The IRIS Shell: "How to Build ITSs from Pedagogical and Design Requisites"

Ana Arruarte, Isabel Fernández-Castro, Begoña Ferrero, Jim E. Greer

International Journal of Artificial Intelligence in Education (1997) 8, 341-381

The IRIS Shell: "*How to Build ITSs from Pedagogical and Design Requisites*"

Ana Arruarte[1], Isabel Fernández-Castro[1], Begoña Ferrero[1] and Jim Greer[2]

[1] *Department of Computer Languages and Systems*
Computer Science Faculty
University of the Basque Country UPV/EHU
649 Postakutxa, 20080 Donostia, Spain

[2] *ARIES Laboratory*
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada S7N 5A9

Abstract. The goal we pursue in our research is to build a shell for helping human instructors to develop intelligent teaching-learning systems in a wide range of domains. We aim to provide a system where a previously defined architecture can be adapted automatically into a new tutor using a set of instructor-generated requirements. Trying to provide a sound basis for this tool, we use a theory of instruction that integrates *cognitive processes*, *instructional events* and *instructional actions* within a three-level framework that relates them. In this paper we extract, through the analysis of both the cognitive theory and the generic architecture, the requirements of the different components of a tutor and integrate them in IRIS (IRakaste-Ikaste Sistema; *Teaching-Learning System)*, a shell for building teaching-learning systems. Moreover, we embed these requirements, cognitive principles, and design requisites in a shell in order that human instructors can follow them easily. Various design issues and an example of building a tutor for mathematical differentiation using IRIS are presented.

INTRODUCTION

Research in computer-aided instruction has been varying its focus according to new trends in the educational and psychological fields. Ranging from Computer Assisted Instructional systems, to Intelligent Tutoring –ITSs, and now to Intelligent Learning Environments –ILEs, systems permit successively greater degrees of freedom for the learner. This evolution has led us to develop new architectures in the area of tutor modelling, learning, and domain representation.

Thus, during recent years, while our research group has been working in the ITS area, our goals have likewise evolved. First, we developed a generic ITS architecture evaluated with two applications: TUTOR (Fernández et al., 1993), which focused on conceptual domains, and INTZA (Gutiérrez, 1994), which focused on conceptual and procedural domains. At present our goal is to work towards developing a set of tools valid for building ITSs and ILEs in a wide range of domains (Figure 1).

With the idea of defining computer-based tools for constructing ITSs supported by a sound theoretical basis, we proposed a cognitive theory of instruction, the CLAI Model: Cognitive Learning from Automatic Instruction (Arruarte et al., 1996a), which integrates cognitive processes, instructional events and instructional actions within a three level framework. This theory addresses two main aspects of tutoring. On the one hand it is suitable for integrating into an instructional plan various recent educational imperatives such as using multiple teaching styles (Srisetamil & Baker, 1995-1996), generating flexible tutorial responses (Reye, 1995), and adapting a tutor's performance to changing situations (Van Marcke & Vedelaar, 1995; Vassileva, 1995a-1995b). On the other hand it is valid for multiple domains (Khuwaja et al., 1996) involving conceptual and/or procedural contents.

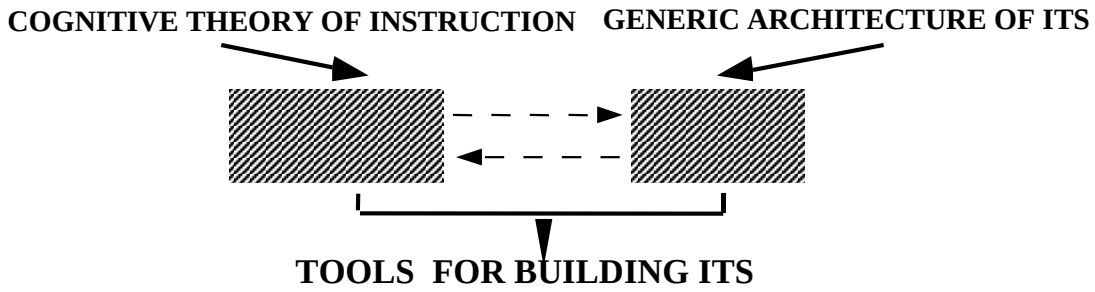

Figure 1. Underlying basis involved in the design of tools for building ITSs

In this paper, after reviewing different shells developed for building ITSs, we describe the requirements for the components of a tutor obtained from a comparative/parallel analysis of a cognitive theory of instruction (Section 2) and a proven generic architecture (Section 3). These requirements are integrated inside the Instructional Plan of the generic architecture in Section 4. In Section 5 we present the IRIS Shell for ITSs, illustrate the process of constructing an actual ITS using the IRIS Shell, and present an example of an actual tutor we constructed for mathematical symbolic differentiation. Finally, in Section 6 we draw some conclusions.

Shells for building ITSs: A review.

The goal of developing shells for building ITSs has been considered as a main focus by the ITS community. In this subsection we briefly describe some of existing systems relevant to our work.

IDE-Interpreter (Russell, 1988) is a planner-based adaptive tutoring system which automatically generates instructional courses. It uses knowledge structures produced by **IDE** (Instructional Design Environment) as knowledge sources to guide the process. IDE (Russell *et al.*, 1990) is a hypertext-based tool whose aim is to provide assistance to the designer in creating a course; for this purpose the knowledge describing the course's content and structure and the set of rules corresponding to the instructional method must be supplied. Due to the great amount of knowledge that must be specified when generating a course, initial course construction is a complex task for the designer. Nevertheless once a course is generated, it can be reused or modified.

GTE (Generic Tutoring Environment) defines a formalism for representing the instructional expertise of experienced human teachers in terms of instructional tasks, methods and objects (Van Marcke, 1992). The underlying assumption of GTE is that this knowledge is not specific to any individual situation, and can be generally re-applied in a variety of situations or to completely new domains. *Instructional tasks* are the building blocks of an instructional process; the great majority are very general (test, teach,...). An *instructional method* is a knowledge-based description of a procedure for carrying out an instructional plan. It includes context-dependent knowledge in order to execute tasks. The *objects* are instructional primitives manipulated directly by the knowledge sources and the instructional methods. GTE provides a library of instructional tasks, methods and objects that can be used to author a new teaching strategy. It collects the knowledge that human teachers use during the instructional process but lacks a formal theoretical basis. The author argues that existing theories are so weak that they are largely irrelevant for computational purposes.

FITS (Framework for ITSs) (Ikeda & Mizoguchi, 1994) is a framework for building ITSs. It is developed to examine what functions can be realized as a domain-independent framework among those which are needed in ITSs. Its final goal is to identify those generic instructional functions and provide a set of *building blocks* useful to cover essential tasks for teaching. Following expert system technologies, each building block is designed as a domain-independent problem solver for its corresponding generic task. FITS is totally oriented to knowledge engineers and does not offer any suitable interface for human teachers who may be non-experts in the computational area.

COCA (CO-operative Classroom Assistant) is a system developed to allow for authoring ITSs. It makes a clear distinction between three types of knowledge relevant to the design of an ITS (Major & Reichgelt, 1991). The first type of knowledge concerns the representation of the material to be taught or domain knowledge representation. COCA uses a simple object-oriented representation language by which each fragment of domain knowledge is represented as a frame with a number of user-defined attributes and attribute values. The second type of knowledge, *i.e.* teaching strategy, concerns the way in which the material is to be taught. Finally, COCA includes some meta-strategic knowledge to determine the conditions under which to apply a certain teaching strategy as well as to revise a teaching strategy in the light of previous results. For representing both strategic and meta-strategic knowledge COCA uses production rules that must be constructed by teachers. The difficulty of specifying a large amount of knowledge with rules is one of the bigger weakness attributed to COCA (Major, 1995). Trying to solve this problem and using COCA as a kernel, REEDEM (Reusable Educational Design Environment and Engineering Methodology) (Major, 1995) offers graphical windows to assist teachers with the task of authoring.

Regarding most of the shells introduced above, one aspect that deserves special mention is the difficulty instructional designers find when they confront the interfaces necessary for developing the different components of an ITS. First, many instructional designers are non-experts with computational field; moreover, building even the most simple tutor requires such a large amount of data that is very hard for the instructor to specify the system requirements. Another weakness of these shells is that, although some of them include or observe various pedagogical principles, none of them incorporates a whole teaching-learning theory.

Many other ITS shells have been and are now being developed, but they are less relevant for our work. In some of them, for example in the Eon system (Murray, 1996; 1998), the initial approach is quite different. Eon has been designed under the assumptions of a user base-line similar to typical users of commercially available authoring systems for traditional computer-based teaching. As such, many of the tools supported in standard CBT authoring systems are included as basic tools in Eon. In other cases, for example CREAM-Tools (Nkambou *et al.*, 1996), they are focused in only one of the components associated to the architecture of an ITS. The main goal of CREAM-Tools is to facilitate instructional designers to generate curricula.

A PRACTICAL COGNITIVE THEORY OF INSTRUCTION

Advances in psychology of learning, especially in our understanding of higher-order cognitive processes, have led to a steady evolution in understanding the teaching-learning process. Instead of viewing the learner as an agent reacting to the stimuli generated by the teacher, the learner is considered to be an active participant in the teaching-learning process (Weinstein *et al.*, 1986). From this learner-centred perspective we developed a cognitive theory of instruction, the CLAI Model (Arruarte *et al.*, 1996a), that integrates issues about human learning, cognitive processes, and learning strategies together with aspects from teaching processes. The theory, based mainly on Gagné's (Gagné *et al.*, 1988) and Shuell's (Shuell, 1985) ideas, can be used to guide the practical development of ITSs. With the same goal of driving automated instructional design and development, Merrill and the ID_2 Research Group (1996) developed Instructional Transaction Theory, a set of prescriptions for determining appropriate instructional strategies to enable learners to acquire instructional goals. As with the CLAI Model, Merrill's theory is based on Gagné´s ideas.

The CLAI Model integrates *cognitive processes* (CPs), *instructional events* (IEs) and *instructional actions* (IAs) within a three-level framework that relates them. CPs are the psychological processes or mental activities that have to happen inside the student in order to achieve learning about a particular chunk of content. IEs are classes of events that occur in a learning situation. Each event works by providing the external conditions of learning. External conditions refer to various ways in which the instructional events outside the learner activate and support the internal cognitive processes of learning. Finally, IAs are instances of instructional activities that the system uses to provide instruction about both specific contents of the teaching domain and learning strategies.

Gagné (Gagné *et al.*, 1988) made an early attempt to relate learning and instruction across CPs and IEs and his theory has been shown to provide an adequate basis for integrating more modern educational practices (Petri *et al.*, 1987). Gagné's IEs are appropriate for various kinds of learning and they provide a general framework that must be adapted for each particular instructional situation (tutorial programs, drill and practice, simulations, etc.). Figure 2 shows a snapshot of the links relating the levels of CPs and IEs (two superior levels) adopted in the CLAI Model from the ideas of Gagné.

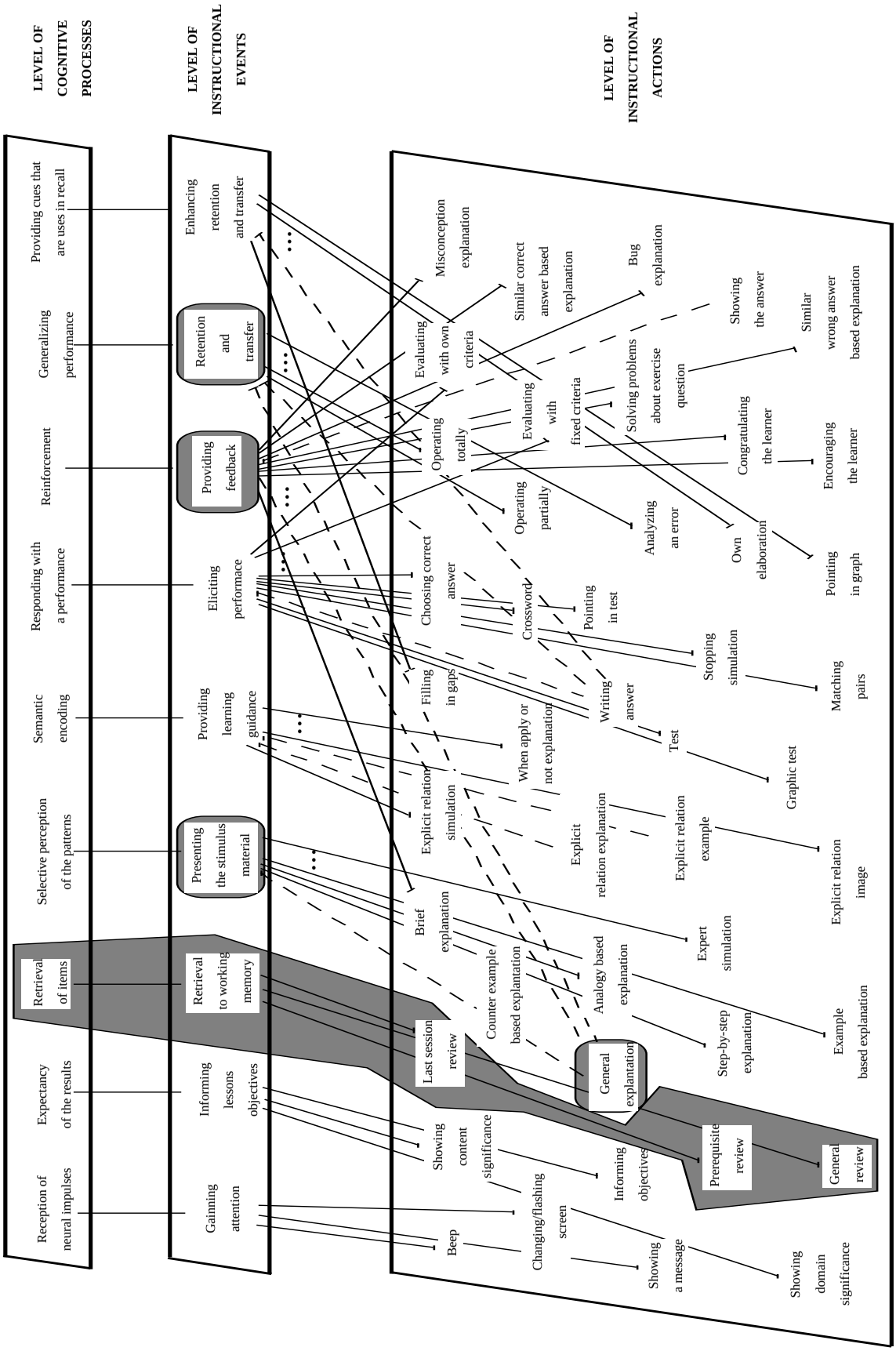

Figure 2 - A snapshop of links relating three leves of the CALI Model (adopted from Arruarte *et al.*, 1996a)

Each IE has been instantiated in our theory in a group of concrete IAs. In this way a single IA may refine several IEs depending on the context of use, and therefore it may trigger several CPs. The remarked/shaded area of

Figure 2 indicates that the three IAs *Last session session review*, *Prerequisite review*, and *General review* can be used indistinctly to refine the IE *Retrieval to working memory* and therefore each of them can trigger the CP *Retrieval of items*.

By using IAs the system provides instruction about both the contents of the teaching domain and about learning strategies. In this way, the more complete the set of IAs, the greater the flexibility and power of the system and, as a result, the possibilities of successful learning increase. So far we have identified 45 IAs, many of which were integrated and tested in INTZA (Gutiérrez, 1994). The rest of them were adopted after a deep analysis of didactic software widely used in the local school environment by the ORIXE team (Folch, 1993). IAs have been described as objects with three associated attributes:

- **Initiated-by**: indicates whether the instructional action can be initiated by the instructional agent, by the learner, or by either of them.

- **Instructional-event-refined**: the instructional event/s that can be refined using this instructional action.

- **Cognitive-process-triggered**: the cognitive processes that are eventually triggered by the instructional action. Although this information can be also obtained from the instructional events, it has been included in order to provide greater expressivity.

For example, the instructional action *General-explanation*[*] is represented in this way:

General explanation
INITIATED-BY: Instructor, Learner
INSTRUCTIONAL-EVENT-REFINED: Presenting the stimulus material, Providing feedback, Retention and transfer
COGNITIVE-PROCESS-TRIGGERED: Selective perception of the patterns, Reinforcement, Generalizing performance

Most of the identified IAs can be initiated either by the system or by the learner. Thus following her own preferences, the learner can use an individualized learning strategy. In order to successfully trigger the adequate CPs, these strategies can be initiated by a wide variety of IAs integrated into the ITS. Nevertheless, we think it is necessary to extend this set of possibilities by providing the learner with a set of additional learning

[*] The ruled boxes and dashed lines of Figure 2 indicate graphically that the instructional action *general-explanation* can be used to initiate either the instructional event *presenting the stimulus materia, providing feedback* or *retention and transfer* and therefore, can trigger the cognitive processes *selective perception of the patterns, reinforcement* or *generalizing performance.*

techniques. These techniques, named *supported actions,* can be initiated by the learner at any point of the teaching-learning process. *Note-book, Summary window, Schema window and Underlining tool*[*] are the four supported actions that are widely recognised in the educational literature (Hernández & García, 1991) as useful learning supports. Their goal is to provide support for learning via *scaffolding* (Soloway *et al.*, 1992), i.e., *"providing a set of mechanisms that enable a student to perform a task, but which fade away as the student becomes more expert"*. Using instruction, we try to encourage the student to activate the appropriate cognitive processes in order to learn the presented information. However, we have no guarantee of what the student will do. The student is free to choose the kind of instructional actions she prefers and can select the supported actions most suitable for her.

Two main requirements arise when we try to integrate this instructional model into a learning environment or tutoring system. First, it is necessary to define the required kind of instructional tutoring according to the previously referenced three levels: CPs, IEs and IAs. Second, it is necessary to select the most suitable subset of instructional and supported actions depending on both the domain and the kind of learner.

INTZA'S KERNEL: A GENERIC ARCHITECTURE. SYSTEM REQUIREMENTS FOR THE IRIS SHELL

Although most of the existing ITSs are individually crafted for specific domains, an interest arises in developing generic ITS Shells valid for a wide range of domains. TUTOR (Fernández *et al.*, 1993) and INTZA (Gutiérrez, 1994) were ITSs developed by our research group with this eventual aim. TUTOR treated conceptual domains while INTZA is able to work with both conceptual and procedural domains. By learning a conceptual domain we mean learning and understanding both the concepts of the domain and the relationships existing between conceptos. Meanwhile learning a procedural domain implies also learning how to execute procedures consisting of a sequence of defined steps. So we can say that the architecture of INTZA generalises the architecture of TUTOR by extending the kind of domains it can manage. TUTOR and INTZA

---

[*] *Note-book:* The learner uses an auxiliary tool, for instance a window, for recording/copying relevant ideas from the text or paragraph.
Summary-window: The learner elaborates and writes out the essential ideas or concepts of a text establishing their inter-relationships.
Schema-window: This is used to construct graphical representations of relevant concepts and their relationships.
Underlining: The learner is allowed to underline some words or fragments of a text.

(both developed in LISP) were instantiated over the domains of teaching elementary programming and electrical power plant process training, respectively.

Figure 3 shows the components of the architecture of INTZA that constitute its kernel. The components correspond directly to those of the ITS basic architecture (Wenger, 1987; Sokolnicky, 1991). This running architecture amounts to a crude Shell which can be used as a general schema to develop new ITSs in similar domains. Next we briefly describe each component and identify a set of requisite attributes related to its internal structure.

- Domain is the explicit representation of the content to be taught to the student. The Domain Component is composed of the *Pedagogic Domain* and the *Domain Expert.* The *Pedagogic Domain* contains the subject matter *i.e.* the set of topics to be taught (procedures, facts, and typical problems or malfunctions) organized from a pedagogical point of view. The *Domain Expert* encodes the expert's abilities for carrying out procedures and for detecting and solving problems and correcting malfunctions. It is used to provide the trainee with exemplars of expert operations as well as to compare trainee and expert performances in order to identify differences and potentially reveal deep errors. The hierarchical representation of the *Pedagogic Domain* (based on an extended genetic graph) is formed by a conceptual network where each node represents a topic to be taught. Relationships existing in the domain are expressed using links between nodes. Moreover topics are described using a set of descriptors which give the necessary meta-information for traversing the network.

  It is necessary for this component to be able to represent the domain in terms of the referenced structural elements, *i.e.* topics, relationships and descriptors. INTZA works with physical processes and has the ability of simulating these processes. If an intended domain has such characteristics and simulation is required, one would have to specify and implement the Domain using procedural and heuristic information. This would be needed to facilitate the analysis of the learners' responses to different scenarios presented as practical exercises.

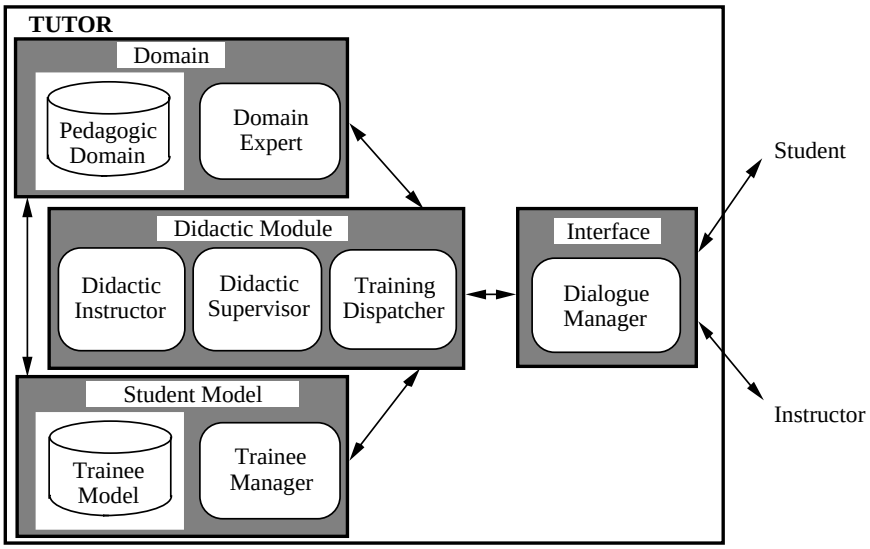

Figure 3. INTZA's kernel

- Student Model records information about the student´s skills and knowledge acquired during the teaching sessions. The Student Model consists of the *Trainee Model* and the *Trainee Manager*. The *Trainee Model* records the long-term characteristics of the user together with her acquired knowledge and skills. The *Trainee Manager* analyses, critiques, and evaluates the trainee's interactions, and consequently updates the Trainee Model. The Student Model is based on an overlay approach extended in order to represent misconceptions. A declarative description is used to characterize both the knowledge of the student and her learning objectives.

  It is necessary for this module to specify both the initial characteristics of each learner and the kind of learner interactions that will be treated by the tutor (*student´s objectives)*.

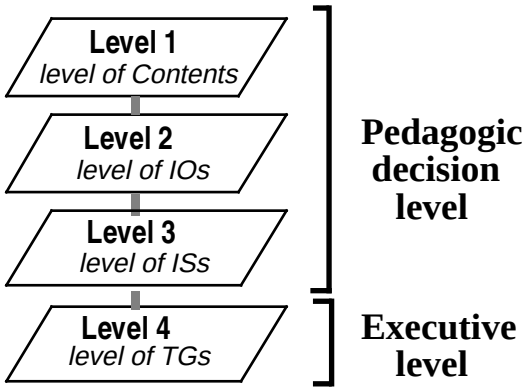

Figure 4. Layers in the Instructional Plan of INTZA

- Didactic Module plans and performs the teaching/learning session. The Didactic Module supports two different training styles: *guided*

training and *free-exploration*. This module generates, re-plans and carries out the Instructional Plan (IP) for the session. In INTZA, the IP is organised into four layers (Figure 4): *Contents*, *Instructional Objectives* (IOs), *Instructional Strategies* (ISs) and *Tutor Goals* (TGs). The first one consists of a sequence of contents *(concepts* and *procedures)* to be acquired by the learner. The second layer represents a sequence of IOs (*cognitive skills*) that the tutor wants the student to achieve in the training session – currently implemented IOs are extracted from Bloom´s taxonomy (1956). At the third layer, each IO is refined into a sequence of ISs. ISs include activities for both programming the session and guiding and motivating the student during the accomplishment of an IO. Finally, at the fourth level each IS is refined into a sequence of TGs. The TGs are the basic actions that the tutor has to execute in order to accomplish an IS. The first three layers in the IP correspond to the *pedagogical decision level*, which reflects the pedagogical decisions developed by the tutor, and the last one corresponds to the *executive level*, which reflects the refinement of the IP in directly executable tutor goals. During the session, each IO is refined into a sequence of *Instructional Strategies* (ISs), meanwhile, each IS is refined in a sequence of the *Tutor Goals* (TGs).

This component is structured in several cooperating submodules: i) the *Didactic Instructor* generates the IP for the session, deciding which IOs should be reached and which IEs should be applied during the training process; ii) the *Training Dispatcher* carries out the IP; iii) the *Didactic Supervisor* decides how to treat the new conditions in the session caused by the trainee's interactions and how to integrate this treatment in the IP.

The Didactic Module uses a set of IOs, ISs, TGs and all the rules and plans related to their selection and refinement. To treat special situations such as errors, students requests, unforeseen changes in the Student Model, and changes in the available time for the session, the tutor introduces special strategies for describing, correcting and recovering from errors as well as for responding to the student´s objectives and changes in the student model. So, the Didactic Module is described in terms of Instructional Objectives, Instructional Strategies and Tutor Goals together with the set of plans and rules to select them. In Section 4 we show the relationship between the CLAI Model (CPs, IEs and IAs) and the pedagogic decision level of the instructional plan of INTZA (Contents, IOs, ISs).

- Interface / Dialogue Manager copes with the communication process between the Tutor and the human agents.

The Interface / Dialogue Manager would normally be developed separately for each ITS, as it is completely depending on the domain. So at interface level, it is necessary to specify the set of necessary, tutor-specific graphical tools and communications channels.

The INTZA kernel comprises an already implemented core which can be adapted to building new tutors sharing the same structure according to the requirements specified by human instructors. The INTZA system was developed following an object-oriented design and implementation, and contains several rule-based modules which determine the nature of the instructional support tools described in this paper.

INTEGRATING THE CLAI MODEL WITH A TUTOR SHELL

Taking into account the architecture schema defined by the INTZA kernel and its concrete specification and implementation, the production of a new tutor maintaining its structure imposes a set of requirements over the contents of each one of the identified components.

The IRIS Kernel: Instructional Planner

The integration process is especially relevant in the module responsible for the pedagogic decisions of the system, *i.e.* in the *Didactic Instructor* module that builds and executes the Instructional Plan (IP). The IP defines the behaviour of the tutor during the whole teaching-learning session. It is responsible for deciding what to do next at each point during the session. In order to be effective, the IP requires both domain-dependent and domain-independent knowledge about instruction (Macmillan *et al.*, 1988). The former is required for capturing instructional methods of expert teachers for a particular subject area. The latter is required in order to improve cost-effectiveness in developing processes for tutors in new domains. In section 5 we will show how both kinds of knowledge are treated in our system.

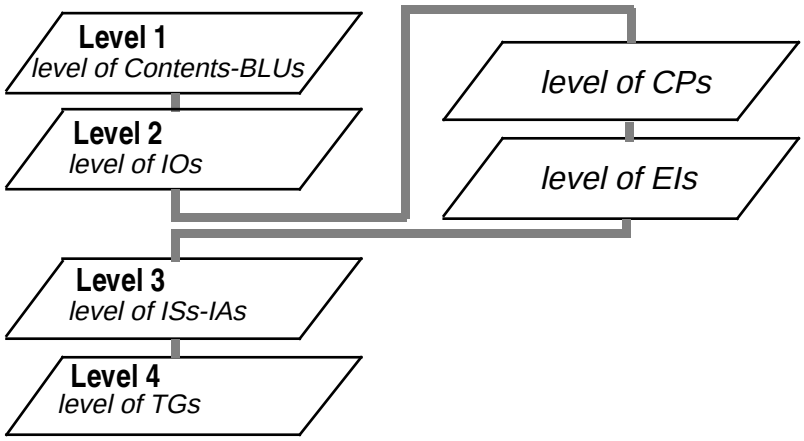

Figure 5. Layers in the Instructional Plan

In order to fully integrate the aspects considered in the CLAI model we include two new levels in the INTZA's IP: those of the *Cognitive Processes* (CPs) and *Instructional Events* (IEs). The former corresponds to the set of mental activities in which the student must be engaged for the learning to take place; the latter corresponds to the events that the system carries out in order to activate the required cognitive processes in the student's mind. Figure 5 shows the modified/final IP including the two new layers.

Focusing just on the pedagogic decision level, formed by the first five layers, the main requirement for building an ITS that integrates aspects from the CLAI Model and from the generic tutoring system architecture, is to specify the preferred instructional tutoring method in terms of BLUs* , IOs, CPs, IEs and IAs.

The structure of the instructional plan determines the basic cycle of general activities of the tutor:

1. determine the next Basic Learning Unit to teach
2. determine the Instructional Objective to reach
3. select the Cognitive Processes for reaching the previously chosen IO
4. refine the CPs into a sequence of Instructional Events
5. select the most suitable Instructional Actions for applying these IEs

An example of an IP is shown in Figure 6. It is based on the mathematical symbolic differentiation domain and illustrates a possible refinement of the IP for teaching a particular BLU, specifically a procedural rule for differentiating a product expression.

* Basic Learning Units (BLUs), described in detail in subsection 4.2, refer to those minimal contents of domain knowledge to be taught to the learner.

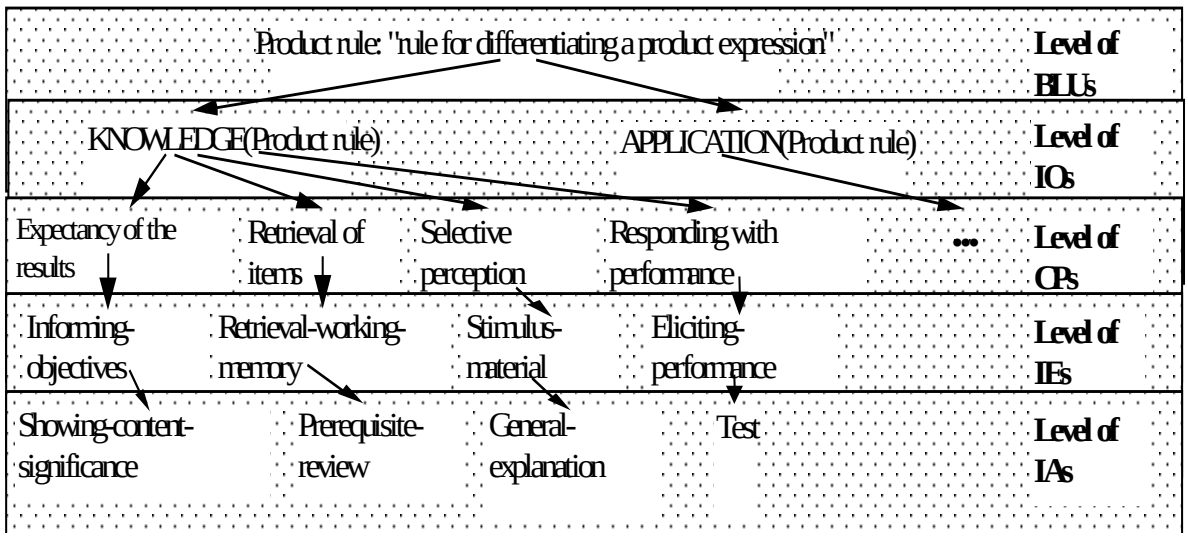

Figure 6 - Example of a IP

Once the last layer of the pedagogical decision level, *i.e.* level of IAs, is obtained, the tutor refines each IA in a sequence of Tutor Goals (TGs) or *directly executable* procedures. For example, the IA *Prerequisite-review* is refined as the sequence of TGs: *obtain prerequisite list, show prerequisite until last prerequisite.* Each one of these TGs can be executed directly by an implemented algorithm.

The process of building the Instructional Plan is implemented by means of a rule-based paradigm. Each level of activity involves a particular set of rules used to refine the current level of the IP in terms of the subsequent level. The set of rules is automatically customized for each tutor, depending on the requisites specified by the human instructor; namely characterization of the domain, the kinds of student who are going to use the teaching-learning system, and the pedagogical aspects of the tutor. These requisites have a direct relationship with the rules identified in an initial hierarchy. In Section 6 we will show how the initial hierarchy is customized according to the specified requisites.

Domain knowledge requirements for IRIS

Instructional Design Theories study in detail knowledge representation paradigms from an educational perspective; they are primarily concerned with prescribing optimal methods of instruction to bring about desired changes in learner knowledge and skills. In particular, these optimal methods must specify what must be learned (Scandura, 1983) and some way to represent knowledge. There are three basic features relating to ITS design that are taken into account in different Instructional Design Theories: learning units or kinds of teaching-learning contents, relationships between elements, and skills to be reached.

Basic Learning Units

Instructional design theories based on Merrill's (1983) 'Component Display Theory' present *facts*, *concepts*, *procedures* and *principles* as the Basic Learning Units (BLUs). From this perspective it is possible to determine a complete schema of knowledge representation organized from three different views: *conceptually*, when a conceptual structure (taxonomy of parts or types) is used to organize the concepts and the facts, e.g. in the TUTOR system (Fernández-Castro *et al.*, 1993); *procedurally*, when a procedure based structure is used for the domain organization, e.g. in the INTZA system (Gutiérrez, 1994); and, *theoretically*, when a structure based on principles or theories is used for this organization, e.g. in the WHY system (Stevens *et al.*, 1982).

Once a particular domain is chosen, the immediate task is to determine the kind of characterization required to cover the contents to be conveyed to the student. For instance, in the mathematical symbolic differentiation domain, for practicing differentiation rules, two types of BLUs, namely concepts and procedures, are sufficient to represent the domain. The selection of one BLU or another will determine, amongst other things, the kind of specific internal representation. For example, a concept (see subsection 5.1) will have, as well as other attributes, several associated texts, a difficulty level, and assessment items. Meanwhile, a procedure will need to include its steps or actions, some application examples, and practice-tests.

Relationships between elements

In order to establish a pedagogical view useful for selecting and/or sequencing the content, Reigeluth (Reigeluth *et al.*, 1978) references four different kinds of relationships between teaching contents of the same type:

- *Requisite relationships*. E.g.: "The learner must know X (or must be able to do X) before learning Y (or be able to do Y)". They appear in the TUTOR (Fernández *et al.*, 1993) and INTZA (Gutiérrez, 1994) systems.
- *Conceptual relationships*. E.g.: "X is Y-type ", "X is part of Y". These appear in TUTOR and INTZA as well.
- *Procedural relationships*. These are either *order relationships*, e.g.: "The learner must do X before doing Y" (INTZA) or *decision relationships*, *e.g.*: "Given a condition A, the learner must do X rather than Y or Z".

- *Theoretical or principles-based relationships*. These can be *cause-effect relationships, e.g.:* "Y is the effect of X" as in the WHY system (Stevens *et al.*, 1972), or *prescriptive relationships e.g.:* "In order to achieve Z it is necessary that X and Y happen in a specified order".

So, the BLUs include in their representations some attributes which relate them to one another. Some relationships we propose and use are *is-a, part-of* and *prerequisite* (see subsection 5.1). In our mathematical differentiation domain we identified three type of relationships: requisite, conceptual and procedural.

Instructional Objectives

Instructional Objectives (IOs) refer to the application of particular skills over BLUs. They form a useful part both in planning the teaching process and in creating procedures to assess the learner's knowledge. The instructional objectives can be hierarchically organized in order to establish a structure for the didactic activities. The most accepted taxonomical classifications in the psycho-educational field have been the *taxonomy of teaching objectives* (Bloom *et al.*, 1956) and the *taxonomy of learning objectives* (Gagné *et al.*, 1988). The former identifies three different learning categories: *cognitive*, *affective*, and *psychomotor.* Inside the *cognitive* category, six IO have been defined: *knowledge, comprehension, application, analysis, synthesis* and *evaluation*. INTZA uses the objectives *knowledge, applications* and *analysis*. Gagné identifies six categories of learning: *intellectual skills, cognitive strategies, verbal information, motor skills,* and *attitudes*; only the first three are valid for acquiring static knowledge and problem-solving skills.

Depending on the domain and the characteristics of the various learning activities proposed by the instructor, each BLU in the domain can be completed by adding the skills (*i.e.* IOs) that must be developed in the learner. For instance, a BLU procedure can be learned superficially just by knowing its associated steps, or in a deeper way by knowing how to execute it.

In the example of Section 5, we specify two of Bloom´s IOs: *knowledge* or remembering of ideas or phenomena (*e.g. the learner knows the definition of the procedural differentiation rule for a product expression*) and *application* or correct use of the procedure (*e.g. the learner is able to differentiate a product*).

These three main domain knowledge features (*basic learning units, instructional objectives* and *relationships between elements)* are fundamental to many Instructional Design Theories, and also are needed by IRIS for generating the domain model related to each new tutor. Other systems integrate these features, but not in a particularly explicit way. IRIS

requires an explicit representation of these features and so provides the instructor with tools (as shown in subsection 5.1) for specifying their actual values. These features identify two different levels in the domain representation (Arruarte & Fernández, 1995), i) *Concrete Level,* focused on how to represent the basic contents or elements associated with the teaching-learning process (*i.e.* basic learning units) and ii) *Pedagogical Level,* focused both on the representation of the skills to be mastered by the learner (*i.e.* instructional objectives) and the relationships between these elements in order to get an effective teaching-learning process (*i.e.* relationships). Similar pedagogical relationships between *learning units* or contents to be learned by the students are also represented in the ECSAI system (Gavignet, 1994).

Learner Requirements in IRIS

Although the Learner Model or Student Model is a widely recognized component of ITSs there is no consensus about the information it should include. Moreover, there is debate about whether or not it is an essential component in order to achieve an effective and efficient instructional system (Holt *et al.*, 1994).

From a pragmatic point of view we consider the learner model as a needed component whose information is a basis for the Pedagogic Module to make individualized planning decisions and thus to produce a more efficient instruction process. Even though "learner models by themselves achieve nothing" they do provide other components with useful data for diagnosis, sequencing content, determining the level of explanations, and so forth (Self, 1990).

Different criteria have been used to classify learner models. Kok (Kok, 1991) classifies these criteria on the basis of several parameters: *why are users modelled*, *who is modelled*, *what is modelled*, and *how are users modelled*. Verdejo (Verdejo, 1992) classifies them along the following six dimensions: *the use of the model*, *the extent of the user's domain knowledge to be represented, shadow or deep model, generic or individual model*, *permanent or temporal model* , and *predefined or inferred model*. These six dimensions are valid for both classifying and defining learner models; moreover, the last three are related to model construction (Verdejo, 1992).

The ideal learner model should include all the aspects related to behaviour and knowledge that influence the learning process (Wenger, 1987). In spite of this assumption, IRIS generates a pragmatic learner model that, in the sense of Evertsz and Elsom-Cook (Evertsz & Elsom-Cook, 1990), is "accurate enough as is necessary for guiding the tutorial actions of the teaching-learning system".

The learner is modelled in IRIS following a successfully tested classical approach, based on the characteristics identified in the learner model of INTZA, is a passive-descriptive model that represents the acquired knowledge by means of the overlay technique (Carr & Goldstein, 1977). It is a shallow, individual, system inferred student model containing a combination of temporal and permanent information. The model, which is built up during the teaching-learning process, defines the individual learning characteristics and the evolution of each student. It includes new characteristics about the student such as learning preferences, used supported actions, etc.

The IRIS shell also requires that the grain-size of the learner model representation be taken into account in the final tutor. As a consequence, the level of adaptability in tutors generated with IRIS can vary; the more fine and specific is the representation of the learner specified by the instructor (see subsection 5.2), the more adaptation is obtained.

Pedagogic Knowledge Requirements for IRIS

Pedagogic knowledge is the core of any tutoring system. It is responsible for the pedagogic decisions of the system as it builds and executes an Instructional Plan (IP) adapted to the individual learner considering his or her curriculum needs and learning characteristics. Specifying the pedagogic knowledge for planning instructional sessions is one of the most difficult aspects that instructors face in the process of building ITSs using shells. This becomes even more difficult if this knowledge has to be represented by the instructor in a rule-based mode (Major, 1995).

Several planning approaches supported by decisions about different aspects give rise to a range of tutoring styles. In particular, tutors generated by IRIS share an incremental planning schema of minimum commitment. In subsection 4.1 we introduced the general tutor planning activities in the generation of the instructional plan. In the first level a decision is made about the next BLU to teach but not about the whole session plan. The IP (Figure 5) is carried out by successive refinements to the basic actions to execute. Later during the session time, possibly in response to the changing tutorial situations, an adaptive opportunistic plan-based approach is used to allow the plan to be revised and adapted. Each level is refined in the next level by means of rule-based systems which implement instructional decisions. In order to build the planner for a new tutor, it is necessary to know the contents for each level of the IP and the corresponding sets of refinement rules. The level contents are: BLUs and kinds of IOs considered in the particular domain, CPs and IEs identified in the teaching-learning process and IAs chosen for refining the IEs in that particular tutor.

With the aim of facilitating the description of planning knowledge, IRIS provides the instructor with the possibility of specifying planning knowledge in a *non-direct* or *non-explicit* form (a concrete example of specification in a non-direct way is illustrated in subsection 5.3). In this way the instructor is not obliged to explicitly define the different set of refinement rules needed for planning.

USING THE IRIS SHELL

Taking the previous analysis as a starting point, our goal has been to build a set of tools for helping to develop ITSs. We aim to facilitate teachers attempts to build ITSs for those domains in which they are experts. The human instructor must establish the characteristics or requirements of the target tutor, which in turn are used to adapt and to produce the final tutor. Thus this resulting tutor uses a general planning cycle generating, executing and re-planning the Instructional Plan, while maintaining the representation structure and the reasoning schema of INTZA kernel with changes only in the content of the components. This methodology leads to the design of the IRIS Shell (IRakaste-Ikaste Sistema or, in English, *Teaching-Learning System*) which utilizes the planning cycle and basic architecture of INTZA and incorporates the CLAI Model for instruction. Thus, the IRIS shell has the following functionalities:

- assisting with data acquisition related to the nature of the desired tutoring system, the semantics of the teaching domain, and to the final users of the tutoring system;
- determining a tutor architecture derived from the requirements specified;
- providing the final tutoring system by selecting and/or generating the necessary set of rules and objects in terms of the previously specified properties of the tutor, domain and users.

The internal structure of the tutors generated with IRIS closely resembles the INTZA kernel and shares with INTZA certain activities *(i.e. instructional planning, selection of contents and/or instructional activities, and so forth).*

The process of authoring a new tutor using the IRIS Shell consists of two phases (Figure 7). During Phase 1 a skeletal architecture of the resulting tutor is generated on the basis of the requirements specified by the human instructor. These are introduced and used to adapt the generic architecture of INTZA to the tutoring domain. The requirements are grouped around four basic aspects: the characterization of the *tutor*, the description of both the Concrete Level and the Pedagogical Level of the teaching-learning *domain*, the *learner* characterization, and the *interface*

characterization. These aspects, as we will see in next subsections, are not independent from one other. In Phase 2 the contents of each module established in Phase 1 are defined and the final tutor system is produced. Some of these contents are authored by the teacher (*e.g. contents of the domain by means of a set of related BLUs instances, concrete presentation forms, etc.*) and some others are generated automatically by IRIS (*e.g. the rules for refining the instructional plan*).

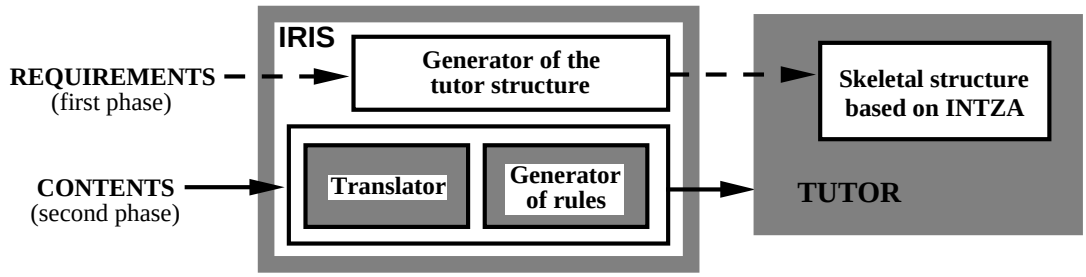

Figure 7. Phases in building a tutor

Next we will discuss the requirements associated with the Domain, the Learner, and the Tutor that must be known in order to fully specify the IRIS Shell; it is responsibility of the human instructor authoring them. A study of the different types of requirements for the Shell together with the internal structure of INTZA leads to a minimum basic architecture for the IRIS Shell (see Section 5.4). On the other hand, interface issues and diagnosis requirements in the IRIS Shell are undertaken in other research work, which is beyond the scope of this paper (Ferrero *et al.*, 1996).

Throughout the remainder of this paper we will use the domain of mathematical symbolic differentiation to illustrate the process of building a specific ITS, called "***Maisu*** tutor", by using the IRIS Shell. In short, as the human instructor or course developers uses the IRIS Shell to construct each component of the new tutor, the following choices must be made:

- With respect to the Domain: specify basic learning units, instructional objectives, presentation and evaluation forms, and relationships between contents.
- With respect to the Learner: specify learner characteristics and learner goals.
- With respect to the Pedagogic: specify instructional method, supported tools and motivation resources.

The generic requirements for the ***Maisu*** tutor are listed below:

Domain (mathematical symbolic differentiation)

- The required Basic Learning Units are *concepts* and *procedures.*

- In general, the skills to be developed in the learner are *knowledge* and *application.*
- The tutor should use the following types of presentation and evaluation forms: *texts*, *examples*, *tests*, and *fill-gaps.*
- The sequencing relationships used by the tutor are *prerequisite* and *next.*

Learner (assuming an adult learner of mathematical symbolic differentiation)

- The adaptability of the system is based on the following learner characteristics: *type, motivation, preferred learning method,* and *preferred session duration*
- The learners´ interventions will refer to the control of the session *(sleep, follow, finish)* and the session development *(ask for explanation, solve exercise, agree, disagree, repeat explanation, repeat exercise, ask for exercise)*

Pedagogy (assuming a coaching style of instruction)

- The instructional method selected is *guided learning.*
- The supported tools selected for the tutor are *summary window* and *underlining tool.*
- The motivation resources include *beep, attention/alert message, provide congratulation* and *give encouragement.*

Keeping in mind these particular design decisions for the Maisu Tutor, we will show incrementally the hierarchy of objects generated by the IRIS Shell for such a tutor. It is generated by growing an initial basic hierarchy representing the main components of the *Maisu* tutor (Figure 8).

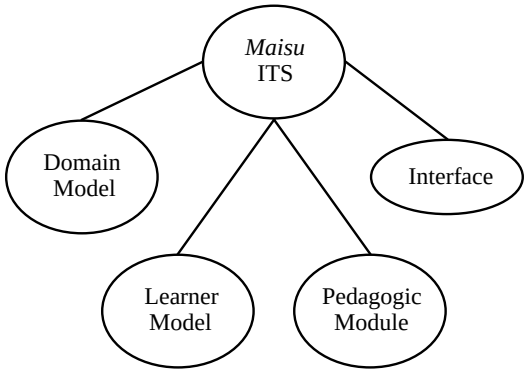

Figure 8. Structure of the main components of the *Maisu* ITS

Building the *Maisu* domain: basic symbolic differentiation

Four different types of information are included to describe the domain: *meta-information for characterizing the domain in a general way, basic learning units, instructional objectives* and *requisite relationships.* The former has been included with the aim of describing the main goal of the tutor and the interest of the domain.

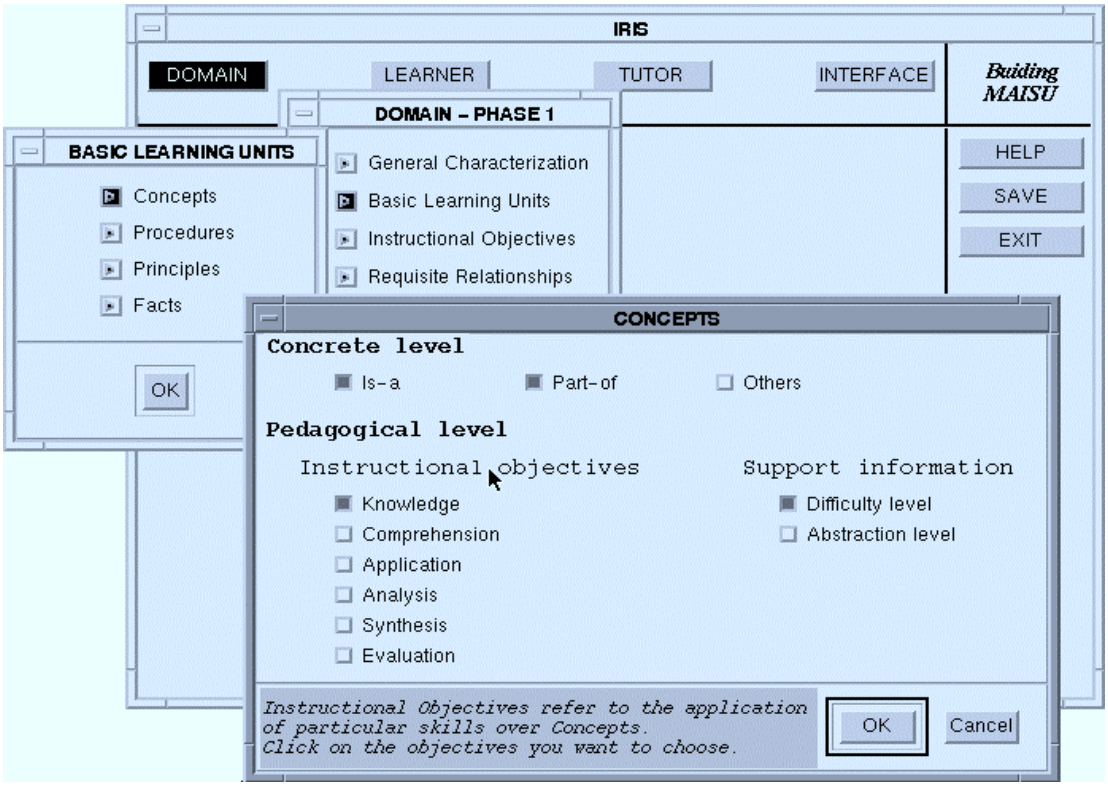

Figure 9. Specification of the BLU Concept in *Maisu*

- Meta-information for characterising the domain in a general way. Two attributes have been included: *General Goal* and *Relevance*. The human instructor will select one or both; however it could be possible to define new informative perspectives. The selection/inclusion of these slots ensures that the final tutor include instructional actions for informing the learner about the goals she can get working with that domain.
- Basic Learning Units. IRIS allows the instructor to define any instructional domain in terms of the already referenced four BLUs: *concepts, procedures, principles,* and *facts*. The differentiation domain can be represented using mainly concepts (ex. *"Derivative concept"* and *"Derivative function")* and procedures (ex. *"Sine derivative rule", "Add derivative rule").* Figure 9 shows the specification of the concept BLU in the *Maisu* domain; the selected characteristics are those shadowed.

Each BLU considered in the domain must be described at the Concrete and Pedagogical levels. At the Concrete Level we record exclusively the curricula requirements, *i.e.* the declarative descriptions of the subject matter to be conveyed to the learner. The Pedagogical Level defines a meta-level description of the domain and includes descriptors such as *IOs* and *Support Information*. For each BLU the set of skills to be worked are chosen; the particular development of each skill is elaborated and further specified later, independent of the associated BLU. At the Pedagogical Level, we also specify the *Support Information* requisite indicating the *Difficulty Level* attribute.

- Instructional Objectives. The default IOs used in the shell are those of Bloom´s taxonomy (1956). In *Maisu* the IOs Knowledge and Application have been selected as the main abilities to develop for the basic symbolic differentiation domain (Figure 10). Both kinds of objectives need a set of *presentation forms* (techniques which can be used to introduce the domain concepts to the learner –*text* in the example), *evaluation form* (techniques for assessing domain concepts –*test* in the example), *difficulty level* (a refinement of the difficulty level property associated to the corresponding BLU) and *estimated-time* for acquiring that IO.

- Requisite Relationships. The defaults *requisite relationships* are *Prerequisite*, *Corequisite*, *Postrequisite* and *Next*. They are used for organizing, selecting and sequencing the BLUs (Reigeluth *et al.*, 1978). In *Maisu* we have identified the *prerequisite* and *next* relationship.

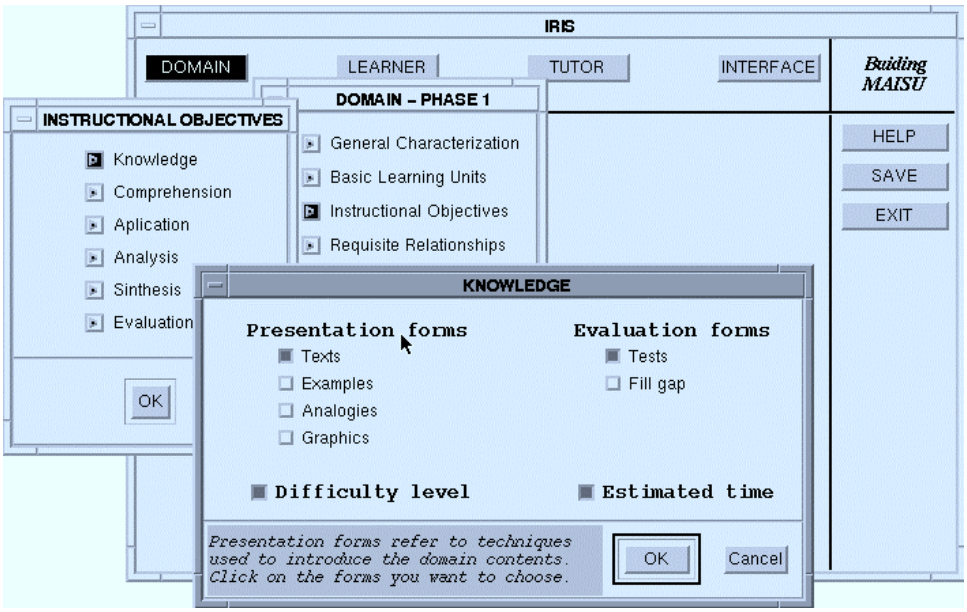

Figure 10. Specification of the kind of Instructional Objective: *Knowledge*

Next the resulting class of concept BLU obtained from the specification phase (Phase 1) is showed in a frame-like form (Table 1). The form will be filled in Phase 2 as the domain expert creates each domain concept. The Procedure BLU is described in a similar way. In the case of Figure10, the prerequisite relationship has been selected and the Knowledge and Application IOs have been chosen.

CONCEPT	**PROCEDURE**
identification:	identification:
is-a:	is-a:
part-of:	part-of:
difficulty level:	steps:
next:	difficulty-level:
prerequisite:	next:
IOs: KNOWLEDGE	prerequisite:
texts-knowledge:	IOs: KNOWLEDGE, APPLICATION
tests-knowledge:	texts-knowledge:
difficulty-level-knowledge:	tests-knowledge:
estimated-time-knowledge:	difficulty-level-knowledge:
	estimated-time-knowledge:
	texts-application:
	examples-application:
	tests-application:
	fill-in-gaps-application:
	difficulty-level-application:
	estimated-time-application:

Table 1. Slots for the Concept and Procedure Frames of a BLU

For example, consider defining a new BLU named *Derivative-function* using IRIS (Phase 2). As shown in Figure 11, first the instructor chooses the type of the new BLU, *concept* in our example. After doing that IRIS prompts the instructor to fill in values for each of the concept attributes related to that concept (those represented above in Table 1). In figure 11 the attribute *identification* of a Concept is being filled with the value "derivative-function". When the instructor is finished with specifying each BLU, IRIS generates all the internal associated instances. Each BLU is created in this way.

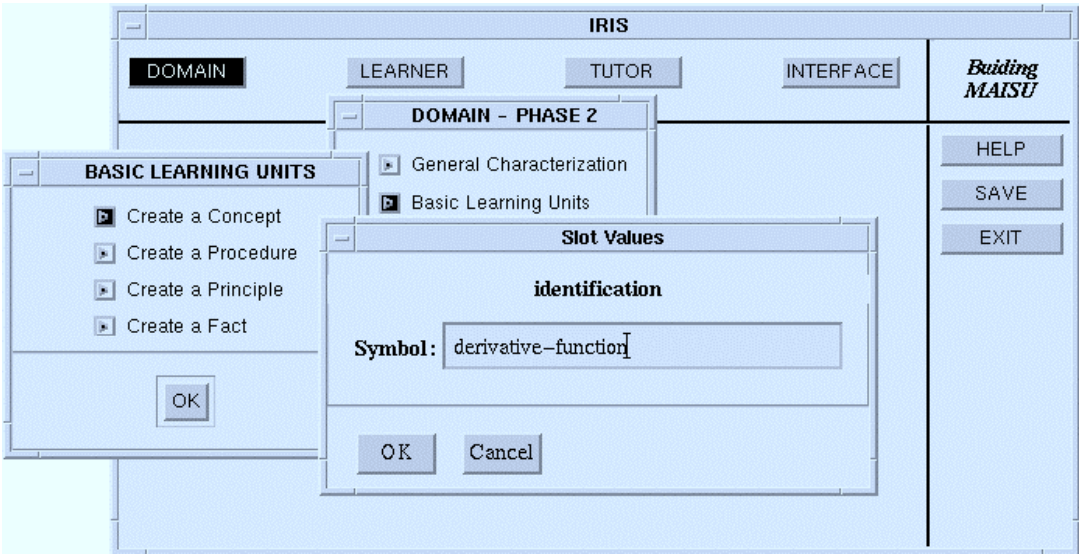

Figure 11. Creating a new Concept

Figure 12 shows the domain objects generated by IRIS for *Maisu* tutor represented in a *part-of* hierarchy. This organizes the Domain Model in three sections: *General Characterization*, *Descriptive Structure* and *Descriptive Taxonomy* of Instructional Objectives. The *Descriptive Structure* groups the BLUs, IOs and the instructional resources used in the representation of the differentiation domain.

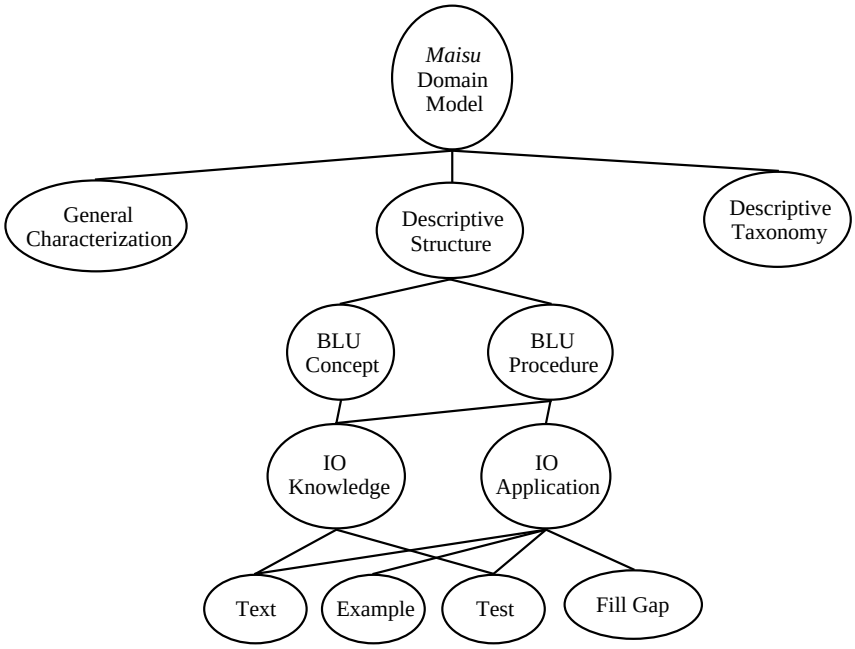

Figure 12. *Part-of* hierarchy of the domain objects generated in the first phase by IRIS for *Maisu* tutor

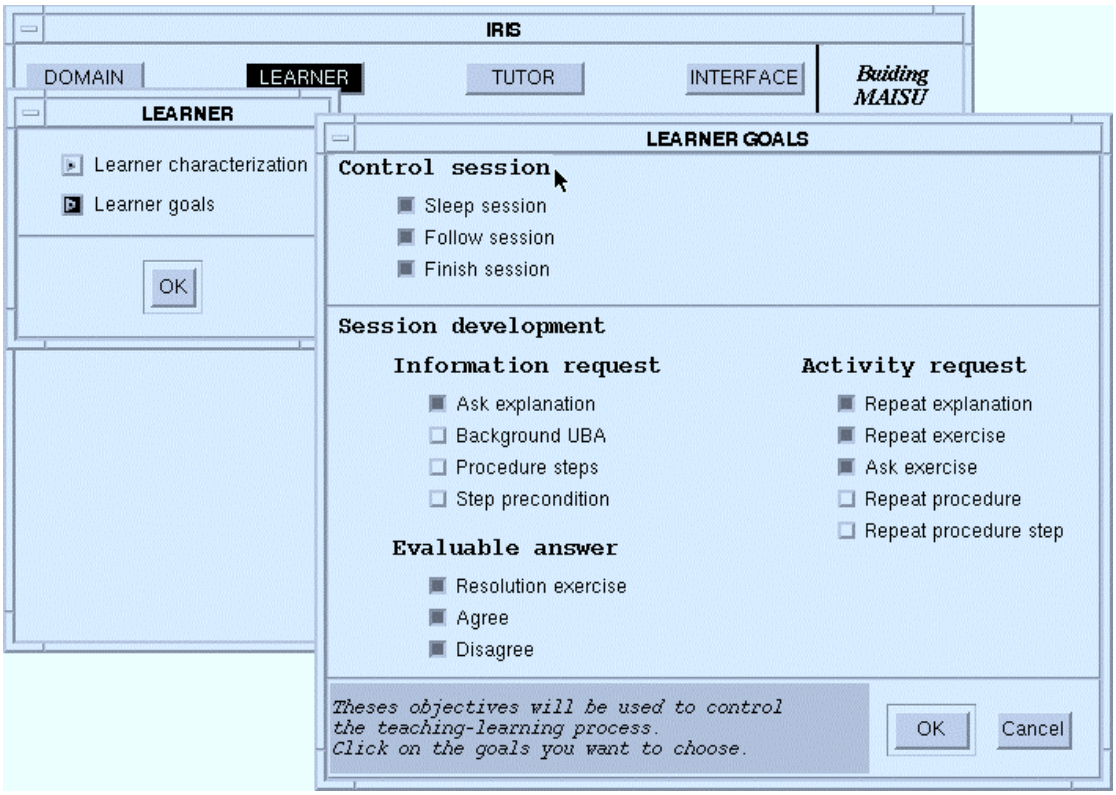

Figure 13. Specification of the Learner Goals

Learner modelling for *Maisu*

The characterization of the general kind of the learner for *Maisu* tutor, *i.e.*, the prototypical aspects *(curriculum*, *learner profile* and *learner goals)* are used not only to generate the Learner Model but also to determine the kind of instruction that learners are going to receive. That is, there are choices of using one or another group of Instructional Actions and the possibility of recognizing different learner goals.

Learner goals refer to the goals that the tutor will be able to identify after interpreting the student interventions. Goals have been split into two groups (Figure 13) according to their functionality: *control of the session* and *development of the session.* They collect the set of learner goals identified in the literature by several systems (Barnard & Sandberg, 1994) (Breuker *et al.*, 1987) (Diaz de Ilarraza, 1990) (Fernández, 89) (Vadillo *et al.*, 1994).

Figures 13 and 14 show the properties used in IRIS for characterizing the kind of learner and her goals in *Maisu* tutor. The learner characteristics are separated into two groups: *curriculum* and *learner profile* (Figure 14).The former refers to the physical and professional aspects of the learner, the latter refer to characteristics that influence the learning process; so, they are the basis for tutorial decisions about Cognitive Processes, Instructional Actions and communications ways between the tutor and the learner. The learner goals are selected directly by the instructor from a predefined set.

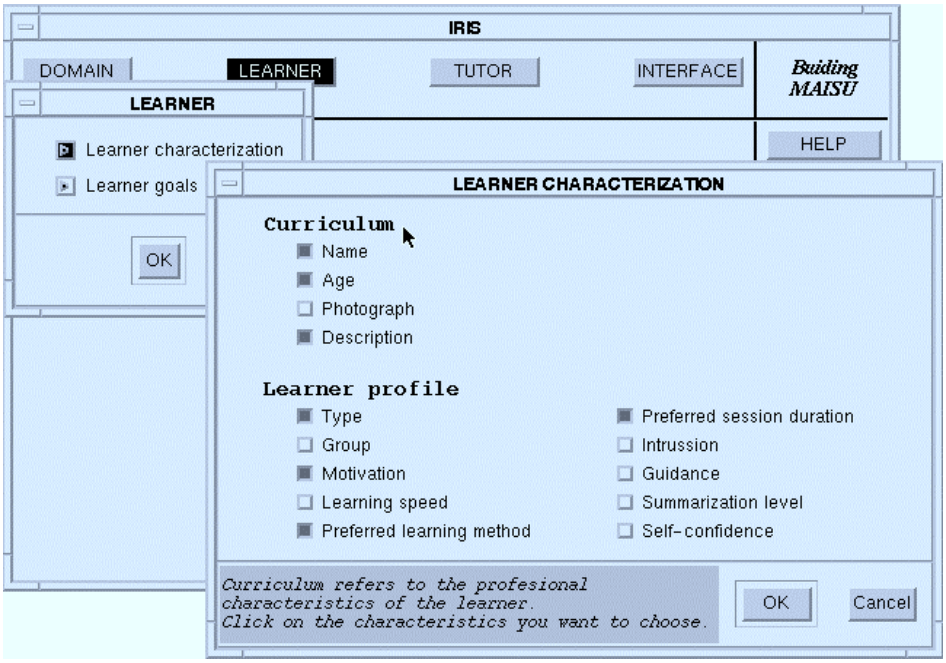

Figure 14. Learner Characterization for *Maisu*

The Learner Model finally generated by IRIS for *Maisu* is organized in two parts *Permanent Model* and *Dynamic Model* (Figure 15). It collects information not only from the requirements related to the learner but also from the domain requisites in order to reflect the knowledge of the learner and how it was acquired. The Permanent Model is updated at the end of each session and is divided into three sections:

- *Learner Characterization*, which consists of *Curriculum-Vitae* and *Learner Profile*;
- *Learner Knowledge*, which records the domain contents acquired by the learner during the learning process including the achieved Instructional Objectives, the errors made and the didactic materials used; and finally
- *Learner History*, which consists of information about the evaluation process of the *Last Session* and the collection of the most important events of the whole *Course*.

The Dynamic Model exists just during the current session and is used for updating the permanent model. It is divided into two components: *Session Characteristics*, *i.e.* events occurring up to the current moment in the session, and *Learner Performance*, which contains interaction information about the learner such as *texts presented, requests made*, and so forth.

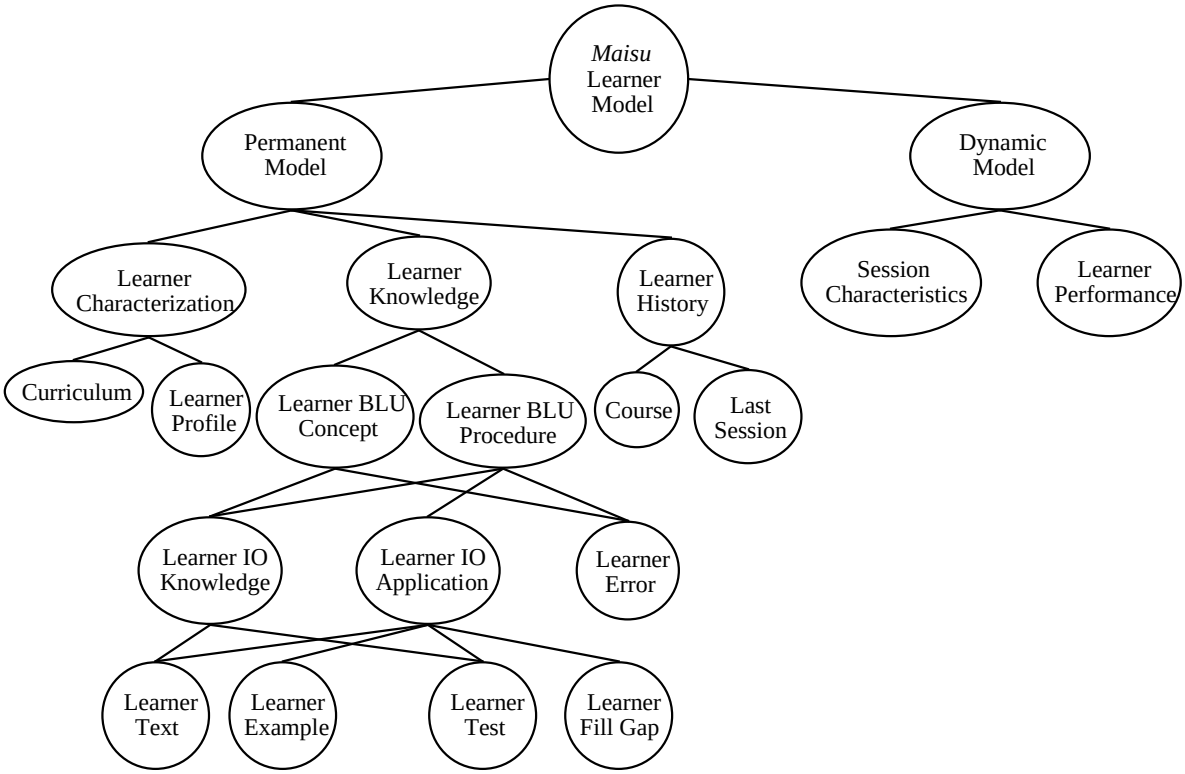

Figure 15. Part-of hierarchy corresponding to the Learner Model of *Maisu* (phase 1) generated by IRIS

Pedagogic Knowledge in Maisu

At this moment, after specifying the domain and the learner characteristics, just a few aspects remain to be fixed. They are the instructional method that the target tutor will use, the supported tools, and the motivation resources.

If BLU has prerequisite/s then learn first the prerequisite/s BLU/s

If
BLU-IPL.current-BLU = $BLU
not(reached-blu-p ($BLU))
not(empty-p $BLU.prerequisite)
not(reached-BLU-list $BLU-prerequisite)
Then
BLU-IPL.current-BLU := first($BLU.prerequisite)
BLU-IPL.BLU-LIFO:= append(rest($BLU-prerequisite), $BLU,
BLU-IPL.BLU-LIFO)

Figure 16. Rule corresponding to the *prerequisite* property

Tutors generated by IRIS share the same planning schema defined in INTZA which is customized for each target tutor in terms of all specified properties. As IRIS holds default rules and objects associated to several defining properties, instructor selection is used to produce the final new planner.

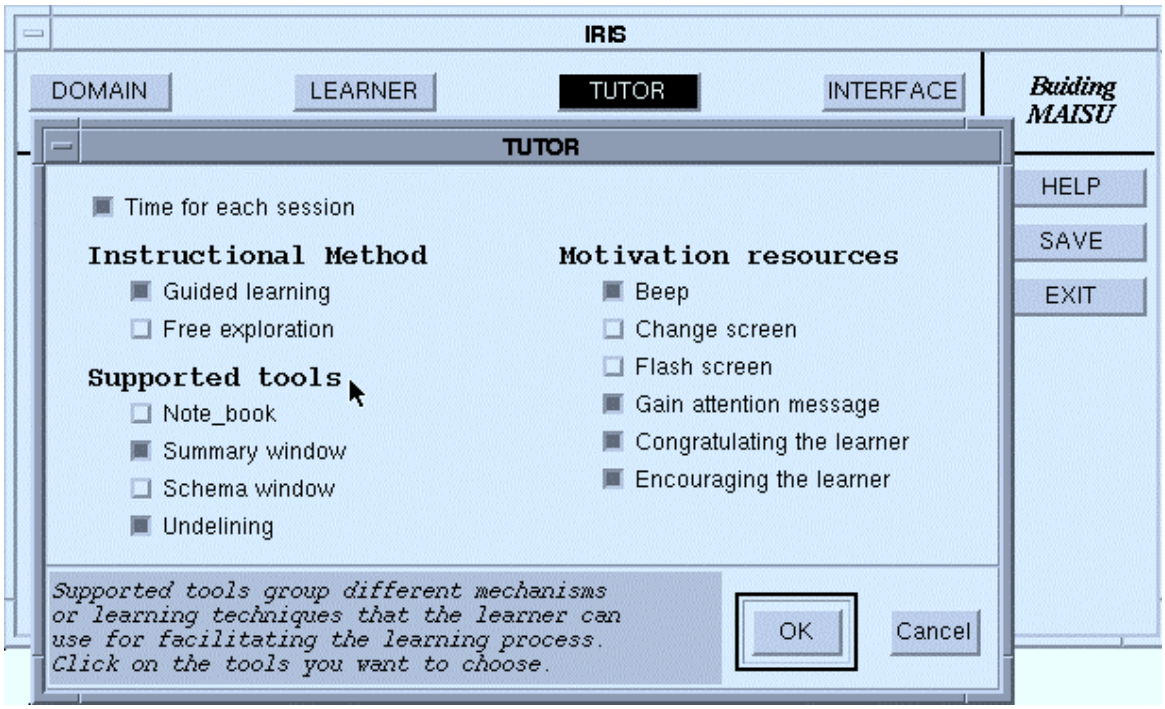

Figure 17. Specification of the tutor requirements

So, for example, if the instructor chooses the property *prerequisite* as a relation requisite, the rule corresponding to prerequisite (Figure 16) is included in the corresponding refinement rule set to be taken into account for selection of BLUs in the new tutor *Maisu.* Thus, the set of rules corresponding to each level of the IP is customized for each tutor built using IRIS by means of the properties chosen by the instructor.

In the specification phase (Phase 1) the human instructor chooses the most suitable instructional actions (e.g., *beep, changing/flashing the screen, congratulating the learner*, etc.) according to the tutor and domain characteristics. In the first IRIS prototype *impact* (audio-visual) *resources* and *messages* have been identified. Finally, the *Supported Tools* indicate supported action tools that the student can use. Shadowed attributes (in Figure 17) are those selected for the symbolic differentiation domain in *Maisu*.

Figure 18 represents a snapshot of the objects of the pedagogic component generated by IRIS in *Maisu*. The module is composed of six main aspects: the *Instructional Method* which in the particular *guided learning* method includes the instructional plan, *Learner Goals, Instructional Strategies, Instructional Events* and *Instructional Actions*. Finally the *Supported Learning Actions* are also considered.

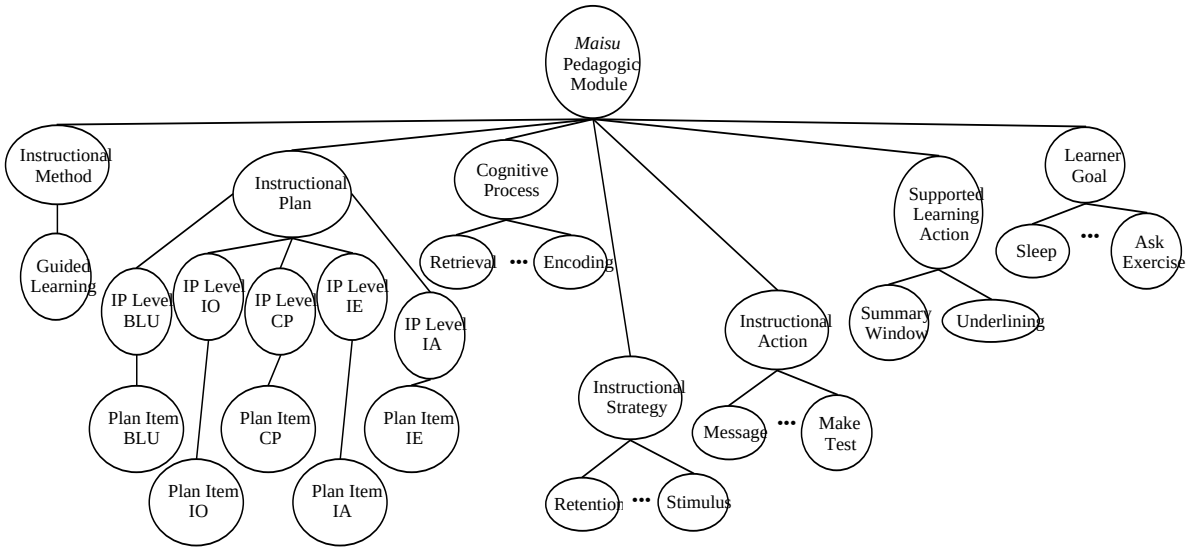

Figure 18. Snapshot of the Part-of hierarchy corresponding to the Pedagogic Module of *Maisu*

Significance of the IRIS Shell for Building ITSs

IRIS is a shell developed with the aim of facilitating the efficient production of ITSs through *software reuse*. The main reason for taking reusability issues into account in the ITS field stems from the fact that building ITSs necessitates large and costly development environments and significant amounts of time. The tools in these development environments tend to require sizeable computing resources and are seldom suited for developing both experimental research prototypes and practical teaching-learning systems. Taking into account that prototype ITSs are built incrementally through successive enhancements and refinements, the time and cost of development is reduced if existing software and knowledge is reused.

Concerning the construction of ITSs, IRIS considers knowledge reusability at different levels (Arruarte *et al.*, 1996b; Arruarte *et al.*, 1997), *knowledge bases/data structures, libraries of teaching resources, libraries of teaching strategies, reusability of modules* and *reusability of architectures.* The tutors built using the IRIS Shell, *e.g. Maisu,* reflects all these levels of knowledge reusability. *Maisu* reuses knowledge and rules bases, uses some of the teaching resources and strategies identified in previously developed tutors, mainly in INTZA and TUTOR, reuses the modules of the architecture of INTZA and also, reuses the whole architecture of INTZA. The tutors built using the IRIS Shell focus on reusability at the highest level, *i.e. reusability of architectures*. Architecture reusability is achieved when capturing the whole architecture of a previously developed system and using it for organizing new systems

(Krueger, 1992). In this sense, IRIS facilitates the development of new tutors by reusing and customizing the whole architecture of INTZA. The extra effort needed to adapt and integrate modular components with functionalities of parts of the original tutor are eliminated.

Furthermore, the IRIS Shell not only reflects the reusability but also promotes it. IRIS is a tool for developing applications generating the code automatically. It is not necessary for the instructor to specify any programming code, IRIS builds different teaching-learning systems following only the requirements specified by human instructors by using different graphical windows. The tutors built using IRIS, *e.g. Maisu* tutor, are modular tutors developed under a modular philosophy. The modular design of the components is the basis for generating reusable ITS software.

Building any system, including any ITS, from scratch requires three main phases: *analysis*, *design* and *implementation.* The utilization of a high level tool like IRIS greatly facilitates the system development process by virtually eliminating the implementation phase and significantly reducing the design phase.

When using IRIS to develop a tutor, the analysis phase involves high-level decisions about each main component of the tutor *(i.e.* domain module, learner model and pedagogic component). The domain component analysis is supported by textbooks in the domain, previous teaching experience of the instructor, and also user help provided by the IRIS Shell. The outcome of the domain analysis will result in the necessary Basic Learning Units (BLUs) for representing the domain, the Instructional Objectives, and the selection and/or sequencing relationships between BLUs.

Analysis involving the desired style of tutoring determines both the learner model and the pedagogic module. The same basic type of learner model is maintained in all tutors developed using the IRIS Shell, but the analysis phase will determine the grain size of the learner model representation, *i.e.* representations range from coarse to fine representation of acquired knowledge. This analysis leads to specification of the learner's goals that the tutor must be able to identify and interpret. Concerning the pedagogic module, the instructor must identify the instructional method for the new tutor, the supported tools that the tutor will be able to offer to the learner, and the motivation resources that the tutor will use.

The design of a tutor constructed with the IRIS Shell is implicitly conducted by the instructor through selecting the different requisites and characteristics offered by IRIS for each component. The implementation phase vanishes as IRIS takes charge of generating the new tutor's code automatically.

Let us suppose that one wants to develop a new tutor for, say, symbolic integration. This domain is very similar to that of differentiation in *Maisu* tutor, so that one can presume many similarities with *Maisu.* In fact, one

can generate similar learner and pedagogic modules but a different domain component would be required. As pointed out in Section 5, the process of building a new tutor using IRIS consists of two phases: *requirements specification* and *content specification.* The specification of requirements for the new domain can be represented using the same types of Basic Learning Units (concepts, procedures), the same Instructional Objectives (knowledge, application) and the same requisite relationships (prerequisite, next). The difference arises in the content specification phase (second phase) when we have to define integration rules instead of differentiation rules.

In conclusion, a reflection on the process of building a new tutor based on IRIS shows that the analysis phase is always required (to plan the structure of the tutor), the design phase is considerably simplified (to select parameters within the IRIS Shell and to design the domain knowledge rules and interface), and the implementation phase is mostly automated (with the exception of the interface).

A DEEPER LOOK AT THE ARCHITECTURE OF THE IRIS SHELL

As mentioned in the previous section, the IRIS Shell works in two phases which run sequentially. In the first phase, the tutor structure is generated on the basis of requirements, and in the second phase the defined modules are filled with contents. The basic architecture, therefore, includes the following four main components (Figure 19): *Generator of the Tutor Structure*, *Object Generator*, *Rules Generator*, and *Interface.*

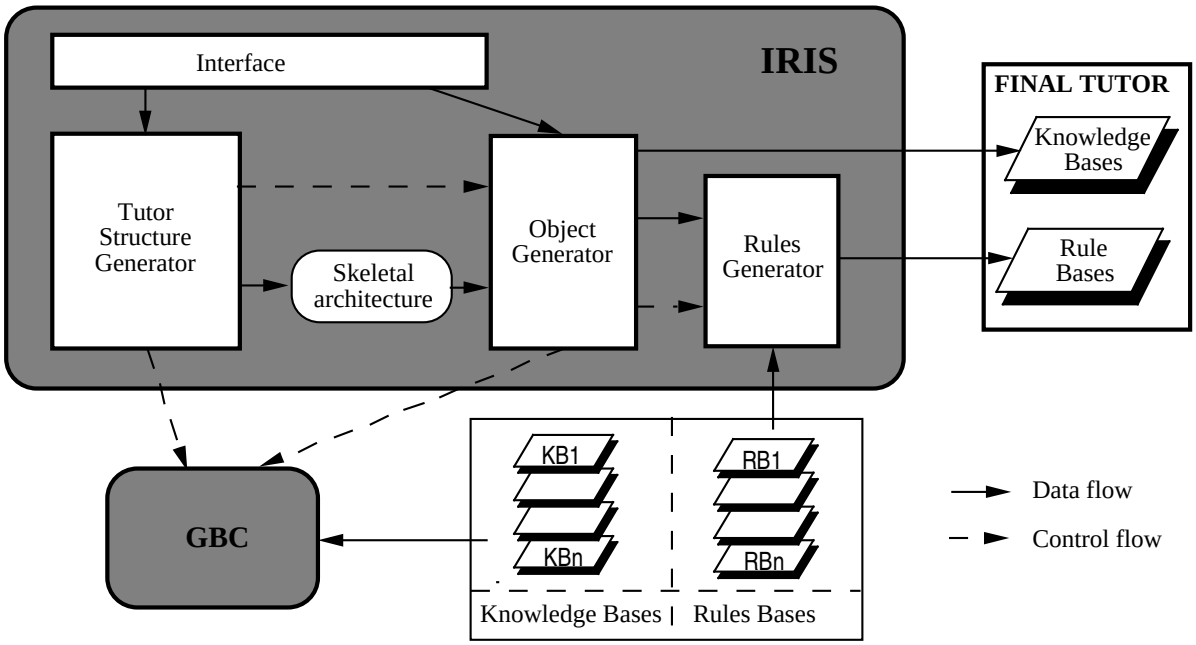

Figure 19. Architecture of the IRIS Shell

The **Generator of the Tutor Structure** has a twofold function: to customize the generic architecture of INTZA and to check the creation preconditions and the completeness of the contents.

In order to customize the generic architecture of INTZA, taking into account the particularities of the domain and the teaching method specified by the instructor, a low level knowledge acquisition tool, GBC (see below) is used together with the various knowledge bases of INTZA. A skeletal architecture is generated by selecting the main components that will comprise the final tutor.

Furthermore, it is necessary to check the creation preconditions and the completeness of the contents in each phase in order to get a viable and reliable tutor. It will not be possible, for example, to introduce particular contents of the teaching-learning domain before characterizing the domain.

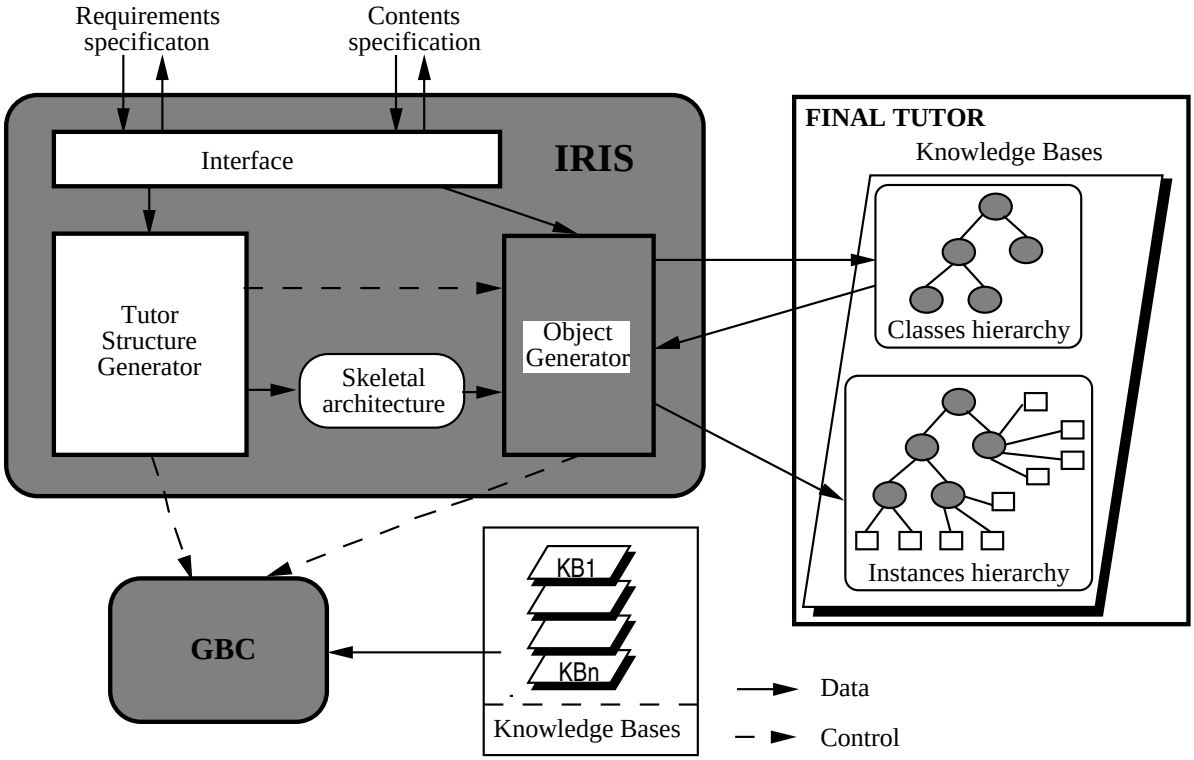

Figure 20. Partial schema related to the *Object Generator*

Once the skeletal architecture of the tutor has been produced, the **Object Generator** completes and saves the declarative knowledge associated to each component of the new tutor (its classes and instances). It works in two different phases (see Figure 20):

1. The first phase, namely the specification of requirements, gives rise to the needed classes of objects corresponding to the components of the new tutor.

2. The second phase, namely the specification of contents, generates the corresponding attribute values to create pertinent instances from the previously defined classes.

Both phases are supported by GBC (Generacion de Bases de Conocimiento, in English, *Knowledge Bases Acquisition)*. GBC (Elorriaga *et al.*, 1995) is a basic tool for generating Knowledge Bases (KBs). Its goal is to facilitate the building of Knowledge Based Systems prototypes. GBC acquires the abstract class specifications of a particular domain as well as its instances. The tool allows us to create new KBs and modify and adapt them to their particular needs using already existing KBs. GBC represents each domain using an updatable internal format that finally is translated to CLIPS code.

The **Rules Generator** works after the skeletal architecture of the tutor and the first phase of the object generator have been completed. Its function is to generate and/or select, the necessary sets of pedagogic decision rules in order to get operative components for the tutor (as we pointed out in subsection 4.1 tutors generated by IRIS are based on a rule-based paradigm). The Rules Generator module is composed of a *Selector* and a *Generator*. If we select default values in the first phase, as it was shown at the beginning of Section 5, the *Selector* sub-module will be the activated. If, on the other hand, we introduce new slot values, the *Generator* will be activated instead.

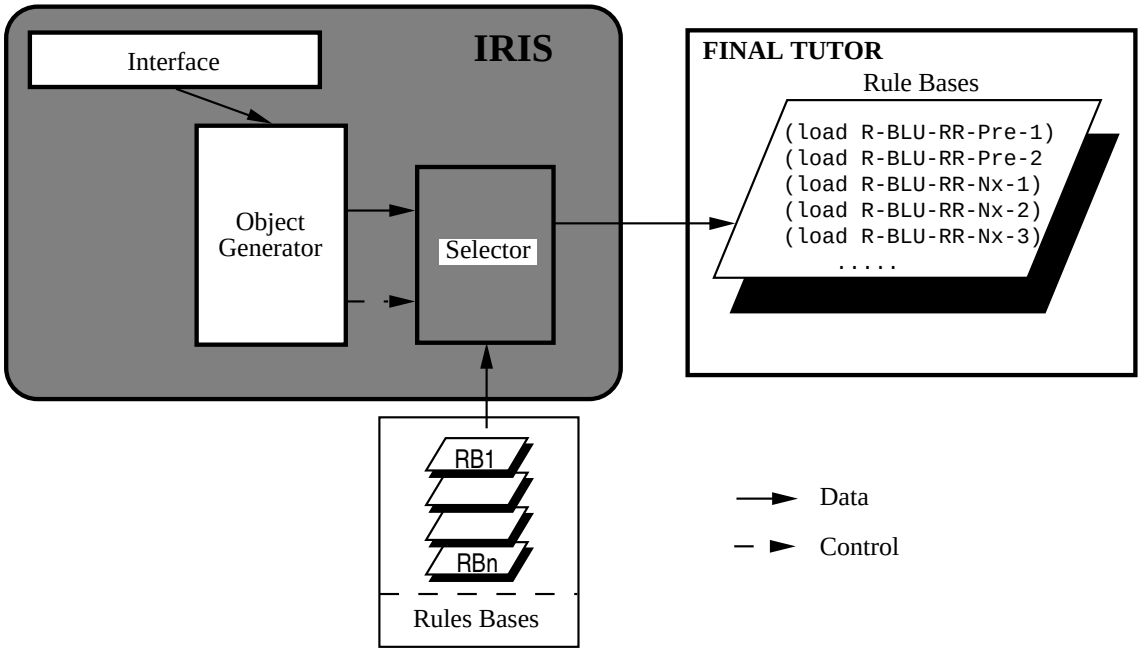

Figure 21. Partial schema related to the *Selector* module

The current IRIS Shell prototype includes only the *Selector* sub-component and does not contain the Generator yet. The *Selector* customizes the whole set of rules existing in the initial rules hierarchy, taking into account the requisites or requirements specified by the human instructor (Figure 21). The different requisites have a direct relationship with the rules identified in the initial hierarchy. For example, if we select only two IOs (*Knowledge* and *Application*) from the six IOs that the tool offers by default, the sets of IO's rules (in the *Pedagogic Module)* will be composed of just rules related to those two IOs.
An initial hierarchy of rules has been identified which groups the different sets of rules according to its final goal in the new tutor. Figure 22 shows a snapshot of rules identified in the initial hierarchy as rules corresponding to the *requisite-relationships* in the first level of the instructional plan, *i.e.* the BLU selection phase.

If the instructor chooses *prerequisite* and *next* as a requisite relationships among BLUs, the set of rules that will be activated in *Maisu* will be just the rules identified in these two objects, specifically: R-BLU-RR-Pre1, R-BLU-RR-Pre2, R-BLU-RR-Nx1, R-BLU-RR-Nx2 and R-BLU-RR-Nx3 (Figure 20).

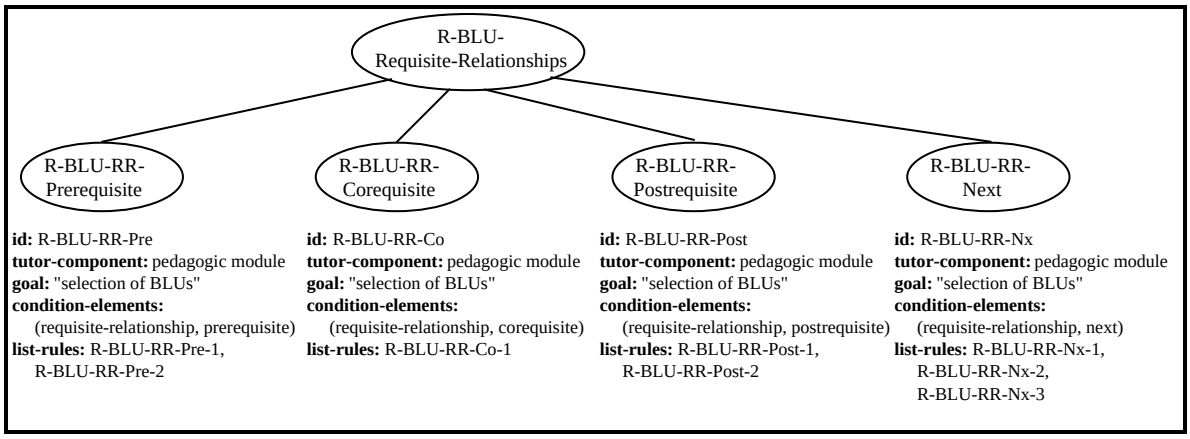

Figure 22. Stage of the requisite-relationships rules in the initial hierarchy

Finally, the **Interface** or communication component maintains the interaction process with the human instructor by means of various graphical windows, some of which were shown earlier in the paper where we illustrated the construction of the *Maisu* tutor. In our initial IRIS prototype the interface is was implemented using MOTIF in C.

CONCLUSION

The goal we have pursued in this paper is to describe the design and implementation of a shell to aid the construction of instructional

environments integrating pedagogical principles with a general ITS architecture. We assume that a human instructor establishes the requirements of the final tutor and these requirements be used to adapt automatically a previously built generic architecture producing a new tutor. INTZA is an already developed and tested ITS, valid for conceptual and procedural domains, that provides a kernel for the IRIS Shell. Thus, any new tutor built with IRIS is provided with a flexible planner and shares many characteristics of INTZA including its types of domains and how it generates, executes and re-plans its Instructional Plan. It will retain the representation structure and the reasoning scheme with changes only in the content of its components.

On the other hand, with the aim of providing a sound basis for computer-based training, we have previously defined a pragmatic cognitive theory of instruction, the CLAI Model, which has been integrated into the IRIS Shell. A deep analysis of both the architecture and the theory has allowed us to extract the necessary pedagogic requirements that affect the different components of a tutor. The information required for building the domain knowledge has been organized in four groups: meta-information for characterizing the domain in a general way, basic learning units, instructional objectives and requisite relationships. For modelling the learner, IRIS requires a curriculum, a learner profile, and learner goals. Finally, the necessary pedagogical information includes the specification of the instructional method that the target tutor will use, the supported tools, and the motivation resources together with the instructional plan refinement rules. All these requirements (which must be specified by human instructors) have directly influenced the design and the basic architecture of the IRIS Shell and allow us to integrate those pedagogic principles with a general ITS architecture.

At this time we have an implemented prototype containing most of the components identified in the architecture, running on a SUN workstation. The *Object Generator* and *Selector* were implemented in CLIPS and tested on an example tutor for *symbolic differentiation.* The *Interface* component of the IRIS Shell was implemented in MOTIF and C. After finishing the whole implementation we plan, as an immediate task, to test and evaluate the global shell with human instructional designers from an educational and a usability perspective. We are aware of the necessity of developing on-line help and adequate methodological guidelines to facilitate the use of IRIS.

Two important aspects remain to be considered: those related to the diagnosis and treatment of errors and to the generation of interfaces. They constitute the current and future research lines of our group. The diagnostic process is being considered currently (Ferrero *et al.,* 1996; Ferrero *et al.,* 1997), and in that work we face the general problem of developing an autonomous general purpose tool, integrable in IRIS and

adaptable to different domains. It would offer to instructors the possibility of defining a customized minimum diagnosis process. The more challenging problem of automatic interface generation will be tackled in future research.

Acknowledgements

This work is partly supported by the Economy Department of the Gipuzkoa Council (Gipuzkoako Foru Aldundia), the Department of Education, Universities and Research of the Basque Government (Eusko Jaurlaritza), and the University of the Basque Country (Euskal Herriko Unibertsitatea).

References

Arruarte, A. and Fernández, I. (1995). A two-phased development shell for Learning Environments. In A design proposal. Tinsley and van Weert (Eds.), *World Conference Computer in Education VI*: Chapman & Hall, 53-65.

Arruarte, A., Fernández, I. and Greer, J. (1996a). The CLAI Model. A Cognitive Theory to Guide ITS Development. *Journal of Artificial Intelligence in Education*, 7(3/4), 277-313.

Arruarte, A., Elorriaga, J.A. and Fernández-Castro, I. (1996b). Knowledge Reusability: some Experiences in Intelligent Tutoring Systems, In *Proceedings of the Workshop Architectures and Methods for Designing Cost-Effective and Reusable ITSs*, Montreal.

Arruarte, A., Elorriaga, J.A. and Fernández-Castro, I. (1997). Reutilización del Conocimiento: Experiencias en los Sistemas Tutores Inteligentes. *Journal of Informática y Automática (in press).*

Barnard, Y.F. and Sandberg, J.A.C. (1994). The Learner in the Centre: towards a methodology for open learner environments, Doctoral Dissertation, University of Amsterdam, 1994-6.

Bloom, B.S., Engelhart, M.D., Murst, E.J., Hill, W.H. and Drathwohl,D.R. (1956). *Taxonomy of Educational Objectives: The Cognitive Domain*, Longmans.

Breuker, J., Winkels, R. and Sandberg, J. (1987). A Shell for Intelligent Help Systems, *Proceedings of the 10th International Conference on Artificial Intelligence*, 167-173,

Carr, B. and Goldstein, I. (1977). Overlays: A Theory of Modelling for Computer-Aided Instruction. *International Journal of Man-Machine Studies*, 5, 215-236.

Diaz de Ilarraza, A. (1990). Gestión de diálogos en Lenguaje Natural para un Sistema de Enseñanza Inteligente, Doctoral Dissertation, University of the Basque Country UPV/EHU, Donostia.

Elorriaga, J.A., Ferrero, B. and Fernández-Castro, I. (1995). GBC: una herramienta para la construcción de bases de conocimiento. In *Actas de CAEPIA-95*, Alicante, 367-376.

Evertsz, R. and Elsom-Cook, M. (1990). Generating Critical Problems in Student Modelling. In Elsom-Cook, M. (Ed.), *Guided Discovery Tutoring. A Framework for ICAI Research*, Paul Chapman, 216-246.

Fernández, I. (1989). Estrategias de Enseñanza en un Sistema Inteligente de Enseñanza Asistida por Ordenador, Doctoral Dissertation, University of the Basque Country UPV/EHU, Donostia.

Fernández, I., Díaz-Ilarraza, A. and Verdejo, F. (1993). Architectural and Planning Issues in Intelligent Tutoring Systems. *Journal of Artificial Intelligence in Education*, 4(4), 357-395.

Ferrero, B., Fernández-Castro, I. and Urretavizcaya, M. (1996). Un Sistema de Diagnóstico para Ayuda al Aprendizaje. Experiencias en el Dominio de las Derivadas. Technical Report UPV/EHU/LSI/TR 14-96, University of the Basque Country UPV/EHU, Donostia.

Ferrero, B., Fernández-Castro, I., Urretavizcaya, M. (1997). DETECTive: a generic diagnostic tool to support learning. Some experiences in the symbolic differentiation domain. In *Proceedings of Computer Aided Engineering Education CAEE'97 (in press).*

Folch, M. (1.993). Una Metodología del Ordenador como Instrumento de Apoyo al Curriculum. *El Grupo ORIXE. Comunicación y Pedagogía*, Marzo, 29-34.

Gagné, R.M., Briggs, L.J. and Wager, W.W. (1988). *Principles of Instructional Design*. Third Edition, Holt, Rinehart and Winston: Orlando.

Gavignet, E. (1994). Instructional expertise in ECSAI. Desalles, J. (Ed.), In *Proceedings of the International Conference on Computer Aided Learning and Instruction in Science and Engineering CALISCE'94*, TELECOM, 249-257.

Gutiérrez, J. (1994). INTZA: un Sistema Tutor Inteligente para Entrenamiento en Entornos Industriales. Phd. Thesis. Euskal Herriko Unibertsitatea, UPV/EHU, Donostia.

Hernández, P. and García, L.A. (1.991). Psicología y Enseñanza del Estudio. In *Teorías y Técnicas para potenciar las Habilidades Intelectuales*, Pirámide: Madrid.

Holt, P., Dubs, D., Jones, M. and Greer, J. (1994). The State of Student Modelling. In. J. Greer and G. McCalla (Eds.), *Student Models: The Key to Individualized Educational Systems*, Springer Verlag, 3-35.

Ikeda, M. and Mizoguchi, R. (1994). FITS: a Framework for ITS - A Computational Model of Tutoring. *Journal of Artificial Intelligence in Education*, 5(3), 319-348.

Khuwaja, R., Desmarais, M., Cheng, R. (1996). Intelligent Guide: Combining User Knowledge Assessment with Pedagogical Guidance. In

Frasson, C., Gauthier, G., Lesgold, A. (Eds.), *Proceedings of the Third International Conference Intelligent Tutoring Systems ITS'96*, Springer-Verlag, 225-233.

Kok, A.J. (1991). A Review and Synthesis of User Modelling in Intelligent Systems. *The Knowledge Engineering Review*, 6(1), 21-47.

Krueger, C.W. (1992). Software Reuse. *ACM Computing Survey*, 24(2), 131-183.

Macmillan, S.A., Emme, D. and Berkowitz, M. (1988). Instructional Planners: Lessons Learned. In Psotka, J., Dan Massey, L. & Mutter, S.A. (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum, 369-402.

Major, N. and Reichgelt, H. (1991). Using COCA to build an intelligent tutoring system in simple algebra. *Intelligent Tutoring Media*, 2(3/4), 159-169.

Major, N. (1995). REEDEM: Creating Reusable Intelligent Courseware. J. Greer (Ed), In *Proceedings of Seventh World Conference on Artificial Intelligence in Education AI-ED'95*, AACE, 75-82.

Merrill, M.D. (1983). Component Display Theory. In Reigeluth, C.M. (Eds.), *Instructional-Design Theories and Models: an overview of their current status*, Lawrence Erlbaum Associated, 279-333.

Merrill, M. and the ID_2 Research Group (1996). Instructional Transaction Theory: Instructional Design Based on Knowledge Objects. *Educational Technology*, May-June, 30-37.

Murray, T. (1996). Having It All, Maybe: Design Tradeoffs in ITS Authoring Tools. Frasson, C., Gauthier, G., Lesgold, A. (Eds.), In *Proceedings of the Third International Conference Intelligent Tutoring Systems ITS'96*, Springer-Verlag, 93-101.

Murray, T. (1998). Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. *Journal of the Learning Sciences*, 7(1), 5-64.

Nkambou, R., Gauthier, G., Frasson, C. (1996). CREAM-Tools: An Authoring Environment for Curriculum and Course Building in an Intelligent Tutoring System. In Díaz de Ilarraza Sánchez, A. & Fernández de Castro, I. (Eds.), *Computer Aided Learning and Instruction in Science and Engineering*, Springer-Verlag, 186-194.

Petri, B., Mouton, H. and Reigeluth, C.M. (1987). A Lesson Based on the Gagné-Briggs Theory of Instruction. In Reigeluth, C.M. (Eds.), *Instructional Theories in Action*, Lawrence Erlbaum, 11-44.

Reigeluth,C.M., Merrill, M.D., and Bunderson,V. (1978). The Structure of Subject Matter Content and Its Instructional Design Implications. *Instructional Science*, 7, 107-126.

Reye, J. (1995). A Goal-Centred Architecture for Intelligent Tutoring Systems. J. Greer (Ed), In *Proceedings of Seventh World Conference on Artificial Intelligence in Education AI-ED'95*, AACE, 307-314.

Russell, D.M. (1988). IDE: The Interpreter. In Psotka, J., Dan Massery, L., Mutter, S.A. (Eds.), *Intelligent Tutoring Systems. Lessons Learned*, Lawrence Erlbaum Associates, 323-349.

Russell, D.M., Burton, R., Jordan, D.S., Jensen, A., Roger, R., Cohen, J. (1990). Creating instruction with IDE: tools for instructional designers. *Intelligent Tutoring Media*, 1(1), 3-16.

Scandura, J.M. (1983). Instructional Strategies Based on the Structural Learning Theory. In Reigeluth, C.M. (Eds.), *Instructional-Design Theories and Models: An overview of their current status*, Lawrence Erlbaum Associates, 213-246.

Self, J.A. (1990). Bypassing the Intractable Problem of Student Modelling. In Frasson, C. and Gauthier, G. (Eds.), *Intelligent Tutoring System. At the Crossroads of AI and EI*, Ablex, 107-123.

Shuell, T.J. (1985). Designing Instructional Computing Systems for Meaningful Learning. In Jones, M. & Winne, P.H. (Eds.), *Adaptive Learning Environments. Foundations and Frontiers*, Springer-Verlag, 19-54 .

Sokolnicky, T. (1991). Towards Knowledge-Based Tutors: A Survey and Appraisal of Intelligent Tutoring Systems. *The Knowledge Engineering Review*, 6(2), 59-95.

Soloway, E., Guzdial, M., Brade, K., Hohmann, L., Tabak, I., Weingrad, P., Blumenfeld, P. (1992). Technological Support for the Learning and Doing of Design. In Jones, M., & Winne, P.H. (Eds.), *Adaptive Learning Environments. Foundations and Frontiers*, Springer-Verlag, 173-200.

Srisetamil, C. and Baker, N.C. (1995). Application and Development of Multiple Teaching Styles to an Engineering ITS. In J. Greer (Ed), *Proceedings of Seventh World Conference on Artificial Intelligence in Education AI-ED'95*, AACE, 75-82.

Srisetamil, C. and Baker, N.C. (1996). ITS-Engineering: A Domain Independent ITS for Building Engineering Tutors. Frasson, C., Gauthier, G., Lesgold, A. (Eds.), In *Proceedings of the Third International Conference Intelligent Tutoring Systems ITS'96*, Springer-Verlag, 677-685.

Stevens, A. (1982). Misconception in student' understanding. In Sleeman, D. and Brown, J. (Eds.), *Intelligent Tutoring Systems*, Academic Press, 13-24.

Vadillo, J.A., Díaz de Ilarraza, A., Fernández, I., Gutiérrez, J. and Elorriaga, J.A. (1994). Explicaciones en Sistemas Tutores de Entrenamiento: Representación del Dominio y Estrategias de Explicación, *Actas II Congresso Ibero-americano de Informática na Educaçao*, 2, 289-309.

Van Marcke, K. (1992). A Generic Task Model for Instruction. In Dijkstra, S., Krammer, H.P.M., VanMerrienboer (Eds.), *Instructional Models in Computer-Based Learning Environments*, Springer-Verlag, 171-194.

Van Marcke, K. and Vedelaar, H. (1995). Learner Adaptivity in Generic Instructional Strategies. In J. Greer (Ed), *Proceedings of Seventh World Conference on Artificial Intelligence in Education AI-ED'95*, AACE, 323-333.

Vassileva, J. (1995a). Dynamic Courseware Generation: At the Cross Point of CAL, ITS and Authoring. In *Authoring Shells for Intelligent Tutoring Systems, Workshop at AI-ED´95*, Washington, AACE.

Vassileva, J. (1995b). Reactive Instructional Planning to Support Interacting Teaching Strategies. In J. Greer (Ed), *Proceedings of Seventh World Conference on Artificial Intelligence in Education AI-ED'95*, AACE, 334-342.

Verdejo, M.F. (1992). User Modelling in Knowledge-Based Systems. In Ezquerro, J. and Larrazabal, J.M. (Eds.), *Cognition, Semantics and Philosophy*, Kluwer Academic Publishers, 23-46.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishing.

Weinstein, C.E. and Mayer, R.E. (1986). The teaching of Learning Strategies. Wittrock, C. (Eds.), *Handbook of Research on Teaching*, New York: Macmillan Publishing Company, 315-327.