Milord II

Josep Puyol Gruart, Carles Sierra

Institut d'Investigació en Intel·ligència Artificial (*IIIA*)
Camí de Santa Bàrbara, 17300 Blanes (Girona)
e-mail: {puyol, sierra}@ceab.es

Abstract

Milord II is a programming language and a tool for the development of expert systems. **Milord II** has been designed and developed at the *IIIA* and it works as a testbed for the implementation of the theoretical developments of our group. Several applications from medicine to biology, pedagogy and pig farms are being developed. **Milord II** is a free prototype available for research purposes. This is a brief summary of the main concepts of **Milord II** in order to the understanding of a software demonstration.

1 Introduction

The main goal of **Milord II** is the programming of real world expert systems, that is, those dealing with real problems and programmed by real experts. Programming in the large and imperfect information treatment are one of the most important characteristics of **Milord II**.

2 A Modular Language

The decomposition of a whole problem into simpler parts is a good and natural programming methodology. **Milord II** is then a homogeneous language based on modules. A **Milord II** program is a hierarchy of modules.

Modules are the basic unit of programming in **Milord II**. All the other components of expert systems as facts or rules belong to modules. Every module can be considered as a specialist on a concrete domain, that is, every module is a complete expert system containing both the domain and control knowledge. In the following we will briefly describe the main components of modules using the simple example of Figure 1.

2.1 Interface

The interface of a module has two components: the *import* and the *export* interface. The export interface of a module is its output result. It represents the facts the module is able to deduce. For instance the module *Gram* exports a set of germs. The import interface of a module is the set of facts provided by the user (*Penicillin* in the module *Gram*).

Modules can also import information from other modules via their declaration as submodules. The module *Gram* has three submodules (*Respiratory Diagnosis*, *Type of Infection* and *Gram of Sputum*). It can import the fact *DCGP* exported by the module *Gram of Sputum* by means of the path *S/DCGP*[1].

```
Module Gram =
    Begin
    Module D= Respiratory_Diagnosis
    Module T= Type_of_Infection
    Module S= Gram_of_Sputum
    Import Penicillin
    Export Pneumococcus, Haemophilus, Staphylococcus, Enterobacteria
    Deductive knowledge
        Dictionary: ; not defined here.
        Rules:
        R001 If S/DCGP then conclude Pneumococcus is possible
        R002 If S/DCGP and D/Bact_Pneumonia
            then conclude Pneumococcus is very_possible
        R003 If S/BGN and D/Aspiration_Pn and T/Nosocomial
            then conclude Enterobacteria is quite_possible
        R004 If S/CBGN and Penicillin
            then conclude Haemophilus is sure
        Inference system:
        Truth values= (impossible, few_possible, sligh_possible, possible,
            quite_possible, very_possible, sure)
        Renaming
            D/False ==> impossible
            D/True ==> sure
            T/False ==> impossible
            T/True ==> sure
            ...
        Connectives:
            Conjunction =
                ((impossible impossible impossible impossible,
                    impossible impossible impossible)
                ...
                (impossible few_possible sligh_possible possible
                    quite_possible very_possible sure))
        end deductive
    Control knowledge
        Evaluation Type: Lazy
        ...
        end control
    end
```

Figure 1: Example of module declaration.

2.2 Deductive Knowledge

Milord II deals with imperfect knowledge, that is, uncertain, imprecise and vague knowledge. The use of a symbolic multi–valued logics allows us to give to the concepts a graduated truth–value. For instance, we can say that *fever is possible.*

The deductive knowledge is composed by the declaration of the dictionary, the rules and the inference system of the module.

[1] *S* is the local name of the submodule *Gram of Sputum.* Local names are defined by **Module** *local_name = module.*

Dictionary: It defines the fact identifiers and some of their attributes. The most important attribute of facts is their type. There are different types of facts: boolean, logic, enumerated and numeric facts[2]. The value of a logic fact is an interval of truth–values. In Figure 2 there is an example of a numeric fact declaration. Enumerated facts are fuzzy sets.

```
Dictionary:
     Predicates:
     ...
     Temperature =
          Name: "Patient's Temperature (in centigrade)"
          Question: "Which is the patient's Temperature?"
          Type: Numeric
          Relation: less-relevant-than AIDS
     ...
;; within the Temperature predicate definition the next
;; meta-predicate instance is present:
;; less-relevant-than(Temperature, AIDS)
```

Figure 2: Example of predicate declaration.

Rules: This component of modules represent the relational knowledge. For instance:

If temperature $> 39°$ **then conclude** fever **is** definite

Inference System: It is the declaration of the local logic of a module. It consists of the declaration of a total ordered set of linguistic terms and the connective operators used to combine and propagate the truth–values when making inference. In the example of Figure 1 we can see a declaration of a logic with seven terms and the conjunction operator.

The inference system declaration of a module also includes the mapping (translation) of the terms of the different logics of its submodules to the own logic[3].

In the example of Figure 1 the renaming declaration D/False ==> *impossible* means that the term *False* from the module *D* is translated to the local term *impossible*.

2.3 Control Knowledge

The current implementation of the meta-language allows the definition of meta–rules, and the type of module execution, which can be lazy or eager. *Lazy* means that facts are evaluated, at the object-level, only when needed, i.e. imported facts and exported facts of submodules are asked only if they may be useful to compute the export interface of the module. On the contrary an *eager* module execution obtains, first of all, values for the imported facts and for the

[2]For the sake of simplicity we avoid to explain *temporal* facts that allows to implement temporal reasoning and facts of type *array* used to implement bayesian reasoning.

[3]These translations of linguistic terms should preserve the deduction among modules with some criteria.

exported facts of the submodules and then the deductive knowledge is used. The interaction between the object level (rules) and the meta level (metarules), reification/reflection step, is made after every import information is obtained, after every submodule execution and after every time object level is extended by deduction.

An example of meta–rule:

M001 **If** **K**(X/$y,$v) then conclude **K**($y,$v)

Given any fact of the submodule *X (X/y)* with any value *v*, then we give to the fact *y* of the current module the same value *v*.

3 The Tool

Milord II is composed by two programs, the compiler and the interpreter. These programs have been implemented using C and Common Lisp respectively. Now we have only a Macintosh version because of the dependency of the window system to the machine.

The operation on **Milord II** interpreter can be summarized as follows:

1. The list of modules or a graphical representation is presented. The user can choose the starting point by selecting a module. Notice that it is not necessary to start on the root module of the application.

2. The list of the exported facts of the module is presented. The user can ask a module for the value of a fact or of a set of facts.

3. The module will generate questions to the user and to its submodules. The local control strategy of every module executed will determine which is the information needed to reach the goals.

4. Finally the answer or conditioned answers[4] to the initial questions are presented to the user.

4 Applications

We are developing several expert system applications in different domains.

Terap–IA: It is a medical application for pneumonia treatment. It is the natural extension of a previous expert system named *Pneumon–IA* for pneumonia diagnosis.

Ens–AI: It is an intelligent tutorial system directed to the diagnosis and orientation assistance in pedagogical processes.

[4]The inference engine of **Milord II** works by *specialization*. The conditioned answers are obtained by the specialization (partial deduction) of the object level of modules.

Spong–IA: It is an expert system for marine sponge classification.

Porc–IA: It is an expert system for the supervision of a pig farm. It is an example of the use of temporal reasoning in **Milord II**.

5 Future Work

We are working on an intelligent editor for **Milord II**. We are also working in the connection of **Milord II** with databases and its distribution on a Mac network.