NFP23 Program; Project No. 4023-27013
“Les systèmes explorateurs intelligents”; Prof. P Mendelsohn and D.K.Schneider.

Intelligent Learning Environments

August 1992

B.BORCIC, P. DILLENBOURG, M. HILARIO, P. MENDELSOHN, D. SCHNEIDER.

TECFA
University of Geneva
Faculté de Psychologie et des Sciences de l’Education

Abstract

The design of educational software has evolved during three decades, reflecting technical advances and changes in theories of instruction. In this project, we implemented some features of advanced interactive learning environments (ILEs): the multiplicity of teaching styles (the same content may be taught in several ways), the multiplicity of learning sources (experience, coaching, hypertext browsing), the use of a rich interface allowing direct manipulation and free exploration, the use of artificial intelligence techniques for creating complex problem situations and for supporting pedagogical reasoning.

We developed two pieces of software: ETOILE and MEMOLAB. The first system is a generic software, an Experimental Toolbox for Interactive Learning Environments. It provides designers with a set of tools and resources allowing them to create their own ILE. Globally, ETOILE provides the pedagogical architecture while the designer provides the domain expertise and creates the specific parts of the interface. ETOILE is not an authoring tool that can be mastered by any genuine author, it is a toolbox to be used by designers with programming skills in Common Lisp. ETOILE namely includes an Object-Oriented Production System, a hypertext, various interface management facilities and pedagogical knowledge bases. Its conceptual architecture is explained in the User Guide.

MEMOLAB is a learning environment for acquiring basic skills in experimentation methodology for human sciences. It illustrates the kind of systems that can be designed with ETOILE. MEMOLAB includes several components. The ‘lab’ allows learners to build an experiment. The simulation produces the results of this experiment. The ‘data tools’ allow the learner to visualize the results and to compute basic statistics. During their activities, learners receive the assistance of a tutor. The system includes five different tutors, implementing various teaching styles. The learner may also explore the various hypertextes in which he can find knowledge about designing experiments and knowledge about human memory.

The design of MEMOLAB reflects our efforts to translate a theory of development in terms of systems specifications, namely to structure a learning session into a sequence of micro-worlds, each microworld being characterized by a specific interface language. The psychological theory is encoded in the relationship between the language of successive microworlds. This architecture of a learning environment is supported by the knowledge structures used in ETOILE.

1. Artificial Intelligence and Education

Research in “AI and Education” is based on the computational study of learning and teaching processes. The design of intelligent learning environments (ILE) raises several challenges for AI researchers, both at the technical and epistemological levels. The reciprocity of the relationship between AI and ILE has been illustrated by the work of W. Clancey (1983). His attempts to use the well-known expert system Mycin as a training tool showed the limitations of early production system architectures. He demonstrated the necessity of explicitly representing problem solving strategies and thereby prefigured the second generation of expert systems (Steels, 1990).

Nowadays, several AI techniques are tested or developed through the implementation of ILEs. Recent developments on belief systems are explored in order to bypass the difficult problem of student modelling (inferring the student’s misconceptions from her behaviour on the interface) (Self, 1991). Dynamic planning methods, especially blackboard systems, are used to allow the tutor component to adapt its didactic plans to the student behaviour (Murray, 1989). Among other techniques, we point out the use of classifier systems (Hron, 1990), case-based reasoning techniques (Bloch and Farrell, 1988), explanation-based learning (Costa and Urbano, in press) and inductive learning (Dillenbourg, 1989) methods, neural networks (Mengel and Lively, 1991), distributed problem solving (Brazdil, in press). Some of these techniques are used in MEMOLAB, the system that we are developing.

However, the scope of artificial intelligence goes well beyond an inventory of techniques. AI is concerned with epistemological issues as well. The main challenge of researchers is to ground the design of systems on theories of learning and teaching. In this research, we explore the notion of cognitive architecture for a learning environment and we investigate how a cognitive architecture can be supported by generic computational tools.

2. Project Overview

Our ambition was to develop a library of *tools for designing* and implementing Intelligent Learning environments (ILEs). However, although generality is our research goal, we started by designing a particular system.We briefly explain why. Many research teams with a similar goal and a very much larger manpower have failed to create powerful authoring tools because they started with the wrong question. If one asks what functionalities are shared by any educational software, the generality constraints are so high that one inevitably comes out with a set of neutral interface tools and with procedures for specifying the sequence of activities. The pedagogical features of the authoring process are almost lost. To avoid this trap, our research methodology was to go from a particular case to general principles. We started this research with the design and the implementation of MEMOLAB. Both the conceptual and the computational work achieved so far have been possible because we tackled a concrete problem, rather than a set of abstract functionalities.

MEMOLAB is a collection of components. Most components are partly or totally domain-independent.The hypertext class for instance enables the author to create his own example of hypertext in any new domain with a very simple syntax. The ETOILE system is composed of the domain-independent components of MEMOLAB. ETOILE stands for ‘Experimental Toolbox for Interactive Learning Environments’. As its names indicates, ETOILE is not an authoring tool that can be mastered by any genuine author. It is a toolbox to be used by designers with programming skills in Common Lisp. It namely includes an object-oriented production system, an hypertext, various screen management facilities and a pedagogical knowledge base. Its conceptual architecture is explained below.

3. ETOILE

The basic idea is:

- The **designer** defines a *curriculum*. The curriculum is the skeleton of the future ILE. It is a network of *goals*. A goal defines a competence C in four ways:
    - a class of problems P that can be solved with C;
    - the situation (or application frame) in which these problems must be solved;
    - a rule base (domain expert) that can solve any problem of class P;
    - a hypertext node presenting the knowledge useful for this goal.
- The **toolbox** generates the learning activities around the curriculum. The basic learning activity is solving problems with the assistance of the expert.The form and amount of this assistance is determined by the active *tutor*.The toolbox includes five tutors. They propose various combinations of theory and exercises, various forms of progression, etc... which cover the range of courseware approaches (CAI versus microworlds). Selecting the most suitable tutor is done by the *coach*, according to the learner’s performance and initiatives.

3.1. Defining the curriculum

Defining a learning environment means creating a set of objects called 'goals'. These objects are pre-defined in ETOILE. The author must create instances of goals and fill their four slots:

- The **problems**
  A goal defines a skill by specifying a set of problems that the learner must be able to solve. 'Problem' is a pre-defined class in ETOILE. The author introduces some text which is presented to the learner when the problem is selected. The author should indicate the difficulty of each problem (with an integer from 1 to 5). The author may add domain-specific knowledge, by creating a sub-class of 'problem', with additional slots, specific to his system (as we did for Memolab).
- The **application frame**
  Learners solve problems through an interface which is specific to the ILE to be created. This interface is basically one or more windows, with a set of commands (menus, buttons, authorized object moves,...). ETOILE is based on the Common Lisp Interface Manager (CLIM) which provides us with the concept of application frame. A frame precisely gathers all the information that defines a situation, i.e. the window, the command tables, etc.
  In many learning environments, the problem solving situation remains the same throughout the learning process. ETOILE allows the author to diversify the situation, for instance to ask learners to solve the same kind of problems with a new set of commands (as it is the case in MEMOLAB). This feature of ETOILE supports the mechanism of language shift (see section 4.1).
- The **domain expert**(s)
  An expert for goal-X is a rule base able to solve any problem stored in goal-X. This rule base determines the interaction between the learner and the machine.For instance, diagnosis is carried out by searching for an expert's rule which implies the screen command that the learner has just performed. The conclusions of the expert's rules include commands which belong to the command table (stored in the goal's situation frame). This relation between the rules and the interface objects is made possible by the object-oriented inference engine we have implemented. The designer must respect the mechanism and the syntax of this engine (see 3.4).
  The author may define several experts for the same goal. This means these experts hold equivalent knowledge. This would support the implementation of multiple viewpoints. However, if the expertise of one is qualitatively or quantitatively superior to the expertise of another, then these experts should be associated to different goals.
- The **hypertext**
  Finally, the designer is expected to add to the goal a more readable representation of the expert's knowledge in the form of a hypertext. The learner will explore freely the information relative to the skills he must acquire. This information must be structured in granularity layers, in such a way that the learner can deepen some points and pass quickly over others. The functionalities offered by the hypertext are described in section 3.3.

3.2. Tutors and coach

In our first implementation, the **teaching style** was an implicit notion. The pedagogical agents (the coach and the tutors) behaved according to parameters, such as the teaching strategy (inductive, deductive or analogical), the level of difficulty, the learner's level of control, the 'inertia', etc. The combination of values for each parameter corresponded to a particular teaching style. The space of all possible combinations theoretically covered the whole range of styles. However, it appeared that some combinations of values were meaningless. The very idea of teaching style implies that some combinations of parameters have more consistency. For instance, a behaviourist approach favours immediate feed-back and a fine granularity in learning steps; it would be meaningless with large learning steps. To implement consistent teaching behaviours, it has been necessary to add complex rules which took into account the interactions between parameter values. This led us to write numerous and complex rules. When we tried to simplify and generalise our pedagogical rules, it became clear that our difficulties were due to the discrepancy between the nature of teaching style and their representation in our system.

The new implementation exploits the fact that the OOPS-engine supports multiple rule-bases (see 3.4). Each tutor has its own rule-base which implements a particular teaching style. Because rule-bases are hermetically sealed, they guarantee the internal consistency of teaching styles. This enabled us to reduce the number and complexity of rules and thereby to get a better control of the system behaviour. Five tutors (i.e. five teaching styles) have been defined: Papert (free discovery), Piaget (guided discovery), Vygotsky (apprenticeship), Bloom (mastery learning) and Skinner (programmed learning). The teaching style (i.e. the tutor) is selected by the coach. This selection is based on efficiency criteria and on the concept of **'pedagogical drift'**. Let us imagine that a learner follows the apprenticeship style, but fails often, receives a lot of feed-back, and asks frequently for help. In this case, the actual interaction will look more like mastery learning. This is what we refer to as a drift. The coach could then decide to 'jump' to mastery learning, rather than conduct an interaction which does not correspond to any style.

3.3. Hypertext

A hypertext system was deemed the most appropriate computational infrastructure for the knowledge resources offered to the learner. In traditional text processing systems, a document is no more than a sequence of character strings which are stored and retrieved in a strictly linear fashion. The essential innovation of hypertext consists in violating this linearity constraint. The hypertext approach views a document as a set of text chunks corresponding to conceptual nodes in a database; a system of machine-supported links between nodes and chunks allows interactive branching within and between documents. Hypertext systems exploit the latest developments in computer interfacing technology, particularly window management and graphics tools. Conceptual entities are displayed as windows of text which may contain any number of icons or buttons representing links to other nodes. The user can navigate freely around the hyperdocument by clicking on any of these graphics pointers.

While commercial hypertext shells are available, we have implemented our own hypertext system for several reasons:

- Building the hypertextes on a commercial shell would lead to portability and copyright problems when we distribute copies of our system for testing and validation purposes.
- An intelligent learning environment should give an overall impression of implementational coherence; by building our own hypertext tool, we can freely tailor it to share the look-and-feel of the other system components.
- Commercially available systems are usually closed products, and embedding them in a larger system leads to inevitable communication problems. This is a crucial point in our architecture, where the hypertextes should interact effectively with the other modules. For example, when the tutor feels that the student needs additional information on a given subject, she should be able to select and display the relevant sections of some hypertext.Conversely, a program can be run from within an hypertext.

In MEMOLAB, the entire hypertext subsystem is implemented as a class of objects called hypernodes. Each hypernode contains a title and a chunk of text with embedded formatting commands. Moreover, each node has an associated list of keywords summarizing the semantic content of its main text. These keywords serve as the basis of an indexing facility.

A hypertext is written by using a small subset of Latex commands which then have to be compiled in order to be used on-line. We included book structures commands (chapter, section,...). This way an author can combine the advantages of linear and 'web' text. A special graphic browser allows to display the linear structure of a hypertext and to jump to a section. Because of the use of a Latex, hypertext documents can be printed easily. For instance, the user manual of Memolab is available both as an on-line hypertext or as a printed document.

3.4. The OOPS engine

The expertise associated with each goal must be expressed by a set of rules. These rules are executed by the inference engine. We have implemented an inference engine in order to achieve a high level of integration between the rules and the other components of the system. The implemented production system has the following features:

- ***tight integration of object-oriented and rule-based programming***

In first-generation rule-based systems, the inference engine was assigned a special working memory, neatly separated from other data structures such as those manipulated by conventional code. This working memory was represented as a list of assertions which were matched with rule conditions to determine applicable rules. The advent of object-oriented languages led to an intermediate version where a global memory of object structures coexisted with a working memory of logical assertions: only the latter was accessible to the inference engine. MEMOLAB strives towards full integration of object-oriented and rule-based representation. On the one hand, rules should refer to and work directly on the hierarchy of classes and objects present in the global memory. On the other hand, rules themselves, their antecedents and their conclusions, as well as the other task-implementation structures needed to operate the inference engine (ex. the current conflict set) are all represented as objects. As a result, this integrated object-oriented, rule-based system provides:

- ***a powerful rule representation language***

To give an idea of the expressive power of the representation language used, the basic rule condition template is the following:

(<object> is <class> with <slot$_1$> <operator$_1$> <value$_1$> &
.. . &
<slot$_n$> <operator$_n$> <value$_n$>)

where <operator> can be =, <>, <, >, <=, or >=.

This rule condition is the object-oriented equivalent of a set of two-place logical assertions where <slot> is the predicate, <object> the first argument and <value> the second argument. Note that here, the rule interpreter allows variables not only for the arguments but also for the predicate, thus approximating the power of second-order predicate logic. In addition, a rule condition may include any function call:

(CALL <function> <argument>* <operator> <result>)

Universally and existentially quantified rule conditions are also allowed:

(FORALL (<variable>*) (<var-domain-condition>*)
<subcondition>*)
(FORSOME (<variable>*) (<var-domain-condition>*)
<subcondition>*)

Finally, any of the above rule conditions (or any conjunction of these) may be negated.

The most important rule actions consist in creating or destroying objects, or modifying slot values of objects. In the general case, these actions are applied to objects of the application domain; they are then base-level actions. But since rules themselves are represented as full-fledged objects, these same actions become meta-actions when applied to rules; that is, the same set of commands allow us to change rule conditions and actions, rule priorities, etc. Furthermore, metalevel-specific actions, like activating a particular rule or rule set, allow the user to modify the inference engine's control strategy. Finally, to enhance the system's flexibility, any user-defined function can be called from a rule's right-hand side.

- ***possibility of concurrent multi-expert reasoning***

Up to this point, we have referred to 'the' inference engine as though only one were possible in a given system. Actually MEMOLAB's production system framework has been designed with a view to implementing multi-expert systems. Any number of inference engines can be created, each associated with its own knowledge base (KB). For example, we could generate two instances of the class of inference engines to model two different tutors working concurrently on the same problem, each with her own knowledge base and problem-solving history. The inference engine specially dedicated to overall control will be a particular instance of the class of inference engines. The blackboard system can be managed by such a control-KB/ meta-engine tandem, while the domain knowledge sources can run their individual knowledge bases on (an)other inference engine(s).

- ***full computational support of metareasoning***

Since rules are represented in exactly the same formalism and within the same global memory as domain objects, rules can reason about rules and can examine their own (and other rules') components, evaluation contexts, history of past successes and failures, etc. Rules can also modify (even destroy) themselves and other rules as well as create new rules. Full metareasoning power is not available in classical production system languages: an OPS5 rule, for example, cannot directly examine its own instantiations because these are represented differently and separately from domain knowledge. To implement metalevel reasoning in such systems, programmers usually have to code special-purpose functions to retrieve metaknowledge inaccessible to rules.

4. Memolab

MEMOLAB is a learning environment based on the ETOILE architecture: its is a network of five goals. The domain of teaching is the methodology of experimental psychology. Basically, the learner's activities form a cycle: design an experiment, run it and analyze the output data. Here is an example of an experiment that the learner can design:

> Present a list of 10 words to two samples of 30 subjects. For the experimental group, the words are selected within the same semantic field. For the control groups, words have no close semantic relation. Let the subjects study the list for 5 minutes and will have to identify the first list words in another list of thirty words. Performance will be measured as the number of words correctly identified.

A challenge of our research was to articulate a computational architecture with psychological theories of learning. ETOILE supports the cognitive architecture of MEMOLAB, which is explained below, but does not coerce the designers to do the same. Our cognitive architecture divides the learning cycle into stages. In previous reports, we used the metaphor of a pyramid with several floors. Floors are discriminated by the operators provided to learners for solving the problem. In ETOILE, the author may define a different set of commands for each goal. In MEMOLAB, we paid attention to the relationship between successive goals and grounded our design choices in Cases's hypothesis on stage transition in children development.

Actually, in MEMOLAB, each goal does not have a completely different application frame (interface). For each goal, the problem solving situation is the LAB, i.e. a workbench where learners can assemble experiments and run them. However, the set of commands available is not the same in each goal. The LAB is the central place of interaction. It is connected to subordinate frames, such as the data-tools (where learners analyse the data collected through the experiment) and the simulation. The simulation is a set of procedures that approximate the results of the learner's experiment by comparing this experiment with the most similar experiment found in a database in which we stored experiments described in scientific literature. The simulation method is inspired by case-based reasoning techniques (see 4.2).

Finally, MEMOLAB includes two hypertextes. The 'handbook of methodology' includes several chapters, connected to the various goals of the system. It includes a theoretical introduction to the design of well-formed experiments. The 'encyclopaedia of memory' is another hypertext, independent from any goal, i.e. that learners can access at any time, and which provides them with the knowledge about human memory they could need, for instance, for determining the relevant variables.

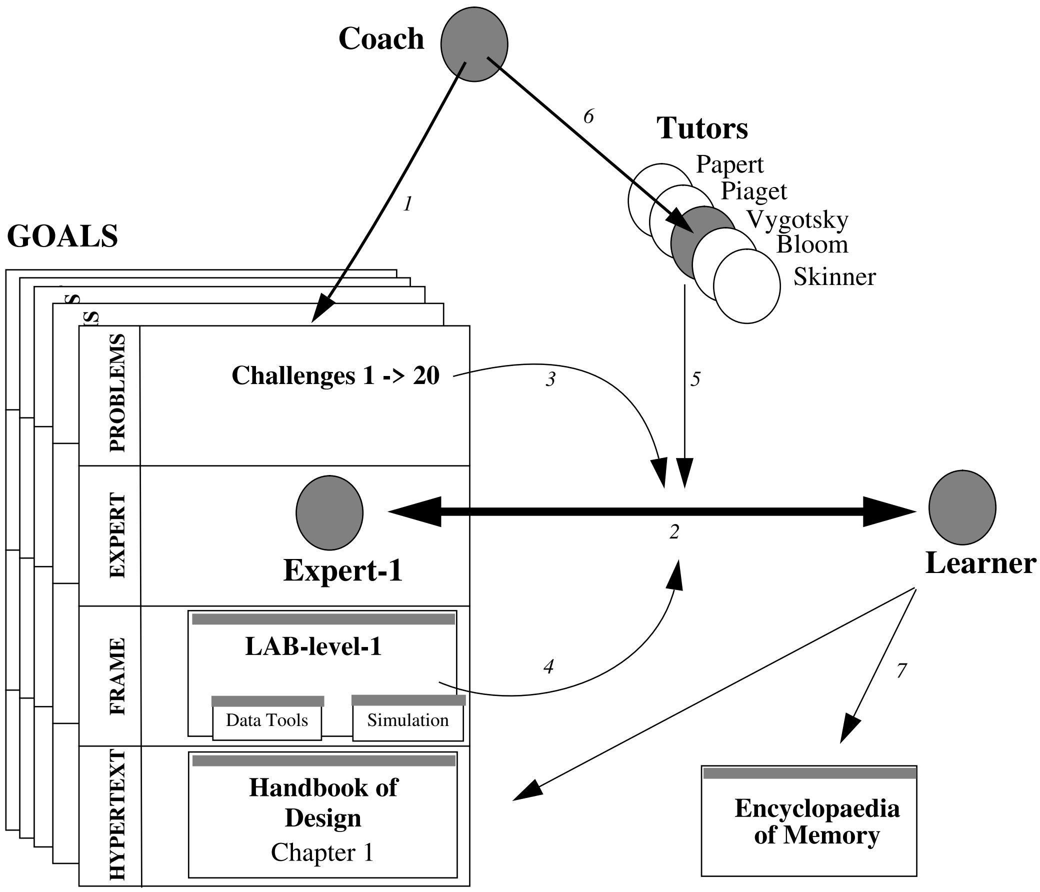

Figure 1 : The architecture of a MEMOLAB or any environment based on ETOILE: (1) the coach selects a goal; (2) the learner and the expert interact in order to (3) solve the problems specific to that goal (4) using the appropriate interface facilities; (5) the interaction between the learner and the expert is tuned by a tutor; (6) the coach chooses the active tutor; (7) some frames may be available through the system (not goal specific).

4.1. The Cognitive Architecture

Designing an intelligent learning environment (ILE) involves implementing some theory of learning and teaching. However, most available theories do not have the level of operationality required for implementation work.This is why the design of an ILE remains a real research work. Our challenge is to develop a framework that builds a bridge between theories and implementation. This framework translates psychological knowledge into terminology more relevant to computer scientists. It specifies the cognitive architecture of MEMOLAB.To present a detailed view of our intermediate framework goes beyond the scope of this report. We present the two concepts of this metaphor: the *pyramid metaphor* and the *language shift* mechanism.

The pyramid is a visual metaphor for the core structure of a learning environment. The pyramid represents the concepts and skills to be acquired by the learner, ranked bottom-up according to their level of "hierarchical integration" (see below). Learning consists in moving up in the pyramid. Each level of the pyramid is defined by two languages: the *command language* and the *description language*. The command language vocabulary is the set of elementary actions that the learner is allowed to do at some stage of interaction. The command language syntax defines how the learner composes sequences of elementary actions. The description language is the set of symbols (strings, graphics,...) used by the computer to show the learner some description of her behaviour. This description reifies some abstract features of the learner's behaviour in order to make them explicitly available for metacognitive activities (Collins and Brown, 1988).

The command and description languages are different at each level of the pyramid. The hierarchical nature of a pyramid implies that each level integrates its lower neighbour. This integration is encompassed in the relationship between the languages used at successive levels: if a description language at level L is used as a new command language at level L+1, then the learner is compelled to use explicitly the concepts that have been reified at level L.This is what we called the *language shift* mechanism (Dillenbourg, 1992): when she receives a new command language, the learner must explicitly use the concepts that were implicit in her behaviour. The meaning of the new commands has been induced at the previous level by associating the learner's behaviour with some representation. This representation is now the new command. The ILE structure can then be described as a sequence of [*action language, representation language*] pairs, a sequence in which the relationship between two successive pairs is described by the language shift mechanism.

Let us consider a simple example on solving equations. At some level of the pyramid, one can show the learner - with some graphics - that a good heuristic is to regroup the X's on the same side of the equation. At the next higher level, we can offer a button "regroup X" in order to compel her to explicitly use this heuristic in her solution. A more complete example will be described when presenting MEMOLAB.

The process by which properties that are implicit at some level of knowledge can be abstracted and explicitly reached at the higher level has been studied under the label of *reflected abstraction* (Piaget, 1971). The language shift mechanism has two uses. Firstly, it translates this psychological concept in a terminology more relevant for ILE designers. Secondly, it describes a pedagogical strategy (mainly inductive) to trigger reflected abstraction.

By applying the framework to ILE design, we ground the structure of this learning environment in a model of cognitive development. Reciprocally, this model of development can be tested through the difficult process of implementation. We found that this intermediate framework can be used as the interface between several theoretical backgrounds. Most psychological theories address actually only a specific facet of learning while an ILE designer must consider learning in its globality and complexity. Therefore, an intermediate framework should integrate multiple theoretical bodies of knowledge, each relevant for some aspect of reality. For instance, an ECS must account for the importance of discovery, for the role of practice and for the effect of coaching, because all of them occur at some stage of learning in the real world. The framework we propose can be read from different theoretical perspectives. We outline two of these psychological theories and concentrate on the one that is central for MEMOLAB: the theory of Robbie Case (1985).

From Campbell and Bickhard's (1986) viewpoint, the language shift mechanism can be viewed as a process of inducing *interaction patterns.* An elementary interaction associates some sequence of user's actions and the computer's description of this sequence. Inferring the meaning of the description language can indeed be described as the result of inducing the relationship between the actions performed and their representation (Dillenbourg, 1992). This corresponds to a view of knowledge as something that stands in the interaction between the subject and her environment. It creates a bridge between our model and current research on *situated learning* (Brown,1990), a "hot" issue in AI and Education.

Our intermediate framework also introduces the designer to the theories of Vygostsky. The *apprenticeship* idea is reified in the pyramid model by sharing control between the coach and the learner: when the learner is able to perform at some level L, the tutor must guide her activities at level L+1. This level L+1 corresponds to the concept of zone of proximal development (Vygotstky, 1978). At each language shift, the learner will assume a more important control of his solution process and the coach's guidance will be reduced. Moreover, Wertsch (1985) proposed a linguistic analysis of the internalization process that relates it to the language shift. He observed (in mother-child interactions) that the move from the inter-individual to the intra-individual plane was preceded by a language shift inside the inter-individual level: mothers replace a descriptive language by a strategy-oriented language (i.e. a language that refers to objects according to their role in the problem solving strategy).

The third but central theoretical background that fits with our framework is the neo-Piagetian theory of Robbie Case (1985). We focused on this theory because it has already a rather operational form. The key idea in Case's theory of intellectual activity and development is what he calls the "executive control structure". He believes that there is a valid general analysis of problem solving across domains viewed as the execution of a "mental plan". This plan is defined as a sequence, or program, of schemata. There are two types of schemata: "figurative schemata" represent states and "operative schemata" represent transformations. The mental plan is divided into three main sub-components.

- A representation of the "problem situation": this is the set of conditions relevant to the plan. The complexity of the representation will depend directly on the complexity of the problem.
- The goals to be achieved defined as a set of new states, or "solution situation".
- The "problem solving process" to be used, stated as a set of operations that transform the problem situation into the solution situation.

These components are further analysed. Elements of the problem situation are mapped to elements in the solution situation, and both are mapped to transformations in the strategy set. The result is a well-defined formal structure that allows Case to associate specific tasks with problem solving processes in a rigorous way.

Case formulates his general theory with reference to developmental stages as they are identified in specific domains. One of the characteristics of his theory is that it relates ***quantitative change***s within a stage to ***qualitative changes*** between stages: for example, an increase in the active unit capacity of working memory occurs within a stage, but helps to explain the transition to the next stage. Case distinguishes activity within a stage (i.e. a "sub-stage") by first defining what he calls "basic units of thought". He then notes that during development (and probably also during skill acquisition) we have the classical four stages:

- Perception of objects and motor activities
- Relations between motor activities
- Manipulation of dimensions (quantifiable variables)
- Second order dimensions (ratios)

How do we explain the formation of new units and the *transition between stages*? According to Case, each new sub-stage within a stage is characterized by the subordination of a new basic unit to the executive control structure: the first sub-stage has two basic units, the second has three and the third has four. The complexity of subordination reached at the final sub-stage (in stage n) is such that it corresponds to a basic unit at the next stage (stage n+1). When the executive control structure of stage n+1 subordinates two of these basic units passed up from below, it will enter its own first sub-stage... and so on. The last sub-stage of stage n can thus be considered as sub-stage zero of stage n+1. In other words, the four-unit control structure of stage n can be translated into a one-unit control structure at stage n+1. It is this formal process which Case calls "hierarchical integration".

It is an increase in "Short Term Storage Space" (STSS) that permits the transition from one sub-stage to the next. This increase is achieved within the "Total Processing Space" (TPS) which also contains the "Operating Space" (OS) utilized to control the active schema. STSS increases with age during development as a result of the maturation of the nervous system. It also increases during the learning of schemata as the result of an increase in the efficiency of the control structure: as the learner masters a task, the compilation of her knowledge frees up *short term memory* to hold new objectives.

There is an obvious mapping between the structure defined by Case and our intermediate framework.The control structures at each level of the pyramid integrate the control structures located at the lower level. The sequence of microworlds within the pyramid is structured as Case's view of development: quantitative variations define the improvement possible within some level (or microworld or stage) while the qualitative variations define the transition between two levels.The concept of stage transition is translated into the language shift mechanism. This transition is necessary when the learner try to solve problems that have too high memory load constraints. After the language shift, the learner has at her disposal new control structures that enable her to solve the problems with a reduced cognitive load. This ILE-oriented re-expression of Case's allow us to ground the design of MEMOLAB in this theory.

MEMOLAB is structured according to the principles of our framework. The MEMOLAB pyramid has three levels,. Creating an experiment means assembling concrete objects on the lab workbench. These objects are 'events' at the first level, 'sequences' at the second level and 'plans' at the third level. An event is the association of some subjects with some task and a measurement device. A task is defined by an activity, a material and some parameters. Building a plan (level 3) means creating a non-instantiated sequence on the workbench and generating the experiment by defining a n-dimensional table of factors (n being the number of independent variables).

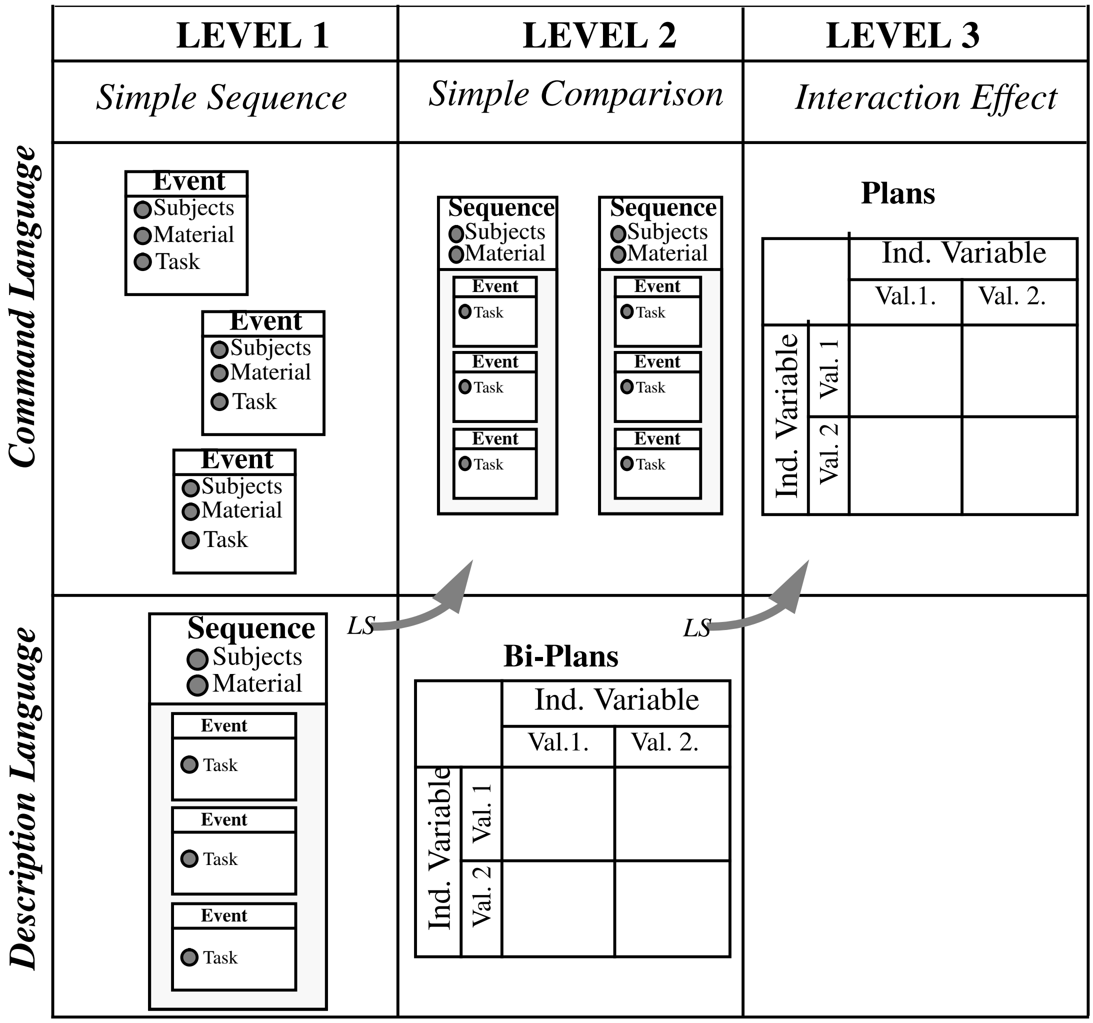

Figure 2 : The Cognitive Architecture of MEMOLAB

The three levels are:

- 1: Building non-comparative experiments (single sample); Units: events
- 2: Building comparative experiments (one factor, several samples); Units: sequences or paradigms
- 3: Building bi-dimensional experimental plans; Units: plans

An overview of the learner's activities at each level of the pyramid is presented in figure 2. For each level, we represent two screens, one displaying the command language and the other presenting the description language. At level one, the novice sequences events on the lab workbench. At the end of level 1,the learner receives challenges that induce the need for comparisons.The concept of sequence is reified at level one and used at level two as the way of designing new experiments. At the end of the second level, the necessity of taking into account the interaction of effects is induced by new challenges.The concept of plan is presented at level two and used for designing experiments at level 3.

The shift from one level to another, i.e. to shift from one language to another corresponds to some *qualitative* jump in learning. Within each level, we defined four sub-levels that are discriminated by *quantitative* differences. These differences result form an increase in the difficulty of the challenges proposed by the coach. More complex challenges compel the learner to handle a larger number of dimensions and hence increase the working memory load. At the end of the second sub-level, the learner receives challenges that already belong to the next level. This shows the learner the necessity to have more powerful control structures to solve the proposed challenge (As in Case theory sub-level i.4 is equivalent to sub-level i+1.0).The "reunitarisation" of the objects used at some level in a new more powerful object frees the memory resources necessary to solve the problem.

4.2. Case Based Simulation

The input of the simulation is the experiment that the learner has built in the LAB. In an experiment, pseudo-subjects are supposed to memorize words or other items and to try later to remember them. The output of the simulation is the list of words that each subject has remembered. The component that performs the simulation is called the Memory Machine (MM). The constraints that justify our design choices are related to the pedagogical function of MM and to the nature of available knowledge.

- ***Pedagogical constraints.***

The simulation has to produce results as similar as possible to the results that one would obtain if the experiment was performed on real subjects. However, psychological experiments have *de natura* a low fidelity. Moreover, this simulation is clearly not a substitution for real experimentation. Therefore, the validity of simulated results must be assessed in the light of pedagogical goals: to acquire basic skills in methodology of experimentation. This goal determines the lower limit for the validity of MM results. This lower limit can be expressed by the following rule:

> Minimal Validity Rule: if the learner builds an experimental plan with a factor F and based on a paradigm P, and if the literature includes knowledge on the effects of factor F, within the paradigm P, then the simulation must produce data in which the effects F can be identified.

More than being precise, the results must be explainable. Several ILE designers (White and Frederiksen, 1988; Roschelle, 1988) have shown that a qualitative simulation, although less efficient, is better suited to pedagogical purposes. Simulating an experiment involves an analysis of the structure and content of this experiment. The simulation produces a trace that can be shown to the learner.

- ***Knowledge Acquisition***

Contrary to many physical devices, human mnemonic behaviour cannot be predicted by a set of formulas or rules. Available theories do not constitute a consistent and exhaustive body of knowledge. Knowledge is distributed among a large set of experiments. Therefore, learners' experiments will be simulated by comparison to a similar experiment from the literature. This process is implemented with cased-based reasoning (CBR) techniques (Riesbeck and Schank, 1989).

Moreover, the literature does not cover the very large space of experiments that can be designed in MEMOLAB. Within the space of possible experiments, psychologists have concentrated their work on some avenues, or paradigms. (In experimental psychology, a paradigm is defined as a class of experiments.)The comparison of experiments within a paradigm brings more information than comparing experiments which differ in too many ways. Therefore, the case-retrieval process is not a simple search through a flat set of cases, with some similarity metrics. The set of cases (literature experiments) is partitioned into paradigms and a main stage during the simulation consists in identifying the relevant paradigm (see below, the paradigmatic analysis).

Another peculiarity of this domain is that the adaptation of the retrieved case to the target case cannot be driven by universal rules. Let us imagine that the learner builds an experiment where subjects have to memorize a long list of words and that MM retrieves an experiment which is similar on all points but the list length. The effect of list length is not universal, but varies according to other factors such as the delay between the memorizing and recall events. Therefore, case-adaptation is performed by a set of rules (named Vertical Adapters) that differ for each paradigm. In short, case-adaptation in MM is governed by rules that are themselves case-dependent.

- ***The Case Library***

The library includes the paradigm discrimination tree (figure 3) and a set of experimental sequences (cases). Each leaf of the paradigm tree corresponds to a subset of sequences found in the literature. Once we identify the leaf node corresponding to the learner's sequence, we retrieve the experiment which is the most similar to that of the learner. The vertical adapters (VA) are stored at different levels of the tree, according to their generality. Indeed a set of VA forms a simple production system.

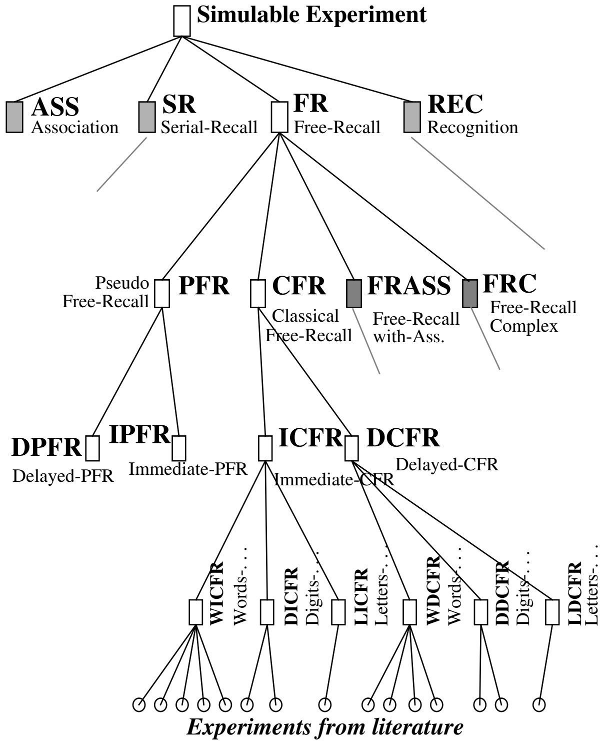

Figure 3 : The Paradigm Tree (as developed so far)

- ***The Algorithm***

MM must reason at the same time about the content and the structure of experiments. The content of the experiment refers to the sequences composing the experiment: what are the tasks, the material features, etc. This information is necessary to determine the paradigm to which the learner's experiment belongs. The structure of an experiment refers to the relationship between sequences. The fact that, for instance, two experimental sequences are identical on all points but the material length indicates that the learner intends to observe the effect of the material length. This 'factor' must be identified in order to respect the 'minimal validity rule' (see above). The *structural analysis* returns the factors identified within the learner's plan and the value (modality)of this factor in each sequence. This knowledge feeds the *paradigmatic analysis* process which identifies the most specific paradigm (leaf node) corresponding to the learner's sequence.

Let us consider first the simplest simulation process, i. e. computing the results for a sequence s_i. This sequence s_i is matched against the discrimination rule of each node N of the paradigm tree. If the match is positive, we store the vertical adapters of N and recurse on descending nodes. When a leaf node is reached, the associated set of literature sequences $S'_i = \{s'_{i.1}, s'_{i.2}, \dots, s'_{i.n}\}$ is retrieved. In addition, vertical adaptation rules have been collected through the tree: $VA_i = \{va_{i.1}, va_{i.2}, \dots, va_{i.m}\}$.

The selection process reasons on VA_i in order to select among S'_i the experimental sequence s'_i which is the most similar to s_i. The last stage is to retrieve the data of s'_i - denoted D'_i - stored with s'_i, and to execute VA in order to compute D_i:

$$VA\ (D'_i) \Longrightarrow D_i$$

Let us imagine that the learner's experiment includes two sequences s_1 and s_2. We can apply independently the same algorithm on each sequence and generate results for each sequence:

$$VA\ (D'_1) \Longrightarrow D_1$$
$$VA\ (D'_2) \Longrightarrow D_2$$

However, this independent processing does not respect the minimal validity rule expressed above. Let us assume that between s_1 and s_2, the learner has created some systematic differences in order to observe the effect *e* of factor *f* (denoted e_f). With independent sequence processing, we cannot guarantee that, when learners will compare D_1 and D_2, they will find e_f. In order to guarantee that differences between D_1 and D_2 correspond to the expected effect, one must generate D_2 from D_1 and what we know about e_f. When s_1 and s_2 are compared to a tree node, the effects stored with that node are compared to the structure of (s_1,s_2). This structure was determined during the structure analysis stage. If e_f is identified, the Horizontal Adapters *(HAs)* that are associated with e_f are stored. Then, the experiment is 'reduced', i. e. only s_1 is passed to lower nodes. Data for s_2 will be generated from D_1 and the VA of e_f.

```
VA(D'1) ===> D1
HA(D1) ===> D2
```

These two strategies are summarized in figure 4. The optimal strategy is applied when a known effect is identified within the learner's experiment and if this effect does not interact with other factors. The optimal solution can be applied on parts of the learner's experiments.

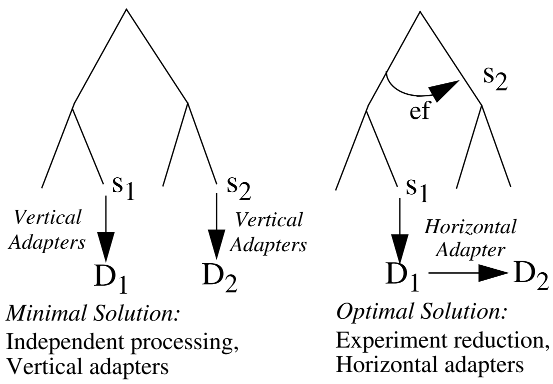

Figure 4 : Strategies for the Paradigmatic Analysis

5. References

BENTO C. et COSTA E. (1988) *Automatic Cognitive Modelling in Hierarchical Domains*. Proceedings of ECAI'88.

BLOCH G. and FARRELL R. (1988) *Promoting Creativity through Argumentation*. Proceedings of ITS-88, Montreal,pp. 243-249.

BRAZDIL P. B. (in press) Integration of knowledge in multi-agent environments. In E. Costa (Ed.). *New Directions for Intelligent Tutoring Systems*. Berlin: Springer-Verlag.

CAMPBELL and BICKHARD (1986) *Knowing Levels and Developmental Stages*.Karger.Basel

CASE R. (1985) Intellectual Development form Birth to Adulthood. Academic Press. New York.

CLANCEY (1983) *The advantages of abstract control knowledge in Expert system design*. Report N° STAN-CS-83-995, Stanford.

COLLINS A and BROWN J. S. (1988) The Computer as a Tool for Learning through Reflection, in H. Mandl and A. Lesgold (Eds), *Learning Issues for Intelligent Tutoring Systems. Springer* Verlag. New York, pp. 1-18.

CONKLIN, J. (1987). Hypertext: An Introduction and Survey. *Computer*, Vol.. 20, No. 9, pp. 17-41.

COSTA E. and URBANO P (in press) Machine learning, explanation-based learning and intelligent tutoring systems. in E. Costa (Ed) *New Directions for Intelligent Tutoring Systems*. Berlin: Springer-Verlag.

DILLENBOURG P. (1989) Designing a self-improving tutor: PROTO-TEG. *Instructional Science*, 18, 193-216.

DILLENBOURG P. (in press) The Language Shift: a mechanism for augmenting learner's control and abstraction. To appear in P. WINNE and M. JONES (Eds). *Foundations and frontiers in Educational Computing Systems*. Berlin: Springer-Verlag.

DILLENBOURG P. and MENDELSOHN P. (in press). The genetic Structure of the Interaction Space. To appear in E. Costa (Ed). *New Directions for Intelligent Tutoring Systems*. Berlin: Springer-Verlag.

DILLENBOURG P. and SELF J. A. (in press) Designing human-computer collaborative learning. in C. E. O'Malley (Ed), *Computer-Supported Collaborative Learning*. Wiley & Sons.

HRON A. (1990) *A diagnostic system for qualitative knowledge*. Paper presented at the third ITS Seminar in Arhus, Denmark, 5 - 9 December.

MENDELSOHN,P. and DILLENBOURG,P. (in press) Le développement de lenseignement intelligemment assité par ordinateur, in Symposium Intelligence Naturelle et Intelligence Artificielle, Pari, PUF.

MENGEL S. and LIVELY W. (1991) On the Use of Neural Networks in Intelligent Tutoring Systems. *Journal of Artificial Intelligence in Education*. Vol.. 2. , N 2. , pp. 43 - 56.

PIAGET J. (1971) Biology and Knowledge. The University of Chicago Press. Chicago

RIESBECK C. K. and SCHANK R. C. (1989) *Inside Case-Based Reasoning*. Lawrence Erlbaum Publishers. Hillsdale, New Jersey.

SELF J. A. (1991) *Formal approaches to learner modelling*. Technical Report AI-59, Computing Department, University of Lancaster (UK).

STEELS L. (1990) *Components of Expertise*. AI Magazine, Vol. 11, n 2, pp. 28 -49.

VYGOTSKY L.S. (1978) The Development of Higher Psychological Processes (edited by M.Cole, V. John-Steiner, S.Cribner and E. Souberman). Harvard University Press. Cambridge. Mass..

WERTSCH J. (1985) Adult-Child Interaction as a Source of Self-Regulation in Children. in R. Yussen (Ed). The growth of Reflection in Children. Academic Press, pp. 69-97.

6. Research Team

TECFA is an academic team active in the field of educational technology. It belongs to the School of Education and Psychology of the University of Geneva. It is directed by Professor Patrick Mendelsohn and includes about 10 collaborators. The team is in charge of the computer courses within the school and provides services within the computer-based learning community. TECFA main research interests are the applications of artificial intelligence to education, the cognitive effects of educational software and the communication issues with new technologies (distance education, multi-media systems,...).

This research project is funded by the Swiss Research Fund under a special research program (PNR 23), entitled 'Artificial Intelligence and Robotics'. The contractors are Prof. P. Mendelsohn and D. Schneider. The research team is multidisciplinary. It includes P. Mendelsohn (psychology), B. Borcic (computer science), M. Hilario (computer science), D. Schneider (political and computer sciences), P. Dillenbourg (educational and computer sciences). Address for correspondence:

• TECFA, Faculté de Psychologie et des Sciences de l'Education, Université de Genève, Route de Drize, 9, 1227 Carouge. Switzerland. Phone +41.22.705.96.93. Fax: +41.22.342.89.24
• E-mail: pdillen@divsun.unige.ch.