems into computer simulation models that allow one to understand the structure and behavior of such systems. SD uses components to describe the structure of a system: stocks, flows, constants, auxiliaries or converters, and relations or connectors. Co-Lab's Model Editor supports the specification of SD models using mathematical formulas as well as qualitative relations. The latter are defined in an interactive way by selecting among a set of qualitative relation types, which makes this task very easy and intuitive for students.

Co-Lab as a whole is organized around a building metaphor. Each building represents a course in a specific domain. The building contains floors, which represent the different modules of the courses. A specific system is the subject of study on each floor. Students' goal is to create a model of the system described at each particular floor. Therefore, there is a modeling problem assignment for each floor, and the set of all floors make up a modeling problem memory. Four rooms on each floor (theory, lab, meeting and hall) organize the learning activity.

Fig. 1 shows a screenshot of a Co-Lab session. On the left side of the user interface there is a menu with different tools available, a panel with the session members, and tools for navigation through rooms and coordination. The bottom part contains a Chat and a tool to move objects across rooms. The central work area houses the Model Editor, and the Graph and Table tools to represent the value of the variables during simulation in a numeric and graphic way. A more detailed description of Co-Lab and its tools can be found in [13].

[1] The Co-Lab environment was developed with support of the European Community under the Information Society Technology (IST) School of Tomorrow programme (contract IST-2000-25035).

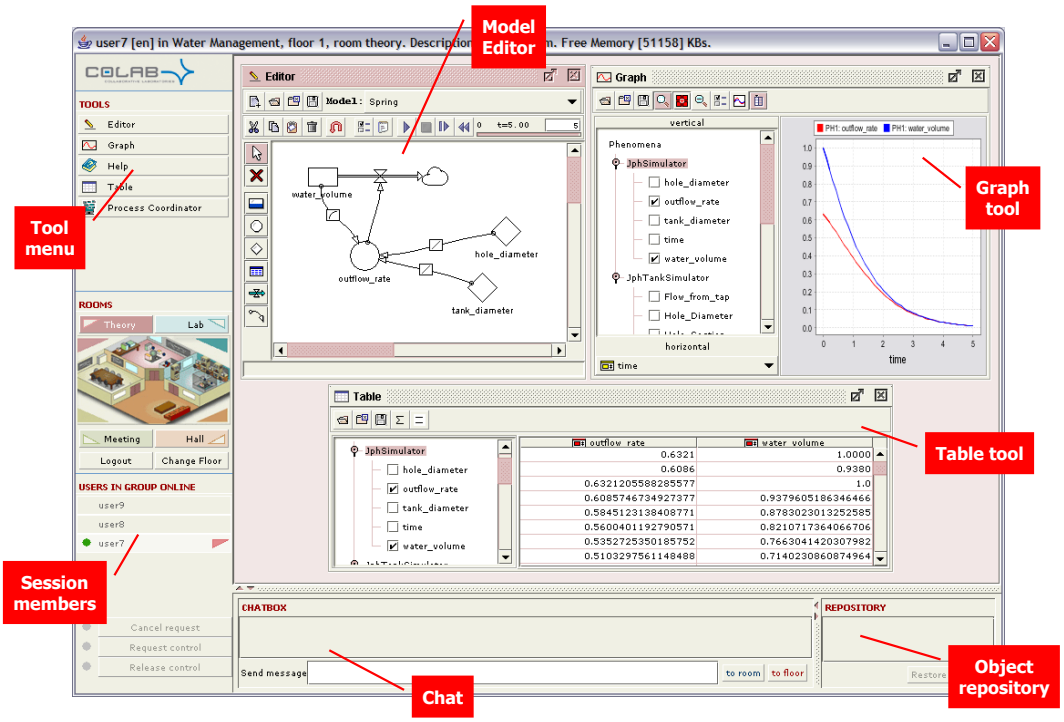

Fig. 1. Screenshot of a Co-Lab session.

Using the Model Editor (which is in the theory room) students can make and execute models. This tool follows the typical structure of a CAD tool: a whiteboard and a toolbar with the components (stocks, auxiliaries, etc.) to be inserted on it (see Fig. 1). The representation of the model follows an icon-based structure. In addition to the component toolbar, a simulation toolbar is available, which controls operations such as starting, stopping and pausing the simulation, and defining simulation parameters. These parameters are the speed (delay) with which the model will be run; the step time, which indicates the time steps with which the model will calculate its output; and the start and stop time.

The Model Editor supports a continuous simulation. However, one can calculate the model results without having to know precisely how the program performs the calculations. Accordingly, the editor's simulation engine supports several integration methods of the differential equations, ranging from the Euler method to Runge-Kutta 4th order. A model starts its calculations with the starting values of the model variables and then calculates how these values change after the specified time step using the formulas describing the relationships between variables. The new values (after one time step) are used to calculate the model changes in the next time step. In this way, the model calculates in time steps until it reaches the specified end time.

3. Formalization of modeling problems and solutions

In order to develop intelligent and automatic advice support we need reference models. These models can be instantiated and operations and transformations can be applied to them. In this section we first describe the models for representing the solutions built by the students and, second, a model for problem specifications that contain a problem statement as presented to students, a reference solution, additional information concerning this reference solution, and parameters to configure the advice.

3.1. Describing solutions

We represent a solution to a SD modeling problem by means of a set of object classes and relationships between them. The object classes identified in the SD domain are *Stock*, *Auxiliary*, *Constant*, and *Flow*. The first three objects are called variables; the *Flow* represents the derivative (with respect to time) of the value of the *Stock*. A variable is described by a name, a type, a unit, and the expression that defines it.

The types of relationships are *FlowRel*, *QualitativeRel*, and *InitializationRel*. Each relationship definition includes the object classes it refers to. A *FlowRel* represents a flow-stock structure. There are four sub-types of *FlowRel* depending on the class of the source and destination (stock or sink). For the relations (connectors) between variables (stocks, auxiliaries and constants), we focus on *qualitative* types. According to Löhner et al. [17], qualitative specifications are appropriate during the initial stages of the modeling process when students still lack a clear understanding of the model they are building. As the advice is meant to support these initial stages, we made the choice to handle qualitative relations. There are five sub-types of *QualitativeRel*: positive linear, parabolic minimum, negative linear, exponential, and parabolic maximum. An *InitializationRel* connects a constant or auxiliary with a stock in order to initialize its value in the first simulation step. All the relationship types are unidirectional. For instance, a *QualitativeRel* between auxiliaries A and B means that A is used to calculate B, but B is not used to calculate A.

This object-oriented description of models facilitates its representation by means of computable structures such as directed graphs, which can be easily implemented by means of matrices. An object model built according to the syntax underlying the aforementioned domain model is not always correct. It needs to conform to the domain semantic constraints (see Subsection 4.1), that is, the formulas defining the variables must be semantically correct.

These models are of a conceptual nature. A graphic representation is required to visually represent a set of objects and relationships. Typically, objects are represented by icons and relationships by lines. Fig. 2 shows a graphic representation of the Water Tank Level system. This model allows the students to investigate the water level in a leaking tank and the influence of the diameter of both the tank and the bottom hole in it. The model contains five objects: one stock, one auxiliary, two constants and one flow, and five relationships: four connectors and one relationship of *FlowRel* type.

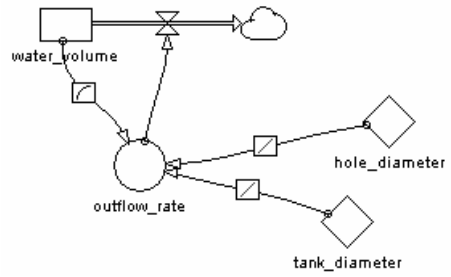

Fig. 2. The model of the Water Tank Level system.

The system implementing this approach represents and stores a specific SD model by means of an XML-based specification. Table 1 shows an excerpt of an example. Each variable in the model is described in a *varspec* label inside the *variables* section. The relationships are defined in the *links* section inside the *layout* section. Each relationship includes its conceptual definition (variables related and relationship type) as well as some graphic properties such as the x-y coordinates and color of the variables involved. The graphic aspects of the variables are included in the *nodes* section. This kind of specification allows students, among other things, to save their models and send them to others.

```
<?xml version="1.0" encoding="UTF-8"?>
<model>
<descriptor>
  <symbol>Condenser Discharge Solution</symbol>
</descriptor>
<variables>
  <varspec>
    <variable>
      <descriptor><symbol>current_intensity</symbol></descriptor>
      <type>double</type>
    </variable>
    <specification>
      <auxSpec>
        <expression>1/resistance * electric_potential</expression>
        <unit>A</unit>
        <exprType>1</exprType>
      </auxSpec>
    </specification>
  </varspec>...
</variables>
<layout>
  <nodes>
    <naux symbol="current_intensity" x1="289" y1="152" x2="321" y2="184"
      label="south" color="0,0,0"/>...
  </nodes>
  <links>
    <lrelation symbol="relation_5" start="capacity" end="electric_potential"
      type="2" x1="93" y1="210" x2="185" y2="195" cx1="123" cy1="200" cx2="155"
      cy2="201" color="0,0,0"/>...
  </links>
</layout>
</model>
```

Table 1. Excerpt of the specification of a student's solution.

3.2. Modeling problem specification

The learning environment in which the advice approach presented in this article is applied contains a set of modeling problems that can be presented to students so that they can get to know what systems need to be modeled. In Co-Lab these modeling problem assignments are presented in the hall room.

In our proposal, the modeling problems and their solutions are represented by means of XML-based specifications to facilitate their definition and computational manipulation. The definition of a modeling problem includes four sets of information: (1) a research question that specifies for the student what needs to be modeled, (2) a *reference solution* (RS) in the form of a model that represents a satisfactory solution for the problem, (3) additional information about the components of this solution, and (4) some parameters configuring the automatic advice support for that modeling problem.

The RS is represented, like any other solution, as a separate XML file following the specification presented in the previous subsection (see Table 1). This is the second set of information in the problem specification.

However, many modeling problems do not have one single solution; instead, multiple equivalent solutions are possible. Therefore, we use a more flexible model that identifies each variable as optional or obligatory to represent a family of solutions for a modeling problem (this is the third set of information). Thus, an optional variable can be or not be in the student's solution. The family of solutions is built by generating all the combinations of optional variables starting from the RS. The number of solutions in this family is the same as the number of combinations among optional variables. This generation method requires the conversion of connectors between variables. Fig. 3 shows a model (left) that contains an optional variable *C*. The model on the right is an alternative solution that does not contain this variable, but it includes a new connector between *B* and *D* built according to a transitivity table. This transitivity table defines conversion rules to describe how to convert the

connectors connected to an optional variable to new connectors in a model in which this variable is not present.

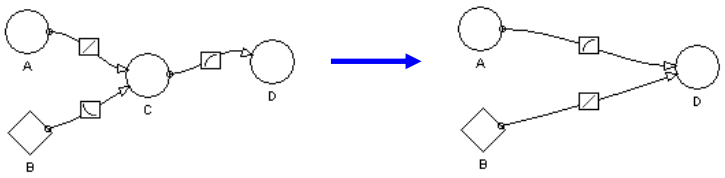

Fig. 3. Elimination of an optional variable in a solution.

In this strategy the stocks are always obligatory to represent a family of solutions with a unique reference solution. This can be a suitable approach for a majority of the modeling problems that can be presented in Secondary Education and first year University courses. In these areas of education the models to build can fit in this approach (requiring a fixed number of stocks to represent the modeled system's state) depending on the nature of the specific domain. Such domains include Population Dynamics, Water Management, Electricity, etc.

To be able to compare students' solutions with a RS, we need to consider that students may name the variables differently than in the reference model. We attempt to overcome this by associating a list of possible words (aliases) with each variable in the RS, hypothesizing that the students will use these words in their variable names. This heuristic will be explained in Subsection 4.1.1.

The fourth set of information allows the teacher to configure the advice method for that specific modeling problem. Table 2 shows an excerpt of a modeling assignment aiming at representing the discharge process of a condenser. The RS (see Table 1) completes this specification. The *variables* section contains the list of solution variables. The optional character of a variable is defined in the *optional* label. The alias definition for a variable is found in the *aliases* section. Finally, the parameters for configuring the advice are included in the *checkParameters*, *modeChangeParameters* and *adviceSelectionParameters* sections. This configuration will be described in Subsection 4.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<model>
  <descriptor>
    <symbol>Condenser Discharge Modeling Assignment</symbol>
  </descriptor>
  <variables>
    <variable>
      <descriptor>
        <symbol>current_intensity</symbol>
      </descriptor>
      <optional>false</optional>
      <aliases>
        <alias>intensity,current</alias>
        <alias>intensity </alias>
        <alias>current</alias>
        <alias>curr</alias>
      </aliases>
    </variable>...
  </variables>
  <parameters>
    <checkParameters>
      <minimumNumberOfElements>3</minimumNumberOfElements>
      <timeBetweenChecks>60</timeBetweenChecks>
    </checkParameters>
    <modeChangeParameters>
      <timeToIntermediate>600</timeToIntermediate>
      <numberOfChecksToIntermediate>3</numberOfChecksToIntermediate>
      <modelChangesToIntermediate>4</modelChangesToIntermediate>
      <timeToFinal>1200</timeToFinal>
      <numberOfChecksToFinal>5</numberOfChecksToFinal>
      <modelChangesToFinal>6</modelChangesToFinal>
    </modeChangeParameters>
    <adviceSelectionParameters>
      <messagesToShow>99</messagesToShow>
      <messageCategoryOrder>
        <errorOcurrencesOrder>0</errorOcurrencesOrder>
        <errorTypeOrder>1</errorTypeOrder>
        <objectTypeOrder>0</objectTypeOrder>
        <objectNameOrder>2</objectNameOrder>
        <errorLevelOrder>0</errorLevelOrder>
      </messageCategoryOrder>
      <filterFirstElementInCategory>false</filterFirstElementInCategory>
    </adviceSelectionParameters>
  </parameters>
</model>
```

Table 2. Excerpt of the specification of a modeling problem.

Given these specifications of the four sets of information making up the definition of a SD modeling problem, a problem P can be formalized as follows:

P=(modelingAssignment, RS, optionalVars, aliases, checkParameters, modeChangeParameters, adviceSelectionParameters)

The matching of a student's solution to an ideal model or possible solution is quite a common process in ITS. In Constraint-based Tutors (see, e.g., [6]), there is an ideal solution that is compared against the students' solution. In KERMIT [18], the ideal solution really represents more than one solution, because there are rules that allow the system to transform the solution to equivalent ones having different forms, which is similar to our case. Other approaches, instead of comparison to an expert model, use one or more solutions built by a teacher. This is the case for CTAT [19], an authoring tool for Pseudo Tutors. Pseudo Tutors are developed by demonstration [20], instead of programming. This way, the author demonstrates correct and incorrect solutions, which are used for matching and giving feedback.

4. A flexible advice approach

Fig. 4 shows the structure of the advice method. The advice generation takes place in two phases. First, the student's solution (SS) is matched to the reference solution (RS). Second,

the advice messages are generated based on the differences detected. The student builds the SS using Co-Lab's Model Editor. The RS is built by the teacher using an authoring tool. The following subsections describe both phases and components of the method, its configuration, and its integration in Co-Lab.

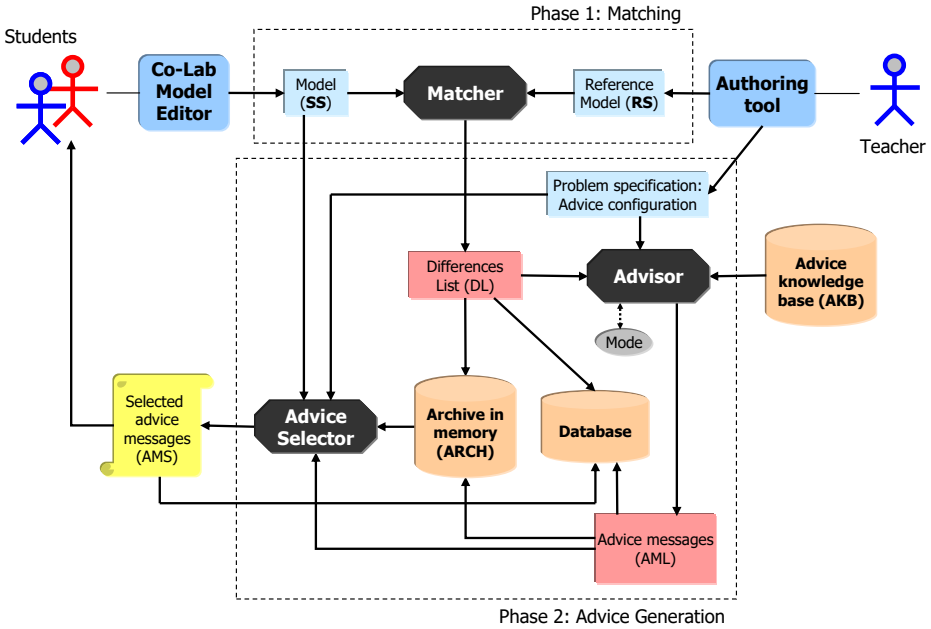

Fig. 4. Structure of the advice method: phases, processes and data structures.

4.1. Analyzing differences between solutions

The advice procedure works on demand, avoiding unnecessary intrusion into the student's task performance. When student asks for advice (*check* to request advice), the matching process is started (see Fig. 4).

For the Matcher to do its work, the SS must be syntactically and semantically valid. The syntactic and semantic rules have been coded into the Model Editor. Graph Grammars [21] would be an option for expressing and checking syntactic aspects. However, their use is restricted by efficiency [22]. This way, the Model Editor constrains the student by preventing syntax errors or allowing their immediate correction. Before starting the matching process, the Matcher does a final semantic check, informing the student of the semantic errors so that they may be corrected.

Every difference found between the SS and the RS is an opportunity for the system to present advice and for the student to acquire new knowledge and skills [23]. A *difference* is defined by an identification and five parameters, which are used to store the names of the objects and relationships involved in the difference. Two difference levels are identified: general differences, which do not refer to a particular element, and specific or detailed differences, which refer to a specific object or relationship represented by means of the parameters. The difference types are organized in seven categories: STOCKS, FLOWS, AUXILIARIES, CONSTANTS, CONNECTORS, UNITS and OTHER. The majority of them refer to SS variables that do not belong to the RS, which are therefore *unnecessary* variables, and to RS variables not considered by the students in the SS, which are therefore *missing* variables. Table 3 shows a few sample difference types.

Category	Difference ID	Meaning	Level	Parameters
STOCKS	STOCK-	Some stocks are missing	General	-
	STOCKX+	The stock <object> is unnecessary (STOCKX+)	Specific	objectType, objectName
CONNECTORS	CONNSD_RTYPE	The connector (<object1>,<object2>) is not of the correct type	Specific	objectType1, objectName1, objectType2, objectName2, connectorType

Table 3. Sample of differences generated by the matching phase.

The matching process consists of two stages: linking SS variables with RS variables, and then comparison of both solutions to detect structural differences.

4.1.1. Linking variables

Linking SS variables to RS variables is not trivial. Names chosen by students have to be understood by the Matcher to associate them with RS variables. A list of possible names for each variable is included in the modeling problem specification. We call such a list an *alias*. When all the words of an alias are contained in the SS variable name, then the SS and RS variables match, and the SS variable is supposed to correspond with the RS variable. In the example given in Table 2, the aliases for the variable *current_intensity* admit any combination of the *intensity* and *current* words; also the word *curr* matches with this variable.

The teacher should take the RS variable names from the modeling assignment, so that these names will be familiar to the students. Moreover, in the case of Co-Lab these variables could be extracted from the simulations available in the lab rooms, in which the students can experiment with the behavior of the models that they have to build in their modeling tasks. KERMIT [18] avoids the problems of name matching by forcing the student to select the name of each entity in the modeling assignment by highlighting it.

If there are SS variables not assigned to RS variables after the matching, the process stops and the students are requested to complete the linking of variables manually. The manual linking is also required when the unit of a RS variable does not match with the unit of the corresponding SS variable. If the number of SS variables of a specific type is the same as the number of obligatory RS variables, that is to say, both solutions could potentially match, but exactly one SS variable is not yet linked (it has been not able to be assigned automatically to a RS variable) the matching process applies a heuristic consisting of assigning this SS variable to the first obligatory RS not assigned, because this linking is likely to be successful.

Fig. 5 shows the user interface to support the manual linking of variables. The names and units for the SS and RS variables are shown in a table giving the present linkings. The students have to select the RS variables corresponding to their variables from a selection field (*Solution var.*). The check box *Change name?* allows the students to change their SS variable name to the RS variable name. The students can mark the check box *Change Unit?* to indicate that they want to assign the correct unit to their variable. This interface is also shown when the manual linking is not necessary, so that the students can review the assignments made by the system.

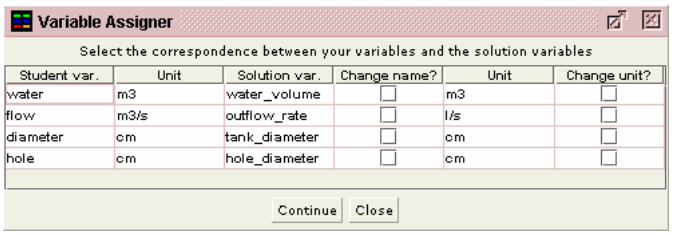

Fig. 5. The user interface for manual linking of SS variables with RS variables.

4.1.2. Matching solutions

When the automatic linking has been successful or when the students have pressed the *Continue* button in the user interface (see Fig. 5), the matching process continues to the second stage, the matching of the solution structure, which can generate three kinds of differences: RS variables that are missing, SS variables that are unnecessary, and other differences in relation to units, connectors, etc. As mentioned before, the differences are generated at two levels. Typically, a general level difference means that the number of variables of a specific type in the SS and RS is not the same. A detail level difference usually implies that a specific variable or relationship is missing or not required, or that there are units or relationship types that do not match. All differences detected are inserted in the differences list (DL), which is the output of this phase and the input of the next phase (see Fig. 4).

The algorithm describing the solution matching process is shown in Table 4. This process works in two steps. The first step analyzes the solution at a general level, generating only general differences. The function *matchNumberOfObjects* matches the number of objects in SS and RS. Let *#objects* be a function to calculate the number of objects of a specific type in a solution. In general, if *#objects*(SS) is less than *#objects*(RS) a difference statement of the type "*There are <variables> missing in the SS*" is inserted in the DL. On the contrary, if *#objects*(SS) is greater than *#objects*(RS) the difference statement "*There are unnecessary <variables> in the SS*" is inserted in the DL. There is a distinction between matching stocks and flows and matching constants and auxiliaries. A stock or flow is always obligatory, whereas a constant or auxiliary can be obligatory or optional. Therefore, in order to determine that constants or auxiliaries are missing in the SS, their number must be less than the number of obligatory elements in the RS; and to determine that there are unnecessary constants or auxiliaries, their number must be greater than the total number of both obligatory and optional elements. In regard to flows, apart from matching their total number with the *matchNumberOfObjects* function, they are matched by number of each flow type with the *matchNumberOfFlows* function.

```
// step 1: match number of objects
matchNumberOfObjects(STOCK,SS,RS)
matchNumberOfObjects(CONSTANT,SS,RS)
matchNumberOfObjects(AUXILIAR,SS,RS)
matchNumberOfObjects(FLOW,SS,RS)
matchNumberOfFlows(SS,RS)
```

```
// step 2: match specific variables and relationships
matchVariables(SS,RS)
matchUnits(SS,RS)
matchSpecificFlowTypes(SS,RS)
matchConnectors(SS,RS)
matchOptionalVariables(SS,RS)
```

Table 4. Algorithm for solution matching.

The second step focuses on detecting detail level differences regarding specific variables and relations. The *matchVariables* function calculates unnecessary SS variables and missing RS variables, and inserts the appropriate differences in the DL. A difference is detected for each SS variable defined with a different unit than its corresponding RS variable (*matchUnits* function). The *matchSpecificFlowTypes* function detects flows that do not match in number and type in SS and RS, and also identifies missing and unnecessary flows. The technique of identifying missing or unnecessary objects is also applied for matching connectors (*matchConnectors* function). In this case, connectors that are in both SS and RS but with

different connector type are identified. Finally, the *matchOptionalVariables* function inserts in the DL difference statements expressing that other variables not considered by the students (optional variables in the RS) could also be used; this makes it possible to give suggestions about that concern.

The differences calculated play the role of activation conditions from which to generate advice. But they can also be used to measure the quality of the SS compared to the RS. In order to evaluate the quality of SD models, Löhner et al. [17] propose a calculation consisting of giving a positive score for each correct relationship and subtracting a penalty for each redundant relationship. Similarly, positive and/or negative scores can be assigned to all the types of differences. Thus, the solution quality can be estimated simply by summing the scores of the differences occurred in a specific matching. This quality estimation can be challenging and motivating feedback for students, especially for success-oriented ones.

4.2. Giving advice

The second phase of the advice method is the advice procedure itself. It takes the differences detected between the SS and RS (DL) as well as the differences and advice that have been detected or generated in previous checks (ARCH), and generates a list of advice messages as output (AMS). This procedure is performed in two sequential stages (see Fig. 4). The first stage consists of the generation of advice statements based on the differences. This is carried out by the Advisor process. In the second stage, the advice messages are sorted and selected according to different criteria and then presented to the students. This is carried out by the Advice Selector process. These two stages are commonly used in tutoring or coaching systems that offer advice (e.g., [24, 25]).

4.2.1. Advice generation

Before designing the advice method, we first performed an empirical study to identify the way students approach a modeling task within Co-Lab, with a remote teacher who gave advice when asked. We saw that students build an initial understanding of the modeling assignment and of the model that solves it and think in terms of general structures at the beginning. When their understanding progresses they add elements to their general structure, so that they build a more detailed model and start running simulations in order to complete it. Finally, they refine their solution by adding small details. Therefore, we concluded that the advice should consist of general messages related to students' general structures in the beginning, and more detailed messages at the end of the tasks, when students can experience difficulties reaching a good solution. Accordingly, the Advisor has three modes of operation: *initial*, *intermediate* and *final*. The mode of operation changes depending on the model's specific structure or on the amount of students' work (number of changes made to the model in a specific time).

The advice generation is based on an advice knowledge base (AKB) that contains a set of rules. Table 5 shows a subset of advice rules from the AKB. Upon each advice request, the Advisor analyzes all the rules from the AKB and calculates which rules to activate. When a rule is activated, its associated advice message is considered for presentation to the student. The texts of the advice messages were selected taking into account the suggestions of experts and teachers and the aforementioned study. Similar to Paolucci et al. [25], the advice style consists of suggestions and questions, so that the students have to think about the advice instead of simply executing it.

Rule	Advice message	Advice Type	Advice Level	Matching Type	Activation Conditions		
					Initial Mode	Intermediate Mode	Final Mode
R1	This is a nice basic structure for your model! Try running it and compare the results with your experiments	C	G	ALL	Not(STOCK-), Not(STOCK+), Not(FLOW-), Not(FLOW+), Not(FLOW_TYPE), Not(FLOWX_TYPE), Not(FLOWXSD-)	Not(STOCK-), Not(STOCK+), Not(FLOW-), Not(FLOW+), Not(FLOW_TYPE), Not(FLOWX_TYPE), Not(FLOWXSD-)	Not(STOCK-), Not(STOCK+), Not(FLOW-), Not(FLOW+), Not(FLOW_TYPE), Not(FLOWX_TYPE), Not(FLOWXSD-)
R2	You may consider adding or removing flows (revise your flow-stock structures)	E	G	SOME	FLOW-, FLOW+, FLOW_TYPE, FLOWX_TYPE, FLOWXSD-	-	-
R3	Think about what "things" flow in and out and from/to where	E	G	SOME	STOCK-, STOCK+, STOCKX-, STOCKX+	-	-
R29	You do not need all the links in your model	W	G	ALL	-	CONN+	-
R31	<variable> does not depend on <variable>	W	D	ALL	-	-	CONNSD+ < R33
R33	You should remove the connector between <variable> and <variable>	W	D	ALL	-	-	CONNSD+ > R31

Table 5. A subset of advice rules from the Advice Knowledge Base.

A rule is defined by a text message, a list of activation conditions (ACL) for each advice mode, and three attributes: *type*, *level* and *matchingType*. The type (*error* (E), *warning* (W), *comment* (C)) refers to the importance of the advice. The level (*general* (G) or *detailed* (D)) classifies the advice. The Advisor analyses the ACL corresponding to its current mode. Each list includes a number of activation conditions (AC). The matching type indicates whether all AC must be activated to activate the rule (*ALL* value) or if it is only necessary that at least one AC be activated to activate the rule (*SOME* value).

An AC is defined by the identification of a difference, the definition the activation as occurring when this difference is present or when it is not present (called *presence* attribute), and information indicating which rules must obligatorily have been activated in past checks (called *afterAdviceMessages* attribute) or which rules must have not been activated in past checks (called *beforeAdviceMessages* attribute) as a requirement to activate the present rule. In Table 5, the symbols *Not*, < and > are used in the definition of the *presence*, *beforeAdviceMessages* and *afterAdviceMessages* attributes respectively. The *beforeAdviceMessages* and *afterAdviceMessages* attributes make it possible to have several different messages for the same differences in the same mode, facilitating the presentation of many different messages depending on the prior sequence of messages received, which enriches the advice process and makes the Advisor flexible and dynamic.

The matching and the advice generation constitute a production-rule model. The advice rules described above work like the rules for recognizing both errors and correct steps in building the solutions as in cognitive tutors [5].

The change of advice mode acquires importance in this rule-based advice generation method, since the messages given depend on the mode (see Table 5). The basic idea is to advance the mode (from initial to intermediate or from intermediate to final) if the students are not building a correct solution or if they have spent too much time in a specific mode. The mode can also move back, as when the model gets worse from check to check. To implement the change of mode, we define two types of model structures: basic structure, representing a model whose stocks and flows match with the ones in the RS, and complete structure, which consists of a model with a basic structure and whose auxiliaries, constants and connectors match with the ones in the RS. We call TI and TF the time that has to pass to change from initial to intermediate and from intermediate to final modes respectively; CKI and CKF are the number of checks to change from initial to intermediate and from intermediate to final

modes respectively; and CI and CF are the number of changes in the model to advance from initial to intermediate and from intermediate to final modes respectively. The mode is calculated before generating advice, each time the students request advice. In order to advance from a given mode to the following mode at least one of the two following conditions should be verified: (1) that a specific period of time (TI and TF) passed and the number of checks and of changes in the model (insertions, deletions, etc.) was greater than some minimum (CKI, CKF, CI and CF), or (2) that the model had a specific structure (basic or complete).

The existence of three advice modes makes the advice more progressive (from general to detailed, from suggestions to imperative messages) than other approaches. For instance, SYPROS [10] basically considers a state in which the students have made many errors; if it seems clear that they are not able to find a solution, it gives more concrete explanations.

A high level algorithm that formalizes the advice generation described is shown in Table 6. The advice mode is calculated at each advice request with the *calculateMode* function. For each advice rule (AR) in the AKB, the Advisor checks if the rule should be activated (*ruleIsActivated* function) starting from its definition, the DL and the *adviceMode* (see Fig. 4). If it is activated, an instance of an advice message (AMI) is created, containing the text, type and level of the advice message and some parameters to replace some marks in the message text. These parameters are extracted from the difference corresponding to the first AC that is activated. Then, the AMI is inserted in the advice message list (AML). Finally, the differences and advice messages generated are stored in a database for later analysis, as well as in an archive in memory (ARCH) to be taken into account in subsequent advice requests.

```
AML=∅
adviceMode=calculateMode()
for each AR in AKB do
  if ruleIsActivated(AR, adviceMode, DL) then
    AMI=createAdviceInstance(AR, DL)
    insertInList(AML, AMI)
  end-if
end-for
registerInDB(AML, DL)
registerInArchive(AML, DL, ARCH)
```

Table 6. Algorithm for advice generation.

Table 7 shows an excerpt of the XML-based definition of two example rules. The first one (left) is for a general comment to be shown in the initial mode that indicates that the structure of the student' model is right. This occurs when any stock is missing (*ruleError* label), etc. The second one (right) is a warning that exemplifies the use of parameters: the values of the difference parameters replace the marks ('%1' and '%2') in the advice text. In the case of the *UNIT* difference, the parameters 3 and 4 are the names of the units.

```
<adviceKnowledgeBase>
  ...
  <adviceRule>
    <adviceId>BasicStructureNice1</adviceId>
    <adviceText>This is a nice basic structure for
your model! Try running it and compare the results
with your experiments
    </adviceText>
    <adviceDescription>
      <type>comment</type>
      <level>general</level>
      <errorMatching>all</errorMatching>
    </adviceDescription>
    <ruleErrors>
      <ruleError>
        <errorId>STOCK-</errorId>
        <isPresent>false</isPresent>
        <adviceState>initial</adviceState>
      </ruleError>...
    </ruleErrors>
  </adviceRule>
  ...
```

```
  <adviceRule>
    <adviceId>UseUnitToMeasure</adviceId>
    <adviceText>Use '%2' to measure '%1'
    </adviceText>
    <adviceDescription>
      <type>warning</type>
      <level>detailed</level>
      <errorMatching>all</errorMatching>
    </adviceDescription>
    <parameters>
      <parameter>3</parameter>
      <parameter>4</parameter>
    </parameters>
    <ruleErrors>
      <ruleError>
        <errorId>UNIT</errorId>
        <isPresent>true</isPresent>
        <adviceState>final</adviceState>
      </ruleError>
    </ruleErrors>
  </adviceRule>
  ...
</adviceKnowledgeBase>
```

Table 7. An excerpt of the XML-based specification of the AKB.

4.2.2. Sorting and selection of advice messages

Since the number of advice messages generated can be large, it would not be effective to show all of them. According to Paolucci et al. [25], the advice is often more than a student can be expected to absorb and respond to at one time. Moreover, it would be difficult for the student to approach their resolution in an organized way. In line with this, we designed a procedure for sorting and selecting advice messages. Comments are always shown, but errors and warnings are chosen by this selection process.

With respect to sorting, we identified five ordering criteria: (1) the frequency of the difference that activates the advice, (2) the advice type, (3) the type of the objects to which the advice refers, (4) the name of the objects, and (5) the advice level. The first criterion allows the students to focus on the most frequent differences and develop skills in relation to the area of knowledge in which they seem to be failing. The second criterion leads the students to approach the more serious advice messages first. The third and fourth criteria organize the advice messages around the objects and relationships and their types. This facilitates students' understanding, since they can focus on model parts or mini-models (concrete objects and relationships). The fifth criterion presents first the general advice messages referring to structural aspects of the model, which should be solved in the first place.

Once sorted, the messages on the AML can be filtered. When the filtering function is activated, the Selector chooses the advice messages with the first (highest) value according to the sorting category specified. Thus, if the first sorting criterion is the advice level, only the general messages will be shown. A second complementary filtering functionality consists of selecting the first *n* messages.

This sorting and selection approach is more complete than the ones incorporated in KERMIT [18] and BELVEDERE [9]. KERMIT shows an advice message only for each solution submission. BELVEDERE calculates several advice statements, but it selects only the first in priority order. An interesting approach found in COLER [24] is sorting the advice messages by preference. This preference can change during the session according to the group's performance. COLER also selects one advice message (the most preferred) from the advice list, whereas the remaining messages can be given on demand. Suthers [26] sorts advice messages using selection heuristics. Two example heuristics are "say something you haven't said before" and "minimize the amount of information to be processed". Our selection process supports the former by means of the *before* and *after* attributes of an advice rule. The latter has been implemented in the two aforementioned filtering functionalities.

4.3. Configuration of the advice method

One of our aims when we designed the advice method was that it should be flexible enough to allow the teacher to adjust it. In line with this, many parameters are configurable. These parameters are included in the XML-based modeling assignment specification (see Table 2) and can be easily modified by the teacher according to his/her preferences, resulting in different ways of generating advice, each one with different results for the students' error correction process and for their learning.

The teacher can configure the advice at the modeling assignment level. The configuration areas (see the sections *checkParameters*, *modeChangeParameters* and *adviceSelectionParameters* in the specification given in Table 2) are as follows:

- Constraints for advice request: The advice works on demand, which means that if students use it very frequently the advice more or less solves the modeling problem for them. To avoid this situation, two parameters are available: the minimum number of model elements that need to be present and the time that has to pass between checks in order to give advice.
- Change of the advice mode: The change of mode is implemented with the TI, TF, CKI, CKF, CI and CF parameters previously described.
- Sorting and selection of advice messages: In order to define a specific sorting, the priority of each sorting criterion has to be indicated. The activation of the filtering must also be defined.

The complete advice knowledge base (AKB) is specified in a separate XML file. In this way, by making use of a suitable tool, the teacher can adjust and extend the rules. An initial configuration containing 44 advice rules is included in Co-Lab.

4.4. Integration in Co-Lab

The models, data structures and algorithms that made up the advice method have been implemented and integrated in Co-Lab. Fig. 6 shows a session of modeling with Co-Lab. The system that needs to be modeled according to the modeling assignment is a Water Tank Level system. In the example user interface shown, the student has requested advice. Therefore, besides the Model Editor, there are two open windows: the Advisor console and the Variable Assigner table for linking variables.

The student has to press the *check* button to request advice. Because the processing of the Advisor may take a few seconds, the *State* label indicates whether the Advisor is working (*Active*) or not (*Inactive*). The Advisor process triggers the Matcher process. The Matcher can require the students to manually link their own variable names to variable names present in the reference model (for those variables not assigned automatically), using the Variable Assigner window as described above (see Fig. 5). This window also opens when all variables are linked automatically, so students can check the linking the Matcher has made. After the linking of variables (manual or automatic) the Advisor generates the relevant advice messages, and their texts and types are shown in the advice console.

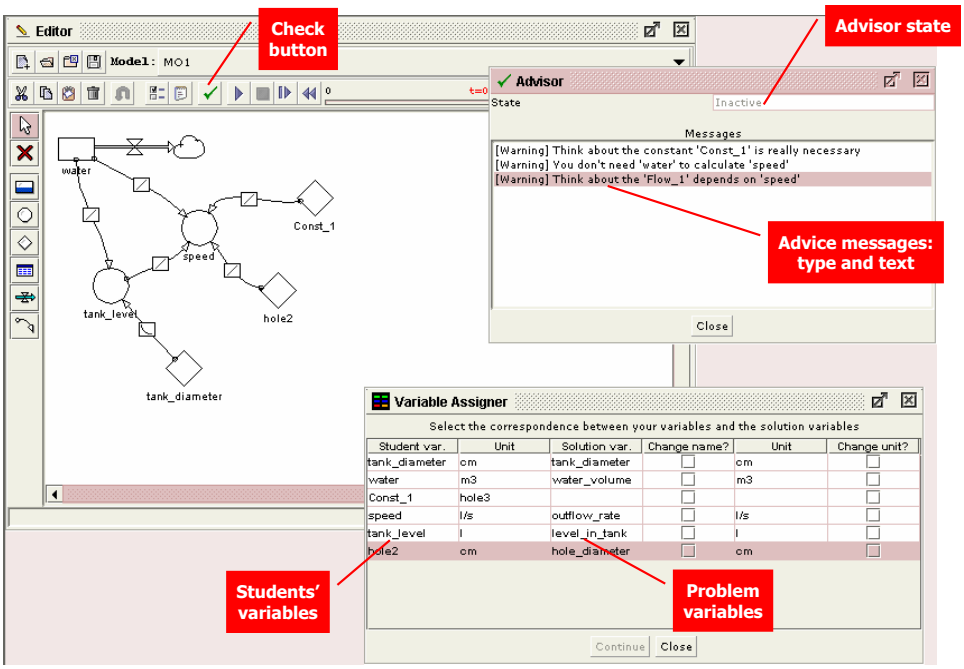

Fig. 6. Example of a model and the advice generated.

The three advice messages in the example in Fig. 6 ("Think about the constant 'Const_1' is really necessary", "You don't need 'water' to calculate 'speed' ", "Think about the 'Flow_1' depends on 'speed' ") highlight some differences between the student's model and the reference model: on the one hand, the flow should be connected to an auxiliary; on the other hand, a constant and a connector are not necessary.

5. Evaluation

In order to validate the advice approach, we carried out a summative evaluation. This kind of evaluation aims at assessing the effectiveness of the system [27]. In doing so, we performed two studies. A more detailed description of these studies can be read in Bravo et al. [28].

The first study consisted of applying the advice method, integrated in Co-Lab, to some final student solutions for a specific modeling problem. The objective of this assignment was to model the behavior of a tank that lost water by a hole. Three experienced teachers evaluated the advice that was generated. A total number of 21 solutions were analyzed in this study. However, only 6 of them were syntactically and semantically correct, so that advice could only be generated in 6 cases. A total of 36 advice messages were generated for these 6 cases, distributed over 8 errors, 26 warnings, and 2 comments. No filtering criterion was selected and the sorting criterion was the advice importance (error, warning and comment).

The main results of this evaluation were:

– The aliases approach for students' variables was promising, since the method was able to link 74% of the students' variables automatically to the reference solution variables in this real modeling task. This is a positive result since a simple technique for matching variable names is used, instead of using complex natural language processing techniques. In addition, the student will always have the capability to complete the linking of variables manually.

– The teachers said that the general approach as well as the advising style was "fine". However, they agreed on the necessity of considering a wider variety of reference solutions, since there are many different ways of modeling the behavior of a given system. They also highlighted that students should be helped more to think about the system in the initial phase.

The second study consisted of using the system to build a model of a physics system. In this case the model being studied was the discharge process of a condenser. Twelve students took part in the study. They had previously received instruction in the use of the modeling tool. The modeling task took one hour. During this time they requested advice 102 times, receiving 396 advice messages. There were an average of 8.5 checks and 33 advice messages per user, and an average of 3.9 advice messages per check.

The main results of our analysis were the following:

– All students used the Advisor frequently. They got used to making *checks* and they took into account the different advice messages shown in the majority of the cases.
– The advice guided the students to correct solutions. After one hour of working, 83.3% of the students were able to complete the model. Therefore, we believe that the advice was successful.
– The students had a positive view of the advice support. In a questionnaire with a five-point Likert scale ranging from 1 (completely disagree) to 5 (totally agree), they expressed their opinion about (1) whether the language used by the Advisor in its advice messages was understandable and clear, (2) whether the advice or help they had received was appropriate for their model and its state, (3) whether they thought that the help they had received was good, (4) whether the advice they had received was similar to that a teacher would give, and (5) whether they believed that the Advisor had made them work and reflect. The average scores for answers to these questions are shown in Table 8.

Question	M	SD
Do you think that the language used by the Advisor in its advice messages was understandable and clear?	3,81	0,81
Do you think that the advice or help you have received was appropriate for your model and to its state?	3,67	0,73
Do you think that the help received was good?	3,61	0,70
Do you think that the advice you received was similar to what a teacher would give you?	3,00	0,84
Do you think that the Advisor has helped you to improve your model making you reflect and work so that you can reach a solution?	4,10	0,62

Table 8. Average scores for answers to the questions on the post-test.

6. Conclusions and future work

In this article we have dealt with modeling and simulation-based learning environments. In these environments students can have difficulty building models for a number of reasons. Therefore, supportive measures to scaffold the modeling tasks are required. Our initiative for such support has been the design of an intelligent advice method to be integrated in these environments. The main characteristics of this method are:

– Each modeling problem in the memory of the learning environment includes a family of solutions (reference solution).
– The student's solution for a modeling problem is compared to the reference solution. In this matching a set of *differences* are generated. We have proposed a heuristic to tackle the problem of matching variable names.

– The strategy for generating advice is based on an advice knowledge base containing a set of advice rules that define which advice messages to give based on the differences detected and the present advice mode.
– This strategy is completed with the definition of different criteria to sort and filter the advice messages.

The aim has been to design a type of advice that makes students work and reflect instead of looking at the advice and mindlessly executing the indicated behavior. However, since some students can require more directed help, the detail level and imperative tone of the advice increase if time goes without improvement of the model's quality. The method is flexible and configurable to fit the requirements of different teaching styles, modeling problem types and learning goals. All method parameters are included in XML files for an easy configuration.

We have carried out some studies with this method to validate the approach. The results confirm that the method is promising and can produce good results in specific content areas and for specific modeling problems, guiding the students to correct solutions. Some teachers stated that the advice approach and style were good, although they recommended a greater variety of reference solutions to approach a wider problem area. Students highlighted that the Advisor made them reflect and work on the model, pushing them to improve it. However, more work is required for the advice to become more similar to the natural advice of a teacher (see Table 8). The method used also presents some limitations that need to be addressed. For example, we should find a solution for the case where a modeling problem has more than one family of possible solutions, allowing the application of this method to more complex problems. The advice procedure should be also able to generate advice even when the model contains syntactic or semantic errors.

We believe that this advice can improve and scaffold the modeling process. The advice challenges students to improve their models, and, in collaborative sessions, to discuss the advice and the best way of proceeding. Although the advice system was designed to work within Co-Lab, a collaborative environment, its working does not depend on this collaborative nature. Whereas the advice given in a collaborative setting may stimulate discussion between students, in an individually-oriented working environment a student may also be stimulated to reflect on the model and modeling process, based on the feedback generated by the advice system.

We are also studying other domain and application areas in which this approach can be applicable and useful. The matching procedure is domain-specific, since its comparison process depends on the domain represented as a set of specific object classes and relationships, although this matching algorithm, as described in Subsection 4.1.2, can be easily adapted to other domains. On the contrary, the advice generation procedure is domain-independent. It uses a list of differences to generate advice messages without knowing anything about the domain. Thus, this approach could be applied to other areas for which a domain model and a matching algorithm can be devised. Along this line, the next step is to make the solution matching process domain-independent by means of appropriate formalisms for describing domains, solutions and matching algorithms in a more abstract and declarative way.

Acknowledgements

The authors would like to express their gratitude to Jakob Sikken, Ard Lazonder, Sarah Manlove, Petra Hendrikse, Sylvia van Borkulo (University of Twente), Elwin Savelsbergh (Utrecht University), Thorsten Bell (IPN, Kiel) and Ernesto Martin (University of Murcia) for supporting this research. The work reported here was done during the stay of Crescencio

Bravo at the Department of Instructional Technology of the University of Twente. We would also like to thank the students and teachers of University of Castilla – La Mancha for their participation in the evaluation experiments.

References

1. Hansen, E. The role of interactive video technology in higher education: Case study and proposed framework. Educational Technology. 1990; 30(9):13-21.
2. de Jong, T., van Joolingen, W.R. Discovery learning with computer simulations of conceptual domains. Review of Educational Research. 1998; 68:179-201.
3. de Jong, T. Computer simulations - Technological advances in inquiry learning. Science. 2006; 312:532-533.
4. de Jong, T. Scaffolds for computer simulation based scientific discovery learning. In: Elen, J., Clark, D., editors. Handling complexity in learning environments: research and theory. London: Elsevier Science Publishers, London; 2006. p. 107-128.
5. Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R. Cognitive Tutors: Lesson Learned. Journal of the Learning Sciences. 1995; 4 (2):167-207.
6. Mitrovic, A., Mayo, M, Suraweera, P., Martin, B. Constraint-Based Tutors: a Success Story. In: Monostori, L., Váncza, J., Ali, M., editors. Engineering of Intelligent Systems: 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (LNCS); 2001; Budapest, Hungary. Berlin: Springer, p. 931-940.
7. Ohlsson, S. Constraint-based student modeling. In: Greer, J.E., McCalla, G., editors. Student modeling: the key to individualized knowledge-based instruction. Berlin: Springer-Verlag, NATO ASI Series; 1994. p. 167-189.
8. Mitrovic, A., Koedinger, K., Martin, B. A comparative analysis of cognitive tutoring and constraint-based modeling. In: Brusilovsky, P., Corbett, A., Rosis, F.D., editors. User Modeling 2003: 9th International Conference (LNAI); 2003; Johnstown, PA, USA. Berlin: Springer, p. 313-322.
9. Toth, J.A., Suthers, D., Weiner, A. Providing Expert Advice in the Domain of Collaborative Scientific Inquiry. Proceedings of 8th World Conference on Artificial Intelligence in Education; 1997; Kobe, Japan. p. 302-308.
10. Rothenhöfer, D., Herzog, C. SYPROS: An Intelligent Tutoring System for Parallel Programming. In: Chan, T.W., editor. Proceedings of International Conference on Computers in Education; 1993; Taiwan. p. 300-305.
11. Suraweera, P., Mitrovic, A. An Intelligent Tutoring System for Entity Relationship Modelling. International Journal of Artificial Intelligence in Education. 2004; 14:375-417.
12. Bravo, C., Redondo, M.A., Ortega, M., Verdejo, M.F. Collaborative environments for the learning of design: A model and a case study in Domotics. Computers and Education. 2006; 46 (2):152-173.
13. van Joolingen, W.R., de Jong, T., Lazonder, A.W., Savelsbergh, E.R., Manlove, S. Co-Lab: research and development of an online learning environment for collaborative scientific discovery learning. Computers in Human Behavior. 2005; 21:671-688.
14. Bravo, C., Redondo, M.A., Ortega, M., Verdejo, M.F. Collaborative Distributed Environments for Learning Design Tasks by Means of Modelling and Simulation. Journal of Networks and Computer Applications. 2006; 29 (4):321-342.
15. Forrester, J.W. Industrial Dynamics. Waltham, MA, USA: Pegasus Communications; 1961.
16. Forrester, J.W. Collected Papers of Jay W. Forrester. Waltham, MA, USA: Pegasus Communications; 1975.
17. Löhner, S., van Joolingen, W.R., Savelsbergh, E.R. The effect of external representation on constructing computer models of complex phenomena. Instructional Science. 2003; 31:395-418.
18. Suraweera, P., Mitrovic, A. KERMIT: A Constraint-Based Tutor for Database Modeling. In: Cerri, S.A., Gourdères, G., Paraguaçu, F., editors. Intelligent Tutoring Systems (LNCS); 2002; Biarritz, France. Berlin: Springer, p. 671-680.
19. Koedinger, K.R., Aleven, V., Heffernan, N., McLaren, B., Hockenberry, M. Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. In: Lester, C.L., Vicari, R.M., Paraguaçu, F., editors. Intelligent Tutoring Systems (LNCS); 2004; Maceió, Alagoas, Brazil. Berlin: Springer, p. 162-174.
20. Lieberman, H., editor. Your Wish is My Command: Programming by Example. Morgan Kauffman Publishers; 2001.
21. Blostein, D., Fahmy, H., Grbavec, A. Issues in the practical use of graph rewriting. In: Cuny, J., Ehrig, H., Engels, G., Rozenberg, G., editors. Graph Grammars and Their Application to Computer Science (LNCS); 1996; Williamsburg, VA, USA. Berlin: Springer, p. 38-55.

22. de Lara, J., Vangheluwe, H. AToM^3: A Tool for Multi-Formalism and Meta-Modelling. In: Kutsche, R.D., Weber, H., editors. Fundamental Approaches to Software Engineering (LNCS); 2002; Grenoble, France. Berlin: Springer, p. 174-188.
23. Matsuda, N., Takagi, S., Soga, M., Hirashima, T., Horiguchi, T., Taki, H., et al. Tutoring System for Pencil Drawing Discipline. Proceedings of International Conference on Computers in Education; 2003; Hong Kong. p. 1163-1170.
24. Constantino-González, M.A., Suthers, D.D., Icaza, J.I. Coaching Web-based Collaborative Learning based on Problem Solution Differences and Participation. In: Moore, J.D., Redfield, C.L., Lewis Johnson, W., editors. Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future; 2001; San Antonio, Texas, USA. IOS Press, p. 176-187.
25. Paolucci, M., Suthers, D., Weiner, A. Automated Advice-Giving Strategies for Scientific Inquiry. In: Frasson, C., Gauthier, G., Lesgold, A., editors. Intelligent Tutoring Systems (LNCS); 1996; Montréal, Canada. Berlin: Springer, p. 372-381.
26. Suthers, D. Preferences for Model Selection in Explanation. Proceedings of 13th International Joint Conference on Artificial Intelligence; 1993; Chambery, France. p. 1208-1213.
27. Mark, M.A., Greer, J.E. Evaluation Methodologies for Intelligent Tutoring Systems. Journal of Artificial Intelligence in Education. 1993; 4 (2/3):129-153.
28. Bravo, C., van Joolingen, W.R., de Jong, T. (submitted) Using Co-Lab to build System Dynamics models: Students' actions and on-line tutorial advice. Perspectives on collaborative inquiry learning: An International network for research, curriculum and technology. Bell, T., Slotta, J., Schanze, S., editors. Special issue submitted to International Journal of Science Education.

Short biography

Crescencio Bravo is associate professor of Computer Systems and Languages at the University of Castilla – La Mancha, Computer Engineering School. He has focused his research in the development of structured mechanisms to support collaborative learning and of computational methods to analyze the collaboration process and the resulting products. Recently, he has investigated the generation of intelligent advice to students during modeling tasks in problem solving settings.

Wouter van Joolingen is associate professor of Instructional Technology at the University of Twente, Faculty of Behavioral Sciences. He has worked on the development of both the SimQuest and Co-Lab learning environments for the support of (Collaborative) Inquiry Learning. He investigates the role of computer-supported modeling by learners in inquiry learning environments, with a special interest in the influence of model representations.

Ton de Jong is full professor of Instructional Technology and of Educational Psychology at the University of Twente, Faculty of Behavioral Sciences. His main interests are in problem solving in science, scientific discovery (computer-simulation based) learning environments, learners' cognitive processes, and instructional design. Ton de Jong is (co-)author of over 100 journal papers and book chapters and was the editor of three books. He is associate editor of Instructional Science, and is on the editorial boards of the Journal of Computer Assisted Learning, Contemporary Educational Psychology, International Journal of Artificial Intelligence in Education and Educational Research Review.