**Design and implementation of a reasoning engine
using generalized meta-rules
to express control knowledge**

F.J. Garijo
F. Ribas
K. Urzelai

Report LSI-88-26

**Design and implementation of a reasoning engine
using generalized meta-rules to express control knowledge**

**Francisco J. Garijo
Francesc Ribas
Karmelo Urzelai**

Sección de Inteligencia Artificial
Dept. De Lenguajes y Sistemas Informáticos
Universitat Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain

Abstract:

This paper describes the design and implementation of the reasoning engine developed for the interpretation of FLORIAN rule language. A key feature of the language is to allow the specification of control knowledge using generalized meta-rules. The user can define how to solve conflicts at the object-level, at the meta-level or at any higher level using meta-i- rules. Object-level rules and generalized meta-i-rules share the same rule format. Several examples of meta-rules and higher level rules are presented using the rule syntax. The architecture and working of the rule interpreter is analysed describing the main algorithms and abstract data types implementing the reasoning engine.

I - Introduction

FLORIAN rule language has been designed and implemented in the framework of FLORIAN project. The overall design guidelines were the following:

Modularity: Language constructs are provided allowing the user structuring concepts and actions.

Readability: Syntactic primitives include keywords facilitating the expression of sentences close to natural language.

Reliability : Object declaration and typing mechanism allowing static detection of errors before execution .

Portability: The compiler generates code for a virtual inference machine easily portable to another systems.

A key feature of the language is to allow the specification of problem solving strategies using meta-rules. In most of the existing rule languages e.g. OPS-5 [Brownston et al. 85], KEE [KEE 85] ART [ART 87] , Knowledge Craft

[Knowledge Craft 86] ,control strategies are predefined within the rule interpreter, they are expressed as a fixed number of rule selection criteria that should be defined by the user at the beginning of the inference process, for example LEX and MEA in OPS-5. Hybrid systems like KEE offers to the user a wider range of control criteria : Rule sets can be interpreted either backward or forward . When backward mode is specified the user can define one strategy for goal selection - deep first, breath first or best first - , in both modes it is possible to select one of the predefined rule selection criteria - least premise complexity, greatest premise complexity , weighted least premise complexity, etc - . The previous approaches reduces control knowledge to the selection of a fixed combination of alternatives within an enumerated set of control choices. The justification of this limited view of control problem is based on efficiency considerations in the implementation of rule interpreters, but in practice the knowledge engineer is constrained to express complex control behavior as low level constructs such as control attributes added to domain objects.

Solutions to the control problem in rule base systems are based on meta-rules [Davis 80], [Davis&Lenat 82], [Beetz87], [Jensen et al 87]. As stated in [DAVIS 80] expressing control knowledge through meta-rules offers a number of advantages:

Flexibility. Strategy knowledge is declarative, its application depends on the state of the problem solving process.

Introspection. The system is able of making decisions reasoning about the content of its knowledge.

Uniformity . Meta-rules and object-level rules are defined using the same formalism and could be interpreted by the same inference engine.

The use of Meta-rules involves the introduction of an introspective phase in the traditional machine inference cycle - form the conflict set, select a rule and execute the selected rule - . Knowledge encoded in meta-rules is employed to select the rule from the conflict set. The selection process implies changing the level of abstraction from domain objects to conflict set, and reasoning about the rule structure and content. So an elegant implementation of that introspective phase could be made using the same inference engine employed at the object level . The rule selection step is then performed starting a new introspective cycle using the conflict set as the working memory , object-level rules as domain objects and meta-rules as object-level rules . Most of the existing systems e.g. TEIRESIAS [Davis&Lenat 82], ESB [Jensen et al 87] are based on two levels of abstraction object level and meta-level. This approach only works if control knowledge encoded in meta-rules allow the selection of an object-level rule in every problem solving cycle; unfortunately this is not realistic when meta-rules are user defined. In practice at the meta-level we can find control conflicts similar to that existing at the rule level :

a - Many meta-rules can be applied to the same situation. There is a conflict set at the meta-level.

b - The interpretation of the meta-rules existing in the conflict set are incomplete in the sense that more than one rule can be selected.

To deal with the previous problems the concept of meta-rule has been generalized in FLORIAN, providing high level meta-rules. The user can define how to solve conflicts at the meta-level using meta-meta-rules. In general meta-i-rules can be defined expressing conflict resolution at lower level. When user meta-i- rules do not allow the selection

of a unic object-level rule, the system utilizes predefined meta-rules to make a choice . The user can also indicate to the inference machine which of those default meta-rules should be used to complete their meta-i-rules.

High level meta-rules and object-level rules share the same formalism and are interpreted by the same inference machine. The interpretation of meta-i- rules is performed by the rule interpreter changing the level of abstraction considering as objects the rules at lower meta-level and meta-i- rules as inference operators.

In the next section we briefly present the characteristics of FLORIAN rule language focussing on the expression of control knowledge. The architecture and working of the rule interpreter are described on section three showing trough examples the process of rule and meta-rule interpretation. Implementation and performance aspects are discussed in the conclusion.

II - Expressing control guidelines

FLORIAN rule language belong to the family of production rule languages based on frames. Rules are made up of conditions expressing situations to be verified and actions to be executed when conditions are satisfied. Situations and actions refers to related objects belonging to a global data base called Working Memory (W.M) . The goal of the production system or rule interpreter is to determine which production rules are satisfied given the current state of the W.M and perform the corresponding actions. The main aspects differentiating existing production languages are the formalism used to define objects on the W.M , the syntax for expressing situations and actions and the semantic used by the rule interpreter. Object definition in FLORIAN is based on the frame formalism [Brachman & Schmolze 85], [Fikes&Kehler 85]; rule syntax has been designed aiming to help users express situations and actions in a structured and readable way. Features of rule formalism include: Typing mechanism for variable in rules, quantification operators, pattern matching operators, definition of constraints associates to patterns using logical expressions. A detailed definition of the rule syntax can be found in [Garijo et al. 88]. Examples of the formalism to defining control knowledge follows.

II.1.Control rules

The most important feature of FLORIAN rule interpreter is its capability of dealing with control knowledge expressed using Meta-i-rules. The syntax of Meta-i-rules is the same as ordinary rules. Rules and Meta-i-rules share the same syntax and are interpreted in the same way. They only differs in the category of objects satisfying their premises; while ordinary rules refers to user objects belonging to the problem domain, Meta-i-rules condition's refers to predefined object such that conflict set , ordinary rule condition and/or actions. Actions in Meta-i-rules refers to the conflict set while actions in ordinary rules refers to W.M. .

Predefined object allowing the definition of Meta-i-rules are the following :

CONFLICT SET. Is defined as a tuple of classes of particular conflict set corresponding to each class of meta-i-rules

(TUPLE Conflict-Set) { The values are of type Conflict-Set-Meta-Class}

Conflict-Set-Meta-Class represents the class of Conflict Set corresponding to each class of meta-i-rules . Their values are tuples (i-rule,j-context,i-action), where i-rule is a rule which conditions are satisfied by the objects belonging to context j-context, and i-action representing the action to be performed if the rule is fired. This tuples are represented by the schema Satisfied-Rule.

(Schema Conflict-Set-Meta-Class

(Set-of-Satisfied-Rules :< Satisfied-Rule>) { The values are of type Satisfied-Rule }

(Level :< integer >) { Type is integer }

)

The schema Satisfied-Rule represents the information about rules whose left hand side conditions are satisfied. The schema is made up of the following slots:

- Name to identify the rule que la identifica.
- Weight to express priority criteria
- Application context containing the collection of objects satisfying the rule
- Actions to be performed when the rule would be fired

(Schema Satisfied-Rule

(Name :< identifier >) { Type is identifier }

(Weight:< integer >) { Type is integer }

(Application-Context :< collection of objects >) { Type is collection of objects }

(Actions:< action >) { The values are of type Action }

)

Actions are represented by the name of the action and their parameters .

(Schema Action

(Parameters :< collection of objects >)

(Operation :< identifier >)

)

II.2 - Action Primitives to express Meta-Rules

Meta-Rules are used to express control knowledge about how to select ordinary rules or other Meta-Rules. The rule language provides the user with a set additional action primitives to be used with Meta-Rules. Their definition is as follows:

-Select : Set-of-Satisfied-Rules x Conflict-Set-Meta-Class -> Conflict-Set-Meta-Class

input : A class of satisfied rules and a selection criteria defined by a logical expression

output : A subset of the input class of Set-of-Satisfied-Rules containing the rules satisfying the selection criteria.

Example: Select rules with Weight >10

If there are satisfied rules with Weight >10 => select them

```
( Rule select-weight-bigger-than-10
  (exist-set $?selected-rules = [ ?rule /
    ( schema Conflict-Set-Meta-Class
      (Level 0)
      (Set-of-Satisfied-Rules $? ?rule $?
        & ?rule:(schema Satisfied-Rule
          (Weight ?weight & (?weight > 10)))
      )
    )
  )
  =>
  Select($?selected-rules)
)
```

-Order : Set-of-Satisfied-Rules x Ordering-Criteria x Modality x Conflict-Set-Meta-Class -> Conflict-Set-Meta-Class

input : The set of satisfied rules to be ordered

output : The Set-of-Satisfied-Rules ordered according to the ordering criteria and modality

The pre-defined ordering criteria are the following:

- Weight : Satisfied rules will be ordered according to their weight.
- Application-Context- Cardinality : the ordering criteria is the number of objects contained in the application context.
- Premise-Complexity : the ordering criteria is the number of operators contained in the left hand side of the rule
- Conclusion-Complexity : the ordering criteria is the number of actions contained in the right hand side of the rule
- Recency : the ordering criteria is the inference step when the objects in the application context has been added to the working memory

The ordering Modality may be:

- Ascending.
- Descending.

Example : Order the rules according with its Weights

If there are rules (Meta-level = 0) => Order them in a descending way according with their Weight

```
(Rule order-rules-by-weight
  (schema Conflict-Set-Meta-Class
    (Level 0)
    (Set-of-Satisfied-Rules $?rules ))
  )
=>
Order( $?rules, Weight, Descending)
)
```

-Delete : Satisfied-Rule x Conflict-Set-Meta-Class -> Conflict-Set-Meta-Class

input : A class of satisfied rules and a deletion criteria defined by a logical expression

output : A subset of the input class of Set-of-Satisfied-Rules where the rules satisfying the deletion criteria has been deleted.

Example : Delete rules with Weight 0

If there are satisfied rules with Weight 0 => delete them

```
(Rule Delete-rules-Weight-0
  (Schema Conflict-Set-Meta-Class
    (Level 0)
    (Set-of-Satisfied-Rules $? ?rule $?
      & ?rule:(schema Satisfied-Rule
        (Weight ?weight & (?weight = 0)))
    )
  )
=> Delete(?rule) )
```

II.3 - Examples of use of Meta-rules

Example 1 : Delete rules creating objects

Delete Ordinary Rules creating objects, that is containing as right hand side the action create (object)

```
(Rule Delete-creater-rules
  (Schema Conflict-Set-Meta-Class
    (Level 0)
    ( Set-of-Satisfied-Rules $? ?rule $? ))
  ?rule:(Schema Satisfied-Rule
    (exists (Actions $? ?modif $?
      & ?modif:(schema Action
        (Operation create)))
    )
  )
=> Delete ( ?rule) )
```

Example 2 : Breath first Strategy

For every rule => order all the rules in descending ordering using the time tag criteria

```
( Rule Breath-first-strategy
  (schema Conflict-Set-Meta-Class
    (Level 0)
    (Set-of-Satisfied-Rules $?rule ))
  )
=> Order( $?rules, Recency, Descending))
```

example 2 : Deep first strategy

For every rule => order all the rules in ascending ordering using the time tag criteria

```
( Rule Deep-first-strategy
  (schema Conflict-Set-Meta-Class
    (Level 0)
    (Set-of-Satisfied-Rules $?rule ))
  )
=> Order( $?rules, Recency, Descending) )
```

Exemple 4 : Meta-meta-rule . Execute first a Meta-Rule selecting rules

If it does exist a Meta-Rule selecting rules => select that Meta-Rule for firing

```
( Rule First-selecting-rules
  (schema Conflict-Set-Meta-Class
    (Level 1)
    (Set-of-Satisfied-Rules $?rs1 ?rule $?rs2 ))
  ?rule:(schema Satisfied-Rule
    (exists (Actions $? ?Action $?
      & ?Action:(schema Action
        (Operation Select))) ) )
  )
=> Select( ?rule $?rs1 $?rs2))
```

Exemple 5 : Meta-meta-meta-rule .Ordering Meta-Meta-rules according to the number of objects contained in their application context

For all Meta-Meta-rule => order them in a ascending order according with the cardinality of their application context

```
( Rule Order-by-Cardinality
  (schema Conflict-Set-Meta-Class
    (Level 2)
    (Set-of-Satisfied-Rules $? rules-to-order ))
  )
=> Order( $?rules-to-order, Application-Context-Cardinality, Ascending ) )
```


III - The Rule Language Translator

The rule language translator performs two major tasks:

Analysis : Verifying the syntactic-semantic correction of the collection of the rules defined by the user.

Synthesis: Transforming the information contained in rules into code executable by an abstract inference machine.

The computational model of the inference machine is based on the production system architecture. The basic cycle is made up of the following steps:

- 1- Form the conflict set interpreting left hand side of the rules on the working memory.
- 2- Resolve the conflict selecting a rule.
- 3- Execute the selected rule.

The main feature of FLORIAN's inference machine is the way of performing step 2 - selecting a rule from the conflict set -. Conflict resolution is directed by the meta-level knowledge contained in meta-rules. The selection process is achieved by starting a new inference cycle:

- a) To find out the choice criteria (meta-rules) matching the state of the conflict set
- b) To select the set of criteria allowing the selection of a rule, and
- c) To execute those criteria.

This extra-cycle is implemented changing the level of abstraction of the inference machine: the conflict set is used as working memory, object-level rules as domain objects, and meta-rules as object-level rules. Conflicts may arise in step b), to solve them if meta-meta-rules exist, they are interpreted in the same way as meta-rules. In general all existing classes of meta-i-rules are interpreted uniformly, starting a new selection cycle. When conflicts arise at meta-i-level and there are not user defined meta-i+1 rules applicable to solve that conflicts, general default meta-i+1 rules are used. These rules express general choice criteria, as for example, choose the first rule in the collection, choose the rule having minimum weight, least premise complexity, etc. A detailed description of the component parts and working of the inference machine follows.

III.1 - Architecture of the reasoning engine

The inference machine is made up of the following elements:

Working Memory (W.M.): On each problem solving cycle the W.M represents the state of all the solution elements defined by the user to solve his problem. The W.M. could be initially set up of a structured collection of objects -schema, tuples or constants- or it could be empty. Rules are interpreted as operators generating, modifying, or deleting objects in the W.M.

Rule Memory : It records user defined rules. The rule translator transforms the original rule set into a structured collection of rule classes according to the "level" of each rule. Object-level rules are on level =0, meta-rules on level=1, meta-meta-rules on level =2, etc. . Code generated for each rule have two main parts: Left hand side rule code performs the verification of the conditions stated in the left hand side of the rule. It also recovers from the W.M. all the application contexts satisfying rule conditions. Code corresponding to the right hand side of the rule express the actions to be made on the W.M. when the rule would be executed. High level meta-rules are interpreted in the same way as object-level rules, the only difference is that application conditions refers to the Conflict Set instead of on the W.M. .

Conflict Set (C.S.) : For each inference cycle, the C.S. represents the set of rules whose left hand side condition patterns matching the elements existing in the present state of the inference machine. C.S. is structured in classes according to the level of each rule. Object-level rules are on level =0, meta-rules on level=1, meta-meta-rules on level =2, etc. . Each class contains a collection of pairs (i_rule, i_context), where i_rule represents a rule instance, and i_context, represents a collection of objects matching the left hand side of the rule. Contexts in pairs belonging to object-level class are formed by W.M. elements, while contexts in higher level classes are formed by C.S. elements .

III.2 Design and Implementation of the Reasoning Engine

Major component elements of the Inference Machine - Working Memory, Rule Memory and Conflict Set - have been defined as Abstract Data Types. Abstract algorithms corresponding to the main operations are the following:

BASIC-CYCLE (Rule-Set, Working-Memory)

The operation implements the machine's inference cycle. The process starts using an initial -and possibly empty - Working Memory and a Rule Memory containing code generated by the rule translator. Inference cycle halts when the following conditions hold:

- a- Conflict Set is empty . There is no rules matching the existing W.M
- b- Halt condition stated by the user is satisfied. The selected rule has an explicit halt operation - Stop - in its right hand side part.

**Procedure BASIC-CYCLE (var Rule-Set : Rule-Set-Type,
var Working-Memory : Working-Memory-Type);**

INPUT : The set of rules defined by the user. This set is structured in classes according to the level of each rule. Object-level rules are on level =0, meta-rules on level=1, meta-meta-rules on level =2, etc.

OUTPUT: The Working Memory obtained executing selected rules.

```
Vars Conflict-Set : Conflict-Set-Type;
Rule-Application-Context : Rule-Application-Context-Type; { Rule and context instances }
Begin
{ Evaluating user defined rules using the W.M. }
Conflict-Set <- Evaluate-Rule-Set ( Working-Memory, Rule-Set )

{while there are rules which left hand side is satisfied by objects in the W.M and the explicit STOP
condition is false }

While Not (Empty? (Conflict-Set )) & Not (Stop-Condition ( Working-Memory )) do
{ Select one rule and an application context }

Rule-Application-Context <- Select-Rule-to-Execute (Conflict-Set)
{ Execute the selected rule using the context }

Execute-Rule ( Rule-Application-Context, Working-Memory ) { Working-Memory <- }
{ Evaluate rules continuing process until no satisfied rules exist or stop condition holds }
Conflict-Set <- Evaluate-Rule-Set ( Working-Memory, Rule-Set )

Endwhile

End
```

EVALUATE-RULE-SET (Working-Memory , Ordered-Rule-Set)

The goal of **EVALUATE-RULE-SET** is the construction of the Conflict Set. C.S. is made up of a finite collection of pairs (rule, context), where rule is a rule which L.H.S is satisfied by the set of objects in context . Contexts for each rule can be obtained running the code generated by the compiler. Object-level rules are evaluated first giving rise to level zero conflict set class; next meta-rules are interpreted creating level one conflict set class. The process continues interpreting high level meta-rules until one empty conflict set class is found or all meta-i-rules have been interpreted.

Function **EVALUATE-RULE-SET** (Working-Memory : Working-Memory-Type,
Ordered-Rule-Set : Rule-Set-Type) : Conflict-Set-Type

INPUT : The set of rules defined by the user.

OUTPUT: The Conflict-Set formed by tuples (rule, context) .

PRECONDITIONS: The Rule Set and the Conflict Set are structured in classes according to the level of each rule. Object-level rules are on level =0, meta-rules on level=1, meta-meta-rules on level =2, etc.

Vars Conflict-Set : Conflict-Set-Type;

Rule : Rule-Type;

Application-Context : Application-Context-Type; { Context instances of each rule }

Application-Context-List : Application-Context-List-Type;

Begin

Conflict-Set <- Create-Conflict-Set

While Not (Empty-rule-set? (Ordered-Rule-Set)) **do**

{ Ordinary rules - on level 0 - are interpreted first, then meta-rules, meta-meta-rules and so on }

Rule <- Get-first-rule (Ordered-Rule-Set)

{ All the contexts belonging to W.M. or C.S where there is a rule or a Meta_Rule that could be applied , are obtained }

Application-Context-List<- Get-Application-Context-List(Rule, Working-Memory)

While Not (Empty-Application-Context-List? (Application-Context-List)) **do**

{ Tuples (rule, context) are added to the Conflict-Set }

Application-Context <- Get-first-Application-Context (Application-Context-List)

Add (Rule , Application-Context , Conflict-Set) { Conflict-Set <- }

Delete-First-Application-Context(Application-Context-List) { Applic.-Context-List<- }

Endwhile

{Next rule is obtained }

Delete-Fisrt-rule (Ordered-Rule-Set)

Endwhile

{ Conflict-Set is structured according to the level of each rule }

EVALUATE-RULE-SET <- Conflict-Set

End

SELECT-RULE-TO-EXECUTE (Conflict-Set)

The operation **SELECT-RULE-TO-EXECUTE** receives as input the Conflict Set constructed by the operation **EVALUATE-RULE-SET**. C.S. is structured in classes according to the level of each rule. Object-level rules are on level = 0, meta-rules on level = 1, meta-meta-rules on level = 2, etc. Each class is made up of a collection of pairs (rule, context) representing rules and their application context. The application context corresponding to ordinary rules in level 0 refers to the W.M, while application context corresponding to high level meta_i-rules is made up of objects belonging to the meta_{i-1} class of the C.S. That is, the level =0 class represents all possible operators that can be applied to the W.M. in all possible contexts, while the level =i class represents all the choice criteria that can be used for selecting the criteria contained in level = i-1.

The goal of the operation is the selection of a pair (rule, context) belonging to class =0 representing the rule to be executed and their application context. The selection process is performed by interpreting the choice criteria represented as (rule, context) pairs belonging to level =1 and higher classes. First the higher meta-i rule class is interpreted, let be i their class level, obtaining the criteria to be used on class i - 1. These criteria will be used on class i - 2 obtaining the criteria to be used on class i - 3, etc. . It may occur that user defined criteria stated through meta-i-rules are not sufficient for the selection of the rule to be executed, in that case predefined meta-rules complementing user's meta-i-rules are used to perform the selection.

The detailed working of the algorithm follows:

The variable **Selector-MetaRule-Set** represents the meta_i class, that is, the set of criteria being used to select the rules of level meta_{i-1}; **Selected-MetaRule-Set** represents meta_{i-1} class, it contains the set of pairs (meta-rule, context) representing criteria to be selected. The invariant of the most external loop is the following :

Actual-Metalevel >0

Selector-MetaRule-Set represents a subset of the meta_i-rules class where i = Actual-Metalevel

Selected-MetaRule-Set represents a subset of the meta_{i-1}-rules class

The termination condition is true when Actual-Metalevel = 0 and when that occur :

Selector-MetaRule-Set = Selected-MetaRule-Set = a subset of the class of rules

The goal of the nested while loop is the application of the rules contained in Selector-MetaRule-Set upon Selected-MetaRule-Set. This will be possible if the following conditions hold:

Selector-MetaRule-Set is not empty, that is, **Cardinality** (Selector-MetaRule-Set) > 0)

There are rules to be selected, that is, **Cardinality** (Selected-MetaRule-Set) > 1

The termination condition for the while loop holds when one of the previous conditions is false.

On every iteration step one meta-rule belonging to Selector-MetaRule-Set is selected and it is applied upon Selected-MetaRule-Set. As a result a subset of the previous Selected-MetaRule-Set is obtained. The changes produced into the C.S. by the execution of a meta-rule may make false the application condition of existing meta-rules, in consequence, the application conditions of every meta-rule must be checked before execution.

Function SELECT-RULE-TO-EXECUTE (Conflict-Set : Conflict-Set-Type):

Rule-Application-Context-Type;

INPUT : Conflict-Set

OUTPUT: The pair rule-context obtained from the Conflict-Set by the interpretation of Meta-Rules

PRECONDITIONS: C.S. is structured in classes according to the level of each rule. Object-level rules are on level =0, meta-rules on level=1, meta-meta-rules on level =2, etc..

Const Basic-Metalevel = 1 { Level of Meta-Rules }

Vars Actual-Metalevel : Integer;

Selector-MetaRule-Set , Selected-MetaRule-Set : Conflict-Set-Type;

{ Selector-MetaRule-Set , Selected-MetaRule-Set are subsets of Conflict-Set }

Rule-Application-Context : Rule-Application-Context-Type; { Rule and context instances }

Begin

{ Obtaining the level of the highest level class belonging to Conflict-Set }

Actual-Metalevel <- Highest-Metalevel (Conflict-Set) {Top-Down selection}

{ Obtaining subset of Rules of the highest level class belonging to Conflict-Set }

Selector-MetaRule-Set <- Meta-Subset (Conflict-Set, Actual-Metalevel)

{ Selector-MetaRule-Set represents the set of choice criteria to be used for the selection of the rules belonging to the next meta - level.

Selected-MetaRule-Set contains the rules to be selected. The process starts by applying the criteria (Meta-rules) of level = i for selecting the Meta-rules belonging to i-1 class. As a result a set of $Meta^{i-1}$ -Rules smaller or equal to the original set will be obtained . The set of $Meta^{i-1}$ -Rules will be applied upon the i-2 class and the resulting set upon the next class; the process continues until the set of Meta-Rules (level = 1) where the rule to be executed will be selected. }

```

{ While there are Meta-Rules that can be applied to the next low level Set of Rules }
While Actual-Metalevel  $\geq$  Basic-Metalevel do
    { Obtaining from the C.S. the class of ( Meta ) rules when the selection criteria contained in
      Actual-Metalevel will be applied }
    Selected-MetaRule-Set <- Meta-Subset (Conflict-Set, Actual-Metalevel - 1)
    While (Cardinality (Selected-MetaRule-Set) > 1)      {There are selection criteria }
        & (Not (Empty? (Selector-MetaRule-Set)))      {There is more than one rule to be selected }
    do
        { Obtaining the selection criteria represented by the pair Rule-Application-Context }
        Rule-Application-Context <- Get-first-Rule-to-Execute (Selector-MetaRule-Set)
        If Is-Applicable( Rule-Application-Context, Selected-MetaRule-Set) then
            Execute-Meta-Rule(Rule-Application-Context, Selected-MetaRule-Set )
            { the execution of the selected criteria will produce as a result a new Selected-MetaRule-Set }
        Endif
        { obtaining the next selection criteria }
        Delete-First-Rule-to-Execute ( Selector-MetaRule-Set )
    Endwhile
    { The set of Meta-rules represented by Selected-MetaRule-Set is used as
      Selector-MetaRule-Set upon the meta lower level }
    Actual-Metalevel <- Actual-Metalevel - 1
    Selector-MetaRule-Set <- Selected-MetaRule-Set
Endwhile
{ The outer loop terminates when the value of the variable Actual-Metalevel is zero, that is when its value
represents the level corresponding to ordinary rules. This value is attempt after Meta-rules has been applied
to select rule instances (rule, context) contained in level =1 corresponding to ordinary rules. When the
inner loop terminates, the variable Selected-MetaRule-Set contains the subset of pairs (rule, context)
obtained by the interpretation of the upper meta-level rules, that is when Actual-Metalevel is of
value 1 -corresponding to meta-rules- the variable Selected-MetaRule-Set holds the subset of pairs
(rule, context) belonging to ordinary rule level, obtained by the interpretation of meta-rules. User
defined meta-i-rules should guarantee that the cardinality of Selected-MetaRule-Set is one. If it does
not occur chose the first pair is selected. }

SELECT-RULE-TO-EXECUTE <- Get-first-Rule-to-Execute (Selector-MetaRule-Set)
End

```

IV - Working example

To illustrate the working of the inference machine we are going to use a reduced knowledge base obtained from a real expert system application on emergency assistance [Garijo 87]. One of the first goals to achieve in emergency assistance is to get information about the person(s) injured in order to evaluate his/her state to make a global diagnosis . Let us assume the following scenario : there is an observer watching a possibly injured patient. The expert system simulating the operator behavior analyzes the information obtained from the observer in order to make a global diagnostic about the graveness of the patient.

The objects representing observer 's information are the following :

(Schema observation

(Slot owner : <person identifier>)	{ person being observed }
(Slot parameter : < parameter value >)	{ the parameter being observed }
(Slot measure) : <observation value >	{ the measure of the parameter })

The possible parameters and their corresponding measures are:

Parameter values	Measure values
bleed	nothing, little, abundant, unknown
consciousness	conscious, semiconscious, nonconscious, unknown
breathe	imperceptible, weak, normal, quick, unknown
pulse	imperceptible, weak, normal, quick , unknown

(Schema patient

(Slot name: < identifier>)
(Slot graveness: **one of** { unhurt, light, grave, great-graveness, critical})

(Schema informer

(Slot name: < identifier>)
(Slot profession: < profession- identifier>))

To simplify the expression of object instances and to improve readability of rules we are going to use a table based notation.

A reduced subset of rules allowing the inference of the state of the patient are:

RULE		PRECONDITION				ACTION
Name	Weight	Bleed	Consc.	Breathe	Pulse	State
bleed-conscience-light	2	nothing	semic.			light
breathe-pulse-grave	6			quick	weak	grave
conscience-breathe-pulse-critical	12		semic.	quick	weak	critical

Rules in the table are characterized by a name e.g. bleed-conscience-light and a weight e.g. 2, the precondition and action parts are made up of slots and their corresponding values . The first rule in the table should be interpreted as follows: **Rule bleed-conscience-light weight 2 :** " if there is a patient being observed with no bleed (the value of bleed parameter is nothing) and semiconscious (the value of conscience parameter is semiconscious) **then** the patient state is light.

Control knowledge is represented through:

a - Meta-rules :

Rule Only-doctors-feel-pulse " if there is an applicable rule containing in the application context the parameter pulse and the profession of the informer is not doctor **then** delete the rule " .

The expression of the meta-rule using the rule language is:

```
(Rule Only-doctors-feel-pulse
  (schema Conflict-Set-Meta-Class
    (level 0)
    (Set-of-Satisfied-Rules $? ?rule $?))
  ?rule : (schema Satisfied-Rule
    (Application-Context $? ?obj $?))
  ?obj : (schema observation
    (parameter pulse))
  (schema informer
    (profession ?prof & ?prof <> doctor))
=>
Delete(?rule))
```

Rule Order-by-weight " if there are applicable rules **then** order the rules by weight in descending order

```
(Rule Order-by-weight
  (schema Conflict-Set-Meta-Class
    (level 0)
    (Set-of-Satisfied-Rules $?rule $?)) ] )
=>
Order ($?rules,weight,Descending) )
```

Rule Select-abnormal-parameters " if there are applicable rules containing in their application context observation schemes and the value of the slot measure is neither nothing or conscience or normal then select the rules for execution "

```
(Rule Select-abnormal-parameters
  (exist-set $?rules = [?rule /
    (schema Conflict-Set-Meta-Class
      (level 0)
      (Set-of-Satisfied-Rules $? ?rule $?))
    ?rule : (schema Satisfied-Rule
      (for-all (Application-Context $? ?obj $?)
        ?obj : (schema observation
          (measure ?degree & ?degree /= nothing &
            ?degree /= conscious &
            ?degree /= normal)))) ] )
=>
Select($?rules))
```

b - Meta-Meta-Rules :

Control strategy based upon the application of the meta-rules named Only-doctors-feel-pulse and Order-by-weight .

Rule Doctors-&-Order-by-weight " if there are applicable meta-rules containing in their application context Satisfied-Rule schemes and the value of the slot name is Only-doctors-feel-pulse then select the meta-rules for execution using the weight "

```
(Rule Doctors-&-Order-by-weight
  (exist-set $?doctors-rules = [?doctors-rule /
    (schema Conflict-Set-Meta-Class (level 1)
      (Set-of-Satisfied-Rules $? ?doctors-rule $?))
    ?doctors-rule : (schema Satisfied-Rule
      (name Only-doctors-feel-pulse )))
    (schema Conflict-Set-Meta-Class (level 1)
      (Set-of-Satisfied-Rules $? ?weight-rule $?))
    ?weight-rule : (schema Satisfied-Rule
      (name Order-by-weight ))
  =>
  Select($?doctors-rules ?weight-rule ))
```

Let us assume that as a result of making a number of observations the state of the the working memory is as follows:

SCHEMA TYPE : observation			
INSTANCES	ATTRIBUTES		
	parameter	mesure	owner
bleed-parameter-i	bleed	nothing	patient-instance-i
conscience-parameter-i	conscience	semiconscience	patient-instance-i
breathe-parameter-i	breathe	quick	patient-instance-i
pulse-parameter-i	pulse	weak	patient-instance-i

SCHEMA TYPE : Informer			
INSTANCES	ATTRIBUTES		
	name	profession	owner
informer-instance-i	J Smith	mechanic	patient-instance-i

SCHEMA TYPE : patient		
INSTANCES	ATTRIBUTES	
	name	graveness
patient-instance-i	T Kortabarria	unknown

Fig 1 Working Memory

The evaluation of the rule base give rise to a conflict set which is made up of three conflict set meta classes: conflict set metaclasses of level 0 containing applicable object level rules, conflict set metaclasses of level 1 containing applicable meta- rules and conflict set metaclasses of level 2 containing applicable meta-meta-rules, their application context are shown in figures 2 and 3

SCHEMA TYPE : Conflict-Set-Meta-Class		
INSTANCES	ATTRIBUTES	
	Level	Set-of-Satisfied-Rules
Conflict-Set-Meta-Class-level-0	0	SR-bleed-conscience SR-breathe-pulse SR-cons-breathe-pulse
Conflict-Set-Meta-Class-level-1	1	SR-Only-doctors-feel-pulse-1 SR-Only-doctors-feel-pulse-2 SR-Order-by-Weight SR-Select-abnormal-parameters
Conflict-Set-Meta-Class-level-2	2	SR-Doctors-&-Order-by-weight SR-Select-abnormals-&-Order-by-weight

SCHEMA TYPE: Satisfied-Rule				
INSTANCES	ATTRIBUTES			
	Name	Weight	Application-Context	Action
SR-bleed-conscience	bleed-conscience-light	2	bleed-parameter-i conscience-parameter-i patient-instance-i	modify (patient-instance-i)
SR-breathe-pulse	breathe-pulse-grave	6	breathe-parameter-i pulse-parameter-i patient-instance-i	modify (patient-instance-i)
SR-cons-breathe-pulse	conscience-breathe-pulse-critical	12	conscience-parameter-i breathe-parameter-i pulse-parameter-i patient-instance-i	modify (patient-instance-i)

Fig 2 Conflict Set Meta-Classes

SCHEMA TYPE : Satisfied-Rule				
INSTANCES	ATTRIBUTES			
	Name	Weight	Application-Context	Action
SR-Only-doctors-feel-pulse-1	Only-doctors-feel-pulse	0	Conflict-Set-Meta-Class-level-0 SR-breathe-pulse pulse-parameter-i informer-instance-i	delete (SR-breathe-pulse)
SR-Only-doctors-feel-pulse-2	Only-doctors-feel-pulse	0	Conflict-Set-Meta-Class-level-0 SR-cons-breathe-pulse pulse-parameter-i informer-instance-i	delete (SR-cons-breathe-pulse)
SR-Order-by-Weight	Order-by-Weight	0	Conflict-Set-Meta-Class-level-0 SR-bleed-conscience SR-breathe-pulse SR-cons-breathe-pulse	Order (Conflict-Set-Meta-Class-level-0)
SR-Select-abnormal-parameters	Select-abnormal-parameters	0	Conflict-Set-Meta-Class-level-0 SR-breathe-pulse SR-cons-breathe-pulse bleed-parameter-i conscience-parameter-i breathe-parameter-i pulse-parameter-i	Select (SR-breathe-pulse SR-cons-breathe-pulse)

SCHEMA TYPE : Satisfied-Rule				
INSTANCES	ATTRIBUTES			
	Name	Weight	Application-Context	Action
SR-Doctors-&-Order-by-weight	Doctors-&-Order-by-weight	0	Conflict-Set-Meta-Class-level-1 SR-Only-doctors-feel-pulse-1 SR-Only-doctors-feel-pulse-2 SR-Order-by-Weight	Select (SR-Only-doctors-feel-pulse-1 SR-Only-doctors-feel-pulse-2 SR-Order-by-Weight)
SR-Select-abnormals-&-Order-by-weight	Select-abnormals-&-Order-by-weight	0	Conflict-Set-Meta-Class-level-1 SR-Select-abnormal-parameters SR-Order-by-weight	Select (SR-Select-abnormal-parameters SR-Order-by-Weight)

Fig 3 Objects contained in Conflict Set Meta-Classes

Once that the conflict set has been build the inference machine start interpreting rules from upper to lower meta classes until obtaining the object level rule to be executed. According to the figure 2 the inference machine select for execution the first meta-meta-rule in Conflict-Set-Meta-Class of level 2 - SR-Doctors&order-by-weight- . As a result the meta-rule SR-Select-abnormal-parameters is deleted from the Conflict-Set-Meta-Class of level 1 because the action part of rule SR-Doctors&order-by-weight specify the selection of rules named Only-doctors-feel-pulse and Order-by-weight. The meta-meta-rule Select-abnormals& Order-by-weight is also deleted from the Conflict-Set-Meta-Class of level 2 because their condition part is not longer satisfied. The resulting conflict set is shown below.

SCHEMA TYPE : Conflict-Set-Meta-Class		
INSTANCES	ATTRIBUTES	
	Level	Set-of-Satisfied-Rules
Conflict-Set-Meta-Class-level-0	0	SR-bleed-conscience SR-breathe-pulse SR-cons-breathe-pulse
Conflict-Set-Meta-Class-level-1	1	SR-Only-doctors-feel-pulse-1 SR-Only-doctors-feel-pulse-2 SR-Order-by-Weight

Fig 4

Meta-rules in Conflict-Set-Meta-Class of level 1 are interpreted in a similar way. First meta-rule instance Only-doctors-feel-pulse-1is selected and executed leading to the conflict set show in figure 5

SCHEMA TYPE : Conflict-Set-Meta-Class		
INSTANCES	ATTRIBUTES	
	Level	Set-of-Satisfied-Rules
Conflict-Set-Meta-Class-level-0		SR-bleed-conscience SR-cons-breathe-pulse
Conflict-Set-Meta-Class-level-1		SR-Only-doctors-feel-pulse-2 SR-Order-by-Weight

Fig 5

then the second instance of the rule is executed obtaining the conflict set in figure 6

SCHEMA TYPE : Conflict-Set-Meta-Class	
INSTANCES	ATTRIBUTES
	Level Set-of-Satisfied-Rules
Conflict-Set-Meta-Class-level-0	SR-bleed-conscience
Conflict-Set-Meta-Class-level-1	SR-Order-by-Weight

Fig 6

Now there is only one ordinary rule in Conflict-Set-Meta-Class of level 0 in consequence it is selected for execution .
When executed the graveness of the patient named Kortabarria should be obtained.

SCHEMA TYPE : patient	
INSTANCES	ATTRIBUTES
	name graveness
patient-instance-i	T Kortabarria <i>light</i>

Fig 7

V - Conclusions

This paper has presented a new production rule language allowing the specification of control knowledge using generalized meta-rules. FLORIAN rule language aims to be a simple, uniform, and powerful formalism to express both object level knowledge and control knowledge. Our approach generalizes existing knowledge systems based on meta-rules [Davis 80], [Davis&Lenat 82], [Beetz 87], [Jensen et al 87], without losing the uniformity of the rule formalism, and adding to it expressiveness and extensibility. The architecture of the inference machine keeps the simple structure of a production machine but it differs from traditional production systems on its capability of making control decisions at different levels of abstraction by interpreting the meta-i-rules. This capability of making control decisions implies the introduction of introspective phases within the traditional production cycle to achieve the decision goals. FLORIAN inference machine performs introspective phases changing the level of abstraction from domain objects to conflict set rule and meta-i rule classes, and reasoning about the rule structure and content using recursively the same inference engine employed at the object level.

Different prototypes of the system have been implemented using reduced versions of the rule language. A complete version of the language is being implemented on TI Explorer using Knowledge Craft and Prolog.

The structural complexity of the FLORIAN inference machine is equivalent to that of production systems, however from computational point of view the utilization of introspective cycles to interpret control knowledge increases the computational cost needed to perform each primary cycle. Time complexity for transforming objects in the working memory might be multiplied by a factor proportional to the number of existing meta levels and the number of satisfied rules on each level. Future research should focus on improving system performance avoiding redundant calculation on satisfied rules on introspective cycles. The approach being investigated is based upon the utilization of the RETE approach [Forgy 82] to monitor the changes within the conflict set meta-classes and their utilization to avoid the calculation of satisfied rules on each introspective cycle.

Acknowledgement

This work was sponsored in part by a grant from the Spanish Council of Scientific and Technical Research Council CICIT (nº PB85-0394) and by European Community Cost 13 program.

IV - References

- [Aiello 84] L. Aiello. The Uses of Metaknowledge in AI Systems. Proc. ECAI 84: Advances in AI-Elsevier Science, 1984
- [ART 87] Inference ART Reference Manual. Inference Corporation, Los Angeles, California, 1987

- [Attardi et al] G. Attardi et al. Metalanguage and Reasoning across Viewpoints. Proc. ECAI
- [Barachini 87] F. Barachini. PAMELA User Manual. ALCATEL Austria-ELIN Research Center, Vienna 1987
- [Barachini 87] F. Barachini. PAMELA - Eine Deklarative Programmiersprache für Echtzeitanwendungen. Austrian Conference on Artificial Intelligence, 1987
- [Barachini 87] F. Barachini. How Production Systems Can Survive in Real-Time Process Control. ALCATEL Austria-ELIN Research Center, Vienna 1987
- [Batali 83] J. Batali Computational Introspection. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, AI Memo No. 701, Cambridge Massachusetts, 1983
- [Bowen 82] K. A. Bowen. Amalgamating Language and Metalanguage. Logic Programming, Academic Press, 1982
- [Brownston et al. 85] L. Brownston, R. Farrel, E. Kant, N. Martin. Programming Expert Systems in OPS5. An Introduction to Rule Based Programming. Addison Wesley Series in Artificial Intelligence . 1985.
- [Brachman & Schmolze 85] R.J. Brachman, J.G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, vol. 9, no. 2. April 1985.
- [Davis 80] R. Davis. Meta-Rules: Reasoning about Control. Artificial Intelligence 15,1980
- [Davis 80] R. Davis. Content Reference: Reasoning about Rules. Artificial Intelligence 15,1980
- [Davis & King 77] R. Davis, J. King . An overview of production systems. Machine Intelligence Vol 8 1977.
- [Fikes&Kehler 85] R.Fikes,T.Kehler. The role of frame-based representation in reasoning. CACM Vol 28 number 9, Sept 1985.
- [Forgy 79] C.L.Forgy. On the Efficient Implementation of Production Systems. Ph. D. Thesis, Carnegie-Mellon University, 1979
- [Forgy 82] C.L.Forgy. RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem. Artificial Intelligence, Vol. 19, pp 17-37, 1982.
- [Garijo 87] F.J.Garijo. 1987 S.O.S.- E.U. An Expert System to help Human Operators In Urban Emergency Assistance. II Int. Conference on Human Computer Interaction. Honolulu, Hawaii. August 1987.
- [Garijo et al. 88] F.J. Garijo, F. Ribas. K. Urcelai . Definition and examples of FLORIAN rule language. Internal working report RT 88 / 08 . Dept. LSI. Barcelona 1988
- [Genesereth 83] M. R. Genesereth. An Overview of Metalevel Architecture. Proc.AAAI 83
- [Gupta 86] A.Gupta. Parallelism in Production Systems. Ph. D. Thesis, Carnegie-Mellon University, 1986
- [Gupta et al 86] A.Gupta, C.L.Forgy, D.Kalpy, A.Newell, M.Tambe. Results of Parallel Implementation of OPS5 on the Encore Multiprocessor. Draft Report, Department of Computer Science, Carnegie-Mellon University, 1986
- [Gupta et al 86] A.Gupta, C.Forgy, A.Newell, M. Tambe, R.Wedig. Parallel Algorithms and Architectures for Rule-Based Systems. 13th Annual International Symposium on Computer Architectures, IEEE & ACM, 1986
- [Gupta & Forgy 83] A.Gupta, C.L.Forgy. Measurements on Production Systems. Technical Report, Carnegie-Mellon University, 1983

- [Jensen y Vici 87] F. R. Jensen, A. D.Vici. The Expert System Builder (ESB). ESPRIT'87 Achievements and Impact. Project No. 96
- [Jensen 87] F.R. Jensen. Flame Reference Chart. ESPRIT Project p96, Report 381-WP-03204-535
- [KEE 85] KEE, the Knowledge Engineering Environment. Intellicorp, Menlo Park CA
- [Knowledge Craft 86] Knowledge Craft, an Environment for Developing Knowledge Based Systems. Carnegie Group, Pittsburgh PA.
- [Laird 84] J. E. Laird. Universal Subgoalting. Carnegie Mellon University, Department of Computer Science, Technical Report CMU-CS-84-129, Pittsburgh, Pennsylvania, 1984
- [Laurent 84] J.P. Laurent. Control Structures in Expert Systems. Technology and Science of Information, Vol. 3, No 3, 1984
- [Maes 86] P. Maes. Introspection in Knowledge Representation. Proc. ECAI 86
- [McCracken 78] D. McCracken. A Production System Version of the Hearsay-2 Speech Understanding System. Ph. D. Thesis, Carnegie-Mellon University, April 1978
- [McDermott 83] D. McDermott. Generalizing Problem Reduction : a Logical Analysis. Proc. 8th IJCAI Karlsruhe, 1983
- [McDermott et al 78] J. McDermott, A. Newell, J. Moore. The Efficiency of Certain Production System Implementations. Waterman D. A. and Hayes-Roth F., Ed., Pattern-Directed Inference Systems, Academic Press, New York, 1978, pp. 155-176
- [Miranker 86] D.P. Miranker. The Performance Analysis of TREAT : A DADO Production System Algorithm. International Conference on Fifth Generation Computing, Tokyo 1984, revised article 1986
- [Miranker 87] D.P. Miranker. TREAT : A New and Efficient Match Algorithm for AI Production Systems. Ph. D. Thesis, Columbia University, 1987
- [Scales 86] D.J. Scales . Efficient Matching Algorithms for SOAR/OPS5 Production System. Report No. KSL 86-47, Stanford University, 1986
- [Schor et al 86] I.M. Schor, P.T. Daly, H.S. Lee, B.R. Tibbits. Advances in RETE Pattern Matching. Proceedings of the AAAI-86, Philadelphia, 226-232
- [Simon 83] H. A. Simon. Search and Reasoning in Problem Solving. Artificial Intelligence 21, 1983
- [Stefic et al 82] M. Stefic et al. The Organization of Expert Systems, a Tutorial. Artificial Intelligence 18, 1982
- [Tenorio 84] M.F.M. Tenorio. Parallelism in Production Systems. Ph. D. Thesis, University of California, 1984
- [Weyrauch 80] R. W. Weyrauch. Prolegomena to a theory of Mechanized Formal Reasoning. Artificial Intelligence 13, 1980
- [Wos et al] I. Wos et al. Automated Reasoning: Introduction and Applications. Chapter 3, section 3.1. Prentice Hall, Englewood Cliffs NJ.