Design Principles for a System to Teach Problem Solving by Modelling.

Gérard Tisseau[1], Hélène Giroire[1], Françoise Le Calvez[2],
Marie Urtasun[2], and Jacques Duma[3]
[1] Equipe SysDef - LIP6, Université Paris6, Boîte 169, Tour 46-0 2° étage,
4 Place Jussieu, F-75252 Paris Cedex 05.
e-mail: <Surname>.<Name>@lip6.fr
[2] CRIP5, Université René Descartes, 45 rue des Saints Pères,
F-75270 Paris Cedex 06.
e-mail: <Surname>.<Name>@math-info.univ-paris5.fr
[3] Lycée technique Jacquard, 2 rue Bouret, F-75019 Paris.
e-mail: dumajd@club-internet.fr

Abstract. This paper presents an approach to the design of a learning environment in a mathematical domain (elementary combinatorics) where problem solving is based more on modelling than on deduction or calculation. In this approach, we want to provide the student with a presentation which is close to the natural language formulations that she tends to give spontaneously, while ensuring a rigorous mathematical reasoning. To do so, we have introduced three modelling levels : first a mathematical formalisation of the students' intuitive process, then a conceptual and computational model allowing mathematical reasoning as well as communication with the student, and finally a presentation consisting in several interfaces, each one grouping problems of some class. These interfaces are viewed as nondeterministic "machines" that the student uses to build a configuration satisfying some constraints.

1 Introduction

Classically, the activities offered by an intelligent tutoring system (ITS) are centred on problem solving. In the domain of mathematics, proposed problems are often solvable by a deductive method: starting from the given facts, considered as a set of axioms, the learner can apply inference rules to get new facts and so find a way to the property to prove or to the result of the requested calculation. While this kind of process is promoted in such domains as transformation of algebraic expressions [9] and proof of geometric properties, it is not suitable to all domains. In other domains such as combinatorics (counting the number of ways of arranging given objects in a prescribed way), this unsuitability is apparent in the pupils' reactions to the problems: "I can't represent the problem", "I understand the solution given by the teacher but I don't understand why mine is wrong", "I'm proposing a solution but I have no way of finding out if it is right or wrong". We interpret these reactions as representational difficulties: the main part of the solving process does not come from a clever chaining of inferences or calculations, but from the elaboration of a suitable representation and from the transformation of one representation into another

equivalent one. The importance of representations in problem solving is studied for example in [1], [3], [11].

We are particularly interested in these domains because they offer a good opportunity to acquire some important abilities, such as the ability to elaborate models. But because of the difficulties they present and of the particular processes they require, they are not given much attention to in the official curriculum. In France, combinatorial mathematics in secondary school is very little developed. There is a deficiency here and we think that an ITS approach can help to fill the gap.

This paper presents an approach to the design of a learning environment in elementary combinatorics at secondary school level (final year), taking into account the characteristics of this kind of domain. We first present the "COMBIEN?" project within which this approach takes place, then we describe the activities proposed to the learner and the related interaction mode: the interfaces are viewed as nondeterministic "machines" that the student uses to build a configuration satisfying some constraints. We then discuss the modelling choices we made to offer the student a presentation close to the natural language formulations that she tends to give spontaneously, while ensuring a rigorous mathematical reasoning.

2. The COMBIEN? Project

Our project, called "COMBIEN?" ("how many?" in French) is to design and implement a learning environment allowing a student to familiarize herself with the mathematical concepts necessary to rigorously solve combinatorics problems.

2.1 Project Review

Since the beginning of the "COMBIEN?" project, we have carried on simultaneously theoretical reflections and implementation experiments. From our experience of teaching combinatorics in the classroom, we have defined the mathematical bases of a solving method, which we will call "the constructive method" in the rest of this paper. It is adapted to the usual student's conceptions and gives access to the mathematical theory of the domain [8]. We have defined a classification of the domain problems and solution schemata associated with the different classes. This classification has been used as a basis for an IA system that solves combinatorics problems [7]. We have defined a language to represent a problem and a solution according to the constructive method and we have implemented an IA system that can verify a solution written in this language. This language proved difficult to use because of its generality and abstraction, so we have defined a conceptual object-oriented model better suited to an interactive use. This model has been implemented in experimental data input interfaces [2]. In the same time, we have implemented different interface prototypes to explore various interaction possibilities [5]. From this, we selected a certain kind of interface that we called "configuration building machines" [8].

The paper presents the design principles that lie at the basis of these machines.

2.2 Pedagogic Objective

The combinatorics problems that we consider have all an analogous form: given a set or sets (e.g. a pack of 32 cards), count within some "universe" of configurations (e.g. the set of all possible five-card hands) the elements satisfying some constraints (e.g. including two spades and two hearts). Rigorously solving this kind of problem requires an appropriate, often abstract and complex representation using the concepts of set theory (e.g. sets, mappings, sets of sets). For a beginner, this approach is inaccessible because the representation is very different from the usual mental image.

However, some students are able to solve these problems [6]. Their answer, though correct, is not a real mathematical proof and they are generally unable to justify every aspect of it. Our objective is to introduce some basic concepts (e.g. sets, lists, mappings from a set to another, equality of sets defined by constraints) and some reasoning schemata that can be applied to them. The point is to acquire representations and practices of the domain in order to prepare a subsequent theoretical course, so that abstract theorems become understandable and can be effectively used. To reach this goal without excessive formalisation, we give the learner the possibility to express herself in terms similar to those of the usual student's answer.

To achieve our project, we first carried out a theoretical study of the domain in order to specify what we wanted the student to learn and what activities could help her to reach this goal. We have elaborated a mathematical formalisation of the method corresponding to the typical answer given by students [8]. We have called this method "a constructive method for counting", as it is based on a dynamic perspective describing a sequence of actions. Let us start from what the learner knows to bring her progressively to the awareness of the underlying abstract structures and of the rigorous methods of mathematical reasoning that apply to these structures.

The aim is not so much to make counting experts, as to train students to a modelling task and to make them able to represent a situation by a complex structure. This ability is essential in modelling and design activities, as shown by the increasing importance attributed to modelling methods in software engineering. We think that counting problems are a good starting point for that and that similar processes can be found in other domains like probabilities and algorithmics. Moreover, solving counting problems does not request only to know how to describe a structure, but also to reason about such a description.

3 The Student's Activities

The activities that we offer to the learner do not consist in counting but in describing. Before asking "how many?", it is necessary to first ask "what?", in order to give a precise representation of the configurations that must be counted. In the course of an activity, the student must fulfil three requirements, that we specify in this section: she must produce some result, she must use some specific resources to produce this result and the process that she follows must be generic.

The *result* that the student must produce at the end of each activity is an example of a configuration satisfying some constraints. For example, a five-letter word including two occurrences of the letter A. A possible answer is “ALPHA”. But we do not ask her to simply write an example, we force her to use certain resources to build it, in order to make her aware of the structure of the configuration.

The *resources* offered to the student look like a “machine” to which the student must give instructions so that it produces the desired result. We have chosen the instruction set and the inner workings of such a machine to familiarise the learner with the underlying representations and principles. For example, an instruction can be something like “select 3 positions and fill them with letters taken randomly in the set of the letters different from A”. The machine carries out this task and replies something like: “I have selected the positions 2, 4 and 5. At the position 2, I put an L, at the position 4, I put a P, and at the position 5, I put an H”. This presentation suggests the mathematical concept of a mapping from the set of the possible positions (1 to 5) into the set of letters. In order to solve the above problem, the learner will have to say something like:

1. I declare I am going to use 5 positions, from 1 to 5.
2. I declare I am going to use the usual alphabet with 26 letters.
3. At the start, all positions are free.
4. First, I want the machine to select 2 free positions and to fill them with “A”.
5. Next, I want the machine to select 3 free positions and to fill them with letters taken randomly in the set of the letters different from A, with possible repetitions.
6. After that my work will be done: the generated word will be a solution.

The learner’s activity is thus a kind of *programming*, with declarations and instructions. But, unlike classical programming, programs here can include nondeterministic instructions. It is not the learner who selects positions or letters, it is the machine that does it randomly, according to the specifications of the learner.

The *process* followed by the learner must be generic: she must write instructions so that, if the machine executed them in all possible ways, it would produce all solutions, nothing but solutions and each solution exactly once. This rule is what we call the *basic principle of counting*. Note that the machine does not actually execute the instructions in all possible ways, but only in one way, randomly chosen.

4 Interface and Interaction

A machine appears on the screen in a window like a form including interaction elements (widgets) of a classical type: buttons, menus, fields. Each machine concerns a particular problem class, but all machines have the same use pattern: the learner selects a problem to solve, she gives a model of this problem and then she gives a construction of a solution, i.e. a program. Problems are stored inside each machine not only as texts, but also in an internal representation.

The student selects the text of a problem in natural language by a selection device (a menu and a text field). She must then specify some elements in order to model the problem. For example, for the problems of the type “build a five-card hand with two

hearts and two spades", she must specify how many elements she wants and in which set the machine must select them. The possible sets are predefined in a menu, each answer being connected to her usual experience. The student can for example ask: "I want to build a set of [five] elements, taken from [a pack of 32-cards]" (the words in brackets are chosen from menus). This first modelling stage is thus very short while bringing an awareness of representation choices.

The student then describes her construction in the form of an instruction sequence. In this example, all instructions have the same type, like the following example: "I want [two] elements whose [suit] [is] [spades]".

The presentation is like a text written in natural language, with blanks to fill in. The syntax and the type of words suggest the role of each element, so it is not necessary to label them with abstract concepts like "attribute", "comparator", "value". In the internal representation, these labels exist and the elements that look like strings are in fact composed objects ("suit" is an instance of an Attribute class, and is related to its possible values and to the sets to which it can be applied). The menus' item lists are dynamic: for example the last menu to the right (possible values) has been updated when the student selected the attribute "suit".

The learner then asks the machine to execute the instruction, which provides a set of elements (for example a seven of spades and a queen of spades) that are automatically added to the current configuration, which always stays visible on the screen. This automatic addition makes what would be the final instruction of the construction implicit: form the reunion of all generated pieces. The resulting configuration could be for example: {seven of spades, queen of spades, ten of hearts, eight of hearts, king of diamonds}.

The machine restricts the learner's possibilities in order to guide her through her progression and to prevent mistakes caused by inconsistencies or confusions without pedagogic interest. However, the machine leaves her the possibility to make mistakes concerning combinatorics concepts and signals them. But the most interesting errors are the violations of the basic principle of counting: "I refuse to execute the instruction 'select two spades' because you already asked 'select three spades' in a preceding instruction. If I accepted, certain configurations would be generated more than once, for example …". The machine is specialised in a particular program schema, so it possesses the necessary knowledge to make this diagnostic.

5 Modelling Issues

The above interfaces are based on an internal representation implementing a model of the domain and of the problems. We discuss here modelling issues, the choices we made and the lessons that can be learned.

5.1 Problem classes and program schemata

As the learner's activity is a kind of programming, it is necessary to define the concepts involved in this type of programming as well as a language to express them.

We first defined a language based on basic set theory, which uses such concepts as interval, cartesian product, reunion, complement, subset, combination, function, injection, singleton and others. A program to build a configuration is composed of a sequence of declarations of set and declarations of mappings. These mappings are nondeterministic instructions to build pieces of the result, the building of the result being a reunion of pieces. We implemented a prototype using this language, but it became clear that it could not be proposed to a student. Although all instructions are elementary, it is not easy to understand how they are arranged to produce the result, and it is even less easy to invent this arrangement from scratch. The language is then like an assembly language: powerful and general, but unstructured and at too low a level. This phenomenon is likely to happen in most of the domains where problem solving relies on a modelling phase: a general mathematical theory that can represent the problems may be unsuited as a communication mean with the student. We then abandoned this approach and introduced the idea of a program schema, which we present below.

In fact the program has a structure: it first includes declarations of fixed sets necessary to the construction, then non deterministic instructions (mapping between two sets) using auxiliary instructions for selecting subset of former sets and finally the building of the result as a reunion of the pieces. Moreover this structure naturally results from a plan according to which it is possible to build a configuration as an union of mappings.

But asking a beginner to invent such a plan from scratch is too demanding. For an expert, this structure and this plan correspond to a classical resolution schema associated with a certain problem class. An expert knows different classes and their related resolution schemata, and this helps him to solve classical problems: he first analyses the problem to identify its class and then, he applies an associated resolution schema. In fact, using a classification to solve problems is a general method. J.G. Dubois [4] proposes a classification for simple combinatorial configurations, but it does not exactly cover the problems that we consider and it was not conceived to be used in a learning environment. We have defined our own classification of the problems of the domain allowing to solve the major part of the problems that are posed at the considered level. This classification served as a basis for Syrclad-Combinatorics, a system that automatically solves combinatorics problems [7].

The existence of classes on the screen, is suggested by the existence of different windows, each showing a “machine to build configurations”. Each machine is associated with a problem class and a resolution method (a program schema to build a configuration). Each machine proposes problems that can be modelled according to the class and solved by the associated method. The important idea is that this class and this method are embedded in the look and the inner workings of the machine.

The program structure is predetermined: first declaration schemata, then a sequence of instructions having all the same schema. The learner does not have to reinvent this structure, but she has still choices to make to reach her goal.

What can be remembered from this discussion is the idea to classify problems and to present each class in an appropriate way, rather than to use a unique general formalism.

5.2 Domain and problem modelling

A machine must build configurations according to the learner's instructions and reason about these instructions, for example to verify that they respect the basic principle of counting cited above. It must include a computational model of the domain that allows mathematical reasoning. On this point, we agree with [9] when he stresses the importance of models explicitly described at knowledge level, and the necessity to elaborate a "resolution theory" of the domain. Despite the anticipated variety of the machines (about fifteen), this model must be the same for all machines to reflect the internal consistency of the domain. It must allow to represent not only mathematical concepts, but also the problems and the learner's activity, which is centred on a specific method (our constructive counting method [8]).

We have defined such a model as an object-oriented conceptual model in the sense of software engineering. The higher level classes are Problem and Construction, thus reflecting the learner's activity: she is given a problem and she must represent it and solve it by providing a construction (a program). Each machine represents a certain construction type associated with a problem type. The definition of the conceptual model is based not only on the logical structure of the main concepts (problem, construction), but also on a concern that the student can use a representation close to her usual language. This has been made possible by the fact that the underlying theoretical method (the constructive counting method) has been itself elaborated from the students' usual answers.

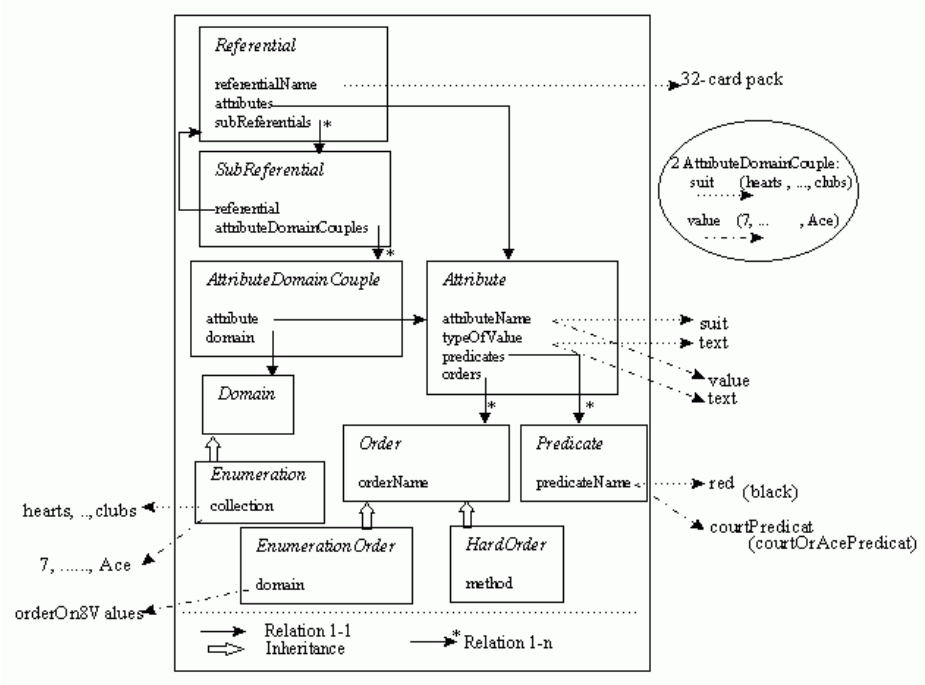

Fig. 1. The class "Referential" and the instance 32-card pack.

To compensate for the excessive generality of mathematical language, where every structure can be assimilated to a set, we have assigned different *roles* to the structures involved in counting, and namely to those involved in the constructive method. For example, the sets of objects (a pack of cards, the letters of the alphabet, tokens) used to form configurations have been called “referentials” and the set of configurations in which the solutions are selected has been called “universe” (it is similar to the possibility space in probability theory). This vocabulary and these concepts do not appear, to our knowledge, in usual courses on combinatorics, but they have proved helpful. A referential (Fig.1 above) is the union of subreferentials, each of them being defined as a set of couples attribute-domain. The "suit" attribute (for a card-pack) derives its values from the domain composed of the values hearts, spades, diamonds, clubs. A domain is defined as an enumeration of values or as an interval of numerical values or alphabetical values. To each attribute, we can associate predicates applying to values (for instance the "being red" predicate applies to suits of cards).

There are 2 types of universe: PartUniverse (used to model a set of parts of a referential, for instance the set of the 5-card hands from a 32-card pack) and AssociationUniverse (used to model a set of mappings from a set into another, for example the set of 5-letter words, each word seen as a mapping from the set of 5 places into the set of letters).

Some concepts do not concern directly mathematical objects (sets, elements, universe) but the means of expression offered to the learner. This introduces a metalevel, which we discuss below.

5.3 Metamodelling

To communicate with the machine, the learner must speak of objects included in referentials, such as playing cards and tokens that can be coloured and numbered. It is often heard that combinatorics does not require to know the nature or the properties of the involved objects, because the only goal is to count them. This is not quite true. In order to count configurations including playing cards, it is necessary to know that each card has a suit and a value, and what the possible values of these attributes are. Likewise, it is necessary to know what an even number is or what it means for a letter to be a vowel. The interface must therefore offer the learner some means of expression to refer to these objects, these attributes and these properties, and the conceptual model must include the related concepts.

In fact, the only important aspects are that the objects have attributes, that these attributes take their values in predefined lists and that it is not necessary to interpret or decompose these values: they are merely atomic. However, these values can have unary properties (a number can be even, a letter can be a vowel) and can be compared to each other. We have restricted the representation of objects to these modelling primitives, which are close to those of the relational model for databases.

To define a subset of a set of objects, the learner can use a declarative *selection formula*. The set of spades in the pack can be designated by: “the cards whose ‘suit’ attribute has the value ‘spades’”. This concept of selection simplifies the constructions that the learner must write. Instead of defining in advance the auxiliary

sets that she will need for her construction (and to do it for example with cartesian products, which is not very natural), she can introduce them only when she needs them, as a selection and without naming them. This comes close to a formulation in natural language: “I want two cards whose suit is spades”. The student can introduce such a selection formula, called "filter" in the model. It is a conjunction of properties, each of them being a comparison, (the suit = spades), a membership (the suit is in {spades, clubs}) or an assertion using a predicate (the number being even).

Finally, our choice consists in introducing a metalevel: instead of including predefined classes (e.g. the Card class, the Token class), we explicitly represent a representation formalism. The model includes for example the classes Attribute, Predicate, Domain, Comparison, Comparator. One benefit of this approach is to allow a teacher (or even a student) to introduce himself new types of objects without having to modify the conceptual model or to program the computer. Through the interface, the student uses this formalism to express her solutions, but the presentation hides the details of the formalism.

Problem Modelling: this formalism is used to model the problem. For each exercise, a wording in natural language is stored as well as an internal representation (class Problem). A problem is defined by a universe describing the type of configurations (for instance: the set of all possible five-card hands) and a set of constraints defining the relevant configurations (for example: the hands including two spades and two hearts). Three classes of constraints are defined: NumberConstraint, RestrictionConstraint, DistributionConstraint. An example of NumberConstraint is "with exactly 2 elements whose suit is spades". An example of RestrictionConstraint is "the letters at the positions 1 and 3 are vowels". An example of DistributionConstraint is "with exactly 2 cards of the same card-value and 3 cards of another card-value ".

Solution Modelling : In the constructive method, the solution of a problem is given in the form of a program of construction (class Construction), as defined in Sections 5.1 and 5.2. There are 2 types of constructions corresponding to the two types of universe: PartConstruction and AssociationConstruction. An example of partConstruction is : "choose 2 cards whose suit is hearts and 3 cards whose suit is spades". An example of associationConstruction is: "choose 2 places, fill them with the letter A, then fill the other places with distinct letters all of them other than A".

A construction is composed of subconstructions, so that the final construction (a 5-card hand) is the union of the sets given by the subconstructions (2 hearts, 3 spades). The constructive method demands that the constructions satisfy some mathematical properties in order to be valid. For example, for a part construction (choose 2 hearts then 3 spades), the subsets from which the choices are made (hearts, spades) must be disjoint. For each exercise the system has a complete model of the exercise (class Problem) and an associated solution (class Construction).

6 Conclusion

This paper shows the approach we are proposing within the “COMBIEN?” project for elementary combinatorics, a mathematical domain where problem solving presents difficulties that are more of a modelling and representation kind than of a

deduction or calculation kind. The implementation of an ITS in such a domain must take this specificity into account with regard to the activities offered to the student, the conceptual model used by the system, the presentation of the interface and the mode of interaction. We propose activities that require the student to model the problem and to carry out a construction with the help of nondeterministic machines. Two obstacles are avoided: the first is to be too abstract, with an excessive mathematical technicity, and the second is to be too concrete and not to lead to a real learning of mathematical concepts. To reduce abstraction, we introduce several machines associated with different problem classes and corresponding to methods that are effectively used by students. To link the student's activities with a rigorous mathematical reasoning, we have introduced two intermediary conceptual levels between the basic combinatorics theory and the interface presentation: first a constructive counting method and then a conceptual model that can be used in the reasonings as well as in the interface presentation.

We have implemented several prototypes to test these ideas and we are now beginning to integrate them in a running system. In order to implement different machines more easily, we have devised a general declarative formalism to specify the interfaces [10], and we have implemented a code generator that generates most of the interface code from specifications in this formalism. Our next work will be to specify and generate different machines with this generator, and to test them with students.

References

1. Abidin B., Hartley J.R., Developing mathematical problems solving skills, Journal of Computer Learning (1998) 14, 278-291.
2. Annell M., Malthet V., Giroire H., Tisseau G., Construction d'interface de saisie d'un problème de dénombrement. Rapport interne Lip6 janvier 99.
3. Cox R., Brna P., Supporting the use of external representations in problem solving: the need for flexible Learning Environments. JAIEd, 6, 2/3, 1995, p.239-302.
4. Dubois J.-G., Une systématique des configurations combinatoires simples. Educational studies in Mathematics 15 (1984), p. 37-57.
5. Duma J., Giroire H., Le Calvez F., Tisseau G., Urtasun M., Mise en évidence de styles de résolution, évolution de l'interface dans le projet COMBIEN ?, Actes des 4èmes journées francophones EIAO, ENS Cachan, T.2, Guin, D. & al. (eds.), Eyrolles, 1995, p. 245-256.
6. Fischbein E., Gazit A., The combinatorial solving capacity in children and adolescents. Zentralblatt für Didaktik der Mathematik, 5, 1988, 193-198.
7. Guin N., Changing the representation of a problem in order to solve it : use of classification, in proceedings of AI-ED 97, Kobe (Japan), August 18-22, 1997, p. 583-585.
8. Le Calvez F., Urtasun M., Giroire H., Tisseau G., Duma J., Les machines à construire. Des modèles d'interaction pour apprendre une méthode constructive de dénombrement. EIAO'97, actes des 5èmes journées EIAO de Cachan, Baron, M., Mendelsohn, P., Nicaud, J.-F., (eds), Hermès, 1997, p.49-60.
9. Nicaud J.-F., Building ITSs to be used: Lessons Learned from the APLUSIX Project, Lessons From Learning, Lewis, R & al. (eds), IFIP, North Holland, 1994, p.181-198.

10. Tisseau G., Duma J., Giroire H., Le Calvez F., Urtasun M., Spécification du dialogue et génération d'interfaces à l'aide d'interacteurs à réseau de contrôle. Actes de la 11ème Conférence francophone IHM'99 p. 94-101.
11. White B.Y., Frederiksen J.R., Causal model progressions for intelligent learning environments. Artificial Intelligence, 42, 1990, p.99-157.