Handbook of Fuzzy Computation

Editors in Chief

Enrique H Ruspini, Piero P Bonissone and Witold Pedrycz

Institute of Physics Publishing
Bristol and Philadelphia

F1.3 Control methods

Ramon Lopez de Mantaras and Carles Sierra

Abstract

Reasoning patterns occurring in complex tasks cannot be modeled only by means of a pure classical logic approach. This is due to several factors, for instance, incompleteness of the available information, need of using and representing uncertain or imprecise knowledge, combinatorial explosion of classical theorem proving when knowledge bases become large, or a lack of methodology in building complex and large knowledge bases (KBs).

To deal with these problems an advanced architecture for knowledge base systems (KBSs) must combine control, implicit and explicit, with modularization techniques, together with an approximate reasoning component based on many-valued logics. In this section we survey the main control techniques needed by means of a generic modular architecture. The implicit control is usually based on a subsumption mechanism together with a rule elimination process. The explicit control, declarative in nature, is, in advanced architectures, a metalevel approach, based on reflection techniques and equipped with a declarative backtracking mechanism. Reflection and subsumption techniques can be used to tackle the problem of incompleteness of the available knowledge. The metalevel approach permits assumptions based on the current state of the object deductive process, and the reflection technique makes them effective at the object level. If, later on, these assumptions are proved to be erroneous a declarative backtracking mechanism allows them to be retracted. Other complex reasoning tasks can be implemented using a combination of the overall set of control methods.

The paper starts with an introduction in which the problem of the control of inference is introduced in the context of a simple KBS, that is a system based on pure classical logic. Next, the main control methods for complex KBSs, using a generic KB architecture, are described and finally some conclusions are presented.

F1.3.1 Introduction: a brief reminder of the basic reasoning process in simple knowledge base systems

In this section we recall the basic reasoning process that takes place in simple knowledge-based systems, in other words systems that are based on pure classical logic and therefore do not deal with the incompleteness, uncertainty, and imprecision of the information.

A simple KBS contains an inference engine or interpreter that performs a simple reasoning process. The reasoning process is based on the *modus ponens* in the most common case where the general knowledge is expressed by means of 'If...Then...' rules. There are two basic reasoning strategies: one is data driven (also called forward inference) and the other goal driven (backward inference). Let us very briefly recall both of them.

F1.3.1.1 Backward inference strategy

The basic algorithm is as follows:

Start with a statement of one or more goals to be achieved
Select a goal
Repeat until goal is achieved:
Look for rules whose conclusion matches the goal
All matched rules become potentially applicable
One rule is selected
Its conditions are the new subgoals to be achieved

F1.3.1.2 Forward inference strategy

The basic algorithm is the following:

Start with a set of facts
Repeat until all applicable rules have been processed:
Match the facts in the set of facts against the conditions of the rules
Those rules whose conditions are fulfilled are potentially applicable
Select one of these rules
Apply the rule by adding its conclusions to the set of facts

F1.3.2 The control of inference problem

In spite of its simplicity, in backward inference neither the order in which the goals or subgoals are to be considered nor the rule to apply are specified. In forward inference the selection of the rule to apply is not specified either. This poses a serious decision problem because in many applications one can have KBs containing many rules and with many goals. There have been many solutions proposed in the literature. The simplest one consists in selecting the first rule among the applicable ones. Other solutions include selecting the rule using the most recently deduced fact, selecting the rule with most conditions and selecting the rule with most conclusions. However the best solution is using *metaknowledge*, i.e. knowledge about how to use knowledge. Such metaknowledge can be expressed by means of metarules. An example of a metarule is

If
the patient has risk factors, and
there are applicable rules mentioning pseudomonas, and
there are applicable rules mentioning klebsiellas
Then
prefer those rules related to the pseudomonas

In large complex KBSs addressing complex tasks, the problem of the control of inference is a crucial one and furthermore they face the additional problem of dealing with different types of inexact knowledge. In the following we will focus on control methods for such complex KBSs.

F1.3.3 Control methods for complex knowledge base systems

Reasoning patterns occurring in complex tasks cannot be modeled only by means of a pure classical logic approach. This is due to several factors, for instance, incompleteness of the available information, need of using and representing uncertain or imprecise knowledge, combinatorial explosion of classical theorem proving when KBs become large, or a lack of methodology in building complex and large KBs. To deal with these problems, an advanced architecture for KBSs, should combine modularization techniques with both implicit and explicit control mechanisms and with an approximate reasoning component.

Roughly speaking, in most advanced complex modular metalevel architectures such as BMS (Tan 1992), DESIRE (Treur 1992), MC-SYSTEMS (Giunchiglia and Serafini 1994), ML2 (Balder *et al* 1993), OMEGA (Maes and Nardi 1988), FOL (Weyhrauch 1980) or Milord II (Agusti *et al* 1994), a KB consists of a set of modules interconnected through their so-called export interfaces, that is, the set of facts that are

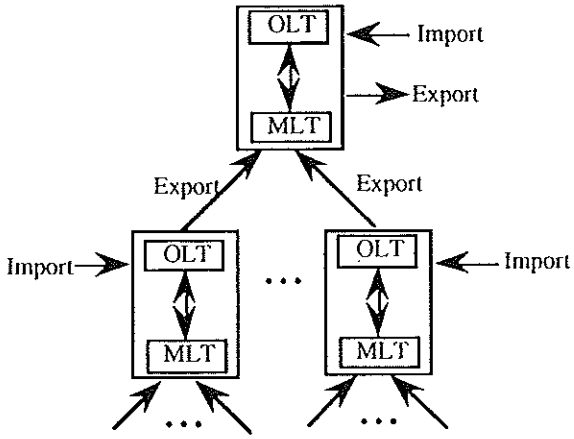

Figure F1.3.1. The structure of an advanced modular KB.

visible from other modules. Each module contains an object level theory (OLT) and a metalevel theory (MLT) interacting through a reflective mechanism (see figure F1.3.1).

A module can be understood as a functional abstraction, by fixing both the set of components it needs as input and the type of result it can produce. From the logical point of view, one can make use of both fuzzy logic and epistemic metapredicates to express the truth status of propositions. For further details on fuzzy logic, the reader is referred to Part B of this Handbook.

In this paper we will focus our attention on the control techniques that determine the global flow of a KBS execution. The explicit part of the control, declarative in nature, is generally based on a metalevel control approach together with the use of reflective techniques and a declarative backtracking mechanism. In this context, reflection makes sense as a control mechanism if there is a clear separation between domain (object level) and control (metalevel) knowledge. On the other hand, the basic implicit control components are a subsumption mechanism and a rule elimination process, both concerning the object level.

Next we list some of the usual control requirements for a complex KBS together with possible ways of fulfilling them.

F1.3.3.1 Locality of control

All the explicit control mechanisms are specified locally to each module. This allows us to identify a module as the complete description of a problem. The separation between domain and control knowledge is a mandatory characteristic of KBS languages to provide a clear and declarative programming style.

F1.3.3.2 Specificity versus generality

To solve problems, human experts are able to reason at different levels of precision depending on the number of data at hand. For instance, a physician cannot always obtain all the relevant data to make a complete and accurate diagnosis. This is the case, for example, when a patient is in a coma and thus the physician cannot ask him any questions. Nevertheless, the physician has to make a diagnosis despite the missing data. To represent these situations most advanced KB architectures provide the knowledge engineer with two different control options:

(i) to encode default-like rules (by means of metarules) that might generate plausible assumptions to be used in the reasoning flow in the case of missing relevant information or

(ii) to write rules with different levels of specificity (using more or less information, that is, putting more or fewer conditions) deducing the same conclusion with possibly different levels of certainty. To deal with this kind of rule a subsumption mechanism is often used.

Subsumption mechanism. When expressing the deductive knowledge of a module experts might write different rules concluding the same fact to represent the possibility of either having different unrelated sets of conditions entailing that fact, or having different sets of conditions related by an inclusion relation

that may allow conclusion of that fact (with different certainty values) depending on the completeness of the information at hand. Subsumption is the mechanism that ensures that only the most specific sets of conditions will be used. With this technique the KBS uses the more specific knowledge whenever possible. We will show a simple example to clarify this concept. Consider the following two rules from a KBS to advise on the treatment of pneumonia:

R051 If cotrinodazol then conclude cotri_low_dose is slightly_possible
R052 If cotrinodazol and seriousness then conclude no (cotri_low_dose) is definite.

These two rules conclude over the same fact ('cotri_low_dose'). It is easy to see that rule R052 is more specific than rule R051, in the sense that whenever we can apply the more specific rule, we could also apply the more general one. We say then that there is a subsumption relation between these rules. That is, rule R052 subsumes rule R051. In this case, when the fact 'seriousness' is unknown but the fact 'cotrinodazol' has a certainty degree, we want to apply the rule R051. However, in the presence of a patient with a serious illness, that is, when both facts are true to some extent, both rules can be applied, but the intended behavior is that only rule R052 has to be applied. The task of the subsumption control mechanism is to ensure that only rule R052 is actually fired, and therefore to prevent contradictory conclusions about the fact 'cotri_low_dose'. In this example, rule R051 together with the subsumption mechanism behaves as a default rule for the case when there is no knowledge about the seriousness of the illness.

F1.3.3.3 Avoidance of unnecessary work

It is possible to take advantage of a specialization deductive mechanism (Puyol *et al* 1998) to eagerly detect when a rule cannot improve the result of the current goal. When a rule is fired, it can be decided whether other rules concluding the same fact are able to improve the result. If they cannot, they are eliminated in order to avoid unnecessary work.

F1.3.3.4 Precision acceptance level

If the certainty values of facts are represented by intervals of truth values as is the case in most systems based on possibilistic logic, intervals may represent the imprecision we have on the certainty of a fact. The wider the interval, the lower precision we have. In some cases, experts are interested in programming modules whose results are only interesting when they are above a minimum truth level. This is done by declaring a threshold local to each module. Whenever a rule acquires by specialization a truth interval with its minimum value below the threshold, it is inhibited. This implies that the contribution of a rule to the truth interval of its conclusion is either a truth interval with the minimum value over the threshold or unknown, that is, the most imprecise interval. In some sense the expert considers as valuable information only those facts over the threshold.

F1.3.3.5 Data gathering

Given a query to a module, different strategies for the module to obtain an answer may be considered. How and in which order the necessary external information is obtained are essential points of the different evaluation strategies that a KBS should allow us to declare.

F1.3.3.6 Declarativity of control

Generally, metarules are used as a declarative language to implement several control actions, such as elimination of rules, generation of plausible assumptions, dynamic changing of the module hierarchy, or dynamic creation of modules.

F1.3.4 Control overview

A modular metalevel KB architecture consists of a set of interconnected modules, each containing possibly different kinds of knowledge and different logics, that is, different truth values, connective operators, and so on, structured as sketched in figure F1.3.2.

```
Begin
    Hierarchy of submodules
    Import: ...
    Export: ...
    Deductive knowledge
        Rules: ...
        Inference System: ...
            Truth Values: ...
            Connectives: ...
    end deductive
    Control knowledge
        Evaluation Type: ...
        Truth Threshold: ...
        Deductive Control: ...
        Structural Control: ...
    end control
end
```

Figure F1.3.2. Knowledge components of a module.

From a logical point of view, a module is composed of an OLT and an MLT. The OLT is composed by the set of rules of the deductive knowledge definition. These rules are formulas belonging to the object level language OL_n, for instance a propositional language based on fuzzy-valued semantics as in the case of Milord II. Formulas of this language are of the form (r, V), where r is a rule and V can be either a fuzzy set, a certainty value, a possibility interval, a probability value or interval, or a fuzzy truth value. An example, in the case of Milord II, is an interval of truth values belonging to a finite and totally ordered set of values, also specified in the module declaration. Deduction at the object level is mainly based on a specialization inference rule, a straightforward generalization of the many-valued version of *modus ponens*, which allows us to simplify rules as soon as we know truth intervals for any of their conditions. On the other hand, the MLT is composed by the set of metarules of the deductive control definition. These metarules are now formulas of the metalevel language ML, a restricted first-order classical language of Horn rules in the case of Milord II. Variables in metarules, if any, are considered always universally quantified. Deduction at the metalevel is based on *modus ponens*. Reasoning inside a module is then performed by reasoning at, and interacting between, both levels, resulting in a sequence of modifications of the initial OLT and MLT. For a deeper insight into this logical view of Milord II modules, the reader is referred to the articles by Sierra and Godo (1992) and Agusti *et al* (1994).

From an operational point of view, a module can be identified with a process used to compute values (truth intervals in the case of Milord II) for all the propositions contained in its export interface. Namely, a module execution consists of the reasoning process necessary to compute the values for the propositions in the export interface the user queries about. Possibly, the execution of a module can activate the execution of submodules in the hierarchy. These executions only interact with the parent module through the export interface of the submodules, giving back formulas as a result. It is worth noticing that the interaction is made only at the object level.

Conceptually the execution of a module involves two deductive subprocesses, object and metalevel, that act as coroutines, and three operations. Two of them, upwards reflection and downwards reflection, are resume operations between the coroutines that, besides acting as resuming operations, modify the knowledge used by the deductive subprocesses. The third operation is the communication with the user and/or other modules and has the effect of adding new formulas to the object level coroutine. Figure F1.3.3 shows the structure of a module and the relations between its components. Besides, the module evaluation type determines in which way subprocesses and operations are combined to obtain the global control behavior of a module execution.

Next we describe each one of the above-mentioned processes and operations.

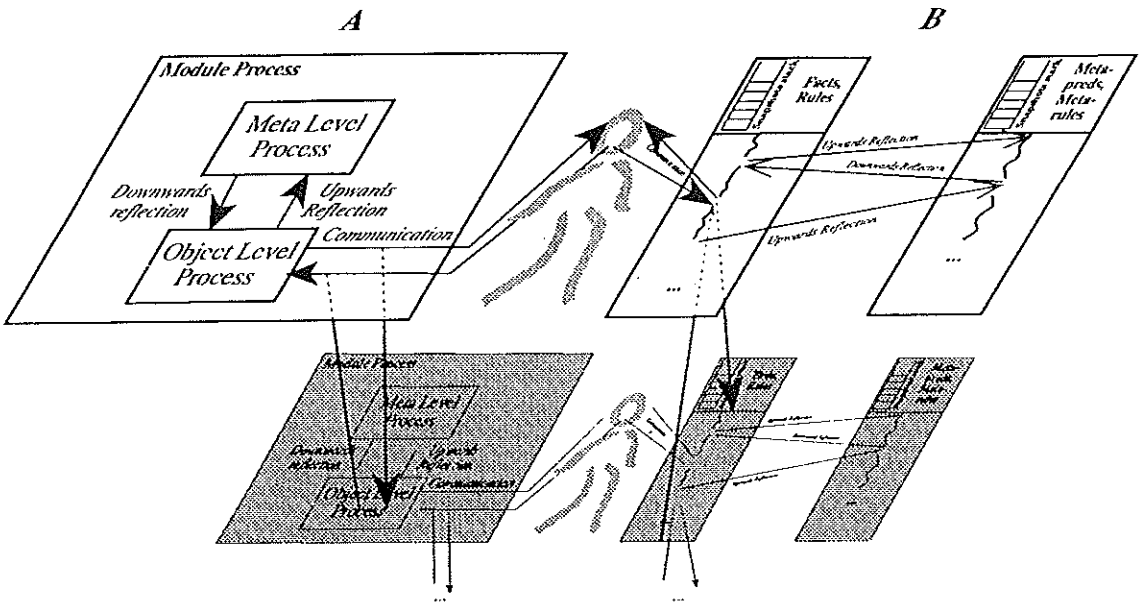

Figure F1.3.3. (*A*) The structure of the components of a module. (*B*) The coroutine view of a module.

F1.3.4.1 Object level process

This process uses as data the set of facts and rules of the module updated by the previous downwards reflection and communication operations. With these data and a goal to be solved, the task of the process is to obtain a value for the goal and potentially for other facts of the module. The obtention of a value for a fact can be achieved by one of the following methods:

(i) by using the communication operation, either by querying the user (when the fact is declared as 'Imported') or querying a submodule (when the fact contains a path);
(ii) by constraint propagation (out of the scope of this section);
(iii) by functional evaluation, in the case of facts with a function attached to them; or
(iv) by deduction when the fact is the conclusion of a rule. The process follows the rule specialization algorithm with two implicit control mechanisms, namely the subsumption and rule elimination, and a declarative one, the truth threshold rule elimination.

The type of evaluation strategy determines when the control is passed to the reification or to the communication operations.

F1.3.4.2 Reification

This operation translates a subset of the current object level formulas in the object process to metapredicate instances in the metalevel process. Once the operation concludes, the metalevel process is resumed.

F1.3.4.3 Metalevel process

The metalevel process takes as input the set of metarules of a module and the set of metapredicate instances generated by the reification operation, together with the metapredicate instances that had been previously deduced. The process then makes use of a forward inference engine with a depth-first control strategy, following the writing order of metarules. The stop condition is the impossibility of applying any new metarule. In that case, the process resumes the object level process through the reflection operation.

F1.3.4.4 Reflection

This operation is the dual of the reification one. It translates formulas from the metalevel process to the object level and executes the actions generated by the metalevel process. When the translation is finished, the object level process is resumed. Special mention has to be made when an instance of the special metapredicate 'Assume' is reflected down. The object level coroutine maintains a stack of snapshots. Every time an 'Assume' is reflected down a snapshot of the coroutine state is added to the

stack. Whenever a 'Resume' metapredicate is reflected down, the last snapshot is recovered and the computation resumed. This is how the declarative backtracking mechanism is implemented.

F1.3.4.5 Communication control mechanisms

The last issue considered in this section referring to control methods concerns the communication operation. The object level module process activates the communication operation to either query the user or query some of the submodules. When the operation queries the user, the result is the extension of the current OLT by a fact. However, the communication from a submodule to its present parent module is governed by a set of inference rules concerning the translation between the possibly different corresponding local logics of the modules, and the structural relations concerning the hierarchy. The following evaluation type strategies determine how this communication operation is applied.

Evaluation type strategies. The control mechanisms determine both the algorithmic behavior of the processes themselves and the way processes and operations are combined. The combination of the previous processes and operations is achieved by the explicit declaration of the evaluation strategy inside each module. In general there are three evaluation strategies, *lazy*, *eager*, and *reified*. Each one of them produces a different behavior. On the one hand, the lazy and eager evaluation types are opposite strategies about how to obtain external data (from the user and/or from its submodules). Namely, the lazy strategy always tries to use the minimum relevant information while the eager strategy makes use of as much information as possible. The reified evaluation type does not differ from the eager one in the method of obtaining data. The main difference of a reified strategy with respect to both lazy and eager strategies is that the inference mechanism of the object level is not used at all. Therefore, deduction is also performed in the metalevel process. The motivation behind this evaluation strategy is allowing the possibility of defining interpreters for object level formulas different from the default one based on specialization and fuzzy-valued logical semantics.

F1.3.5 Concluding remarks

We have discussed the main control methods used in advanced modular metalevel architectures. These architectures combine different control methods with reflection techniques to tackle the problem of large KBs with incomplete, imprecise, and uncertain knowledge. Although this section discusses the control methods independently of any particular architecture, occasionally, and for the sake of being more concrete, we have been forced to exemplify some of the issues with a particular architecture, Milord II, developed in our group. We expect to have transmitted to the reader the idea that the use of such control methods, together with an appropriate modularization, is absolutely necessary to develop KBSs able to deal with complex tasks involving complex reasoning.

References

Agusti J, Esteva F, Garcia P, Godo L, Lopez de Mantaras R and Sierra C 1994 Local multi-valued logics in modular expert systems *J. Exp. Theor. Artific. Intell.* **6** 303–21

Balder J, van Harmelen F and Aben M 1993 A KADS/ML2 model of a scheduling task *Formal Specifications of Complex Reasoning Systems* ed J Treur and Th Wetter (Ellis Horwood) pp 15–43

Giunchiglia F and Serafini L 1994 Multilanguage hierarchical logics (or: How we can do without modal logics) *Artific. Intell.* **65** 29–70

Maes P and Nardi N (eds) 1988 *Meta-level Architectures and Reflection* (Amsterdam: North-Holland)

Puyol J, Godo L and Sierra C 1998 Specialisation calculus and communication *Int. J. Approx. Reasoning* at press

Sierra C and Godo L 1992 Modularity, uncertainty and reflection in Milord II *Proc. 1992 IEEE Int. Conf. on Systems, Man and Cybernetics* pp 255–60

Tan Y H 1992 Non-monotonic reasoning: logical architecture and philosophical applications *PhD Dissertation* Vrije University, Amsterdam

Treur J 1992 On the use of reflection principles in modelling complex reasoning *Int. J. Intell. Syst.* **6** 277–94

Weyhrauch R 1980 Prolegomena to a theory of mechanized formal reasoning *Artific. Intell.* **13** 133–70