A Knowledge Level Analysis of Taxonomic Domains

Marta Domingo* and Carles Sierra
Artificial Intelligence Research Institute, IIIA, Spanish Council for Scientific Research, CSIC 08193, Cerdanyola, Barcelona, Spain

The Knowledge Level (KL) is an abstract level of description, prior to the symbol or software level, which aims at discovering the components of expertise without thinking of computational aspects. The KL analysis emphasizes the regularities in knowledge use for knowledge engineering. We consider the knowledge level analysis the AI counterpart of the specification of programs. Then, it must be possible to define formal ways of putting in relation the KL analysis with computational elements that implement it. The ultimate goal of the research presented in this article is to contribute in the filling of the gap between specification at the KL and implementation. To do so we propose (i) a particular interpretation of the three main concepts involved in the knowledge level theories, i.e., tasks, methods, and domain models, and (ii) a mapping between these notions and computational elements of ***Milord II,*** a shell developed at the IIIA Institute and used as the target programming environment of an example in biological identification.

I. INTRODUCTION

The Knowledge Level (KL) notion was introduced by Newell.[1] Afterwards, other authors gave particular interpretations of this approach.[2–6] Although they differ in some ideas and use different terminology, all of them view *tasks, knowledge,* and *reasoning strategies* as underlying principles of a knowledge base at the abstract level (KL) of analysis. Thus, the KL is an abstract level of description, prior to the symbol or software level, which aims at discovering the components of expertise without thinking of computational aspects. The KL analysis emphasizes the regularities in knowledge use for knowledge engineering. Some knowledge is specific of a domain but some other knowledge is applicable to a class of problems. From the viewpoint of Knowledge-Based System (KBS) technology, the elucidation of such regularities in knowledge represents a considerable shortage of efforts in the construction of future applications.

Steels proposed the identification of what he called *components of expertise*

*Author to whom correspondence should be addressed.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 12, 105–135 (1997)
 CCC 0884-8173/97/020105-31

as a KL analysis of problems.[4] He suggested that a problem domain can usually be described from three perspectives. The *task* perspective deals with the specific piece of work that a problem solver intends to perform, for example classification or planning. The *model* perspective focuses on domain-dependent knowledge and is concerned with the specific domain concepts. The *method* perspective focuses on how domain knowledge is to be used to accomplish the intended task. A method can be seen as a piece of knowledge that organizes and controls the execution of a set of subtasks. This is why we can explain what each method represents by means of its decomposition into a subtasks structure. At the same time, each subtask assumes a particular domain model and method.

Despite the effort made by these authors in the standardization of the terminology and in the setting of methodologies to make the analysis at the knowledge level, the final step to the implementation, that is mandatory to have real working systems, is not considered as a part of the methodology. Basically, it has been so because there were philosophical suspicions that a relation from such an abstract description and a concrete implementation framework could ever be found. This gap between the knowledge level analysis and the implementation has been usually the source of the main criticisms to this approach to KBS building. However, we consider the knowledge level analysis the AI counterpart of the specification of programs. Then, it has to be possible to define a methodology to put in relation the KL analysis with the computational elements that implement it. Some authors share the same opinion and have proposed frameworks to develop applications such as the *componential framework* (COMMET) and the KREST *workbench,*[7] whose goal is to support the design of knowledge systems on the basis of the *components of expertise* methodology.

The ultimate goal of the research presented in this article is to contribute to filling in the gap between specification at the KL and implementation at the code level. To do so we propose (i) a particular interpretation of the three main concepts involved in the knowledge level theories, i.e., tasks, methods, and domain models, and (ii) a mapping between tasks, methods, and domain models and the computational elements of ***Milord II,*** a shell developed at the IIIA Institute and used as the target programming environment of an example in biological identification.

In the last decade important efforts have been made to introduce automatization to the difficult task of classification and identification in taxonomic domains. Several approaches have been explored to satisfy taxonomic needs such as database systems, interactive identification programs or knowledge-based systems (KBS). The latter is a powerful new approach in the systematics field. The knowledge needed to model taxonomic identification processes may be obtained from direct elicitation of human expertise. In order to achieve this goal, one is faced with the challenge of acquiring and formalizing this expertise. In practice, doing so represents such a great effort that some systematists are discouraged at the first steps. To overcome this problem, the above mentioned mapping serves as a methodological approach to KBS construction that simplifies the knowledge modeling. To illustrate this methodology, we present an example of the use of the mapping to define an architecture for the construction of KBS dealing with taxonomic domains.[8,9]

In Section II we introduce the modular constructs of ***Milord II*** and some basic operations that will give sense to the mapping proposed in Section III. Also in Section III we make a revision of the concepts involved in the KL theory. In Section IV we state how to model taxonomic knowledge from the knowledge level perspective. Section V presents an architecture for taxonomic domain applications. Finally, Section VI discusses briefly the related work on Knowledge Level and Taxonomic Applications.

II. *MILORD II*

The ***Milord II*** programming environment[10,11] was developed using COMMON LISP and runs on a Macintosh environment. It is an extension of *MILORD*,[12] which has been successfully applied to several medical KBS.[13,14] The ***Milord II*** programming environment is specially suitable to cope with large and complex applications that need to be tackled incrementally.

A. Basic Constructs of *Milord II*

The structural unit of ***Milord II*** is the *module.* A module may be seen as a black box that admits some information as input and outputs a result. Physically, modules are compound structures (Fig. 1). A detailed description of its components is presented.

1. Interface

The *interface of a module* is the means of communication with the user or with other modules. The interface of a module has two components. The *Import* interface specifies which information can be demanded to the user or to other

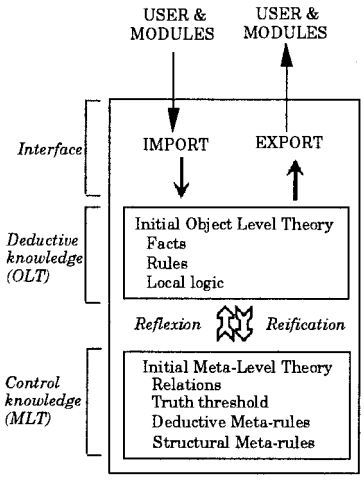

Figure 1. Schema of the components of a module.

modules and the *Export* interface specifies the possible output of the module. For instance, in a module named `Classes` three facts could be deduced, i.e., `Demospongiae`, `Calcarea`, and `Hexactinellida`, and the user may be asked about two facts, i.e., `geographic` and `continue`, if needed in the inference process.

Module Classes =
Begin
    Module S= Skeleton
    Inherit External
    Open Ecology
    Import geographic, continue
    Export Demospongiae, Calcarea, Hexactinellida
    . . .

From the Module `Classes` three other modules are accessible, i.e., the module `Skeleton`, the module `External`, and the module `Ecology`. These are called *submodules* of the Module `Classes` and can be consulted to obtain different kinds of data. There are three types of submodule declaration as shown in this example. The implications for each one are to be explained along with the submodule declaration operation.

2. Deductive Knowledge

This is the knowledge concerning the relation between the import and export interfaces. It is also called *object-level* knowledge. The *deductive* or *object-level knowledge* of a module is composed of the following components:

(a) *Dictionary.* It contains the definitions of the concepts of the modeled domain; from now on they will be called *facts.* Each fact is labeled with a *fact identifier,* an associate *question* if the fact is going to be directly asked to the user, and a *type.* Four built-in types are provided. Three of them, *enumerate, Boolean,* and *numeric,* can be understood as multistate, binary (or two-state), and numeric characters, respectively, in the context of taxonomic domains. The fourth one is called *logic* type. It is used to represent facts whose value is expressed by experts with a linguistic certainty term chosen from the set of totally ordered linguistic terms defined locally inside each module. ***Milord II*** models the uncertainty of a fact by assigning to it an interval of linguistic terms.[15,16] Intervals mean a degree of imprecision on the uncertainty of facts. So, we represent observed and deduced logical facts as pairs containing the name of the fact and an interval described by its lower and upper bounds [e.g., (`Calcarea`, [`very_possible`, `sure`])].

Relations between facts may also be defined. For instance, the relation *needs* in the following example expresses a restriction in the order in which the facts `chemical` and `skeleton` must be queried if needed. A value for the fact `skeleton` has to be obtained before we ask for a value for `chemical`. These relations are defined at the object-level for the sake of compactness. However,

they belong to the control knowledge as meta-predicate instances. Types and relations can be defined by users.

Deductive knowledge
Dictionary:
Predicates:
. . .
Chemical = **Name**: ''Chemical composition''
Question: ''Which material is the skeleton made of?''
Type: (calcium_carbonate, silica, organic)
Relation: needs skeleton
. . .

(b) *Rules.* This component represents implicative knowledge. *Rules* establish associations between facts. *Premises* are conjunction of conditions. Conclusions are labeled with a linguistic certainty value. In fact, rules are labeled with an interval that goes from the linguistic term appearing in the definition of the rule to the maximum in the total order of the linguistic terms defined in the module; in the example below, R001 deduces the fact `Demospongiae` with the interval [`possible`, `sure`]. A wide set of expression possibilities for conditions is provided by the ***Milord II*** syntax. An important aspect of the syntax of rules is the way we refer to facts. We will see later that modules are structures in a hierarchy, so a particular module may refer to facts exported by particular submodules. To do this, a fact reference is composed of: (1) a *path* of module names, separated by ''/'', indicating how to access to the fact in the hierarchy of modules, and (2) a condition construct referring to the fact identifier, i.e.,

$$\text{module\_name}_1 \,/\, \text{module\_name}_2 \,/\, \ldots \,/\, \text{module\_name}_n \,/\, \text{fact\_identifier}$$

In the next rule the reference in a condition to the fact `chemical` of submodule S is done as:

$$S/\text{chemical}=(\text{silica})$$

R001 **If** *S*/chemical=(silica)
then conclude Taxon=(Demospongiae) **is** possible
''Comment: this is a rule of Module Classes. The fact *chemical* belongs to Module Skeleton, locally renamed as submodule *S*''

(c) *Local logic.* It defines three main components of an *Inference System*: (i) the set of *truth-values* (or *certainty values*), (ii) a renaming mapping between the truth-values of the submodules, if any, and the truth-values of the module, and (iii) the *connective* operators used to combine the truth-values of premises and propagate them through the implication connective when making inference.

The following piece of code, from an application in marine sponges identification,[17] is an example of module definition containing simple rules and a logic declaration.

Module Classes =
Begin
Module *S*=Skeleton
Inherit External
Open Ecology
Import geographic, continue
Export Taxon
Deductive knowledge
Dictionary: *not defined here*
Rules:
R001 **If** *S*/chemical = (silica)
then conclude Taxon = (Demospongiae) **is** possible
R002 **If** *S*/pres **and**
S/chemical = (silica)
then conclude Taxon = (Demospongiae) **is** possible
R003 **If** *S*/fiber
then conclude Taxon = (Demospongiae) **is** sure
R004 **If** *S*/chemical=(calcium_carbonate)
then conclude not(Taxon = (Demospongiae)) **is** sure
. . .
R014 **If** *S*/chemical=(calcium_carbonate)
then conclude Taxon = (Calcarea) **is** sure
R015 **If** External/brush
then conclude Taxon = (Hexacinellida) **is** possible
R016 **If** association
then conclude Taxon = (Demospongiae) **is** possible
. . .
Inference system:
Truth-values = (impossible, possible, sure)
Connectives:
Conjunction:
((impossible, impossible, impossible)
(impossible, possible, possible)
(impossible, possible, sure))
End deductive
End

Connectives may be defined as a truth-table, represented by rows, and following the order of the linguistic terms. In the conjunction definition of this sample module it is represented that: `impossible` $\wedge$ `impossible = impossible, impossible` $\wedge$ `possible = impossible`, etc.

Milord II provides a criterion for conflict resolution that concerns the order in which rules are applied and a mechanism to combine the certainty values of the rules that are applied.

There are three criteria involved in deciding the order in which rules are applied: subsumption, certainty value, and writing order. They are considered

in this order in such a way that the system, respecting the writing order, tries first the more specific and the more certain rule. The *subsumption* criterion establishes that the most specific rule be tried first. First instance, R001 is subsumed by R002 because the premises of R001 are contained in R002. Thus, R002 would be tried first. In fact, only if the more specific rule fails (because the premise cannot be satisfied) is the subsumed rule to be applied. The second criterion is the rule's *certainty value.* The rule with the highest certainty value is tried first. Thirdly, the *order* in which rules are written is maintained if no other criterion applies.

In the previous module, to obtain a certainty value for the fact `Taxon = (Demospongiae)` the system would proceed as follows: starting with the set of rules in the module {R001, R002, R003, R004, R016}, after considering the subsumption criterion the following set of rules remains: {R002, R003, R004, R016}, because R001 is subsumed by R002. After considering the certainty value over the set {R002, R003, R004, R016}, the set of rules becomes: {R003, R004}, because R002 and R016 have a lower certainty value. Finally, the system tries R003 first because it is the first written rule.

The certainty combination mechanism of ***Milord II*** computes the *intersection* of certainty intervals. This is going to be explained by means of an example in the previous piece of code. Let us suppose that we have a specimen of sponge in which we observe the presence of a spongin fiber skeleton with tracts of siliceous spicules within. In the presence of these facts, two rules would be satisfied: R003 would conclude that `Taxon = (Demospongiae) is [sure, 1]`, and R002 would conclude that `Taxon = (Demospongiae) is [possible, 1]`. The final certainty would be computed as the *intersection* of these certainty intervals, namely [sure, 1] ∩ [possible, 1], which is: [max(possible,sure), 1]. The resulting interval, namely [sure, 1], would be propagated in the inference process.

Actually, in the inference process it would suffice to apply R003 because it concludes the fact `Taxon = (Demospongiae)` with maximum certainty. That is, when a fact is concluded by a rule with maximum certainty no more rules are applied because the result cannot be improved.

This combination mechanism is a very conservative one because the satisfaction of several rules concluding a taxa with a moderate certainty value does not increase the final certainty value that the system will output. For example, if two rules concluding a certainty [possible, 1] were applied, the final certainty would still be [possible, 1], that is, the result of the intersection [possible, 1] ∩ [possible, 1].

3. Control Knowledge

This is the knowledge concerning the management or control of facts and rules within a module. It is also called *meta-level* knowledge because it is able to control the object-level knowledge. The *control* or *meta-level knowledge* of a module is composed of meta-rules. *Meta-rules* are components that manage the rules and the submodules to make the search efficient in the knowledge base

and to model the strategies of the expert's behavior. Meta-rules can be considered structurally similar to rules but their expressive capabilities are stronger. Two different goals can be distinguished in control knowledge. *Deductive control* can affect rules by different kinds of inhibition. *Structural control* can affect modules of the modular hierarchy by filtering or ordering submodules, or by defining dynamic creation of other submodules.

In meta-level knowledge the type of module execution can be defined as either *lazy* or *eager. Lazy* means that facts are investigated only when needed, i.e., imported facts and exported facts of submodules are to be obtained only if they are useful for computing the export interface of the module. An *eager* module execution obtains, first of all, the values for all the imported facts and for all the exported facts of the submodules, and then the deductive knowledge is used to obtain the values of the exportable facts of the module.

Another parameter which is defined in meta-level knowledge is the *truth threshold.* It is a certainty value that imposes a minimum certainty in facts to be considered meaningful in the inference process. The truth threshold is local to each module, in this way, we can control the degree of precision that is significant in each module depending on the problem that it approaches.

So far we have defined two main bodies of knowledge: the object-level and the meta-level. To be operative they need to work in close interaction. Their interaction is based on a *reification/reflection* process, that can be seen as a continuous updating mechanism of the knowledge that arrives in the system. This updating mechanism is a key issue of the reasoning in ***Milord II.***

The *reification* process maps statements of the object-level into statements of the meta-level, whereas the *reflection* process maps meta-level statements into object-level statements. The *reification/reflection* process is made after each import information is obtained from the user, after each submodule consultation and after each time the object-level knowledge is extended by the deduction of new information. In other words, the knowledge of the meta-level is updated each time the object-level knowledge changes. A detailed explanation of this is out of the scope of this article, the reader is referred to.[16,18]

B. Operations Between Modules in *Milord II*

Incremental programming is considered a safe way of getting correct programs. The programming methodology consists on defining small pieces of code that are later on elaborated, combined, or redefined, up to a point in which the final program is obtained. In some languages this is materialized in a three-step process for each piece of the program: definition of an initial specification, definition of an implementation of it, and finally, verification of the correctness of the implementation. The combination of these verified pieces gives the final complex program. This is the way, for example, SML[19] works.

In Knowledge Based Systems the methodology seems also useful despite the fact that precise specifications are, in many cases, difficult to obtain. The idea of working with small pieces of knowledge has been increasingly used in the artificial intelligence field. Objects, agents or modules are the components

from which bigger systems are defined. However, no clear correlate of the incremental programming methodology as stated before has been applied. ***Milord II*** presents some ideas in this direction. The methodology consists on considering knowledge and specification as interchangeable terms. This is done like this because usually modules are nothing but restrictions over the relation between factual data and results, as specifications are. To cope with the idea of ill-defined problems (problems with partial, incomplete or even erroneous specifications) we propose an *n*-step process in which each module (specification) is implemented using another module (new specification) which is again implemented using a module, and so on, up to a point in which the knowledge in the final module is considered to be competent with the problem to solve. Every step is verified as in the classical process. This incremental process is modeled by means of an operation called *refinement* from which some formal properties are outlined in the article.

Besides refinement, ***Milord II*** provides two other basic mechanisms for module manipulation: composition of modules through the declaration of *submodules* and composition of modules through operators defined by the user via *generic modules.*

1. Submodule Declaration

The declaration of *submodules* is identical, syntactically, to the declaration of modules, and is the key to the hierarchical organization of the knowledge base (Fig. 2).

The export interface of a submodule is accessible from the module that contains it in order to entail its conclusions. There are three syntactically correct

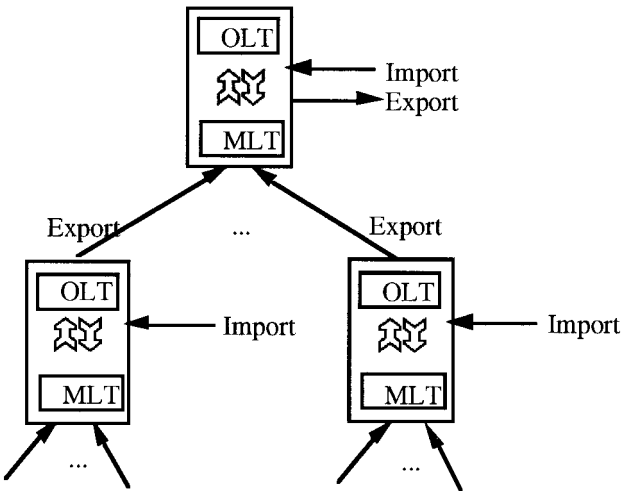

Figure 2. Schema of the submodule operation.

versions for the submodule declaration that allow different prefix mechanisms in the access to the submodule. Recalling the previous module `Classes` definition in Section II-A, the three possible submodule declaration syntax are shown there with an example of the respective correct rule syntax in R014, R015, R016. Notice that the remarkable difference is the fact's prefix in the rule's premise. The definition of a submodule by **`Module`** *`new_name = Old_name`*, means that to access facts from the module *`Old_name`* we will use the identifier `New_name` in the path. **`Inherit`** just keeps the old name; in fact, **`Inherit`** *`A`* is equivalent to **`Module`** *`A = A`*. In the case of a submodule declared with **`Open`**, the access is directly by referring to fact identifier. In this way, the fact *`association`* in R016 belongs to the submodule *`Ecology`* and need not be prefixed. ***Milord II*** takes care, and warns when necessary, of name clashing when two submodules being opened export the same fact identifier.

2. *Refinement of Modules*

The relation between an initial module A and a more elaborated version of it, *B,* is established by means of an operation called *refinement* and is noted $C:A$ (read C is a refinement of A). The central idea in what concerns verification of the refinement operation in our approach is that when we refine a module we want to get a new module with the same interface but with "more precise" knowledge.[20] We understand "precision" with respect to a predefined ordering relation over the set of fact values. In general, the verification of being more precise is computationally untractable. In some cases the complexity can be reduced when the module contents are written in a simple way, as it is the case for the ***Milord II*** architecture (propositional language at the object level).

In the example of Figure 3, module A only specifies the interface, module B is generated by ***Milord II*** if the refinement relation between C and A is satisfied. If so, the resulting module is C extended by some components inherited from A, for example, dictionary definitions or rules. In fact, in ***Milord II*** it is equivalent to write **`Module`** *`B:A = C`* or **`Module`** *`B = (C:A)`*. In Figure 3, B is the result of extending A by an object-level theory, defined in C. In the same way, module E adds a meta-level theory to B generating the module D.

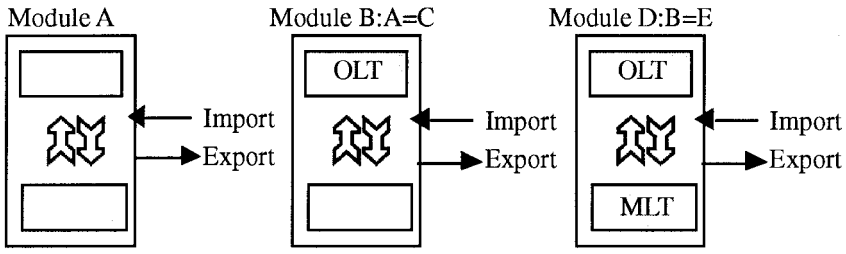

Figure 3. Example of refinement operations.

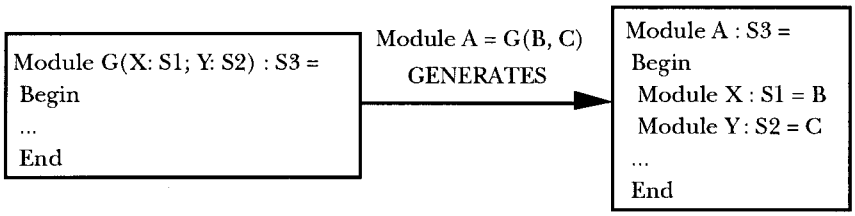

Figure 4. Generic Module application.

3. Generic Modules

Generic modules are the way to define functions from modules to modules. They work in a similar way to a linker in a programming language. An example of a Generic Module is the following:

Module $G(X:S_1;\ Y:S_2):S_3 =$
Begin
;;*Usual module's body definition.*
. . .
End

This generic module accepts two modules as parameters, X and Y, satisfying the *refinement* relation with S_1 and S_2, respectively, and produces a module satisfying the refinement relation with S_3. These generic modules can be defined at top level or at any level in the hierarchy of modules. In the generic module body, it is allowed to make direct reference to the parameters X and Y (Fig. 4). Generic module application may be *static,* i.e., the compiler is responsible for applying the generic modules to their actual parameters, or *dynamic,* i.e., they can be applied at run time, that is, the modules that are parameters of a generic module instantiation are determined at run time.

A generic module can be seen as a module with some submodule names, namely the name of the arguments, not linked to any particular module. When the generic module is applied, those submodule names are linked to the particular modules given as parameters.

We say that a module *combines* other modules when it has access to their exported facts. A module can obtain rights to access the exported facts of another module in two ways: (1) declaring it as submodule, and (2) being the result of the application of a generic module, getting in this case access rights to the exported facts of the arguments of the application. In both cases, the module may use, in its rules and meta-rules, the facts appearing in the export interfaces of the modules declared as submodules and/or the parameters. The access to the exported facts of submodules and/or parameters is made by using the prefix mechanism described previously.

These combinations of modules build up a hierarchical structure of modules. Each module containing their particular object level and meta-level theories.

The distinction between a knowledge base and a piece of a knowledge base is lost in ***Milord II*** because a module is considered as a partial knowledge base itself. Eventually, the full-scale application may be summarized by a single module, which contains other modules as submodules.

III. KNOWLEDGE LEVEL NOTIONS AND THEIR MAPPING TO *MILORD II*

The mapping between the abstract knowledge level concepts and the computational elements of a programming environment is necessary to bridge the gap between design (specification) and implementation (programming) in knowledge bases construction. We think, according to Van de Velde,[5] that any methodology for the construction of second generation ES needs a mapping between the KL model and the knowledge-based architecture.

In this section we deploy a mapping of our notions of task, domain model, and method to the ***Milord II*** programming constructs. This mapping results in a particular methodology for the design and construction of knowledge bases in ***Milord II,*** in which the KL analysis is the starting point for knowledge-based systems construction.

A. Task

We understand a Task as a specification of a problem in terms of the requirements for the input and output behavior. That is, the concepts used, their types, and their possible interrelations. Eventually, a task can contain the same type of requirements over those subtasks which are invariant over method applications.

The methodology to map tasks into ***Milord II*** modules is done by defining (1) a set of fact identifiers that will be used as the input/output language of the module; and optionally (2) a set of subtask name identifiers (when they are defined statically). In the case of ***Milord II,*** the input/output language is specified in the interface of a module. Thus, we map the concept of task to a module whose content is a pure interface, that is, without elements of deduction (rules, meta-rules, etc).

A schema of a task would then be:

Module NodeTask =
  Begin
    Inherit Subtask_1
    . . .
    Inherit Subtask_2
    Import $I_1, I_2, \ldots, I_n$
    Export $E_1, E_2, \ldots, E_m$
  End

or just

Module LeafTask =
  Begin
    Import $I_1, I_2, \ldots, I_n$
    Export $E_1, E_2, \ldots, E_m$
  End

In the previous modules representing tasks, the symbols concerning the

input fact identifiers are declared in the `Import` interface and those concerning the output fact identifiers are declared in the `Export` interface. Thus, the task (module) imposes the minimum input/output behavior of any Domain Model fitting in with it. That is to say, the task specifies the maximum information that the domain model can ever use and the minimum conclusions which it must be able to draw to perform that task. Tasks containing no subtasks are of the type of `LeafTask`, those subdivided in subtasks are of the type of `NodeTask`.

B. Domain Model

A Domain Model is the component of the knowledge base containing domain-dependent knowledge, that is, the knowledge specifically related to the area of study. At the same time, it is responsible for giving contents to a task by establishing concrete relations between input and output of the task. So, in our interpretation of Task, as a specification of input/output behavior, Domain models are particularizations of a general task schema in a concrete domain, by giving the components that relate the input and the output of the task. The distinction between task and domain model is then a matter of granularity. Task is the schematic and domain-independent approach to a domain model. So, the construction of domain models can be seen as an incremental process starting with a task (no domain-dependent knowledge) and ending with a ''complete'' domain model. That is, we can again particularize a domain model by defining a new domain model containing more detailed relations between the input and the output, where *more detailed* is a domain-dependent concept, meaning for example, the increase in the level of observation detail in biological characters, e.g., from macroscopical to ultrastructural.

This intepretation of Domain model differs from the idea of Domain theory in KADS,[6,24] given that Tasks in KADS refer to control of the inference and contain no information about the particular problem being solved. So, Domain theories in KADS cannot be interpreted as particularizations of Tasks.

In ***Milord II*** a domain model is then naturally represented by means of a module. The verification that a domain model is a particularization of a task is done by the refinement operation of ***Milord II.*** In particular, this consists in checking whether the refinement relation *Domain_Model/Task* is correct (read *Domain Model* is a refinement of the *Task*). That is to say, it checks whether the domain model satisfies the input/output language requirements established by the task. That is,

Module Domain_model : Task =
  Begin
    Import $I_1, I_2, \ldots, I_n$
    Export $E_1, E_2, \ldots, E_m$
    Deductive knowledge
  ;;This deductive knowledge deals with domain-dependent knowledge
    End deductive
    Control knowledge

;;This control knowledge deals with control domain-dependent knowledge
End control
End

C. Method

A Method is the component that links a task with a domain model to generate a problem solver for the task. To achieve this goal, a method creates the task/subtask hierarchy and defines the elements of control on top of this hierarchy. Also, it embeds the domain model as the domain-dependent component of the resulting problem solver. In other words, the method defines the steps that may be followed in the inference process to perform a task in the context of a particular kind of domain model. The further determination of which methods are applicable to subtasks has in many cases to be determined depending on the current results of the problem solver. That implies that methods cannot only be considered as static linkers. So, for us, methods have to be considered, at the knowledge level, as manipulators of other methods, that is, methods subdividing tasks into subtasks have to handle the responsibility of dynamically determining which methods are more suitable for the subtasks. Obviously, in simple cases a static decision could be enough.

Methods are then naturally defined as generic modules in ***Milord II*** as follows:

Module Method (DM: T; T: Task) =
Begin
Import $I_1, I_2, \ldots, I_n$
Export $E_1, E_2, \ldots, E_m$
Control knowledge
;;This control knowledge dynamically creates the
;;subtasks of the task and defines the control
;;regime over them
End control
End > T

This expression is a schema of a Method module. In the heading of the generic module the parameters that are needed for this module to be applied are specified between brackets. The first one is a domain model (DM) and the second one is a task (T).

Not every method is suitable for performing a given task. Hence, each method is defined for a particular type of task. That is why in the parameter definition of the module method we have *(DM: T; T: Task).* Any argument T satisfying the refinement operation *T: Task* is a correct argument for that method.

The body of the module contains an expansion of the task, that is to say, the input/output behavior generated by the method module is at least as rich as the behavior specified in the task module (it may contain some additional requirements besides the minimum requirements of the task).

Methods mapped into generic modules may define problem solvers only partially, as required at the knowledge level analysis, that is, at run time the problem solver can use the same or other methods in a recursive way to complete the problem solver definition depending on the partial execution results. This is done by allowing meta-rules in ***Milord II*** to generate submodules resulting from the application of generic modules representing methods. In this sense a method is a lazy operation generating the minimum elements necessary to start the problem solving.

IV. A KNOWLEDGE LEVEL ANALYSIS OF TAXONOMIC DOMAINS

In the following paragraphs, we will illustrate the meaning of the Knowledge Level concepts in the context of taxonomic domains.

A. Tasks

Systematics and taxonomy are the branches of biology concerned with the classification and identification of living organisms. Classification refers to the recognition of groups among a number of species, while identification presumes the existence of groups to which an unknown specimen could be assigned. Although different methods have been developed to facilitate these two activities, they are in essence closely related, and it is difficult to keep them separate. Thus, although taxonomic literature uses mainly the term *classification,* the underlying issue is often one of identification. At the same time, a fundamental purpose in identifying an unknown specimen is to obtain an idea of its classification. In this article we will work on the knowledge level analysis of the classificatory problem as a whole and specifically of the *identification* task.

The identification *task* involved in taxonomic domains may be understood as follows: since we assume that the taxa of a phylum are distributed in a hierarchical arrangement, given an unknown specimen, represented by a set of characters (input), we want to obtain the subtree of the taxonomic hierarchy representing the different taxonomic levels to which it belongs (output). Furthermore, we want this goal to be achieved even when the collection of characters involves incompleteness or uncertainty.

B. Domain Model

As a second step of the KL analysis, an ontology for taxonomic *domain models* is presented. We focus attention on two main groups of taxonomic concepts which are the minimal set of knowledge elements for building a domain model for the construction of any particular taxonomic application: a *structural model,* i.e., knowledge related to the structure of the individual's body, and a *classificatory model,* i.e., knowledge related to classification aspects.

1. Structural Model

- *character:* descriptive domain concept concerned with the features of a specimen. Depending on the kind of information, different groups of characters exist, e.g., morphological, chemical, genetic, etc.
- *character dependence tree:* hierarchical relation between characters in which a character is pertinent to an individual only if the individual presents a different character with a particular value.
- *causal relation:* a character is assumed to be present due to the presence of a different character when a cause–effect relation among these characters exists.
- *incompatibility relation:* the presence of a character is assumed to be not possible after being observed the presence of some other character.

2. Classificatory Model

- *taxon:* groupings of organisms, fitting in with a prototypical definition, considered to be sufficiently distinct from other such groups to be treated as a separate unit.
- *taxonomic tree:* hierarchical arrangement of taxa in which each taxonomic level represents an increasing degree of abstraction from species to phylum. Taxa are exhaustive classes (i.e., each taxon must belong to a higher level taxon) and exclusive classes (i.e., each taxon cannot belong to more than one taxon).
- *diagnostic character:* its presence determines or contributes to the identification of the members of a taxon.
- *confirmatory or auxiliary character:* its presence is not indispensable for the identification but it contributes to verify that identification.
- *character variability:* the property of being variable in form, quality or quantity among the individuals of the same taxon, e.g., the body shape or the color may vary within a given range in some sponge species.

In addition, we must cope with some constraints on this knowledge: *Uncertainty, incomplete or inconsistent data.*

Incompleteness of the domain theory can exist because either the concepts or the known species can be revisited due to new findings in observational techniques. For example, a spicule could be considered smooth under the light microscope, but present spines after an electron microscope examination. On the other hand, the discovery of new species of sponges is still common.

Loss or distortion of information is frequent when examining fragmentary specimens or preserved ones. Fixation of specimens is needed for conservation and microscopic study but some characteristics may be lost as a result of this process, e.g., sponges lose their living color and consistency after being fixed.

Data may involve uncertainty. For instance, the plumo-reticulate skeleton of sponges is an intermediate skeleton between the plumose and reticulate patterns, and the delimitation of each one is slight. Uncertainty in the presence of this feature forces us to make assumptions.

A combination of characters does not necessarily ensure the membership of a specimen to a taxon but it provides a degree of certainty for its identification. Also, erroneous or biased observations may lead to inconsistency in the identification of taxa.

C. Methods

A third aspect to model is the decision-making processes that are significant in taxonomic domains. This applies to what is called *method* in the KL terminology. We are to explore two different problem-solving methods: *refinement* and *propose and revise,* each one appropriate to a particular type of user.

In AI, a standard problem-solving method for classificatory tasks is *refinement,*† also known as *simple classification.*[2] This method (Fig. 5, left) consists of the iterative process of identifying the more promising taxa at a taxonomic level, and refining the taxonomic tree (i.e., pruning the branches that have been rejected). This method forces the user to bring in more and more details about characters, in order to identify each particular level of abstraction. It is a good method to detect inconsistency of the user's answers because it covers the taxonomic tree step by step. In this way, it is a suitable method for users that are not especially familiarized with the domain problem, or for more experienced users facing up to a difficult case. Even if the sample is a new species for the science (i.e., a species never found until that moment) the refinement method could give a ranked list of hypotheses of higher taxonomic rank than species.

Another interesting problem-solving method to be applied in classification is the *propose and revise* method, somehow related to the previous one (Fig. 5, right). The first step is to obtain an identification hypothesis by asking to the user. He expresses this hypothesis on the basis of his own preliminary conclusions for the sample. Then, consistency with the higher path of the taxonomic tree is verified by the system asking the user some essential questions about diagnostic characters. If consistency is proved, the hypothesis is considered as a first identified taxon and the resulting subtree is top–down refined. If the hypothesis cannot be proved to be correct, either because it is not correct or because some questions cannot be answered by the user, the system switches to the refinement method from the rest of the classification tree.

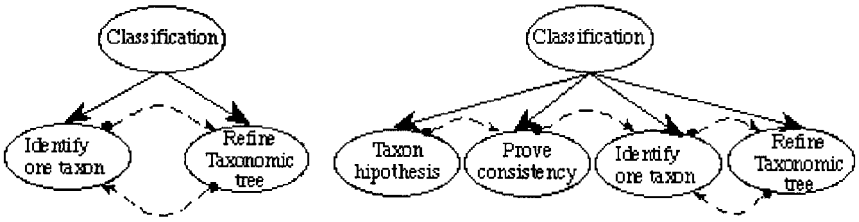

Figure 5. Subtasks decomposition obtained by the Refinement method (left) and the Propose & Revise method (right).

†The term *refinement* concerning methods has not to be confused with the module operation *refinement* of the ***Milord II*** programming environment.

V. AN EXPERT SYSTEM ARCHITECTURE FOR TAXONOMIC DOMAINS

After the KL analysis of taxonomic domains and the mapping of the abstract KL concepts to the ***Milord II*** programming environment, in this section, we particularize this mapping for taxonomic domains.

We use the schema of tasks, domain models, and methods expressed in the ***Milord II*** language to implement the particular KL elements dealing with taxonomic domains, i.e., a *classification task,* a *taxonomic domain model,* and two *classification methods.* This implementation clarifies the ideas proposed so far and provides an architecture for the construction of KBS applications in taxonomy. A simplified schema of this architecture is presented in Figure 6.

A. Classification Task

The *classification task* imposes as the only output requirement that the task provide some values for a variable here named `Taxon`. As explained in the previous section the input is to be obtained from a structural model here named `structures`. In ***Milord II*** terms, the computational representation is:

Module Classification_task =
Begin
Module Structures
Export Taxon
Deductive knowledge
Dictionary:
Types: taxon_type
Predicates: Taxon = **Name:** "Taxon"
Type: taxon_type
Inference system:
Truth-values = (gp, mpop, llp, modp, p, fp, mp, s)
Connectives:
Conjunction: Truth table

(	(gp,	gp,	gp,	gp,	gp,	gp,	gp,	gp	)
	(gp,	mpop,	mpop,	mpop,	mpop,	mpop,	mpop,	mpop	)
	(gp,	mpop,	mpop,	llp,	llp,	llp,	llp,	llp	)
	(gp,	mpop,	llp,	modp,	modp,	modp,	modp,	modp	)
	(gp,	mpop,	llp,	modp,	modp,	modp,	p,	p,	)
	(gp,	mpop,	llp,	modp,	modp,	p,	fp,	fp	)
	(gp,	mpop,	llp,	modp,	p,	fp,	mp,	mp,	)
	(gp,	mpop,	llp,	modp,	p,	fp,	mp,	s	))

End deductive
End

In the dictionary we define an only predicate that we export together with

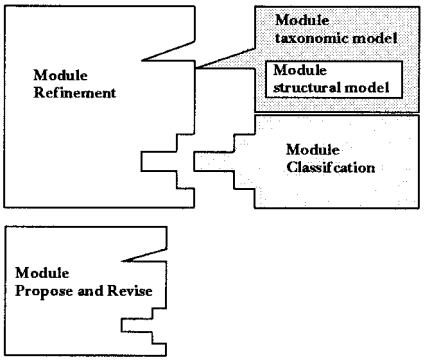

Figure 6. Schema of the expert system architecture for taxonomic domains.

its type; the particular inference system that we want the system to use when drawing conclusions is defined inside the deductive knowledge. We present here a possible inference system as a default inference system in the architecture for the development of applications in taxonomy. The reader interested in defining another inference system in his own application is referred to Ref. 15. Concerning to the default inference system, suffice it to say that it is a conservative one. That is, the conjunction between the premises truth-values is solved by taking the minimum truth-value as a result.

Each original taxonomic application will instantiate the taxon_type with the names of the particular taxa that are dealt with in the application.

B. Taxonomic Domain Model

To represent the knowledge related to a taxonomic domain, two kinds of module schema have been devised: the *taxonomic model* module schema, dealing with the knowledge required to define the membership of a taxon, and the *structural model* module schema, concerning the knowledge required to describe a set of characters dealing with a structure. The skeleton of a taxonomic model module is

```
Module Taxon_model : Classification_Task =
  Begin
    Module structures = Structural_model
    Open Structural_model
    Export Taxon
    Deductive knowledge
      Dictionary:
        Predicates: Taxon = Name: "Taxon"
                            Type: taxon_type
        Rules:
  ;;The knowledge represented in the rules deals
```

```
    ;;with the membership of a specimen to a taxon
        End deductive
        Control knowledge
            Truth threshold: modp
    ;;This control knowledge deals with
    ;;control domain-dependent knowledge
        End control
    End
```

and the skeleton of a structural model module

```
Module Structure_model =
    Begin
        Import I1, I2, . . . , In
        Export E1, E2, . . . , Em
        Deductive knowledge
    ;;The knowledge represented in the rules deals with
    ;;the characters that describe a structure
        End deductive
        Control knowledge
    ;;This control knowledge deals with
    ;;control domain-dependent knowledge
        End control
    End
```

In the architecture, a module following the schema of `Structure_model` is a submodule of a module `Taxon_model`, as noted by **`Module`** `structures = Structural_model` in the above piece of code.

A module following the schema of `Taxon_model` is a refinement of the classification task, as indicated in the heading **`Module`** `Taxon_model:Classification_Task`. Thus, it fulfils the minimum input/output requirements of this task, that is, it provides values for the predicate Taxon on the basis of the data that are input from the Structural model.

In the taxonomic KBS, the truth threshold is an estimate of the maximal uncertainty allowed by human experts to hypothesize membership to a taxon. It is defined within the classificatory model because it is considered as part of the domain knowledge. Its value may be determined heuristically by elicitation from the performance of several human experts in the field. Thanks to modularity, each taxon can be assigned a particular truth threshold if desired. In the definition of the architecture we have taken, as default truth threshold *moderately_possible* (modp).

Two different trends have been followed to associate the concepts of a taxonomic domain model and the computational constructs of ***Milord II:*** One is the combination of modules through submodules declaration, and the other is the assignment of a taxonomic concept to each one of the module's components.

Therefore, in the classificatory model the *taxonomic tree* is represented by

means of a module/submodules hierarchy (each module following the `Taxon_model` schema) that follows the taxonomic levels from phylum to species. Within each module, the object-level theory is concerned with a particular `taxon` that gives its name to this module. Thus, the predicate `taxon` stand for subtaxa. For example, a module that represents an order is declared **`Module`** `order_i`, the predicate `taxon` defined within stand for the families of the order *i*, and the output of this module is the identification of the families of the order *i*. The deductive knowledge of such a module consists of a set of rules following this schema:

If *a* **then** *taxon = (b)* **is** *certainty value*

in which, interpreted as a logic sentence, *a* is a conjunction of sufficient conditions or sufficient and necessary conditions of membership to a taxon and *b* is a taxon instance.

Following the terminology given in the domain model ontology, *a* consists of a particular combination of diagnostic characters. Additional characters may be included in the antecedent *a* to increase the certainty value associated to a rule. The conclusion is the association of a certainty value to the taxon's name (*b*).

Different combinations of characters for the same taxon may be provided in different rules. These rules are to be labeled with different certainty values depending on the discriminancy of each combination. The ideal scenario would be to find a combination of diagnostic characters that characterizes a taxon and, at the same time, is absent from other taxa. For example, it is quite easy to list some characters that would be found in an animal if it were a dolphin, but to make an unequivocal rule with the diagnostic characters for a dolphin is not so direct. However, a rule that mentions the characters *sea animal* and *mammal* would be significant for describing a dolphin although other animals also fit (e.g., whale). In domains where taxa are closely alike, e.g., marine invertebrates, such a combination of characters often is extremely difficult to find. So, several rules are needed to express various possible combinations.

On the other hand, to cope with intrataxon variability several representational facilities are provided by ***Milord II,*** e.g., disjunction of states for a given character or an interval of possible numeric values can be expressed in the premises of the rules.

Similarly, in the structural model, a module/submodules hierarchy (of modules following the schema Structure_model) represents the main structural dependencies between components and subcomponents of the specimens' body. Within each module, the object-level theory is concerned with one of these components. Each one of the module's elements may be assigned to a domain model concept as follows: predicates stand for characters, rules conclude characters that may be deduced from other observed characters, meta-predicate instances express dependence relations, causality and incompatibility between characters, deductive meta-rules link each dependent character to the values of previous characters it depends on and structural meta-rules prune the components and subcomponents of the modular hierarchy.

In Figure 7, we show how the components of a module are assigned a taxonomic concept. On the left, the concepts of a taxonomic model assigned to the module's components. On the right, the concepts of a structural model assigned to the module's components.

C. Classification Methods

1. *Refinement Method*

The implementation of this method in ***Milord II*** is as follows:

Module Refinement_method (DM : T; T: Classification_task) =
Begin
Module Structures
Export taxon
Control knowledge
Evaluation type: eager
Deductive Control:
M0001 **if** $K(=(\$z/\text{taxon}, \$f), \$c)$ **and**
$K(=(\text{taxon}, \$f), \text{int(false,true)})$
then conclude $K(=((\text{taxon}, \$f), \$c)$
Structural control:
M0001 **if** $K(=(DM/\text{taxon}, \$z), \text{int}(\$\text{min}, \$\text{max}))$ **and**
threshold (DM, \$cut) **and**
gt(\$min, \$cut) **and**
submodule(DM, $\$z$)
then Module(=(\$z,Refinement_method(DM/\$z,T)))
End control
End > T

which means the following. Examination of the specimen in question will deter-

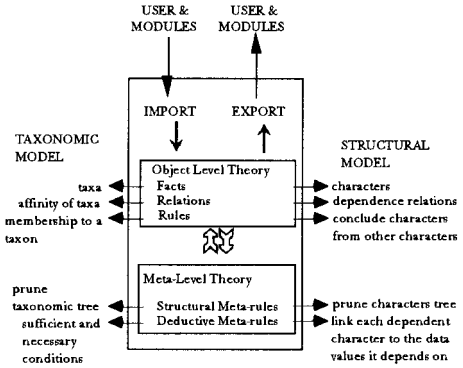

Figure 7. The domain model concepts assigned to the module's component.

mine if there is a fact `DM/taxon` the specimen can belong to with a degree of certainty defined by the interval `int($min, $max)`. If the minimum `$min` is greater than the truth threshold (predefined as a cut level of certainty in the domain model `DM, threshold(DM, $cut)` and a submodule `DM/$z` exists with the same name as the value of fact `DM/taxon`, then, that submodule will be the new domain model for a recursive call to the method, and the module resulting from the application of the method will be included as a new submodule in the next iteration by:

```
(Module(=($z, Refinement_method(DM/$z, T))))
```

The meta-rule conclusion embodies a recursive and dynamic operation of modules. It is recursive because the rule conclusion is a call to the same method that is being defined, over different arguments, and it is dynamic because which concrete submodule is to be considered as the new domain model depends on the current inference process. Dynamic modules are one of the new aspects of ***Milord II,*** specially introduced to cope with the requirements of taxonomic applications.

This method is to be applied until (1) the current domain model is a leaf module, or (2) there is no exported fact in DM with a certainty degree over the truth threshold. Given a specimen, the method compares the inferred certainty of membership to a taxon with the certainty threshold assigned to the type of taxon. So, the threshold tunes up the maximal uncertainty allowed to obtain a suitable behavior in the application in course.

Different additional constraints may be incorporated to this method to contract the result of the refinement method. For example, the next module is a contraction of the module Refinement_method after incorporating two constraints to it:

- consistence preservation: when a case problem belongs to a taxon at a taxonomic level with the value of sure, no other taxa of the same taxonomic level is possible (M0001 of the deductive knowledge).
- inheritance rule: rejection of old possible hypotheses in the higher path of the taxonomic tree (M0003 of the deductive knowledge). The taxon for which none of its subtaxa have been found possible is considered false in the result.

Module Ref_meth_inc(DM:T;T:Classification_task)< Refinement_method(DM,T)=
Begin
Module Structures
Export taxon
Control knowledge
Evaluation type: eager
Deductive Control:
M0001 **if** *K*(=($*z*/taxon ,$*f*), int(true,true)) **and**
K(=($*z*/taxon ,$*ff*), int($min, $max)) **and**
diff($*f*, $*ff*) **and**

```
                neq($min, true) and
                K(=($f/taxon ,$ff), int(false,true))
            then conclude K(=(taxon, $ff), int(false,false))
M0002       if  K(=($z/taxon,$f), $c) and
                K(=(taxon, $f), int(false,true))
            then conclude K(=(taxon, $f), $c)
M0003       if  submodule($father, $son) and
                set_of_instances(
                $exp,
                conj(K(=($son/taxon, $exp),int($min,$max)),
                neq($min, false),
                neq($max, true)),
                $notunknown) and
                cardinal($notunknown, 0)
            then conclude K(=(taxon, $son), int(false, false))
    Structural control:
M0001       if  K(=(DM/taxon ,$z), int($min, $max)) and
                threshold(DM, $cut) and
                gt($min, $cut) and
                submodule(DM, $z) and
                no(K(=(DM/taxon, $y), int(true, true))) and
                diff($z, $y)
            then Module(=($z,Refinement_method(DM/$z, T)))
    End control
End > T
```

The addition of any constraint has to be carefully considered because it may cause undesirable side effects. Let us analyze the two previously considered constraints. The consistence rule defined in M0001 is an obvious requirement: since taxa are exclusive classes a specimen cannot belong to two taxa.‡ Nevertheless, it prevents the system from detecting inconsistency of the input data or incompleteness of the knowledge base.

With reference to the second constraint, its goal is to ''delete'' the taxonomic hypotheses of the branches that have been pruned in the refinement process. As a side effect, it imposes that only complete identifications would be given by the system. In other words, a case whose data does not allow the identification at the leaf level of the implemented taxonomic tree would be declared unknown. This behavior is clearly contrary to the approach presented in this work, since we consider that it is better to provide the user with an identification at a

‡This would happen when a specimen was assigned a membership of *sure* to two taxa of the same taxonomic level, or a value of *sure* to a taxon and any other certainty (above the truth threshold) to another taxon at the same taxonomic level. However, in the Many-valued logic paradigm a specimen can be assigned a membership value of, e.g., *very possible* to two taxa at the same taxonomic level without contravening the basics of this approach.

high taxonomic level than to constrain the identification to a low-level taxon, e.g., species.

2. *Propose and Revise Method*

This method divides a classification task in two subtasks, `Asking` and `Con`. The former is responsible for asking for the hypothesis to be confirmed, the latter checks whether or not the hypothesis is correct. If it is correct then the refinement method is applied to the submodule `submodel` of `con`, otherwise the refinement method is applied to the whole domain model.

The implementation of this method in ***Milord II*** would be:

```
Module Propose_and_revise_method (DM:T; T: Classification_task) =
   Begin
      Module Asking = Askexport_method(DM, T)
      Module Con = Consistenceproving_method(DM, Asking)
      Module Structures
      Export taxon
      Control knowledge
      Deductive control:
   M001  if  K(=($z/taxon,$f), $c) and
             K(=(taxon, $f), int(false,true))
             then conclude K(=(taxon, $f), $c)
      Structural control:
   M001  if  submodule(con, submodel)
             then Module(=(Refine,Refinement_method(Con/submodel, T)))
   M002  if  no(submodule(con, submodel))
             then Module(=(Refine, Refinement_method(DM, T)))
      End control
End > T
```

The subtask `Asking` is solved by the method `Askexport` which asks for a hypothesis from the user:

```
Module Askexport_method(DM:T; T: Classification_task) =
   Begin
      Module Structures
      Import Taxon
      Export Taxon
      Deductive knowledge
         Dictionary:
            Types: taxon_type
            Predicates:
            Taxon =  Name: "Taxon proposed by the user"
                     Question: "What taxon do you suspect the
```

specimen you have at hand belong to?''
Type: taxon_type
End deductive
End > T

The second subtask, namely `Con`, is performed by the method `Consistency_proving`. This method opens the path of submodules from the root up to the module with the same name as the taxon hypothesis obtained by the previous module. If the values obtained in the higher taxa path are over the threshold, then it answers *yes* to the fact `ok`, which means that the check has been successful, and creates a submodule `submodel` by name, containing the subdomain model with the tested hypothesis as a root.

Module Consistence_proving_method(DM: T; T: Classification_task)=
Begin
Module Structures
Export ok, taxon
Deductive knowledge
Dictionary:
Predicates
ok = **Name:** ''Correct user hypothesis''
Type: Boolean
End Deductive
Control knowledge
Evaluation type: eager
Deductive control:
M0001 **if** K(=($z/taxon, $x), int($min, $max)) **and**
threshold($z, $cut) **and**
gt($min, $cut) **and**
K(=($T/taxon, $x), int(true, true))
then conclude K(ok, int(true, true))
Structural control:
M0001 **if** K(=($DM/taxon, $x), int($min, $max)) **and**
threshold($DM, $cut) **and**
gt($min, $cut) **and**
K(=($T/taxon, $tax), int(true, true)) **and**
path($DM/$z, $tax)
then Module(=($z,consistenceproving_method($DM/$z, T)))
M0002 **if** K(=($DM/taxon, $x), int($min, $max)) **and**
threshold($DM, $cut) **and**
gt($min, $cut) **and**
K(=($T/taxon, $x), int(true, true))
then Module(=(Submodel,$x))
M0003 **if** submodule($x, submodel)
then Module(=(Submodel, Drag($x, submodel)))

End control
End $> T$

VI. RELATED WORK

A. Knowledge Level

Several methodologies in knowledge engineering have emerged to solve the dissatisfaction of the construction of knowledge-based systems; heuristic classification,[2,22] generic tasks.[3,23] One of the most outstanding among these methodologies is KADS.[6,24] KADS is a methodology to model expertise. Its authors claim that the knowledge obtained through a knowledge acquisition process can be split into four types of knowledge: *Domain, Inference, Task,* and *Strategic.* Moreover, these four types of knowledge can be organized in a hierarchical way with a fixed operational relation among them. Basically, the inference, task, and strategic layers contain different types of control knowledge acting one on top of the other. Several languages have followed this architecture:[25,26] KARL, FORKADS, $K_{BS}SF$, among others.

Another approach to modeling expertise, and which has already been mentioned in the introduction of this article, is the *components of expertise.*[4] Its authors claim that a problem domain can be described from three perspectives: *Task, Model,* and *Method.* As shown throughout the article, the language ***Milord II*** is more naturally understood as an operationalization of the components of Expertise approach. Tasks are mapped into modules specifying a specification of the input/output relation, Methods are mapped into generic modules that map tasks into a task/subtask division, and domain models are mapped into completely specified modules. However, if we wanted to describe ***Milord II*** in terms of KADS layers we could analyze the components of a single module in terms of these layers. An application using ***Milord II*** will consist of a hierarchy of such modules.

Domain-Layer Primitives. The sublanguage of ***Milord II*** used to describe the domain layer is a many-valued propositional logic extended with set-comparison predicates for variables representing many-valued sorted concepts (the object level component of modules). The language allows for fuzzy and bayesian variables, containing fuzzyfication and defuzzyfication techniques and an algorithm for propagation in bayesian networks. Binary relations between variables can also be defined and used by other layers to take control decisions. The modular structure of the language gives the domain layer a hierarchical structure; as it does for control.

Inference-Layer Primitives. Inference actions are taken in ***Milord II*** by two means: a static parameter, local to each module, selecting the overall inference behavior: Lazy or Eager; and, a declarative theory, written in a type of first-order calculus (deductive meta-control) which determines the tuning of the results obtained by the standard inference mechanism (increasing or decreasing of evidence), and, eventually, implements a set of inference rules as meta-rules, taking profit of the reification/reflection mechanism between the components of a module, as it is done in QIL[25] for the case of first-order logic (in ***Milord II*** the object level language is simpler).

Task-Layer Primitives. ***Milord II*** contains a set of special meta-predicates (used in the structural meta-control component of a module) that permits to determine the control

over the subtasks (submodules of a particular module). This control is done through the elimination, addition or reordering of the submodules.

Strategic-Layer Primitives. The possibility, in ***Milord II,*** of dynamically applying generic modules to get new submodules, takes shape, again, through the use of special meta-predicates in the conclusion of meta-rules (as part of the structural metaknowledge) to decide which generic module apply to a particular module (containing a particular goal as output) in order to generate a new submodule. This component of the KADS methodological proposal is almost never discussed in the different implementations. Only MODEL_K[27] deals with it.

The Primitives Connecting the Layers. in a ***Milord II*** module, the different knowledge elements of the KADS layers are divided in two components: the object (Domain layer) and the meta (the rest of layers). The connection between these two is done through a reflection/reification mechanism that translates formulas of the object level language into terms of appropriate meta-level meta-predicates and vice-versa. The overall control flow between deduction at the object level, deduction at the meta-level, and action of the reification/reflection translation rules is determined by the evaluation type of each module (lazy or eager).

From this analysis, ***Milord II*** can be seen as an attempt to operationalize models of expertise. It also deals with formalization by means of generic constructs (generic modules), operations in the language (refinement), and by an external formalism based on a type of Dynamic Logic, as it is also the case of (ML).[24] However, Dynamic Logic is not used as a component of the language, as it is the case of KARL. ***Milord II*** is programmed in CLOS.

An important aspect of the languages helping in knowledge engineering is their approach to specification. MODEL-K[27] approaches the operationalization of KADS models of expertise by defining a couple of languages: one for the specification (to informally describe the expertise), and one for the implementation. So, the process of building a model consists on a two-step procedure, as in classical software engineering. In ***Milord II*** we provide a language that does not make explicit differences between specification and implementation. Modules go from the most abstract (least committed with a particular solution) to those completely defined, in an n-step procedure. At each step the programmer defines a new module that introduces (implements) particularities that are checked with respect to a previously defined module. Then it proceeds again by particularizing this newly defined module. This idea wants to reflect the usual exploratory and incremental programming used to model expertise.

Finally, the reuse of layers can be easily done in ***Milord II*** by abstracting a module as a generic module over a parameter that will represent, at application time, the abstracted part. Thus, we could abstract a complete module from the domain level, by eliminating it from the module and changing the references to object level components in the meta-knowledge to references to argument object level components.

B. Taxonomy

Computers have been used in taxonomic identification since the 1970s. Pankhurst[28] provides a review of progress to the late 1980s and Edwards and Morse[29] concentrate on progress made in the 1990s.

The first computerized identification tools appeared in the middle 70s as classic programs produced by taxonomists.[28] They were called on-line identification programs to emphasize that they allowed for an interactive process. These classic identification programs emulate any of the traditional identification procedures, either a dichotomous key, a multiple-entry key or a matching method. Many of these programs incorporate a best character selection command that assesses the ''goodness'' of characters. The best character selection procedure may be based on character weighing depending on the observation cost, or depending on the discrimination power of that character. Also, this procedure may be based on taxon weighing depending on the frequency of occurrence. Classic identification programs do not provide true uncertainty handling mechanisms. In fact, many classic identification programs cannot handle uncertainty or missing data at all. However, some classic identification programs permit the ''unknown'' value in the user answers. If so, they enter each of the branches where the unknown character is found, thus increasing the search time and losing efficiency.

The drawbacks of classic identification programs led to the investigation of alternative techniques for taxonomic identification in the middle 80s. These include AI techniques and knowledge-based systems in particular the work of Fortuner.[30] The few examples of knowledge-based systems devoted to taxonomy are reviewed by Edwards and Morse.[29] Some were developed using commercial shells.[21] Others are based on expert system shells specially addressed to biological domains.[31] Some others are expert systems developed for a particular domain.[32]

To our knowledge, there are no examples of more principled approaches to the knowledge level of identification in taxonomy. Virtually, no other attempts to build expert system architectures addressed to the taxonomic field have been proposed apart from the one presented in this article.

VII. CONCLUSIONS

We provide a reflexion on the KL notions of *task, domain model,* and *method.* Our goal has been to state how we understand each of these components as the basic elements of our knowledge level analysis approach. We have deployed a mapping of our notions of *task, domain model,* and *method* to the ***Milord II*** programming constructs. This mapping constitutes a methodological approach to ***Milord II*** KBS design and development. Thus, we have contributed to fill in the gap between the knowledge level and the code level.

We present the analysis of the classification/identification problem in systematics and taxonomy at the knowledge level. It reveals the common background knowledge concerning the *classification task,* the *taxonomic domain model* structure, and the *classification methods.*

Using the previous mapping, we have addressed the construction of an *architecture* for building taxonomic expert systems. This contributes to the accessibility of KBS technology in the field of taxonomy and systematics. Using the architecture, the development of a KBS dealing with taxonomy is reduced to filling in this framework with the particular knowledge of a given taxonomic

domain. In this architecture, the key to reusability is modularity, since modules implement knowledge with different roles separately.

Specifically, we take advantage of our KL notions to define (1) a *classification task* that consists in obtaining a value for the variable *taxon.* Except for the names of the taxa, it is a reusable module in the architecture, (2) a *taxonomic domain model* that embodies the specific knowledge of the particular taxonomic domain. The domain model structure, that is not linked to the studied taxa, is captured in the domain model skeleton. Knowledge concerning structural and taxonomic contents has to be inserted in this skeleton, and (3) some *classification methods* which are generic knowledge in the taxonomic domain. They are reusable modules in the architecture. We have implemented several methods to satisfy particular user's needs (*refinement method, propose, and revise method*) which are *exchangeable* over the same domain model. Methods are designed to infer the taxon membership at *all* the different levels of the taxonomy.

We are gratefully indebted to the anonymous reviewer for his comments that have improved this article. This research has been supported in part by a *Generalitat de Catalunya* fellowship (FI/91-193), the European TMR Number PL93-0186 *VIM,* and the Basic Research Action Number 6156 *DRUMS.*

References

1. A. Newell, ''The knowledge level,'' *Artif. Intell.,* **18,** 87–127 (1982).
2. W.J. Clancey, ''Heuristic classification,'' *Artif. Intell.,* **27**(3), 289–350 (1985).
3. B. Chandrasekaran, ''Design problem solving: A task analysis,'' *AI Mag.,* **11**(4): 59–71 (1990).
4. L. Steels, ''Components of expertise,'' *AI Mag.,* **11**(2), 29–49 (1990).
5. W. Van de Velde, ''Issues in KL modelling,'' in *Second Generation Expert Systems,* J.M. David, J.P. Krivine, and P. Simmons, Eds., Springer-Verlag, Berlin Heidelberg, 1993, pp. 211–231.
6. B.J. Wielinga, A.T. Schreiber, and J.A. Breuker, ''KADS: A modelling approach to knowledge engineering.'' *Knowledge Acquisition,* **4**(1), 5–53 (1992).
7. S. Geldof, L. Steels, and W. Van de Velde, ''COMMET for building knowledge systems,'' *CC-AI,* **10**(3), 195–218 (1993).
8. M. Domingo, ''Towards a knowledge level analysis of classification in biological domains,'' in *Qualitative Reasoning and Decision Technologies,* N. Piera-Carrete and M. Singh, Eds., CIMNE, Barcelona, 1993, pp. 535–544.
9. M. Domingo, ''Evaluating the expert system approach to biological identification through application to Porifera.'' in *Sponges in Time and Space,* R.W.M. van Soest, T.M.G. van Kempen, and J.C. Braekman, Eds., Amsterdam, 1994, pp. 75–82.
10. Ll. Godo, R. Lopez de Mantaras, C. Sierra, and A. Verdaguer, ''Managing linguistically expressed uncertainty in MILORD application to medical diagnosis,'' *Artif. Intel. Commun.* **1,** 14–31 (1988).
11. J. Puyol, ''Modularization, uncertainty, reflective control and deduction by specialization in MILORD II, A language for knowledge-based systems,'' Ph.D. Thesis, Universitat Autonoma Barcelona, April 1994.
12. C. Sierra, ''Milord: Arquitectura multi-nivell per a sistemes experts en classificacio,'' Ph.D. thesis, Facultat d'Informatica, Universitat Politecnica de Catalunya, 1989.
13. M. Belmonte, C. Sierra, and R. Lopez de Mantaras, ''RENOIR: An expert system using fuzzy logic for rheumatology diagnosis,'' *Int. J. Intell. Syst.,* **9,** 985–1000 (1994).
14. A. Verdaguer, A. Patak, J.J. Sancho, C. Sierra, and F. Sanz. ''Validation of the medical expert system PNEUMON-IA,'' Comput. Biomed. Res., **25,** 511–526 (1992).

15. J. Agustı́, F. Esteva, P. Garcı́a, L. Godo, R. López de Mántaras, and C. Sierra, ''Local multi-valued logics in modular expert systems,'' *J. Experiment. Theoret. Artif. Intell.* **6,** 303–321 (1994).
16. C. Sierra and L. Godo, ''Specifying simple scheduling tasks in a reflective and modular architecture,'' in J. Treur, and Th. Wetter, Eds., *Formal Specification of Complex Reasoning Systems,* Ellis Horwood, 1993, pp. 199–232.
17. M. Domingo, ''An expert system architecture for taxonomic domains. An application in Porifera: The Development of *Spongia,*'' Ph.D. Thesis, Universitat de Barcelona, 1995, 290 pp.
18. L. Godo, W. van der Hoek, J.J.Ch. Meyer, and C. Sierra, ''Many-valued epistemic states. Application to a reflective architecture: Milord II,'' in *Proceedings of the 5th International Conference on Information Processing and Management of Uncertainty,* IPMU'94, Paris, Vol. II, 1994, pp. 950–956.
19. R. Milner, M. Tofte, R. Harper, *The Definition of Standard ML,* MIT Press, 1990.
20. Ll., Godo and C. Sierra, ''Knowledge base refinement in Milord,'' *Proc. 14th IMACS World Congress,* Atlanta, 1994, pp. 255–260.
21. J.B. Wooley and N.D. Stone, ''Application of artificial intelligence to systematics: SYSTEX a prototype expert system for species identification,'' *Sist. Zool.,* **36**(3), 248–267 (1987).
22. W.J. Clancey, ''From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ORN final report 1979–1985,'' *AI Mag.,* **7**(3): 40–60 (1986).
23. B. Chandrasekaran, ''Towards a taxonomy of problem solving types,'' *AI Mag.,* **4**(1), 9–17 (1983).
24. H. Akkermans, F. van Harmelen, G. Screiber, and B. Wielinga, ''A formalization of knowledge-level models for knowledge acquisition,'' *Int. J. Intell. Syst.* **8**(2), (1993).
25. D. Fensel, F. van Harmelen, ''A comparison of languages which operationalize and formalize KADS models of expertise,'' *The Knowl. Eng. Rev.,* **9**(2), 105–146 (1994).
26. J. Treur and Th. Wetter, Eds., *Formal Specification of Complex Reasoning Systems,* Ellis Horwood, 1993.
27. W. Karbach, R. Voss, R. Schuckey, U. Drouven, ''MODEL-K: Prototyping at the knowledge level,'' in *Proceedings of Expert Systems and Their Applications, 11th Int. Workshop, Conference Tools, Techniques & Methods,* Avignon, 1991.
28. R.J. Pankhurst, *Practical Taxonomic Computing,* Cambridge University Press, Cambridge, 1991.
29. M. Edwards, D.R. Morse, and A.H. Fielding, ''Expert systems: Frames, rules or logic for species identification?'' *CABIOS,* **3,** 1–7 (1987).
30. R. Fortuner (Ed) *Advances in Computer Methods for Systematic Biology: Artificial Intelligence, Databases, Computer Vision,* The Johns Hopkins University Press, Baltimore, London, 1993, 560 pp.
31. N. Gautier, A. Pave, and F. Rechenmann, ''Object-centered representation and fish identification in Antarctica,'' in *Advances in Computer Methods for Systematic Biology: Artificial Intelligence, Databases, Computer Vision,* R. Fortuner, Ed., The Johns Hopkins University Press, Baltimore, London, 1993, pp. 181–195.
32. W.D. Atkinson and A. Gammerman, ''An application of expert systems technology to biological identification,'' *Taxon,* **36,** 705–714 (1987).

This is Blank Page 136