Diagram, a Learning Environment for Initiation to Object-Oriented Modeling with UML Class Diagrams

Dominique Py, Ludovic Auxepaules, Mathilde Alonso

Jl. of Interactive Learning Research (2013) **24**(4), 425-446

Diagram, a Learning Environment for Initiation to Object-Oriented Modeling with UML Class Diagrams

DOMINIQUE PY, LUDOVIC AUXEPAULES,
AND MATHILDE ALONSO
LIUM, France
Dominique.Py@lium.univ-lemans.fr
Ludovic.Auxepaules@lium.univ-lemans.fr
Mathilde.Alonso@lium.univ-lemans.fr

Learning environments for object-oriented modelling in UML which offer a rich interaction usually impose, in return, strong restrictions on the range of exercises they can address. We propose to overcome this limit by including a diagnostic module that compares the student diagram with a reference diagram. This approach enables to combine the advantages of an open environment (in which the teacher can add new exercises without constraints on the vocabulary or the size of the diagram) with a sophisticated interaction that offers methodological help, encourages self-correcting and self-monitoring, and provides the learner with specific graphic tools. These principles have been developed and implemented through a learning environment for UML class diagrams. Experiments have been conducted in ecological context and show that this approach is achievable and quite effective.

INTRODUCTION

Nowadays, modeling is a core topic in most Computer Science and Software Engineering curricula, especially since the object-oriented modeling language UML (Unified Modeling Language) has been accepted as the standard graphical language for specifying software systems. Recent research works have focussed on the design of computer-supported learn-

ing environments for modeling, especially for object-oriented modeling (Baghaei, 2007; Moritz, 2008) and database modeling (Suraweera & Mitrovic, 2004). In these environments, the relevance of the feedbacks relies on the system's ability to evaluate the correctness of the student's work. Due to the lack of deterministic methods for checking the consistency of a model, performing this evaluation is still a difficult task. To make it easier, designers can impose different restrictions on the interaction, but this also restricts the learner's ways of expression and the problem range of the environment.

We have tried to overcome these limits by exploring a new approach that aims at designing an open learning environment. This approach relies on a specific interaction model and a diagnostic method that evaluates a diagram and enables the system to provide the learner with textual feedbacks that take into account the correctness of his/her diagram.

In section 2 we describe the context of this work and present the state of the art. Section 3 specifies the objectives and approach adopted in Diagram. The model of interaction is described in section 4, the diagnostic method and its results in section 5 and the pedagogical feedbacks in section 6. Lastly, section 7 describes an experiment of the environment and discusses its results.

LEARNING OBJECT-ORIENTED MODELING

Since the development of OOM languages, many empirical studies, pedagogical tools and books have been devoted to OOM teaching. In this section, we present a brief survey of these studies and the limits of these approaches.

UML

UML is a language standardized in 1997 by the Object Modeling Group (OMG) which facilitates the design of programs (OMG UML, 2010). However, UML does not include a complete methodology for analysing and designing OO programs.

The current version of UML 2.1 consists of thirteen types of diagrams, representing different aspects of a software system. Among these diagrams, the class diagram is the core of the modeling process. It is used during the phases of analysis and design and can represent information at various levels of abstraction and precision. For that reason, it is often the first type of diagram taught in the OOM courses. This is why we chose to focus our

study on the class diagram and its main elements: classes, relationships and attributes.

Difficulties of OOM Learning

Empirical research works have studied the difficulties that students face when they learn OOM (Moisan & Rigault, 2009). These studies show that theoretical knowledge of UML syntax and semantics is not enough: high level skills, like problem analysis and abstraction, are essential and can only be acquired by practice. Thus, Habra and Noben (2001) stress that no universal rule applies for the creation of a model, and note that learners need methodological help to guide their creativity. This point of view is shared by Frosch-Wilke (2003) who estimates that modeling is acquired by practice and recommends to provide learners with processes or steps to build their diagram.

Learning Environments for OOM

Different approaches to computer-supported learning environments for object-oriented modeling have been explored so far, mainly the constraint-based approach, illustrated by Collect-UML (Baghaei, Mitrovic & Irwin, 2006), and the curriculum-based approach, used in DesignFirst-ITS (Moritz, 2008).

Constraint-Based Modeling, as defined by Ohlsson (1994), consists in defining the domain rules in terms of constraints, then checking that the learner's answer satisfies these constraints. This approach was used for databases learning in Kermit (Suraweera & Mitrovic, 2004) and OOM learning in Collect-UML (Baghaei, 2007). In Collect-UML, the learner builds a class diagram starting from a textual description of the problem. He/she selects an expression in the text to model the corresponding object. Collect-UML includes a diagnostic tool, based upon domain-specific constraints, that checks whether the learner's diagram satisfies these requirements. Each constraint is composed of a relevance condition and a satisfaction condition. When the former is satisfied and the latter is not, the constraint is violated and an error message is displayed. Usually, the constraint-based approach is well suited for open domains. However, few general constraints exist in UML, so that the semantic constraints in Collect-UML do not actually express domain constraints, but differences that can be accepted between the

learner's diagram and a correct diagram, provided by an expert. For example, a constraint is: "if the ideal solution contains a subclass C, and the learner's solution contains a class C, then C must also be a subclass in the learner's diagram". This has strong consequences on the interaction, because the learner is forced to pick up the class and relation names from the text problem, he/she is not allowed to create elements freely, nor to choose their name. As a consequence, Collect-UML is restricted to problems that do not contain implicit elements. The constraint-based approach, as implemented in Collect-UML, is quite close to the "bug catalogue" approach, because it can only recognize the bugs that have been foreseen by the designers.

The intelligent tutoring system DesignFirst-ITS (Moritz, 2008) is based on a "design-first" curriculum that introduces object-oriented design (using elements of UML) before coding. An author module enables the teacher to create new exercises and automatically generates the solution diagram. An evaluation module compares the learner's diagram to the solution and typical bugs, and a pedagogical agent provides help and explanations. DesignFirst-ITS is quite sophisticated, but the system is restricted to the very first steps of learning, due to the current limits of the expert module. When the teacher creates a new problem, he has to use declarative sentences with basic syntactic structure and active voice. The problem statement must contain all the details about the classes, attributes and methods. Implicit elements and useless details are proscribed. At last, the diagram should not comprise more than five classes. Consequently, the range of problems that can be solved with DesignFirst-ITS is quite small, and the learners have few ways of expressing their creativity when solving a problem.

DIAGRAM'S GENERAL FRAMEWORK

Before presenting more precisely the Diagram environment, we expose in this section the general approach that we adopted, the base of exercises we used to evaluate the environment, and present an example of problem that will be used throughout the article.

Research Motivation and Objectives

The teachers use a great variety of exercises during the first steps of learning, and this diversity is necessary to organize the course and introduce

gradually new concepts. Thus, our first objective was to design an open environment, in which the teacher can add new exercises without constraints on the vocabulary or the size of the diagram. Such constraints strongly limit the expressive power and the range of exercises tackled by environments like Collect-UML or DesignFirst-ITS. In Diagram, the problem statement can include useless details and implicit classes or relations, like problems used in real university courses. The creation of an exercise only requires the teacher to type the statement and to give a correct diagram.

The studies mentioned in section 2 show that two points of interaction design are crucial: first, methodological guidance, in order to mitigate the absence of modeling method, and second, graphic tools for creating the diagram. Our second objective was to design an interface that includes methodological help, encourages self-correcting and self-monitoring, and provides the learner with various and relevant graphic tools. We set up a simple methodological guidance: the learner must follow several steps to create a diagram. We chose to display the statement of the problem on the screen, as the other environments do, but we more largely exploited the possibilities of the graphical interface to reinforce the link between text and diagram (section 4). At last, we provided the learner with contextual assistance to support self-correcting.

Besides these generic helps, some systems also propose specific help messages by comparing the learner's diagram with a solution. DesignFirst-ITS automatically generates this solution, but the constraints that result from this approach (small diagrams, short problem statement, very simple sentences, no implicit elements) do not seem compatible with an open environment. Collect-UML uses a solution provided by the teacher, and so can handle more complex problems, but its constraint-based approach also restricts the learner's creativity. In particular, the names of classes, attributes and methods are imposed, and the problem statement cannot contain implicit elements. Our third objective was to overcome these limits by exploring a new approach, also based on diagram comparison, but without a predefined list of constraints or bugs catalogue. Indeed, we consider that the differences between the learner diagram and the expert diagram are not necessarily mistakes; they sometimes express different modeling choices. The analysis of the differences between diagrams can be performed separately from the interpretation of these differences in a pedagogical objective. We adopted a modular approach, by separately developing a tool that compares the diagrams and enumerates their differences (section 5) and a tool that interprets these differences to provide pedagogical feedbacks (section 6). We make the assumption that this architecture should enable a diagnostic more robust and more relevant than the previous approaches.

Base of Exercises

At the beginning of the project, UML exercises have been collected in order to analyze them and characterize the difficulties encountered by learners. This base includes about thirty exercises. It has been used us to develop Diagram and to carry out the experiments in ecological context. Most of the exercises have been provided by a teacher of the university of Le Mans (France) who took part in the project. He uses these exercises in his OOM courses, with students of second year in computer science. Some exercises found in books or on the Web have also been included. The exercises of the base relate to various fields, the concepts and the vocabulary employed are very diverse. Their size varies from a few sentences to one page. The diagrams comprise four to twenty-seven classes.

This base has been analysed to produce a classification of the exercises according to their difficulty. Several criteria were retained: the length of the text, the simplicity of the vocabulary, the presence of implicit or superfluous information, the presence of clues, the number of information to be modelled. A combination of these criteria provided a scale of difficulty going from one (for "very easy") to five (for "very difficult"). A similar analysis was made to evaluate the degree of difficulty of the diagrams. These two classifications were then used to plan the experiments and propose exercises of increasing complexity.

Example

The functionalities of Diagram are illustrated throughout this paper with an exercise of the base, called "Pen and felt-pen". Figure 1 presents a diagram that will be used thereafter as the expert diagram for this exercise.

> "A pen and felt-pen are two concepts with common attributes: color, brand name, etc. A felt-pen has a top. Both pen and felt-pen have a body with some properties. Pen and felt-pen are used by a person and belong to a person. There is a specific felt-pen that is an eraser felt-pen."

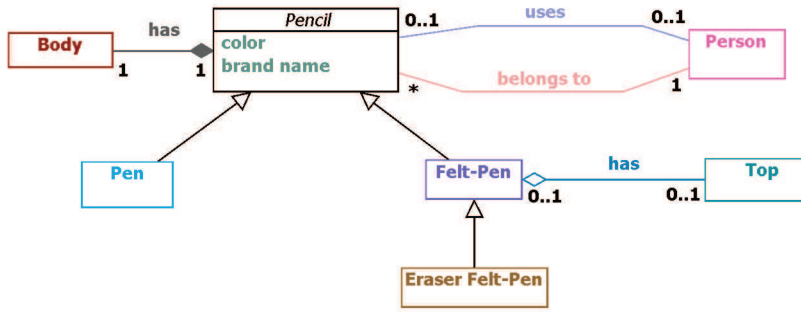

Figure 1. Class diagram for the "pelt and felt-pen" problem.

According to our classification, this exercise is considered as easy (level 2) regarding the statement but difficult (level 4) regarding the diagram because it requires many different UML notions. The statement is short, uses a simple vocabulary and familiar concepts. However, the diagram is not so simple for novices. First, one class is implicit: the word "pencil" does not appear in the text. Some students do not represent this class, but duplicate all the attributes and relations that refer to it. Another difficulty is the presence of two distinct relations between the classes "pencil" and "anybody", which is not common in simple exercises. In addition, this diagram comprises four types of relations: association, composition, aggregation and inheritance. Many novice students cannot make the difference between these relations and often confuse them. At last, classical errors can appear in this exercise, for example an attribute used instead of a class, wrong orientation of relations, or incorrect multiplicities.

This exercise is a classical one, nevertheless it could not be solved with environments like Collect-UML or DesignFirst-ITS, because the concept "pencil" is implicit in the text, so that the corresponding class could not be created in these environments.

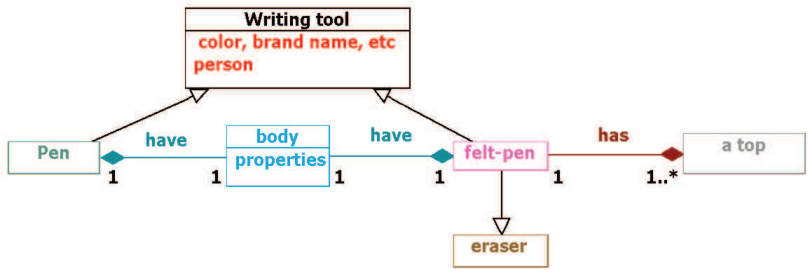

Figure 2. A student's class diagram.

Figure 2 presents a student diagram that contains typical errors. It will be used throughout the article as an example of learner diagram. In this example, the learner has noticed the presence of an implicit class, that he/she named “writing tool”. But he/she duplicated the composition “have” and applied it both to “pen” and “felt-pen”, instead of applying it only to “writing tool”. He/she represented the concept of “person” as an attribute, instead of a class. The relations between a person and a pencil are thus missing. The direction of certain relations is reversed: the inheritance between “felt-pen” and “eraser”, the composition between “felt-pen” and “a top”. Lastly, there are some small errors, like the attribute “color, mark, etc” or the insertion of an attribute “properties” into the “body” class.

THE INTERACTION IN DIAGRAM

Introducing some kind of methodological guidance in the environment can contribute to mitigate the lack of formal method for building a class diagram. The starting point is the method used by the teacher who participated in the design of Diagram. He recommends to its students to follow a three steps method: (1) carefully reading the statement, (2) working out the diagram, (3) reading again the statement and checking the correctness of the diagram. This method is simple and consistent with the recommendations that can be found in books about modeling. It has been transposed into Diagram and enriched it with graphic tools. A fourth stage has been added (which will be developed in section 6), in which the system evaluates the diagram and produces feedbacks to help the student to correct it (Alonso, 2009).

During the reading step, the student discovers the problem statement. Some students use a pencil or a highlighter to mark the important words. To offer the same functionality in the computer environment, Diagram integrates the possibility of underlining words of the statement. A constraint is imposed before the student can start the modeling step: he/she must have underlined the important concepts of the statement (defined by the teacher when he/she creates the exercise).

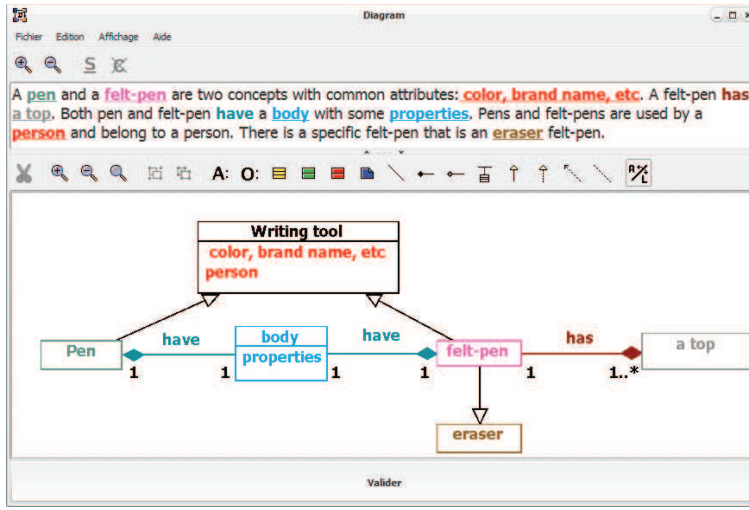

Figure 3. The interface during the modelling step.

The second step consists in creating the class diagram (figure 3). Diagram offers two different ways of creating an UML element (class, attribute or relationship): either from a phrase of the text (« assisted mode ») or freely (« free mode »). In the former case, the element and the expression are displayed in the same color, in order to facilitate visual control and reinforce the link between the statement and the diagram. In the free mode, the learner only selects the kind of element that he/she wants to create. The element is displayed in black and the learner chooses its name. This mode enables to create elements that are not explicitly mentioned in the text, and elements without name, like inheritance relationships. Any free element or free expression in the text can later be linked to another expression, and will take on the same color as the expression to which it is linked. The student remains free to modify the name of any element throughout the session. The free mode is an original concept in Diagram that gives a strong expressive power to the interface and allows the learner to be more creative than in the other environments.

During the last step, the student must read again the text problem and compare it to his/her class diagram in order to check the diagram correctness and completeness. At the beginning of this step, the diagram is hidden and the text alone is displayed in black and white on the interface. When the student moves the mouse over an expression linked to an element of the diagram, this expression recovers its color and at the same time, the corre-

sponding element in the diagram reappears on the interface. The learner can then turn back to the modelling step, if he/she wants to modify his diagram, or continue and validate the diagram.

THE DIAGNOSTIC METHOD

The assistance provided during the interaction described above is generic: it does not take into account the correctness of the diagram. To produce specific feedbacks, based on the analysis of the individual productions of the student, it is necessary to compare the diagram with a solution, as Collect-UML and DesignFirst-ITS do (Baghaei, 2007; Moritz, 2008). However, the diagnostic methods of these two environments impose strong restrictions on the statements and the diagrams. Another approach, which seemed more compatible with the open character of Diagram, has been explored here. It consists in finding all the differences existing between the two diagrams, by using a tool that compares the diagrams. These differences can then be used to produce feedbacks.

Thus, a specific matching method that takes two class diagrams as inputs and matches them (Auxepaules, 2009) has been defined and implemented. The principles of the algorithm rely on the object-oriented metamodel (OMG MOF, 2010) and on graph matching methods and algorithms (Sorlin, Solnon & Jolion 2007). It proceeds in three steps. First, the diagrams are schematized in characteristic structural patterns. Then, these structures are compared by using similarity functions, and similarity scores are computed for each couple of structures. Finally, the resulting matches are labelled with the differences that they express. This method is detailed in the following sections.

Characteristic Structural Patterns

Characteristic structures, called *patterns*, are introduced. They correspond to different granularity levels in class diagrams. The simple patterns represent the basic elements of the object-oriented meta-model, they are organized and structured into complex patterns. The patterns have been introduced because of their interesting structural and semantic properties that can be exploited by the matching algorithm in order to match diagram parts at a higher granularity level.

The main patterns for class diagrams are represented on figure 4.

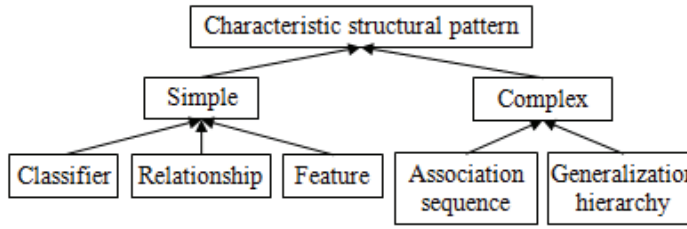

Figure 4. Characteristic structural patterns in class diagrams.

The simple patterns of class diagrams are those defined by the Object Management Group (OMG MOF, 2010). The classifiers include classes, interfaces and objects. The relationships include associations (simple associations, aggregations and compositions), generalizations and dependencies. The features include properties (attributes and association ends) and operations. The complex patterns, respectively Association sequence and Generalization hierarchy, are built from several simple patterns and a relationship type, respectively association and generalization.

Comparison and Matching

In order to automate comparison and matching, class diagrams are considered as graphs, where the vertices are classifiers and the edges are relationships. In this way, the problem of diagram matching can be seen as a variant of graph matching, with graphs characterized by specific features.

Sorlin, Solnon and Jolion (2007) propose a generic similarity measure for graph matching. This measure is parameterized by similarity functions that express domain dependent similarity knowledge and constraints. The matches are either univalent or multivalent. A match is multivalent when a single vertex in one graph is matched with a set of vertices in the other graph.

The diagnostic algorithm in Diagram relies on the same principles. Similarity functions compute a score for each couple of patterns to be compared. These functions are either general or specific to pattern types. They weight and normalize qualitative and quantitative criteria.

The similarity score is composed of two sub-scores. The simple score weights the criteria that do not depend on the other parts of the diagram, as for example name, visibility, aggregation kind, abstraction, nature, and so

on. The complex score weights the scores of the neighbours (parents, children, siblings and linked patterns).

The diagram matching algorithm proceeds top-down. Complex patterns, then simple ones, are compared and the pairs are ordered by decreasing score. A greedy algorithm tries to match complex patterns in an univalent way: if this is unsuccessful, multivalent matches are performed. Once the complex patterns have been matched, univalent or multivalent matches are performed in the same way for each simple pattern of a complex one.

Unlike graph vertices, the basic elements in a diagram (classes, attributes, relationships) are usually given a name. The similarity functions must take this name into account. But, as these names refer to real world objects or concepts, the student may use many synonyms instead of the teacher's names. Moreover, students often mistype words, or make spelling errors. This is why a specific string matcher has been designed and included in the system. It computes a score for each couple of names and selects the pair that matches best. Beforehand, names are filtered by case, gender, number and special characters.

Differences Between the Diagrams

The matching step, described above, produces pairs of fully or partially matched patterns. Then, partial matches are labelled with the kind of difference that they represent. These differences express gaps between structures or characteristics of the two diagrams. They are classified in eight categories:

- Omission of an element: an element of the reference diagram has been omitted by the learner.
- Addition of an element: the learner has added an element that does not appear in the reference diagram.
- Transfer of an element: an element has been transferred to another part of the diagram. For example, a relationship between two classes A and B in the reference diagram is shifted between classes A and C.
- Duplication of an element: an element of the reference diagram is replaced by several elements of the same type in the learner's diagram.
- Merging elements: several elements of the same type are substituted by a single element.
- Misrepresentation: an element of the reference diagram is changed to another one, of a different type. For example, a class is replaced by an attribute or a composition is replaced by an association.

- Reversion of the direction of a relationship: an oriented relationship (inheritance, aggregation or composition) has been reversed by the learner.
- Wrong multiplicity: multiplicities of a relationship in the learner's diagram are different from those in the reference diagram.

In students' diagrams, some differences often go together. For example, a "class omission" usually implies a "relation omission" or a "relation transfer", because the relations supported by the missing class are themselves moved or omitted. When this situation happens, it is better to produce a single feedback covering the group of differences than to produce several feedbacks. Therefore, a set of fifteen compound differences, which can be treated as a whole, has been defined. A compound difference is made of a main difference and a set of secondary differences. For example, the compound difference "relationship duplication and transfer" is composed of a main difference "relationship duplication" and one or several secondary differences amongst "relationship transfer", "direction reversion", "wrong type of relationship" and "wrong multiplicity".

Once all the single differences have been listed in a student's diagram, the compound differences can be searched for. The compound differences have priority on simple differences and lead to specific feedback during the interaction. The simple differences that occur separately will produce the usual feedback.

Example

Table 1 presents the output of the diagnostic algorithm (i.e. the differences between the diagrams) on the example "pen and felt-pen". Among all the differences, three subsets are recognized as compound differences (A, B and C) and the others are simple differences (D, E, F and G).

Implementation and evaluation

The diagnostic algorithm is implemented in Java. UML2 (an Eclipse Tools sub-project) of Eclipse modeling project has been used to represent the diagrams and realize the syntactic validation. An evaluation has been conducted with students' diagrams in order to assess the robustness and speed of the diagnostic method (Auxepaules & Py, 2010). The results show that the diagnostic quality is quite good for simple and medium problems,

and remains correct on complex problems. On the whole test base (n=82), 90% of the matches are relevant at 85% or more. The calculus time depends on the diagram's size: it varies between 0.2 and 4 seconds (on average: 2 seconds), in function of the diagram complexity. This is fast enough to provide pedagogical feedbacks online.

Table 1
Compound and simple differences

Simple differences	Compound differences
Wrong representation of class "Person" by an attribute "person" of class "Writing tool"	(A) Wrong representation of a class and omission of the related elements
Omission of relationship "belongs to" between classes "Person" and "Pencil"	
Omission of relationship "uses" between classes "Person" et "Pencil"	
Duplication of relationship "have" between classes "Pencil" et "Body"	(B) Duplication and transfer of a relationship
Transfer of relationship "have" from "Pencil" to "Pen"	
Transfer of relationship "have" from "Pencil" to "Felt-pen"	
Reversion of the direction of the relationship "has" between "Felt-pen" and "Top"	(C) Wrong representation of a relationship and direction reversion
Wrong type of relationship "has" between "Felt-pen" and "Top"	
Wrong multiplicity of "has" for "Felt-pen"	
Wrong multiplicity of "has" for "Top"	
(D) Reversion of the direction of inheritance between "Felt" and "Felt-pen"	
(E) Representation of the abstract class "Pencil" by an ordinary class	
(F) Merging of attributes "color" et "brand name"	
(G) Addition of an attribute "properties" into class "Body"	

PEDAGOGICAL FEEDBACKS

The output of the diagnostic algorithm is a flat list of differences between the student's diagram and the reference diagram. However, these differences can have quite different meanings, from a pedagogical point of view. Some differences only express a minor variation of the solution while others represent a great mistake. The environment must provide a different response in each situation and the formulation of the feedback messages must vary, according to the degree of certainty about the presence of an error.

Feedbacks formulation

The classification proposed by (Lemeunier, 2000) identifies three main forms of dialogic intervention: notify, question, and propose. These three categories have been used to design the pedagogical feedbacks in Diagram, in a graded manner. "Notify" draws the student's attention to some part of the diagram, or to the way he has modeled a particular concept. This modality is used for differences that are probably due to a slight variation in the students' diagram. "Question" consists in asking the learner about the diagram's properties. The questions are on a binary mode (the answer is "yes" or "no") and encourage the learner to mentally check whether the diagram satisfies a given property. "Propose", the most direct modality, consists in suggesting how to correct the diagram. This kind of help is provided only when the presence of an error is near-certain.

For each type of difference and each form of intervention, a sentence template can be filled with the name of the elements. For example, the template associated with the "relationship omission" difference and the "notify" intervention is "A relationship is missing between classes *C1* and *C2*". The variables *C1* and *C2* are simply replaced with the effective names of the classes when the message is displayed.

The appropriate form of intervention only depends on the type of the difference. For example, adding or forgiving an element is not necessarily an error. In this case, the environment only displays the "notify" and "question" messages. By contrast, when the direction of a relationship has been reversed, the "propose" message can be displayed, because the presence of an error is very likely.

During the interaction, feedback messages are graduated from the more general (notify) to the more precise (propose). For each difference, the most

general message is displayed first, and the learner can ask for a more precise message if he/she wants to.

Example

For each of the seven differences mentioned in table 1, a feedback message is produced following the modalities described in the previous paragraph. Three examples are given below.

The first compound difference means that the student has represented a class by an attribute, and has omitted the associated relationships. The feedback messages focus on the wrong representation of "person" but do not mention the absence of the relationships, because they relate to a class that is missing.

Notify : You have represented person as an attribute of the class Writing tool.
Question : Can the attribute person have properties or a behaviour ?
Propose : If person has properties or a behaviour, it should be a class

The compound difference C is formed of a type error and an inversion of direction (the errors about multiplicities are considered as minor errors, compared to these errors, and are masked). The feedback message focuses on the inversion.

Notify : You say that a top is composed of felt-pen.
Question : Is a top composed of felt-pen ?
Propose : I would rather say that felt-pen is composed of a top.

The simple difference G indicates that the student has added an attribute "properties" into the class "body". The feedback is composed of two sentences. As the insertion of an attribute is not necessarily an error, the "propose" sentence is absent.

Notify : Your class body has an attribute properties.
Question : Is properties an attribute of body ?

EXPERIMENTS

The experiments of the first versions of Diagram focussed on the contextual helps (Alonso, Py & Lemeunier, 2008), and the diagnostic module

has been evaluated apart (Auxepaules & Py, 2010). This article only presents the experiment carried out after the integration of the diagnostic module, which focussed on the relevance of the diagnostic performed in real situations and on the effects of the feedback messages on the students' behaviour.

Sessions

The experiment involved eighteen students in second year of University Diploma for Science and Techniques (DEUST) working with Diagram during four streams of three-hour practice session of UML modeling course. The experiment was conducted in an ecological context, i.e. during the normal schedule of the class. Prior to the session, the students were warned that Diagram's feedbacks are related to one particular solution and that they could ignore some messages if they were convinced that their own solution was a correct alternative. The teacher advised them to read the messages and decide whether a modification was necessary or not. During a session, each student practiced between three and five exercises. For each session, actions on the interface have been recorded.

Results and discussion

All the exercises of the second and the third sessions have been analysed. The first session has not been analyzed because the students were still discovering Diagram. For each exercise, the number of diagnostic calls, the total number of messages and the number of messages read by the learner have been calculated. We consider that a message is read if the student consults at least the first modality of the message (usually, the "notify" feedback).

All in all, 86 exercises were analysed, for a total of 306 calls of the diagnostic, which means that the diagnostic was called 3.56 times per exercise and per student. It results in 1 924 messages, 836 of which (43.45%) have been read. Some students used to read only one or two messages, immediately modify the diagram and call the diagnostic again, which explains this low rate. In some cases, however, the students did not read the messages because they were too many, which probably discouraged them. When the number of messages in a whole session is less than 50, the reading rate is 52.74% whereas if it goes past 50, the rate drops to 31.25%. To address this

problem, a degree of priority could be assigned to each message so that, in a first step, only the most important messages would be displayed.

The same message can appear several times during a session (at the maximum, as many times as the diagnostic is called), so that among the 836 read messages, only 565 are distinct. Among these 565 messages, it can be observed, in 11.6% of the cases, that the diagnostic module did not properly match the two diagrams. As a result, the feedback message is inappropriate. This occurs either when the student's diagram is very different from the reference diagram, both in structure and in class/relation names, or when two classes (or more) have very similar names. The diagnostic algorithm relies on structure and names to compare the diagrams, so that its performances deteriorate in these situations, as it was already noticed in the pre-tests of the diagnostic tool (Auxepaules, Py & Lemeunier 2008). However, the disturbance remained acceptable because the resulting messages were clearly irrelevant and the students ignored them most of the time.

The students' reactions after the relevant messages can be classified into three categories:

- Correction: the student modifies the diagram in accordance with the message (49.7%).
- Modification: the student modifies the diagram in relation with the message but some difference with the reference diagram remains (3.8%).
- No Effect: the student does not take the message into account and does not modify the diagram in relation with the message (46.5%).

The "modification" case is rare, which suggests that the messages are clear: when the students modify their diagram to take into account a piece of advice, this advice is properly understood (49.7%). However, the rate of messages that do not cause any reaction is fairly high (46.49%). Several hypotheses can explain this result. As students were told they could ignore some messages, we can assume that a lot of them did. Moreover, several calls of the diagnostic occurred a few minutes before the end of the sessions. In these cases, even if the students could read the messages, they had not enough time to make all the changes. It can also be assumed that a few messages have not been understood at all by students.

In order to determine what kinds of messages were concerned, the correction rate was analysed more precisely according to the type of difference. Table 2 indicates the percentage of each difference among the messages, and the correction rate for each difference.

Table 2
Percentages Among Read Messages and Correction Rates

Difference	% among read messages	Correction rate
Misrepresentation	28.46	51.41 %
Wrong multiplicity	27.65	50.72 %
Omission	15.83	67.09 %
Addition	10.82	24.07 %
Duplication	8.42	21.43 %
Duplication and transfer (compound difference)	4.61	86.96 %
Merging	2.81	21.43 %
Reversion	1.20	100.00 %
Transfer	0.20	100.00 %
Total	100.00	

The results show that Misrepresentation and Wrong Multiplicity are the most frequent differences (respectively 28.46% and 27.65%). The corresponding messages are easy to understand and quite precise about the modification to perform. The correction rate shows that, in half of the cases, the students decided to follow the advice. Messages related to an Omission difference are most of the time suggestive and easy to follow, especially when the missing concept is present in the problem statement: this explains the quite good correction rate. By contrast, the correction rate corresponding to Addition difference is low: as alternative solutions often include additional items, learners usually chose to keep the extra element. The Duplication and Transfer compound difference often happens when elements of a mother-class are duplicated in the children classes. The high correction rate suggests that the messages are efficient and help the student to understand the process of generalization, an important concept of object-oriented modeling. At last, the messages related to Duplication and Merge differences seem to be too general and could be improved.

It sometimes happens that, after having read a message, a student performs several corrections on neighboring objects, as if he/she was checking all similar objects in the diagram. This suggests that the messages efficiently solicit the checking metacognitive function.

Overall, the time spent on Diagram after the first call of the diagnostic tool represents about one third of the total time. The students devoted much time checking and improving their diagrams, by using the feedback messag-

es. At the end of a session, the diagrams appeared more complete and more accurate than if students had not benefited from the feedback messages.

CONCLUSION

We have presented a new approach for the design of a learning environment dedicated to class diagrams. The main contribution of this work is to overcome some limits of the existing systems. Diagram is an open environment, which does not impose constraints or restrictions on the problem statement, and allows the learner to express his/her creativity through the graphical interface. As a result, the range and the complexity of the problems that can be solved with Diagram are bigger. Moreover, this flexibility is compatible with accurate feedback messages, as in sophisticated tutoring systems. These improvements were made possible by the integration of an original diagnostic algorithm – a diagram matcher – that produces the list of the differences between the learner's diagram and the solution diagram. Several experiments in ecological context have shown that the interaction in Diagram encourages the learner's regulation activities, by supporting checking and self-correcting.

We have focussed on the first stages of UML learning, and that imposes limits on our work. The complete object-oriented modelling process relies on different types of diagrams, not only the class diagram. A first perspective is to extend this framework to other kinds of UML diagrams (object diagrams, component diagrams, and so on), that are required for more advanced students. The features of Diagram, like the task organization, the graphic tools and the help messages, could be adapted to that end.

The diagnostic module works with only one solution and is not able to recognize a correct alternative solution. To overcome this limit, it would be necessary to adapt the algorithm so that it could take into account different solutions. But this would require describing the relations between diagrams having common elements and then to merge the partial matching results, which is currently regarded as a very complex problem.

Lastly, a limit of the assistance is due to the fact that the feedbacks are produced in the same way, each time the diagnostic is performed. Another perspective is to memorize the successive diagnostic, in order to avoid repeating several times the same message, or, on the contrary, to insist if the learner repeats the same mistake for a long time.

References

Alonso M., Py D., Lemeunier T. (2008). A Learning Environment for Object-Oriented Modeling, Supporting Metacognitive Regulations. In *Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies* (pp. 69-73). Santander, Cantabria, Spain.

Alonso M.. (2009). *Conception de l'interaction dans un EIAH pour la modélisation orientée objet*. Doctor thesis in Computer Science, Université du Maine, France. http://cyberdoc.univ-lemans.fr/theses/2009/2009LEMA1007.pdf

Auxepaules L., Py D., Lemeunier T. (2008). A Diagnosis Method that Matches Class Diagrams in a Learning Environment for Object-Oriented Modeling. In *Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies* (pp. 26-30). Santander, Cantabria, Spain.

Auxepaules L. (2009). *Analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet*. Doctor thesis in Computer Science, Université du Maine, France. http://tel.archives-ouvertes.fr/tel-00455992/fr/

Auxepaules L., Py D. (2010). An Evaluation of Diagnosis in a Learning Environment for Object-Oriented Modeling. In Jemni M., Kinshuk, Sampson D. & Spector J.M. (Eds.) *Proceedings of the Tenth IEEE International Conference on Advanced Learning Technologies* (pp 102-104). Sousse, Tunisia.

Baghaei N., Mitrovic A., Irwin W. (2006). Problem-Solving Support in a Constraint-based Tutor for UML Class Diagrams. *Technology Instruction, Cognition and Learning*, 4(2), 113-137.

Baghaei N. (2007). *A collaborative constraint-based intelligent system for learning object-oriented analysis and design using UML*. Philosophiæ Doctor thesis in Computer Science, University of Canterbury, New Zealand, 2007.

Frosch-Wilke D. (2003). Using UML in Software Requirements Analysis - Experiences from Practical Student Project Work. In *InSITE- Informing Science and IT Education Conference* (pp. 175-183). Pori, Finland.

Habra N., Noben K. (2001). Teaching Object Orientation at the Design Level. In *Proceedings of the 5th Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts*, 16th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications OOPSLA (October 2001).

Lemeunier T. (2000). *L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle*. Doctor thesis in Computer Science, Université du Maine, France.

Moisan S., Rigault J.-P. (2009). Teaching Object-Oriented Modeling and UML to Various Audiences. In *Models in Software Engineering, Workshops and Symposia at MODELS 2009* (pp. 40-54). Lecture Notes in Computer Science 6002, Springer-Verlag.

Moritz S. (2008). *Generating and Evaluating Object-Oriented Designs in an Intelligent Tutoring System*. Philosophiæ Doctor thesis in Computer Science, Lehigh University, Pennsylvania, http://designfirst.cse.lehigh.edu/MoritzDissertation.pdf

Ohlsson S. (1994). Constraint-Based Student Modeling. In Greer J.E. & McCalla G. (Eds.) *Student Modelling : the Key to Individualized Knowledge-based Instruction* (pp. 167-189). Berlin: Springer.

OMG MOF (2010). *OMG's MetaObject Facility*. Retrieved November 26, 2010, from http://www.omg.org

OMG UML (2010). *Unified Modeling Language™, UML® Resource Page*. Retrieved November 26, 2010, from http://www.uml.org

Sorlin S., Solnon C., Jolion J.-M. (2007). A Generic Graph Distance Measure Based on Multivalent Matchings. Applied Graph Theory in Computer Vision and Pattern Recognition. *Studies in Computational Intelligence*, 52, 151-182.

Suraweera P., Mitrovic A. (2004). An Intelligent Tutoring System for Entity Relationship Modeling. *The International Journal of Artificial Intelligence in Education*, 14(3-4), 375-417.