Journal of Universal Computer Science, vol. 18, no. 10 (2012), 1279-1307
submitted: 23/5/11, accepted: 24/5/11, appeared: 28/5/12 © J.UCS

Automating the Analysis of Problem-solving Activities in Learning Environments: the Co-Lab Case Study

Rafael Duque
(University of Cantabria, Santander, Spain
rafael.duque@unican.es)

Lars Bollen
(University of Twente, Enschede, The Netherlands
l.bollen@utwente.nl)

Anjo Anjewierden
(University of Twente, Enschede, The Netherlands
a.a.anjewierden@utwente.nl)

Crescencio Bravo
(University of Castilla-La Mancha, Ciudad Real, Spain
crescencio.bravo@uclm.es)

Abstract: The analysis of problem-solving activities carried out by students in learning settings involves studying the students' actions and assessing the solutions they have created. This analysis constitutes an ideal starting point to support an automatic intervention in the student activity by means of feedback or other means to help students build their own knowledge. In this paper, we present a model-driven framework to facilitate the automation of this problem-solving analysis and of providing feedback. This framework includes a set of authoring tools that enable software developers to specify the analysis process and its intervention mechanisms by means of visual languages. The models specified in this way are computed by the framework in order to create technological support to automate the problem-solving analysis. The use of the framework is illustrated thanks to a case study in the field of System Dynamics where problem-solving practices are analysed.

Keywords: Computer-supported learning environments, analysis of problem-solving activities, model-driven development, visual languages.
Categories: L.0.0, L.3.4

1 Introduction

Computer-supported learning environments are now widely used because of their potential not only to facilitate the communication between learners and teachers or to create common information repositories but also to provide workspaces where learners can create and manipulate artefacts to produce a solution to solve a problem proposed previously by a teacher [de Jong, 98]. These learning environments can also integrate analysis features [Duque, 12] that allow the characterization of the learners' activities and provide feedback about the impact of the students' actions and about the properties of the solutions produced [Arts, 02]. The feedback information, therefore, can serve as an evaluation of the work carried out and of the fulfilment of the problem goals, which is intended to improve the knowledge building of the learners [Bravo, 09]. These analysis processes can be particularly useful in pedagogical paradigms such as discovery-based learning, which proposes that learners build their own knowledge through the active participation in problem-solving processes [Dean, 06]. Thus, the analysis process enables the learners to build knowledge by means of feedback about the impact of their actions, aimed at building and manipulating artefacts to solve a problem.

A process aimed at producing feedback in the aforementioned learning settings usually follows an *observation-abstraction-intervention* life cycle [Bravo, 08]. The *observation* phase captures the actions carried out by the students and the solutions created. The *abstraction* phase calculates analysis indicators [Dimitracopoulou, 04] from the information captured usually in the form of raw data. Analysis indicators are variables that characterize aspects of the problem-solving process (e.g., speed, amount of work, etc.) and of the solutions (e.g., quality, size, cost, etc.). Finally, the *intervention* phase focuses on producing feedback, evaluating the students' work and on providing advice on how to better approach a successful problem-solving process. The *intervention* can also include other kinds of actions such as to inform the teacher about the students' work or, as a more advanced technique, to adapt or modify the system's behaviour or user interface to improve the problem-solving activity.

Implementing an analysis system from scratch for analysing problem-solving activities as mentioned above involves costly and difficult tasks. It is necessary to gather large amounts of data about students' actions [Avouris, 05], to infer complex indicators to characterize the students' work and solutions produced [Dimitracopoulou, 05], and to design and implement a set of suitable intervention mechanisms [Mørch, 03] . In order to deal with the cost and effort of these tasks, we looked at the challenge of producing computational support that enables the automation of the *observation-abstraction-intervention* phases.

This article presents a computational framework that allows software developers to specify and automate the problem-solving analysis and the generation of feedback. According to [Booch, 05] [Bosch, 00], a framework is a partially complete software system that can be extended through the instantiation of specific *plug-ins*. Our analysis framework, based on the model-driven paradigm [Mellor, 04], includes computational models that are the *plug-ins* to be instantiated for the generation of software systems to perform the process of analysis. To that end, the framework includes a set of authoring tools that support the specification of models of the analysis processes to be automated. Once specified, the models are processed by a number of transformation tools and, in this way, a software system implementing the analysis specified is produced. To evaluate the proposal, we present a case study where the framework is used to analyse the problem-solving processes supported by Co-Lab [van Joolingen, 05], a computer-supported learning environment that supports problem-solving processes in the domain of System Dynamics.

Section 2 of this article reviews the major scientific contributions related to problem-solving analysis and creation of computational support to automate this kind of analysis. Section 3 presents our approach to specify and automate the problem-solving analysis by means of using models. Section 4 describes a case study showing the application of the framework for the analysis of the problem-solving activity supported by Co-Lab. Finally, Section 5 presents the conclusions drawn from the work carried out and discusses a future research line.

2 Related work

Computer-supported learning environments have integrated analysis features for various purposes such as studying how a group of learners collaborate on a common task or producing feedback that recommends efficient ways of working in groups [Soller, 05], trying to identify which kind of actions a student performed to solve a problem [Muehlenbrock, 05] or to characterize the composition of the solutions designed by the learner [Avouris, 02]. The effective execution of this type of analysis involves establishing methods and computational models to carry out the analysis automatically. Computer-supported learning systems have traditionally followed two different approaches to automate this kind of analysis. The first approach, followed by an important number of ITSs (Intelligent Tutoring Systems), is based on the use of computational models of the application domain [Ohlsson, 94], which is made up of the elements that the students can manipulate to build solutions. In this approach, computational rules are also used to constrain how the students can use the application domain components. In this case, the system decides whether a student action denotes a violation of the constraints and consequently the student should be advised to correct the situation. The so-called constraint-based techniques analyse the validity of the current state of the solution rather than the entire process that leads up to the current state. The computer-supported learning systems falling in the second category are the cognitive tutors [Anderson, 95], which focus on analysing the students' work process and, depending on the results of this analysis, showing feedback to correct the problems identified.

We now review a set of computer-supported learning systems selected from the literature to illustrate the large amount of proposals that have applied both analysis approaches to different domains. For instance, the constraint-based techniques have been applied to diverse domains such as database modelling [Mitrovic, 04], natural language [Menzel, 06], learning foreign languages [Nicholas, 06], UML class diagrams [Le, 06], programming [Le, 10], and Thermodynamics [Mitrovic, 11]. In order to enable the automatic building of this type of ITSs, the WETAS system [Mitrovic, 07] provides a web-based shell that allows the developers to specify computational models to define the application domain. However, this system does not address the automatic construction of computational constraints on how to manipulate the application domain components. This point is addressed by the ASPIRE system [Mitrovic, 09], which enables the instantiation of ontologies to define constraints on how the student should handle the application domain components to achieve a satisfactory solution.

The cognitive tutors usually process the actions carried out by the student and evaluate various aspects of their activity. Thus, for instance, [Conati, 09] have developed an intelligent agent to recognize different student emotions during the interaction with an educational computer game. In other cases, the student is not required to create an artefact to solve a specific problem, but he/she has to carry out a specific working process that is analysed by the system. For example, [Remolina, 04] presents a tutoring system that supports flight instruction in which the student pilots an airplane and this process should be analysed. Following a different approach more similar to ours, in order to enable software developers to produce computational support for analysing the student activity, the Natural-K system [Jung, 10] has an interface that allows one to define rules in natural language on how to analyse the work process. Then, the system automatically transforms these rules into computational support to automate the analysis.

In summary, we observed two trends to produce feedback to help students in their problem solving tasks. The first one is oriented to the display of messages that provide information about how to improve the quality of the solution built by the student. The second one is focused on building feedback to improve the students' work process. In both approaches, the construction of the tutoring system is costly from the developer‘s point of view because it is necessary to define and implement a number of computational components such as the application domain, the problem to be solved, the procedures of analysis of the activities and solutions, etc. At this point, we observed partial solutions to automate this task, such as those aimed at defining the application domain by means of a shell or using ontologies to define constraints or rules in natural language. However, there is a lack of comprehensive proposals that allow a developer to easily specify the analysis process, for example by means of visual languages, and automate it as much as possible. This situation led us to design and build a framework whose main contribution is to enable the configuration and automation of procedures that analyse both the students' work and the resulting solutions in order to produce feedback in some way. In both approaches there is a high degree of dependence between the feedback support and the specific domain and tasks supported by the computer-supported system, so that the different development efforts cannot be easily reused. To avoid this, our framework was developed so that it was independent of the system in which the framework could be integrated to analyse problem-solving activities. Therefore, this framework contributes to the literature because the software developers only need to instantiate a set of models and they will not have to implement the source code of the analysis system. The analysis will evaluate the students' solutions and their work process. Finally, the framework will be an independent support that can be applied to different learning environments.

3 A framework for analysing problem-solving activities

The framework we present here allows a software developer to automate the analysis of problem-solving processes carried out by the students. The framework is a software system whose functionality is auto-configured according to a number of analysis models provided by the developer, with the help of a teacher, by means of authoring tools. Therefore, the following users' roles interact with the framework (Figure 1):

- Student: The students' activity with the computer-supported learning system must be observed to generate data repositories that describe the actions and the resulting products.

- Teacher: He/she is interested in evaluating the students' activity. The teacher specifies the characteristics of the problem that should be solved by the student.
- Developer: Implements the analysis process by means of the specification of computational models. Generally, the teacher provides the developer with information about the features of the analysis to be automated.

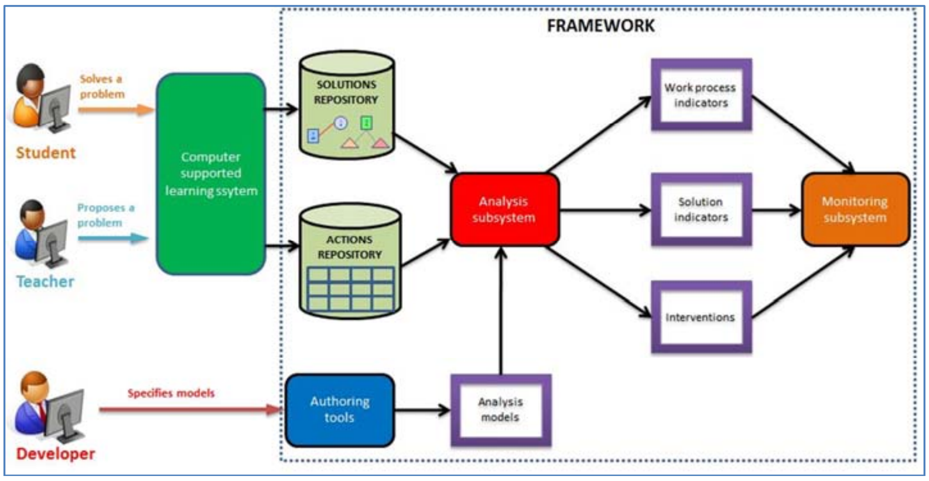


Figure 1: Users' roles of the framework.

The functional architecture of the framework is made up of a set of levels. These levels include a set of subsystems that develop a well-defined functionality. The information generated at a level is processed by another level to carry out their functions. Therefore, this architecture establishes information flow between levels. These three levels are the following:

- Meta-information level: At this level, the analysis is configured according to the features of the collaborative process and the analysis goals. This level includes the Design subsystem, which consists of a set of authoring tools that enable the specification of models that describe the artefacts the learner can create and modify, the characteristics of the problems to be solved, how to calculate indicators that analyse the student's work and solutions, and the interventions considered to provide feedback.
- Analysis level: At this level, the framework builds the computational support that performs the analysis defined in the meta-level information. This level includes an Analysis subsystem that processes the models instantiated at the meta-information level, the actions carried out by the student, and the resulting artefacts are processed to calculate indicators and to define interventions. To that end, it is necessary that the computer-supported learning system is capable of storing both the problem-solving actions and the solutions in repositories that follow a structure predefined by the

framework. Therefore, each computer-supported learning system that stores the solutions and actions generated by the learners in these repositories can be integrated in the framework to automate the analysis processes.

- Interaction level: At this level the framework intervenes in the student' activity according to the results of the analysis process. This level contains a Monitoring subsystem which is responsible for performing the interventions generated at the analysis level.

The Design, Analysis and Monitoring subsystems are described in the following subsections.

3.1 Design subsystem

Using the authoring tools, the developers design computable representations of the analysis process in the form of models created with visual languages. Taking these models as an input, the framework generates an executable analysis system to be integrated into a computer-supported learning application. The implementation of the authoring tools has been carried out using the popular Eclipse integrated development environment. The Eclipse environment provides a framework, called the Eclipse Modelling Framework (EMF), which includes among its features support for the implementation of authoring tools that allow instantiating models. These tools represent the models using the XMI standard (XML Metadata Interchange) for exchanging meta-information using XML. In our case, these models are automatically processed in real time by the framework to implement an analysis system. Therefore, the internal structure of the analysis system depends on the model instantiated. The visual authoring tools integrated in the Design subsystem are the following:

- **Authoring tool for modelling the application domain:** This tool allows one to create a computational model to specify which artefacts are manipulated by the student to build solutions and how these artefacts are made up.
- **Authoring tool for modelling the problem-goal:** This tool allows for the creation of a model that specifies the work process that should be followed by the learner and the characteristics of satisfactory solutions to the problem proposed by the teacher. To that end, the model includes rules each of which defines a characteristic of the work process or solution. A rule can be of two types: *constraint* or *requirement*. On the one hand, a *constraint* rule specifies a prohibition that limits the learner's work process (e.g., to limit the number of accesses to a specific workspace, the time, the use of specific domain components, etc.) or the resulting solution (e.g., to limit the number of elements of the solution, etc.). A *requirement* rule establishes an obligation that affects the student's work process (e.g., the first step of the work process should be to design an outline of the final solution) or the solution (e.g., the solution must include a particular component of the application domain, etc.). These rules can be used by the teacher to specify the most important features the students' solution should fulfil.
- **Authoring tool for modelling the observation:** This tool allows the design of models that specify the subset of student actions that should be captured for analysis purposes. Moreover, this authoring tool enables the classification of the different actions supported by the system according to their meaning

and semantics [Harrer, 04] (e.g., to modify the solution, to test the solution, etc.).

- **Authoring tool for modelling the quantitative abstraction:** This authoring tool allows the developer of an analysis system to specify the calculation of quantitative indicators to characterize the student's activity. The quantitative indicators are usually low-level indicators of four types: (i) information about the time spent by the student in carrying out the learning activities; (ii) information about the actions performed by the student to carry out the activities; (iii) information about the number of specific application domain components used in the construction of a solution; and (iv) information about the number of rules fulfilled or unfulfilled from the ones specified in the problem-goal model. This tool uses mathematical functions (addition, subtraction, multiplication, division, percentage, arithmetic mean and standard deviation) that take a set of low-level indicators as input and high-level quantitative indicators as output.
- **Authoring tool for modelling the qualitative abstraction:** Using this tool, the developer specifies the rules that define how to infer qualitative indicators. Each rule consists of an antecedent and a consequent. The antecedent includes quantitative and/or qualitative indicators calculated previously. The antecedent indicators must take certain values to activate the rule. The consequent is made up of a new qualitative indicator when the rule is activated.
- **Authoring tool for modelling the intervention:** This authoring tool allows the developer to specify the intervention model as a set of intervention mechanisms, each of which is described on the basis of six dimensions: (i) the triggers that specify when the intervention mechanism should be activated (this usually consists of detecting the execution of a certain type of an action or meeting a time cycle); (ii) the conditions necessary to intervene, e.g., that a number of analysis indicators reach specific values; (iii) the form of intervention in the problem-solving process (e.g., showing textual advice, adapting the user interface, etc.); (iv) the target users who will receive the intervention (student, teacher, etc.); (v) the places, i.e., workspaces or tools where the intervention will be carried out; and (vi) the text, advice message or specific information required so that the intervention can take place. For instance, the indicators can inform which rules included in the problem-goal model have been fulfilled and which have not. These rules can specify the features of a correct solution to the problem proposed. Therefore, these indicators can be used in interventions that show a textual advice aimed at correcting the mistakes in the students' solution.

As an example, figure 2 shows an excerpt of the user interface of the authoring tool for modelling the intervention. The authoring tool includes a toolbar on the right with the elements of the visual language used to specify the *intervention* phase. First, the visual language includes icons (Table 1) to specify time cycles or learner's action frequencies, which are the triggers of the interventions. Second, components to express how the analysis indicators should be checked are included. To that end, the visual language represents each analysis indicator as a rectangle in which the developer includes intervals with the maximum and minimum values that the indicator can take. Moreover, the tool enables grouping analysis indicators by means of *and*/*or* logical operators. When the indicators are grouped by an *and* operator, all the values of the indicators must lie outside the intervals to trigger the intervention mechanism. When the indicators are grouped by an *or* operator, a single analysis indicator outside its intervals is enough to perform the intervention mechanism. Finally, the visual language includes a component to define the specific intervention mechanism to be used. This component consists of a rectangular compartment that includes attributes to specify the message or information to be displayed, the place where the intervention will be produced, the users that will receive the intervention and a boolean attribute to express whether the values of the indicators that trigger the intervention should be displayed. A set of relationships labelled with arrow icons are used to define which triggers are used to launch each intervention (*Launch*), which indicators should be analysed to perform each intervention (*Process*) and which indicators are linked by a logical operator (*Connection And*, *Connection Or)*.

Icon	**Semantic**
🕒	Time cycles
⚡	Action frequencies
ℹ	Analysis indicator
⬆	Maximum value
⬇	Minimum value
ᗡ	And operator
≫	Or operator
⟲	Intervention mechanism
✉	Message
🖥	Intervention place
👤	User
🔍	Boolean attribute

Table 1: Visual language to specify the intervention phase.

Figure 2 also shows an example in which a developer is defining the intervention support in a problem-solving activity. It can be seen how the developer has specified an intervention to advise the student that a set of tasks should be carried out and only a few of them are being approached. In doing so, the developer uses a trigger element that specifies that at every 300-second time cycle, the intervention mechanism be activated. Then, the intervention mechanism checks for the value of two analysis indicators. The first indicator is *Task_division*, which evaluates the number of actions carried out by the student in each task. The second indicator is *Time_distribution*, which evaluates how the student has spent the time between the different tasks. When both analysis indicator values lie outside the specified ranges, the intervention takes place and an advice message is shown to the student containing the following suggestion: "*Plan your activities better. You must not focus your efforts on a single task. Review the indicators and try to work on those tasks where you have not worked* *enough*". Finally, the developer specifies that this intervention is shown to the student by means of the Monitoring subsystem and that the values of the related analysis indicators are also displayed. A description in depth of the authoring tools integrated in the Design subsystem can be read in [Duque, 11].

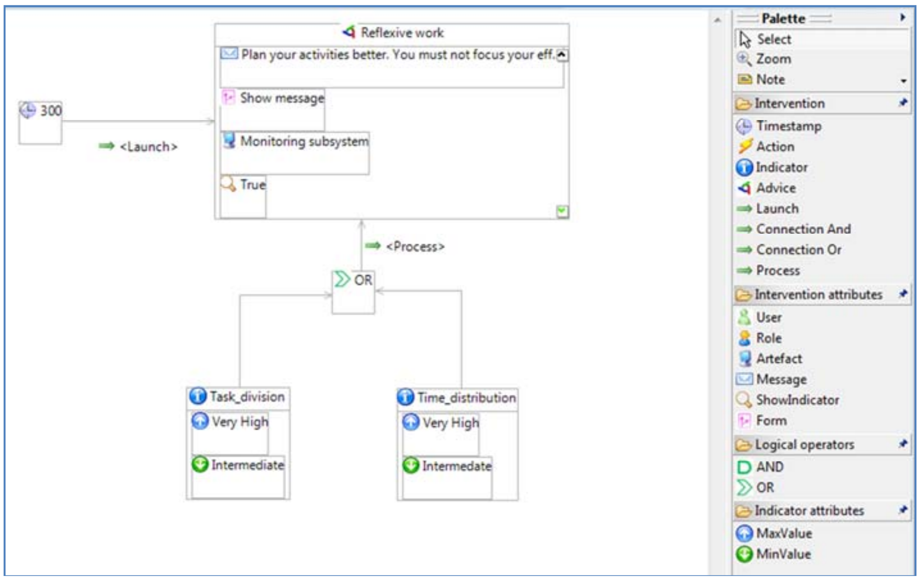


Figure 2: Authoring tool for specifying interventions.

3.2 Analysis subsystem

This subsystem processes the models designed by means of the authoring tools included in the Design subsystem and, according to them, implements the three phases of the analysis life cycle: *observation*, *abstraction* and *intervention*. The Analysis subsystem processes the observation model to determine the subset of actions stored in the repository (see Figure 1) that are needed for the analysis process. The Analysis subsystem also processes the application domain model in order to know which artefacts can be used by the student to build solutions and should therefore be analysed. The repositories of the framework (Figure 1) store basic data about the students' actions (who performs the action, name of the action, time when the action is performed, etc.) and about how the application domain components are manipulated to build a solution.

To automate the *abstraction* phase, the Analysis subsystem processes the quantitative and qualitative abstraction models, as briefly presented in subsection 3.1. The *abstraction* phase is completed by the processing of the problem-goal model that specifies the rules that characterize the type of solution to be built for the problem, so that violations of the established rules can be detected. In summary, the final aim of the *abstraction* phase is to compute a number of high-level analysis indicators as the main information for the *intervention* phase.

Finally, this subsystem processes the intervention model and evaluates whether it is necessary to apply an intervention. This is carried out by testing the corresponding conditions as presented above and triggering the suitable intervention mechanisms.

3.3 Monitoring subsystem

The Monitoring subsystem automates the intervention actions specifically aimed at showing advice messages about the problem-solving process. For this purpose, functionalities for building user interfaces to show feedback are included. Such user interfaces are built with the information provided by the Analysis subsystem. The developers can choose to use this support to visualize advice messages and/or analysis indicators or, instead, use all the information provided by the analysis subsystem freely, creating specific functionalities for monitoring and advice within the learning environment.

Figure 3 illustrates how the Monitoring subsystem generates a user interface when the target user of the intervention is a student. The user interface is structured in three main areas: (i) a list of analysis indicators, (ii) the representation of the value of the selected indicator, and (iii) the comment or advice message.

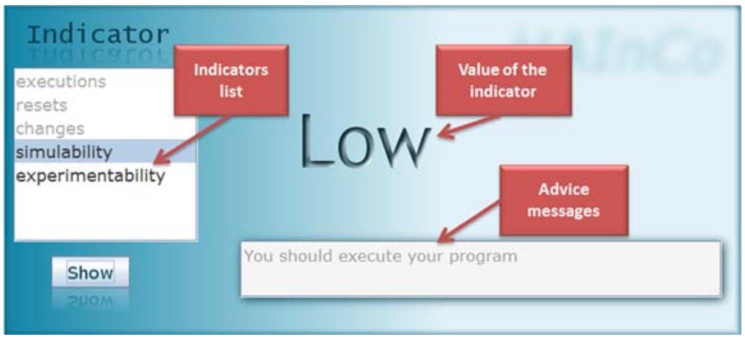


Figure 3: Example of an advice message for the student including the values of the related indicators.

The list of indicators includes the name of those indicators whose values have motivated the intervention mechanism. When the student selects an indicator from the list, the value of the indicator is displayed. At the bottom part of the user interface, an advice message explaining the purpose of the indicator in more detail is shown.

4 Study

A study was prepared to evaluate the functionalities of the framework for characterizing a problem-solving process in a learning environment including intervention support to improve the student's learning process. According to the proposal of [Yin, 94], we followed these steps to carry out this study:

- Determine and define the research questions: This case study tries to answer two research questions. The first question asks if the framework is a suitable computational support for the automation of the analysis of problem-solving activities, which includes intervention and feedback. The second question asks if the analysis outcomes (interventions, analysis indicators, etc.) are useful in the problem-solving process.
- Select the cases and specify data gathering and analysis techniques: We designed a case in which the Co-Lab system was used to solve a problem in the System Dynamics domain [van Joolingen, 05]. Co-Lab stores the actions and solutions produced by the students and for this reason the analysis framework can be integrated with it. We carried out an experiment that tests if the framework enables the automatic execution of analysis in Co-Lab and the adequacy of the analysis outcomes.
- Prepare to collect the data: A teacher proposed a problem in the System Dynamics domain that could be solved using Co-Lab. A software developer configured the framework to automate the analysis process.
- Collect data in the field: A group of 40 students used Co-Lab to solve the problem proposed by the teacher. The students' actions and the resulting solutions were collected and analysed. These students received the intervention actions of the analysis during the problem-solving process.
- Evaluate and analyse the data: With the aim of empirically studying the analysis carried out in this case study, we built a tool that replays the work processes of the students and the interventions they received. This enabled an evaluation of the analysis process.
- Prepare the report: Finally, we show the development of the case study and its results.

The following four sections describe the study development following this outline: (i) *definition of the case study,* in which the learning environment and the application domain are described; (ii) *evaluation method,* in which the participants and evaluation techniques are defined; (iii) *conduction of the study,* in which the participants carry out their tasks and data and evidence are recorded; and (iv) *discussion of the results*, in which the results of the analysis process are discussed and the fulfilment of goals is evaluated.

4.1 Case study

Co-Lab supports problem-solving processes by providing a modelling tool that is able to create and simulate System Dynamics models. System Dynamics [Forrester, 85] is used as part of a methodology that addresses the modelling of complex systems, which consist of several interconnected elements. As a result of interactions between elements, new properties that cannot be explained from the properties of isolated elements emerge. Examples of complex systems are biological populations or economic systems. System Dynamics models are typically made up of the following entities:

- Stocks: They represent a system variable whose value changes over time.
- Auxiliaries: They correspond to mathematical functions.
- Constants: These represent constant values in a model which do not change over time.
- The associations between entities in System Dynamics are:
    - Relationships: They connect entities to make values accessible between entities.
    - Flows: Flows are connections between stocks that represent the change of values over time by decreasing one stock and increasing another, thus building an ordinary differential equation (ODE).

The Co-Lab system is made up of three basic tools. The first one supports the creation of System Dynamics models through the direct manipulation of graphical elements. As shown in Figure 4, this tool provides an editor in which the student inserts, deletes and modifies the components of the model.

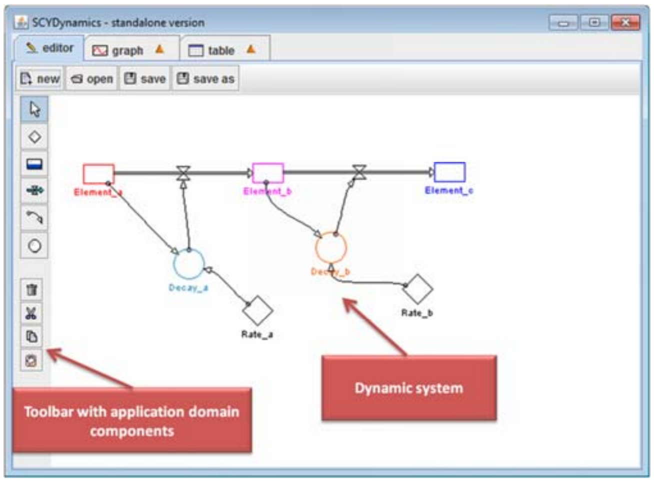
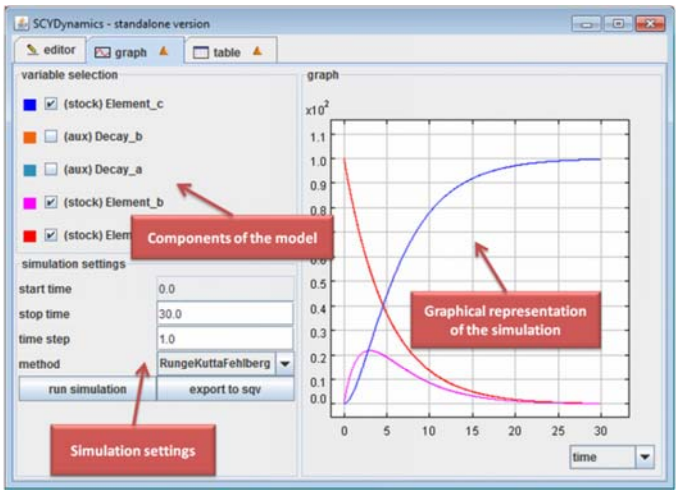


Figure 4: Workspace for modelling a dynamic system.

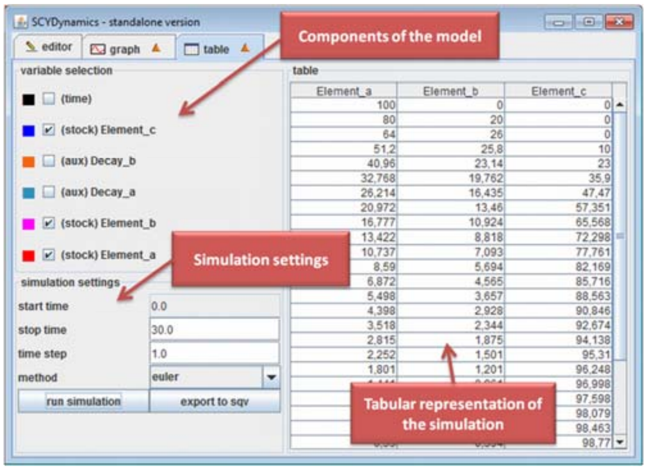
The second tool (Figure 5) enables a student to graphically simulate the models. To this end, the student defines time intervals in which the model is simulated in a process in which the output data are plotted on Cartesian axes. This tool allows the user to select a subset of the components of the model and to apply different methods of iterative approximations of ODEs.



start time	0.0
stop time	30.0
time step	1.0
method	RungeKuttaFehlberg

Figure 5: Workspace for graphical simulation of models.

The third tool (Figure 6) integrated in Co-Lab enables a student to simulate the models designed and visualize the results in tabular form. The student selects the components to be simulated and the simulation methods to be applied, but in this case the results are displayed in a table which shows the value that each variable produces per unit time.



start time	0.0
stop time	30.0
time step	1.0
method	euler

Element_a	Element_b	Element_c
100	0	0
80	20	0
64	26	0
51,2	25,8	10
40,96	23,14	23
32,768	19,762	35,9
26,214	16,435	47,47
20,972	13,46	57,351
16,777	10,924	65,568
13,422	8,818	72,298
10,737	7,093	77,761
8,59	5,694	82,169
6,872	4,565	85,716
5,498	3,657	88,563
4,398	2,928	90,846
3,518	2,344	92,674
2,815	1,875	94,138
2,252	1,501	95,31
1,801	1,201	96,248
[illegible]	[illegible]	96,998
[illegible]	[illegible]	97,598
[illegible]	[illegible]	98,079
[illegible]	[illegible]	98,463
[illegible]	[illegible]	98,77

Figure 6: Workspace for tabular simulation of models.

4.2 Method

To achieve the study goals, three different kinds of actors were involved. Firstly, a computer engineer used the framework to design the required analysis support.

Secondly, a teacher provided a problem description and an ideal solution. Thirdly, a group of 40 students used Co-Lab to solve that problem, which requires the students to model the process of warming the Earth and to simulate its environmental impacts. To implement this objective, the teacher provided the students with a set of basic data about the factors influencing global warming (climate model, albedo factor, capacity and thermal equilibrium, etc.). As mentioned, the teacher provided the developer with an ideal solution to the problem, which is represented in Figure 7.

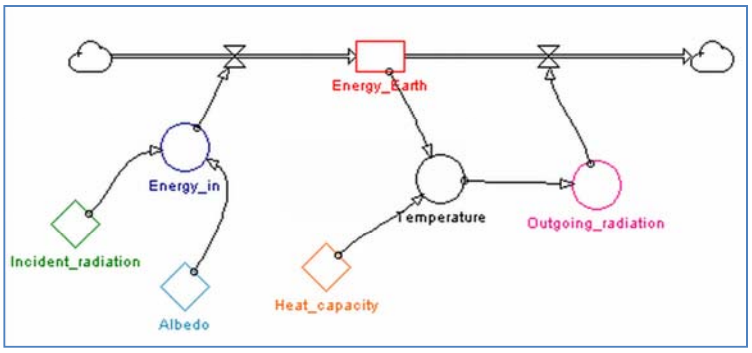


Figure 7: Ideal solution to the problem.

After defining the problem-solving task and the participants involved, subsection 4.3 presents the models built by the developer when designing the problem-solving analysis support and describes a simulation of the intervention provided by that analysis support. Subsection 4.4 discusses the results.

4.3 Designing a problem-solving analysis support for Co-Lab

By applying the framework, the developer started modelling the application domain of System Dynamics with the corresponding authoring tool. This model includes the three kinds of entities of the application domain (*stock*, *auxiliary* and *constant*) (see subsection 4.1) and specifies the possible connections between them by means of two associations (*flow* and *relationship*). Figure 8 shows an excerpt of the application domain model as represented by the authoring tool for modelling the application domain as well as the toolbar of the authoring tool. The toolbar includes icons (Table 2) to represent the domain, the entities and the attributes of both entities and associations.

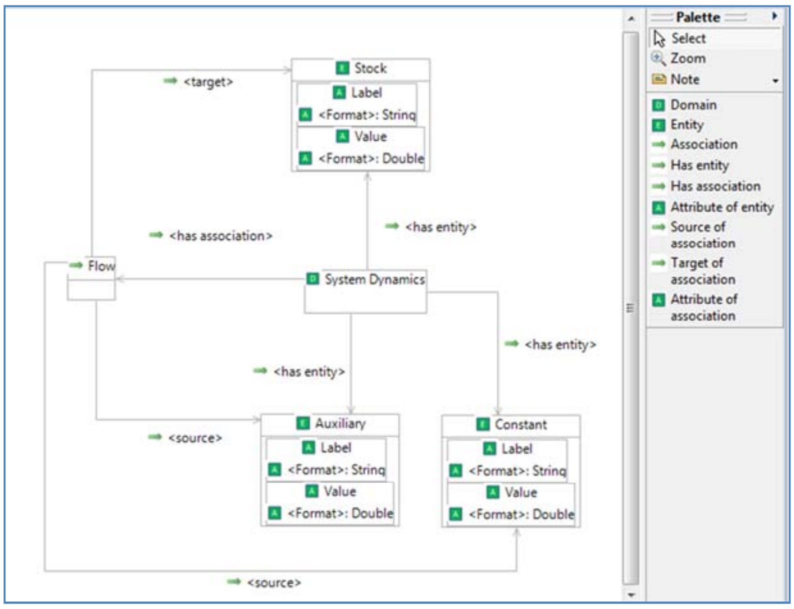


Figure 8: Excerpt of the application domain model.

Different icons with arrows allow the modeller to express (i) the associations of the domain (*Has association*), (ii) the source and target entities of a given association (*Source of association* and *Target of association*) and (iii) the entities of the domain (*Has entity*). The entities of the domain have two attributes: a label for the name of the entity and a numerical value associated with the entity.

Icon	**Semantic**
D	Domain
E	Entities
A	Attributes

Table 2: Visual language to specify the application domain.

Then, the developer specified a problem-goal model that defines four rules that express *requirements* and *constraints* of the student's work when building the solution to the problem. The first rule requires that the student include a *stock* component in the model. The second rule regulates the solution to include three *auxiliaries* as a maximum. The third rule specifies that the learner use at most seven *relationships* in the model. Finally, the forth rule requires that the model include two *flow* components. Figure 9 shows a fragment of the problem-goal model containing the second and third aforementioned rules.

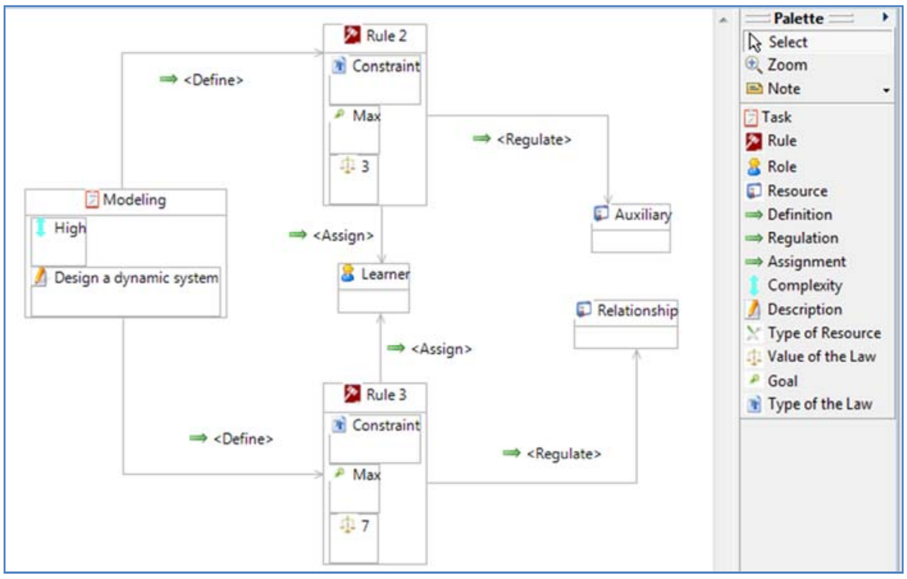


Figure 9: Excerpt of the problem-goal model.

The toolbar of the authoring tool shows icons (Table 3) on the right to represent the learner and each of the tasks to be performed, characterized by its description and its complexity as attributes of the task. The rules are represented by an icon with a hammer. Other icons are used to specify the type of regulation (*constraint* or *requirement*) that establishes the rule, the goal of the rule (e.g., controlling the maximum working time allowed, regulating the minimum number of elements that must be included in the solution, etc.) and a reference value for which the proposed rule is fulfilled (number of components of the application domain that can be used, amount of time that can be spent to solve the task, etc.).

Icon	**Semantic**
👤	Learner
📋	Task
✏️	Description of the task
↕	Complexity of the task
🔨	Rule
T	Type of rule
🔑	Goal of the rule
⚖	Reference value
🗔	Element regulated

Table 3: Visual language to specify the problems-goals.

Finally, it is possible to represent the application domain component or tool of the system that is regulated by the rule, which is called a *resource*. The icons with arrows permit modelling which rules are defined for a particular task (*Definition*), which rules are assigned to a student (*Assignment*) and the elements of the application domain regulated by each rule (*Regulation*).

The next step in the developer's task was to create an observation model with the actions that should be captured in order to analyse the work carried out in each of the three workspaces: (i) the modelling workspace, where the student creates a dynamic model; (ii) the graphical simulation workspace, where the student simulates the system using graphs, and (iii) the tabular simulation workspace, where the student simulates the system and examines the outcome using tables.

The developer later created the abstraction models. The quantitative abstraction model (Table 4) included a set of indicators designed to quantify the learner's work, checking the composition of the model built and the accomplishment of the rules defined in the problem-goal model. A first set of indicators quantifies generic aspects such as the number of elements included in the solution, the number of actions carried out by the learner and the time spent on solving the problem. A second set of indicators quantifies in more detail the development of the three tasks defined in the observation model (modelling, graphical simulation and tabular simulation), quantifying the time spent on each task and collecting the actions related to each one. A third set of indicators counts the type of actions performed on each task: percentage of insertion and delete actions, number of step-by-step and continuous simulations, and number of actions that change the specification of elements of the System Dynamics model. The fourth set of indicators quantifies the fulfilment of the rules included in the problem-goal. Finally, the fifth set of indicators quantifies the composition of the model built discriminating the number of entities and associations. Table 4 also depicts the abstraction level of each indicator: low or high.

The qualitative abstraction model defines a set of indicators (Table 5) that are either inferred from the previous quantitative indicators, or from other inferred indicators. For example, we can infer a global indicator that agglutinates the previous inferred indicators and gives a general assessment of all aspects of the students' activity (see the *quality* indicator in Table 5). In the modelling task at hand, the following qualitative indicators have been considered:

- Speed: This indicator rates the work speed by averaging the number of actions performed per time.
- Cost: It rates the number of components of the application domain used to build the model.
- Experimentation: It assesses whether the model has been tested and simulated enough times to verify its quality.
- Task_division: It measures whether the learner has evenly divided the number of actions among the three tasks (designing the system, simulating it using graphs and simulating it using tables).
- Time_distribution: It assesses whether the learner has evenly divided the time among the three aforementioned tasks.
- Correction: It rates the degree of achievement of the problem-goal.
- Effort: It measures the amount of work from the number of actions performed and objects included in the model per time.

Goal	Indicator name	Abstraction level	Description
Quantifying global aspects	Objects	Low	Number of elements used in the model built
	Time	Low	Time spent in solving the problem
	Actions	Low	Total number of actions carried out
Quantifying the work on each workspace	ActionsTable	Low	Number of actions carried out in the tabular simulation
	ActionsGraphic	Low	Number of actions carried out in the graphical simulations
	ActionsEditor	Low	Number of actions carried out with the model editor
	NDelete	Low	Number of delete actions in the editor
	NInsert	Low	Number of insertion actions in the editor
	TimeEditor	Low	Time used in the model editor
	TimeGraphic	Low	Time used in the graphical simulation
	TimeTable	Low	Time used in the simulation tables
Quantifying specific aspects of the tasks	NSpecifications	Low	Number of actions that change the specification of any element of the model
	PInsertions	High	Percentage of insert actions among all the actions carried out in the editor
	NSimulation	Low	Total number of simulation actions
	PDeletes	High	Percentage of delete actions compared to all actions carried out in the editor
Quantifying the fulfilment of rules	FRulePG1	Low	Fulfilment of the first problem-goal rule
	FRulePG2	Low	Fulfilment of the second problem-goal rule
	FRulePG3	Low	Fulfilment of the third problem-goal rule
	FRulePG4	Low	Fulfilment of the forth problem-goal rule
Quantifying the model structure	NStocks	Low	Stocks in the model
	NAuxiliaries	Low	Auxiliaries in the model
	NFlows	Low	Flows in the model
	NConstants	Low	Constants in the model
	NRelationships	Low	Relationships in the model
	NEntities	High	Entities in the model (stocks, auxiliaries and constants)
	NRelations	High	Associations in the model (flows and relationships)

Table 4: Summary of the quantitative indicators.

- Connectivity_product: It evaluates whether there is a suitable number of associations that connect all the entities of the model.
- Stability: It assesses whether the work follows a stable process where new components are added gradually and the model is not subject to continuous changes and deletions. To do this, we compare the percentage of deletions and the number of changes to the model with insertions performed in the model.
- Quality: It analyses the overall activity and the solution taking into account other qualitative indicators.

Qualitative indicator	**Indicators used in the inference**	**Aspect analysed**
Speed	Actions	Work process
	Time	
Cost	Objects	Solution
Experimentation	ActionsTable	Work process
	ActionsGraphic	
	NSimulation	
Task_division	ActionsEditor	Work process
	ActionsGraphic	
	ActionsTable	
Time_distribution	TimeEditor	Work process
	TimeGraphic	
	TimeTable	
Correction	FRulePG1	Solution
	FRulePG2	
	FRulePG3	
	FRulePG4	
Effort	Actions	Work process and solution
	Cost	
	Time	
Connectivity_product	NEntities	Solution
	NRelations	
Stability	NSpecifications	Solution
	PInsertions	
	PDeletes	
Quality	Stability	Work process and solution
	Effort	
	Correction	
	Time_distribution	
	Task_division	
	Experimentation	
	Cost	
	Connectivity_Product	
	Speed	

Table 5: Summary of the qualitative indicators inferred.

Figure 10 shows an excerpt of the qualitative abstraction model. This model shows a set of inference rules to assign values to the qualitative indicators. The antecedents of the rules specify values of qualitative indicators previously calculated or intervals for the quantitative indicators calculated so that when they take a value in the intervals, a new qualitative indicator is inferred whose value is specified in the consequent of the rule.

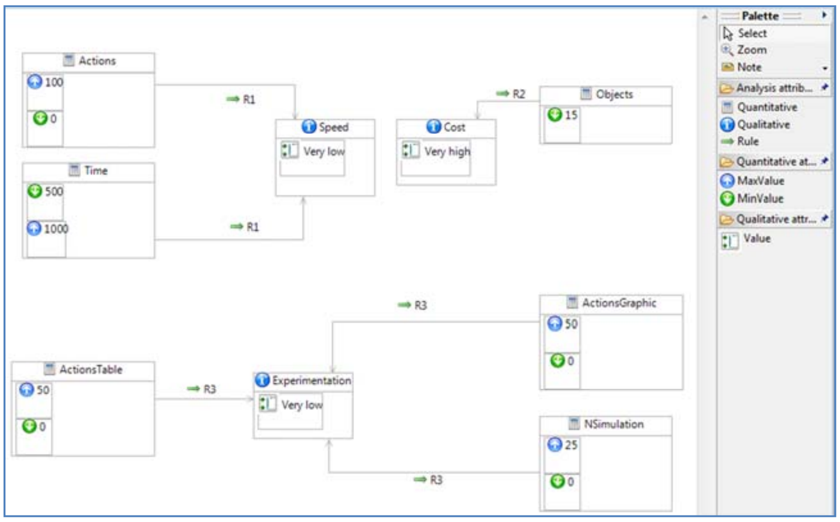


Figure 10: Excerpt of the qualitative abstraction model.

Each rule is represented by a set of links with a specific icon (Table 6) that connects each quantitative indicator of the antecedent with the qualitative indicator of the consequent. A quantitative indicator that is included in the antecedent of the rule is represented with an icon that identifies its name, an icon that represents the maximum value of its activation interval and an icon that represents the minimum value of its activation interval. The qualitative indicators are represented by an icon that identifies their name and another icon that specifies their value.

Icon	**Semantic**
➡	Rule
▦	Quantitative indicator
⬆	Maximum value
⬇	Minumun value
ℹ	Qualitative indicator
↕	Value of the qualitative indicator

Table 6: Visual language to specify the qualitative abstraction.

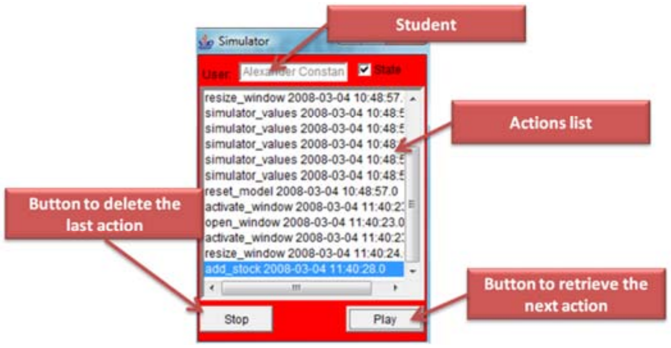
Finally, the intervention model used the inferred indicators to advise the learner how to improve his/her activity. A first group of interventions advises the learner to redirect some aspect of his/her work. This group of interventions tries to correct situations where there is low speed (see I1 in Table 7) or a small amount of work (I2), to better divide the effort between different tasks (I3) or to follow a more reflexive work process (I4). The second set of interventions focuses on the solution and proposes to reduce the components of the model (I5), to review the fulfilment of the problem-goal's rules (I6), to connect all the entities (I7) or to simulate the model more frequently (I8). Finally, a third type of intervention is performed (I9) when the learner closes the work session on Co-Lab and the quality indicator does not reach its highest value; in this case all the analysis indicators are displayed.

Id.	**Indicator's value**	**Trigger**	**Text**
I1	The *speed* indicator is not in the range [intermediate-very high]	3-minute time cycle	"You must work faster. Check the value of the indicator and observe as you perform very few actions per unit of time"
I2	The *effort* indicator is not in the range [intermediate-very high]	2-minute time cycle	"You must work harder. You have performed a low number of actions and included a low number of elements in the solution"
I3	The *task_division* indicator is not in the range [intermediate-very high] OR the *time_distribution* indicator is not in the range [intermediate-very high]	5-minute time cycle	"Plan your activities better. You must not focus your efforts on a single task. Review the indicators and try to work on those tasks where you have not worked enough"
I4	The *stability* indicator is not in the range [intermediate-very high]	4-minute time cycle	"Think about what the solution should be before inserting new components in the model and perform simulations: examine the indicator and notice that you are constantly removing components and changing specifications"
I5	The *cost* indicator is in the range [high - very high]	1-minute time cycle	"The solution includes too many components: review the statement and think about deleting elements"
I6	The *correction* indicator is not in the range [intermediate-very high]	1-minute time cycle	"The problem formulation specifies a number of conditions of using the components of the application domain. However, your solution does not fulfil these conditions. Read the problem formulation and review the solution"
I7	The *connectivity_product* indicator is not in the range [intermediate-high]	3-minute time cycle	"Test that all entities of the solution are linked through associations"
I8	The *experimentation* indicator is not in the range [intermediate-very high]	5-minute time cycle	"Simulate the model more frequently to check its validity"
I9	The *quality* indicator is not in the range [very high-very high]	Close the work session	"Check the value of the indicators: try to improve those which have negative values in future work"

Table 7: Summary of the interventions specified.

4.3.1 The Co-Lab's problem-solving analysis support in action

Figure 11 shows a tool to replay the work processes of the students and the interventions they received. It includes a central panel that shows descriptions of the actions that the learner performed during their work in chronological order. The mechanism to add these actions into the panel is guided by the teacher using two buttons. The *play* button collects the next action the learner carried out. If the action triggers an intervention, the application launches a window with the description of that intervention. Moreover, this tool stores the interventions received by the student in a database. Thus, it allows empirically studying how the framework intervened to help the student in his/her work. The *stop* button allows the teacher to move backwards in the reconstruction of the activity and the intervention taking place. In addition, the tool shows the values of the indicators when the problem-solving process is finished.


Student
Simulator
User: Alexander Constan
State
Actions list
Button to delete the last action
Button to retrieve the next action
Stop
Play

```
resize_window 2008-03-04 10:48:57.
simulator_values 2008-03-04 10:48:5
simulator_values 2008-03-04 10:48:5
simulator_values 2008-03-04 10:48:
simulator_values 2008-03-04 10:48:5
simulator_values 2008-03-04 10:48:5
reset_model 2008-03-04 10:48:57.0
activate_window 2008-03-04 11:40:2
open_window 2008-03-04 11:40:23.0
activate_window 2008-03-04 11:40:2
resize_window 2008-03-04 11:40:24.
add_stock 2008-03-04 11:40:28.0
```

Figure 11: User interface of the tool that replays the student's actions.

First, we studied the usefulness of the framework to empirically study how the student organizes his/her work between the modelling and simulation tasks, and the components used in the solutions. Table 8 collects a set of quantitative indicators that allow a first generic approach on these issues. The table shows that the students typically spent 30% more actions on modelling than on the graphic or tabular simulation tasks (to define time intervals, to select the simulation method, to start, to stop, etc.). Regarding the structure of the solutions, the models have very similar numbers of stocks and flows to the ideal solution structure (see Figure 7), a greater use of auxiliaries, and fewer relationships and constants.

Having studied the students' actions and the solutions' structures from a very general point of view, we analysed the compliance of the rules included in the problem-goal model (Table 9). We observe that most students structured the model around a single stock component that represents the Earth's energy. The second rule, which stipulates the use of three auxiliaries, was fulfilled by 25% of the students. All the students complied with the rule that constrains the use of seven relationships as a maximum. Finally, more than 80% of students fulfilled the *requirement* to use two flows.

Aspect analysed	Average value	SD
Modelling actions (insert component, delete component, etc.)	217.16	98.4
Simulation actions in graphic or tabular form	152.13	85.76
Stocks used in the solution	1.02 (Ideal solution: 1)	0.16
Auxiliaries used in the solution	4.17 (Ideal solution: 3)	1.18
Constants used in the solution	0.55 (Ideal solution: 3)	0.80
Relationships used in the solution	5.57 (Ideal solution: 7)	1.26
Flows used in the solution	2.18 (Ideal solution: 2)	0.54

Table 8: Global analysis of actions and solutions.

Rule description	Percentage of students
Use only one stock	97.5%
Use three auxiliaries as a maximum	25 %
Use seven relationships as a maximum	100%
Use two flows	82.5 %

Table 9: Percentage of students that fulfil each rule of the problem-goal model

Table 10 collects the values that are adopted more frequently by the qualitative indicators and the number of students whose analysis indicators take those values. In this case, we observe that the indicators that characterize the work process never have a value greater than intermediate. This can be explained by the fact that students generally spent more time than expected on solving the problem and the time distribution between the tasks of modelling and simulation was not balanced. However, the indicators that analyse the solution have a value equal to or greater than the intermediate value. Although a significant number of students did not comply with the rule that regulates the use of auxiliaries (see Table 9), this does not imply that the simulation results are unsatisfactory because in most of the cases the students used more auxiliaries to specify values that are defined by means of constants in the ideal solution, but the functionality of the simulation is not altered.

Qualitative indicator	Aspect analysed by the indicator	Most frequent value	Students whose indicator takes this value
Speed	Work process	Very Low	35
Cost	Solution	Intermediate	27
Experimentation	Work process	Intermediate	24
Task_Division	Work process	Low	29
Time_ distribution	Work process	Low	30
Correction	Solution	High	22
Effort	Work process and solution	Intermediate	32
Connectivity_Product	Solution	Very High	24
Stability	Solution	High	35
Quality	Work process and solution	Intermediate	27

Table 10: Most frequent value taken by the qualitative indicators and number of students whose indicator takes this value.

Another purpose of this study was to evaluate the functionality of the framework in replaying the work process and its analysis. Table 11 depicts the number of interventions performed and the percentage of students who received each intervention. The average number of interventions shows that all the interventions, except the interventions that correct the excessive number of components in the solution, were performed at least once on average in each work process. Table 11 also shows that all the participants received an intervention showing an assessment of the quality of the work carried out at the end of their work process. In summary, we observe that, except for the interventions that correct problems to relate entities by means of associations and an excessive number of components in the model built, at least 30% of students received each of the interventions of the analysis process.

Id.	Intervention’s description	Interventions	Percentage of students
I1	Low speed	4.3	87.5%
I2	Little effort	2.8	30%
I3	Time between tasks not well-balanced	3.4	80%
I4	Stability in the modelling	1.1	45%
I5	Excessive number of components in the model	0.7	12.5%
I6	The model does not fulfil the conditions of the problem	3.1	82.5%
I7	Model entities are not related	1.9	2.5%
I8	Lack of simulation	1.8	40%
I9	Summary of the quality of the work performed	1	100%

Table 11: Average number of interventions carried out in each process and percentage of students who received the intervention.

4.4 Discussion

In conclusion, we can say that this analysis shows that the students usually carry out the work process much slower than that which is outlined by the teacher. The indicators that examine whether the students' work has been distributed equally among the three work areas show intermediate values. This is because the students usually focus their efforts more on modelling than on simulating. Thus, it is possible that the students need more experience to build a solution in less time, or the teacher should consider in future experiences that the students need to work more slowly or that the modelling tasks require more effort than the simulation task. However, these students' performances do not necessarily lead to the construction of an incorrect System Dynamics model because in most cases the *correction* indicator, which assesses whether the solution fulfils the rules of the problem-goal model, takes a high value.

The rule that regulates the use of auxiliaries was unfulfilled by a significant number of students. This is because these students used the auxiliaries to represent the albedo factor and the heat capacity of the Earth, and these components should be modelled by means of constants. This is a partial result of the study: the students usually confuse these two types of elements.

Finally, the *quality* indicator that summarizes the value taken by all the indicators has never taken a very high value. The students should basically improve the speed and the time distribution during the work process to reach a very high value of the *quality* indicator in future problem solving activities.

Moreover, the empirical study of the interventions enables us to analyse how the framework contributes to helping the students during the work process. The data show that seven interventions were executed for at least 30% of the students. The analysis process was useful in these cases because it contributed to detecting problems in the students' activity, which was corrected thanks to the interventions. However, some interventions were not frequently performed, such as those aimed at correcting problems of connections between entities or reducing an excessive number of components in the solution. Therefore, the teacher should consider changing these interventions in future activities.

5 Conclusions

The work presented here describes a technological framework that allows software developers to specify and to automate processes to analyse the activity of a student when solving problems and to provide feedback to that student in order to understand the impact of his/her actions. This framework follows a model-based approach in which the software developer specifies representations of the analysis to be automated and the feedback to be displayed. These models are processed by the framework to auto-configure its functionality and to automatically perform the specified interventions. The framework can be integrated into computer-supported learning systems that store the actions and solutions produced by the learners in repositories with a predefined structure. Then, the framework processes these actions and solutions to produce interventions and feedback that in most cases are automated by the Monitoring subsystem.

This framework provides a novel approach with regard to the ITS's traditional proposals focusing on evaluating solutions because our proposal also allows the inference of indicators analysing the work process followed by the student. Moreover, this framework is a flexible computational support that can be used to analyse learning processes in various application domains and problems, and be configured according to the characteristics of each case by means of models.

The model-based approach followed by the framework enables a reduction of the developer's effort because they only need to specify models using visual languages instead of implementing the source code of an analysis system by hand. This modelling task is supported by a set of authoring tools that enable the software developer to configure the analysis (the actions that should be analysed, how to calculate analysis indicators, what intervention should be carried out, etc.).

The framework has been evaluated by means of its application to analyse problem-solving processes in the System Dynamics domain. To that end, a software developer configured the framework to automate an analysis process that was integrated into Co-Lab, a learning environment that supports solving problems in this domain. Thus, this case study has tested that the framework not only supports the modelling of the analysis but also produces effective support to integrate and perform analysis processes in computer-supported learning environments.

The analysis processes produce indicators that assess both the students' problem-solving processes and the solutions built. Therefore, these analyses enable teachers to detect problems during the problem-solving activities that lead to wrong solutions not solving the problem. For instance, an empirical study of the indicators inferred in the Co-Lab case study showed that the problem-solving processes carried out by the students was usually slower (they perform very few actions by unit of time) than predicted by the teacher and with an unbalanced division of tasks.

The framework also automates intervention in the students' activity to correct the problems identified by the analysis indicators. To that end, the analysis processes evaluate all the students' actions to detect problems that must be solved. The intervention actions usually show message that advise the students how to solve these problems. A study of the interventions carried out in the Co-Lab case study showed that they were frequently used to guide the work of students.

In the future, we will explore the use of the framework to specify and automate more complex interventions such as adaptations of the learning system. We are also applying the framework to automate the analysis and interventions in processes where groups of students collaborate in solving a single problem. Thus, the framework would be used to analyse communication and collaboration between the students. In this case, the intervention mechanisms will be enriched to enable the selection of the member of the group of students who should receive the intervention, for instance.

Acknowledgements

The Ministerio de Educación y Ciencia (España) has partially supported this research under Project TIN2011-29542-C02-02. The authors would like to express their gratitude to Ton de Jong, Wouter R. van Joolingen and Sylvia van Borkulo (University of Twente), for supporting this research. The work reported here was done during Rafael Duque's stay at the Department of Instructional Technology of the University of Twente.

References

[Anderson, 95] Anderson, J.R., Corbett, A.T., Koedinger, K.R. Pelletier, R.: “Cognitive Tutors: Lessons Learned”; The Journal of the Learning Sciences, 4 (1995), 167-207.

[Arts, 02] Arts, J.A.R., Gijselaers, W.H., Segers, M.S.R.: “Cognitive effects of an authentic computer-supported, problem-based learning environment”; Instructional Science, 30 (2002), 465-495.

[Avouris, 02] Avouris, N., Dimitracopoulou, A., Komis, V., Fidas, C.: “OCAF: An object-oriented model of analysis of collaborative problem solving”; In: Proceedings of the Conference on Computer Support for Collaborative Learning, Colorado (2002), 92-101.

[Avouris, 05] Avouris, N., Komis, V., Fiotakis, G., Margaritis, M., Voyiatzaki, E.: “Logging of fingertip actions is not enough for analysis of learning activities”; In: Proceedings of The International Conference on Artificial Intelligence in Education, The Netherlands (2005), 1-8.

[Booch, 05] Booch, G., Rumbaugh, J., Jacobson, I.: “Unified Modeling Language User Guide”; Addison-Wesley Professional (2005).

[Bosch, 00] Bosch, J.: “Design and use of software architectures: adopting and evolving a product-line approach”; Addison-Wesley Publishing (2000).

[Bravo, 08] Bravo, C., Redondo, M.Á., Verdejo, M.F., Ortega, M.: “A framework for process-solution analysis in collaborative learning environments”; International Journal of Human-Computer Studies, 66, 11 (2008), 812-832.

[Bravo, 09] Bravo, C., Van Joolingen, W.R., De Jong, T.: “Using Co-Lab to build System Dynamics models: Students' actions and on-line tutorial advice”; Computers & Education, 53 (2009), 243-251.

[Conati, 09] Conati, C., Maclaren, H.: “Empirically building and evaluating a probabilistic model of user affect”; User Modeling and User-Adapted Interaction, 19 (2009), 267-303.

[de Jong, 98] de Jong, T., van Joolingen, W.R.: “Scientific Discovery Learning with Computer Simulations of Conceptual Domains”; Review of Educational Research, 68 (1998), 179-201.

[Dean, 06] Dean, D., Kuhn, D.: “Direct instruction vs. discovery: The long view”; Science Education, 91 (2006), 384-397.

[Dimitracopoulou, 04] Dimitracopoulou, A. et al.: “State of the art of interaction Analysis: Interaction Analysis Indicators”; Kaleidoscope Network of Excelence, Deliverable D.26.1 (2004).

[Dimitracopoulou, 05] Dimitracopoulou, A. et al.: “State of the art of interaction analysis for Metacognitive Support & Diagnosis”; Kaleidoscope Network of Excelence, Deliverable D.31.1.1 (2005).

[Duque, 11] Duque, R., Bravo, C., Ortega, M.: “A framework for the automated model-based analysis of users' activity in social media systems”; Journal of Network and Computer Applications 34, 4 (2011), 1200-1209.

[Duque, 12] Duque, R., Rodríguez, M.L., Hurtado, M.V., Bravo, C., Rodríguez-Domínguez, C.: “Integration of collaboration and interaction analysis mechanisms in a concern-based architecture for groupware systems”; Science of Computer Programming 77, 1 (2012) 29-45.

[Forrester, 85] Forrester, J.W.: “The model versus a modeling process”; System Dynamics Review, 1, 1 (1985) 133-134.

[Harrer, 04] Harrer, A., Bollen, L., Hoppe, U.: "Processing and Transforming Collaborative Learning Protocols for Learner's Reflection and Tutor's Evaluation"; In: Workshop on Artificial Intelligence in Computer Supported Collaborative Learning, Spain (2004).

[Jung, 10] Jung, S.-Y., VanLehn, K.: "Developing an Intelligent Tutoring System Using Natural Language for Knowledge Representation"; In: The 10th International Conference on Intelligent Tutoring Systems, USA (2010), 355-358.

[Le, 06] Le, N.-T.: "A Constraint-based Assessment Approach for Free- Form Design of Class Diagrams using UML"; In: The 8th International Conference on Intelligent Tutoring Systems, Taiwan (2006), 11-19.

[Le, 10] Le, N.T., Menzel, W., Pinkwart N.: "Considering Ill-Definedness of problem tasks under the aspect of solution space"; In: The 23st International Conference of the Florida Artificial Intelligence Research Society (2010), 534–535

[Mellor, 04] Mellor, S., Scott, K., Uhl, A., Weise, D.: "MDA Distilled: Principles of Model-Driven Architecture"; Addison-Wesley Professional, 2004.

[Menzel, 06] Menzel, W.: "Constraint-based modeling and ambiguity"; International Journal of Artificial Intelligence in Education, 16 (2006).

[Mitrovic, 04] Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A.: "DB-suite: Experiences with Three Intelligent Web-based Database Tutors"; Journal of Interactive Learning Research, 15 (2004), 409-432.

[Mitrovic, 07] Mitrovic, A., Martin, B., Suraweera, P.: "Intelligent Tutors for All: The Constraint-Based Approach"; IEEE Intelligent Systems, 22 (2007), 38-45.

[Mitrovic, 09] Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., Mcguigan, N.: "ASPIRE: An Authoring System and Deployment Environment for Constraint-Based Tutors"; International Journal of Artificial Intelligence in Education, 19 (2009), 155-188.

[Mitrovic 11] Mitrovic, A., Williamson, C., Bebbington, A., Mathews, M., Suraweera, P., Martin, B., Thomson, D., Holland, J.: "Thermo-Tutor: An Intelligent Tutoring System for Thermodynamics"; In: The IEEE Global Engineering Education Conference, Jordan (2011), 378-358.

[Mørch, 03] Mørch, A.I., Dolonen, J., Omdahl, K.: "Integrating agents with an open source learning environment"; In: Proceedings of the international conference on computers in education, Hong Kong (2003), 393-401.

[Muehlenbrock, 05] Muehlenbrock, M.: "Automatic Action Analysis in an Interactive Learning Environment"; In: Workshop on Usage Analysis in Learning Systems at the 12th International Conference on Artificial Intelligence in Education, The Netherlands (2005), 73-80.

[Nicholas, 06] Nicholas, A.: "Dealing with Ambiguity in a Foreign Language ITS"; Department of Computer Science and Software Engineering, University of Canterbury, Christchurch (2006).

[Ohlsson, 94] Ohlsson, S.: "Constraint-Based Student Modeling"; In: Greer, J. E. & McCalla J. E. (Eds.), Student Modeling: The Key to Individualized Knowledge-Based Instruction, (1994) 167-189.

[Remolina, 04] Remolina, E., Ramachandran, S., Fu, D., Stottler, R., Howse, W.R.: "Intelligent Simulation-Based Tutor for Flight Training"; In: The Interservice/Industry Training, Simulation, and Education Conference, USA (2004), Paper No. 1743, 1-14.

[Soller, 05] Soller, A., Martinez, A., Jermann, P., Muehlenbrock, M.: “From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning”; International Journal of Artificial Intelligence in Education, 15 (2005), 261-290.

[van Joolingen, 05] van Joolingen, W.R., de Jong, T., Lazonder, A.W., Savelsbergh, E.R., Manlove, S.: “Co-Lab: research and development of an on-line learning environment for collaborative scientific discovery learning”; Computers in Human Behaviour, 21 (2005), 671-688.

[Yin, 94] Yin, R.K.: “Case study research: Design and methods”; Thousand Oaks, CA: Sage Publications (1994)