Collection de notes internes de la Direction des Etudes et Recherches

Mathématiques, informatique, télécommunications

DES MODELES AU TEMPS REEL : LE SYSTEME KSE DE TRAITEMENT DES ALARMES NUCLEAIRES

COMPILING MODELS INTO REAL-TIME SYSTEMS

EDF-- 93NJ00002

EDF Electricite de France
Direction des Etudes et Recherches

EDF
Electricité de France

Direction des Etudes et Recherches

SERVICE INFORMATIQUE ET MATHÉMATIQUES APPLIQUÉES
Département Traitement de l'Information et Etudes Mathématiques

SERVICE ENSEMBLES DE PRODUCTION
Département Communications en Centrale-Conduite

Août 1992

DORMOY J.-L.
CHERRIAUX F.
ANCELIN J.

DES MODELES AU TEMPS REEL : LE SYSTEME KSE DE TRAITEMENT DES ALARMES NUCLEAIRES

COMPILING MODELS INTO REAL-TIME SYSTEMS

Pages : 14

EDF -- 93NJ00002

Diffusion : J.-M. Lecœuvre
EDF-DER
Service IPN. Département SID
1, avenue du Général-de-Gaulle
92141 Clamart Cedex

ISSN 1161-059X

SYNTHÈSE :

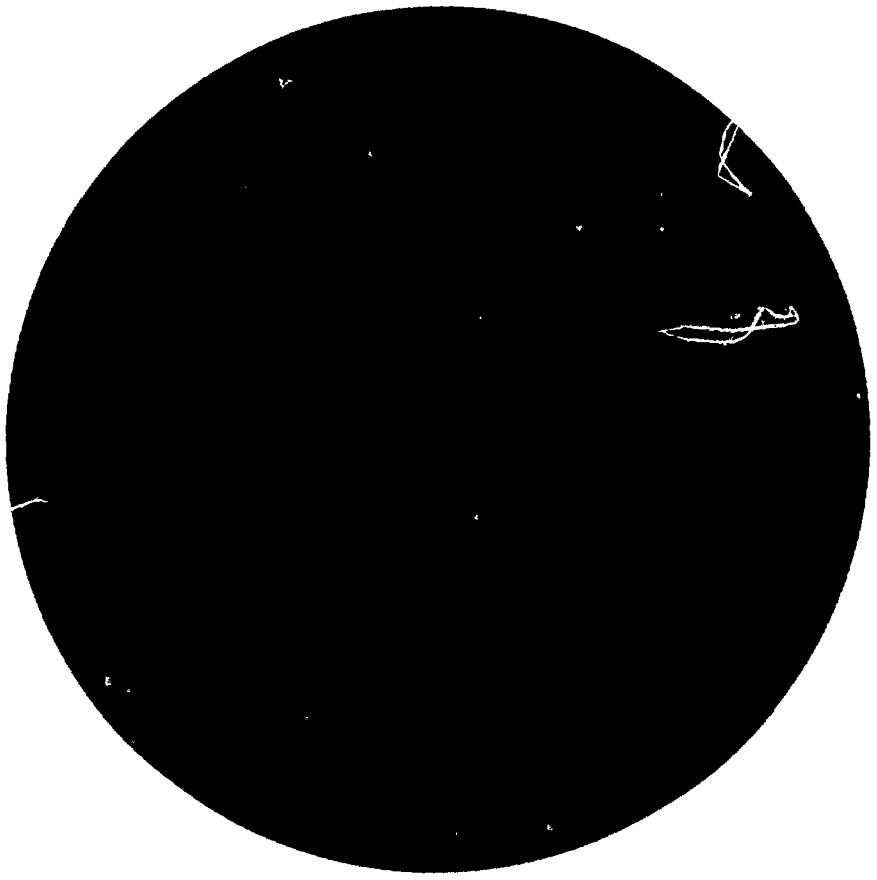
Ce papier présente essentiellement une application, à savoir un système de surveillance en temps-réel basé sur les modèles pour les centrales nucléaires. Ce système a été installé dans une centrale en France. Nous décrivons comment nous avons utilisé des techniques d'intelligence artificielle variées pour le construire : une approche basée sur les modèles, un modèle logique de son fonctionnement, une mise en œuvre déclarative de ces modèles, et des techniques originales de compilation de connaissances visant à automatiquement générer le système expert temps-réel à partir de ces modèles. Certaines de ces techniques ont simplement été empruntées à la littérature, mais nous avons dû en modifier certaines, voire en inventer de nouvelles.

Nous avons été poussés à ces innovations par la nécessité de construire un système véritablement opérationnel. Il nous semble que, considérant l'état de l'art dans la littérature de l'intelligence artificielle il reste une somme de travail importante à consacrer à la "théorie" - même si elle reste d'une portée modeste - et à l'intégration d'approches diverses lorsque l'on ambitionne une application de grande taille. Nous pensons que cela révèle un besoin, qui ne semble pas rempli aujourd'hui, et qui devrait pousser à rediriger les efforts pour les "théoriciens de l'intelligence artificielle *appliquée*".

EXECUTIVE SUMMARY :

This paper presents an architecture for building real-time systems from models, and model-compiling techniques. This has been applied for building a real-time model-based monitoring system for nuclear plants, called KSE, which is currently being used in two plants in France. We describe how we used various artificial intelligence techniques for building it : a model-based approach, a logical model of its operation, a declarative implementation of these models, and original knowledge-compiling techniques for automatically generating the real-time expert system from those models. Some of those techniques have just been borrowed from the litterature, but we had to modify or invent other techniques which simply did not exist. We also discuss two important problems, which are often underestimated in the artificial intelligence litterature : size, and errors.

Our architecture, which could be used in other applications, combines the advantages of the model-based approach with the efficiency requirements of real-time applications, while in general model-based approaches present serious drawbacks on this point.



Compiling Models into Real-Time Systems

1 Introduction

This paper presents a real-time expert system architecture together with model-based knowledge compiling techniques. They have been applied to a system, called KSE, the aim of which is to causally explain undesired events in a nuclear plant, and to provide its operators with a description of the plant operational state. Knowledge-compiling techniques make it possible to automatically generate a real-time expert system from the models of the plant, and from a logical specification of its task. We also discuss two attributes of real-life and real-time applications, which we think should be paid more attention: size and errors.

Though the techniques described here are general, we preferred to present them by reference to the KSE application, in order to 'instantiate' our ideas on a concrete example, and to show how they work in a practical way.

Section 2 describes the aims of the KSE system. Section 3 describes its architecture. First, we show how the plant, the physical and functional behaviors, and the logical model of the system are implemented. Then, we present the knowledge-compiling strategy we use for compiling these models. Section 4 discusses various aspects around our approach, in particular the issue of correctness of a system related to the correctness of its model. Then, we conclude.

2 General aims of the KSE system

2.1 The problem

The KSE project aims at developing a real-time system for alarm-processing in nuclear plants. This means that, whenever an alarm triggers in the plant, the system must be able to provide the operator with the initial cause of the alarm, and with a description of the current state of the plant in terms of functional availability of components.

This is not a diagnosis problem, in the 'pure' sense given in the AI litterature. Here, a 'misbehavior' does not mean that a component is 'broken' -though this is also possible-, but simply that something happens which makes the plant operation troublesome. So, misbehaviors are stated in terms of physical states, and the system must find out the very cause of the current misbehaviors.

Another difference is that there is no lack of information in nuclear plants. On the contrary, the operators are sometimes sunk under an overwhelming flow of information. Up to 300 alarms can trigger in a single minute, the whole set having just a few causes.

2.2 What the KSE system is

The system we have developed has been under operation for more than two years in a nuclear plant near Le Bugey (France). It is fully automatic, i.e. it takes its data from the plant information system, and presents its diagnosis to the plant operator without his intervening.

Indeed, the real-time system is an expert system, but we did not write it. Instead, we have developed a model-based shell, where models of various natures are implemented, and a knowledge compiler, which automatically generates the real-time system from its model. The basic intention was to base KSE on models in an as declarative as possible form, on one hand, and to have an efficient real-time system, on the other hand. As it is well-known, these objectives are contradictory if naively considered. So, our solution has been to develop a knowledge-compiling component.

2.3 What the KSE system is not

The scope of the system has been limited to the electric part of the plant. Indeed, all the plant components which are electrically connected are concerned, but thermodynamical processes are not taken into account. The reason is that fully solving the problem would have been intractable otherwise[1], in particular because of the size of the application. The consequence of these choices is that the system need not be able to reason about time (an instantaneous view of the plant is sufficient), and to handle sophisticated physical models (thermodynamical processes).

3 Building a model, and compiling it

This section is subdivided in two parts. We first describe the kind of model required by our approach, and how the logic of operation of the real-time system task is declared. We then show how this model together with the task specification can be compiled into the real-time system.

3.1 Models of the plant and of the monitoring task

As mentioned in the previous section, our constant intention in KSE was to separate the various sources of knowledge required by the application. We essentially have here three large chunks of knowledge. Firstly, there is a model of the plant, i.e. of what it is made of, of the nature of and the relationships between its components, and of useful information on them. This model fits well the object-oriented or semantic network frameworks. Secondly, there is a description of the causal relationships between the various quantities vehiculated by the plant. The quantities can be of a physical or functional nature. This description is embodied into a set of *principles*, which take the form of "predicate calculus causal implications". Indeed, only a subset of the predicate calculus has been used, which on one hand is sufficient to express the principles, and on the other hand makes it possible to efficiently compile the system. Thirdly, there is a logical model of the intended operation of the real-time system. Though relatively simple, declaring this model was necessary to achieve our goal of separating the knowledge sources.

3.1.1 Description of the plant

Information of various nature is described here.

[1] However, we now think of doing it.

First, there is a hierachical classification of the components (objects) and of the kinds of components (classes) with multiple-inheritance and exceptions. Second, the links between the components are described. There are several kinds of links, according to the nature of the fluid they convene - e.g., electricity or availibility. The links are 'causally' oriented in the possible directions of the flows. Third, there is a model of the attributes of the objects. This essentially describes the possible values of attributes, and their mutual exclusions. Attributes can describe the positions of components (open/closed), the values of quantities (0/1 or low/medium/high), the kind of information provided by the plant instruments, the undesired states, which states can be considered as an explanation of an undesired state, etc. The kind ok knowledge represented here is required to solve the monitoring task, and also for compiling the models (next section).

Indeed, the main problem here is not theory - all this is relatively conventional-, but size. The KSE system works on 12,000 components, and their model adds up to 150,000 attribute-value pairs. An attempt has been made to retrieve this knowledge from existing CAD databases. However, some properties were not described, such as the nature of fluids and the corresponding attributes. So, the largest part of this 'database' has been manually rebuilt for the plant where the system has been installed. A consistency-cheking component has been added, which verifies this database, but experience showed that errors remain. This is a difficult problem which shall be mentioned later on.

3.1.2 Behavioral and functional descriptions

We describe here the causal relationships between quantities of interlinked components of some given types. Let's consider a simple example: some pieces of electric hardware (causally) linked to an electric board (Figure 1).

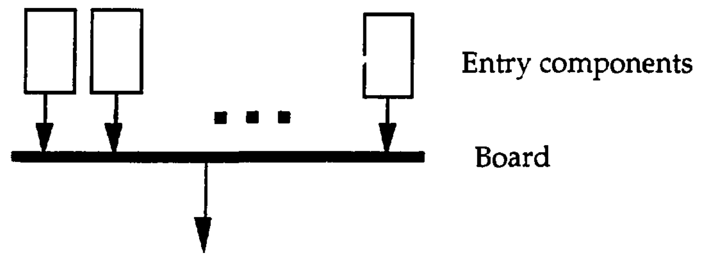


Figure 1: n components linked to an electric board

The corresponding physical principles are as follows:

```
[   x Ako Electric Board
    ∃y    [   y Ako ElectricHardware
              y ElectricallyFeeds x
              y Potential 1]
]
causes-->
[   x Potential 1]
```

```
[   x Ako Electric Board
    ∀y    [   y Ako ElectricHardware
              y ElectricallyFeeds x]
                 ==>   [y Potential 1]
          ]
causes-->
[   x Potential 1]
```

The words Ako, ElectricHardware, Potential, etc. refer to entities in the plant model.

We have here two causal models. Indeed, they could be deduced from each other in some cases, but not in general. The fact that physical laws are causally oriented implicitly introduces some functional aspects: for the class of component described by a given principle, fluids flow in the causal direction. This is global information, which cannot be known unless the whole plant connectivity is known (in particular the fluid sources). Indeed, we precisely intended to separate the principles from the description of the plant. Nevertheless, if causality was removed, the principles would merge into a single equivalence.

Both principles above mention a physical quantity (Potential). Other principles mention other kinds of quantity, such as availability - i.e., the possibility for the operator to use a component. The functional properties we had to deal with had the fortunate property that they could be specified locally, so using principles.

Principles mention classes. Principles can be inherited along the classification, with exceptions.

These principles are simple. There are more sophisticated ones, but it turned out that they all have a common form. Indeed, they all mention an x of a given class of hardware, then other y's from other classes, either universally (with an implication) or existentially quantified. No quantifiers are nested. Moreover, the forms "for all y's different from y_0..." and "there exists y different from y_0..." occur frequently. So, we imposed to express all the principles within this simplified predicate calculus, and we introduced two modified quantifiers, "for all ... except..." and "there exists ... different from ..." for convenience. All the programs processing this knowledge base were limited to this restricted "predicate calculus".

The principles are also checked by a specific component, which verifies that they are not completely stupid. In a practical way, an error in the principles occurred only once.

There are currently about 250 principles.

3.1.3 Model of operation of the task

The task of KSE consists of explaning the initial causes of undesired states. Indeed, the algorithm is relatively simple. From an effect, one can generate all the potential causes of this effect by using the principles. If a component has a state, which is known or deduced from the plant instruments, and which is inconsistent with the assumption that it causes an undesired effect, then this assumtion can be removed. Eventually, all the unremoved assumptions are -potential- causes of the undesired effect.

So, the task of KSE can be subdivided into three subtasks: deducing an as complete as possible description of the state of the plant (in terms of physical quantities) from the instruments data, then generating the possible assumptions, then removing the assumptions. All three subtasks use the description of the plant and the causal principles.

This behavior can be described by six inferences rules which use two extra symbols, namely "$\propto$" for "is a potential cause of" (assumption), and "∂" for "is a cause of" ("definitely"):

A causes B and A	$\rightarrow$	B	*Causal deduction*
A causes B and $\neg$B	$\rightarrow$	$\neg$A	*Contraposition*
A causes B and B and $\propto$B	$\rightarrow$	$\propto$A	*Assumption generation*
A causes B and A	$\rightarrow$	$\neg\partial$B	*Assumption causal negation*
$\neg$A	$\rightarrow$	$\neg\partial$A	*Assumption state negation*
$\propto$A	$\rightarrow$	∂A [Default]	*Assumption confirmation*

The last rule is a default rule. This set of inference rules must start with some assumptions (of the form $\propto$H) corresponding to the observed undesired states. Clearly, the state of the plant is computed by the first two rules, the assumptions are generated by the third, removed by the fourth and fifth, and the initial causes are established by the sixth.

The default here is relatively simple. To obtain a minimal set of initial causes, we just have to apply the first five rules as much as possible, then the default rule. Moreover, the two first rules can be used first, then abandoned, and the fifth used before the fourth.

Contrarily to all the other aspects of the KSE system, this specification of the task of the real-time system has not been incorporated in the system. It is just a formalization for its designers. We shall see the consequences of this in the compiling part of the system.

Remark 1: There are other inference rules which would be consistent with the intended semantics of $\propto$ and ∂, e.g. $\neg\propto A \rightarrow \neg\partial A$. However, they are unuseful in our context.

Remark 2: The assumption generation inference rule mentions B in its LHS. This is not necessary, but generalizing this rule by removing B would come down to generating many more assumptions. In a practical way, including B eventually means "we have found some initial causal explanations of the undesired events, though there might be 'more initial' ones, but, as far as we know, we have no means to prove them". So, this comes down to some kind of "epistemic ignorance".

3.2 Compiling the models

Once the computer has been fed with the models, another program takes them as input and compiles them into the real-time system. Before describing how this is done, we give an idea of what the real-time system consists of.

3.2.1 The target: The real-time system

The real time system works as an infinite loop. At each step of the loop, it takes the data issued from the plant instruments (essentially the values of some physical quantities). It then performs its computations as described above, and provides the user with the initial causes of alarms and undesired states (if any), and with the "functional state" of the plant, e.g. components availability. Then, another step is performed.

So, the real-time system only deals with real-time information (!). All the knowledge about the plant (the components and their links, the principles) can be wired in. Indeed, the real-time system is made up of zero-order production rules, of several kinds. Firstly, there are three groups of rules, corresponding to the three subtasks: deducing the plant state, making assumtions, and eliminating assumptions. Secondly, each rule is particular to a single component, and implements one of these activities for that component. For example, for a particular electric board b112, having three pieces of electric hardware linked to its entry, say e1, e2 and e3, there are four rules which say:

```
if      e1_has_potential 1
then    b112_has_potential 1          (two other rules)

if      e1_has_potential 0
   and  e2_has_potential 0
   and  e3_has_potential 0
then    b112_has_potential 0
```

These rules represent the particular application to board b112 of the principles of Section 3.1.2 under the first inference rule.

So, this gives the idea of the compilation process. The logical inference rules are instanciated against the principles, and applied to the description of the actual plant, to yield the first-order rules. In other words, everything which is not real-time in the models is forgotten or wired in the real-time rules. It just remains real-time information, which cannot be known at compile time, obviously.

In KSE, this yields about 47.000 rules.

3.2.2 Knowledge compiling strategy

Compiling is done in three steps.

The first step consists of "mixing up" the principles and the operation model (the inference rules). This step will produce new "principles", which are not causal ones, but which describe the deductions that can be performed on a generic plant by using the inference rules. To sum up, this first step instanciates the inference rules by the principles. The new "principles" have the same form as the causal ones. We shal call them *inferential principles*. This first step is concerned with a small part of the plant model, namely the description of attributes, of their possible values and their mutual exclusiveness.

The second step takes all the principles available (causal and inferential), and transforms them into a set of first-order production rules which do not use any explicit quantifier. There are two reasons why performing this step. Firstly, we did not have at hand a production rule language which supported explicit quantifiers, in particular our modified ones. Secondly, these rules have a very particular form. In their left-hand sides (LHS), they mention relations on objects which are known in the plant model (for example the linking relations between components). In their RHS, they write a zero-order rule according to a template, the variables of which will be instanciated by actual plant components when the production will be run. This second step is mainly concerned with the hierarchical classification in the plant model.

The third step consists of running the previous first-order set of productions on the working memory made up of the whole description of the plant (actual components, links, ...), and yields the set of zero-order production rules, which is the real-time expert system. This third step is concerned with the major part of the plant model, at least in size, that is its actual description down to the components.

3.2.3 Details of knowledge compiling: from causal to inferential principles

We describe here some particular aspects of the first step previously described. The second and third step are described in the next subsections.

Only the first four rules intervene in the transformation of principles. Each principle is transformed by each inference rule according to a transformational grammar. This grammar states how to transform a causal condition or consequence into a new condition or consequence. This grammar is obvious -it is the identity- for the causal deduction inference rule, which leaves the principles untouched. However, it must be detailed for the other inference rules.

- **Contraposition:** The main problem lies in the transformation of quantifiers, and in the interpretation of negation. The main point is that we want to build non-causal principles which are *Horn clauses* when instanciated. Let us explain this, by taking the example of a causal principle having the form:

  "if x is in class C such that,
  for all y's in class C' linked to x by L(x,y), K(y) is true
  then RHS(x)"

  When variable x is bound to a particular object X through the description of a particular plant, there is only a finite number of y's which make "y in class C' linked to x by L(x,y)" true, say $Y_1,...,Y_p$. So, the whole instanciated principle is equivalent to

  "if $K(Y_1),...,K(Y_p)$ then RHS(X)"

  Now, this clause can be transformed by transposition for all i in [1,p] into the following Horn clauses:

  "if $\neg$RHS(X) and $K(Y_1),...,K(Y_{i-1}),K(Y_{i+1}),...,K(Y_p)$ then $\neg K(Y_i)$"

  If we come back to principles, these clauses are the instanciation of the contraposed principle:

  "if x is in class C such that $\neg$RHS(x) and,
  there exists y_0 in class C' linked to x by $L(x,y_0)$ such that
  for all y's in class C' different from y_0 and linked to x by L(x,y), K(y) is true
  then $\neg K(y_0)$"

  For example, the principle "if all the entries of an electric board have Potential 0, then the board has Potential 0" is contraposed into "if a board has Potential 1, and if all the entries except e_0 have potential 0, then e_0 has Potential 1".

  Now, there is another problem: the interpretation of negation. Potential is an attribute having just two mutually exclusive values, 0 and 1, and this is described in the plant model. So, in this case, the system interpretes "x Potential 1" as the negation of "x Potential 0", and conversely. However, an attribute may have more than two exclusive value. In this case, the system interpretes "$\neg$(x Attr Val)" as "x cannot have value v through attribute Attr". So, for each attribute Attr, the system synthesizes a new attribute Imp_Attr, which reads "Impossible value for Attr". This is the way negation is handled in contraposition.

However, if it is possible at some time to deduce that x.Attr cannot have any of the possible values of Attr except v, then v must be deduced. This is not done in the principles transformation, but in a later step in the compilation process: for any attribute having more than two possible values, and for any component concerned by this attribute, some rules are added to the real-time expert system which make this kind of deduction. Fortunately, in a practical way, there are few at-least-3-valued attributes. So, the real-time system is not overwhelmed by this kind of stupid rule.

Now, we shall not fully describe the grammar of contraposition, instead we just give some hints. Grammar rules locally transform causal conditions and consequences into contraposed conditions and consequences. For example, here are two transformation rules:

($\exists y \in C'$ $L(x,y)$ and $K(y)$) is a causal condition ---->
    ($\forall y \in C'$ $L(x,y)$) is a contraposed condition *and*
    ($\neg K(y)$) is a contraposed consequence

($\forall y \in C'$ $L(x,y) \Longrightarrow K(y)$) is a causal condition ---->
    ($y_0 \in C'$, $\forall y \in C'-\{y_0\}$ $L(x,y) \Longrightarrow K(y)$) is a contraposed condition *and*
    ($\neg K(y_0)$) is a contraposed consequence

Here, x is a free variable w.r.t. the involved formulae, L refers to the structural description of the plant, and K to its state (real-time information).

Assumption generation: The goals are the same as in contraposition, i.e. to get principles which instanciate as Horn clauses, but the means are somewhat different. Indeed, this rule is implemented in a weakened form when there is a universal quantifier of the causal principle being transformed. If we consider a causal principle having the form

```
"if x is in class C such that,
    for all y's in class C' linked to x by L(x,y), K(y) is true
        then RHS(x)"
```

then it is transformed into the following assumption generation principle:

```
"if x is in class C such that RHS(x),
    if ∝RHS(x)
        if y is in class C' linked to x by L(x,y)
        then ∝K(y)"
```

An unweakened form would be:

```
"if x is in class C such that RHS(x),
    if ∝RHS(x)
        if y is in class C' linked to x by L(x,y)
        then ∝[∀y∈C' L(x,y) ==> K(y)]"
```

The same is true if we substitute an existential quantifier for the universal one. In this case, the assumption generation principle will state that there possibly are several causes, but it will "forget" that just one of these causes is necessary to explain the effect.

This means that the assumption generation process does not keep track of "and's" and "or's" of causes. In a practical way, this is not a problem, because eventually all the possible initial causes will be deduced -this could be proved-, but there is no distinction between "and" and "or" sets of causes.

The $\propto P$ and ∂P are represented by synthesized attributes. For example:

$\propto$(x Potential 1) = ∂(x PossibleInitialCause 1)

In a practical way, we do not distinguish the causes of different effects, though this could be possible.

We do not give here the grammar for assumption generation, it works in a similar way to contraposition.

Assumption elimination: There is no particular difficulty here, because, so to speak, this transformation keeps the order of deduction of the causal principles.

In a practical way, the grammars have been implemented as a set of first-order production rules (about 150 rules). From 250 causal principles, 400 inferential principles are deduced. Only a subset of the 250 initial principles are related to physical quantities, and thus deserve to be transformed (the assumption reasoning is irrelevant to 'functional' principles).

3.2.4 Details of knowledge compiling: from inferential principles to RTSE compiler

The second step starts from the inferential principles, and generates a set of productions which, when applied to the full description of an actual plant, synthesizes the real-time expert system (RTES). Indeed, this transformation is relatively straightforward. The only trick is to distinguish what is related to real-time information and what is not. For each inferential principle, a set of production rules is generated, which mentions in their LHS the structural description, and the RHS of which is the zero-order production rule template.

There are 10 production rules generated from each inferential principle in average, that is from 400 principles, about 4,000 first-order rules are built.

3.2.5 Details of knowledge compiling: RTSE compilation

Here, the mechanism is straightforward: the 4000 first-order production rules are run with the whole structural description as working memory (about 150,000 'facts'). This yields about 47,000 zero-order rules in the current version.

3.2.6 Knowledge compilation in practice

Our architecture has some practical advantages: it is possible to disconnect 'conceptual maintenance', which consists of maintaining the principles and the generic knowledge, and the 'structural maintainance', which consists of maintaining the structural description.

Steps 1 and 2 can be performed purely off-line. This means that these steps are fully independant from the structural description of the plant (the actual components, and their connections). In general, the kinds of components a plant is made up of do not change over time, or not very frequently. Would a new kind of component be present, new principles should be added, and steps 1 and 2 should be rerun. So, in general, these steps can be performed in our laboratory, and the corresponding programs need not be used at the nuclear plant site.

Step 3 depends on the structural description. In a practical way, the components of a nuclear plant periodically change (once every three weeks in average). So, the database containing the structural information must be maintained by the plant operators while changes occur. Whenever this happens, the third step must be reexecuted, to regenerate the real-time expert system. We do not have implemented any incremental compilation mode, though this would have been possible (but unuseful). This process takes about 10 hours. Once this is done, the new real-time system is launched.

The real-time system, together with the interface programs, provides a diagnosis every 10 seconds.

4 Other aspects of KSE

We discuss here various issues raised by our architecture.

4.1 Performance

As said above, the models are built, and the first steps of compilation are performed in our laboratory. So, time and memory are of no concern here.

The third step requires about 10 hours, which is compatible with the 'time constants' of real-time operation. Otherwise, we think that it could have been possible to reduce this time by rule-compiling techniques, and/or incremental compilation.

The RTES performs a full run in 5 seconds (10 seconds with the interface). We have developed compilation techniques which would enable us to reduce that time by a factor 15. The reader should not be amazed by this performance: zero-order rules are very simple. So, we can hope to use bigger RTSEs, up to 500,000 rules.

4.2 Correctness and completeness

Though we have not proved it formally, we think that our system always gives a right answer, and always provides the operator with the best possible view of the plant state, at least in terms of initial causes, and functional properties. However, this is true in a perfect world. Obviously, the correctness of an answer strongly relies upon the correctness of the model and of the real-time data provided by the plant information system. Our two-year on-site experiments showed that the system can fail to give a correct answer mainly for two reasons: there is some piece of real-time data which is incorrect, or the structural description is incorrect[1]. The two reasons occurred at the same rate.

To prevent bad operation from that, KSE also implements in the RTES a component which checks its deductions against the available information. So, it can find out that something is incorrect, and informs the operator if so. It could also be possible to distinguish which deductions remain correct despite a known contradiction. However, finding out the cause of a contradiction is a difficult problem. As mentioned, we cannot consider the 'pure' problem where the structual description would be perfect. So, the principles of model-based diagnosis must be reinterpreted here.

We have not solved this problem yet. However, a step has been accomplished by using the system Melodia. Melodia is a zero-order production rule verifier, which from a set of productions yields all the clauses describing the initial working memories which do not lead to inconsistency. In our case, the initial working memory is made up of the real-time data. We got very surprising results when applying Melodia to some instances of the RTSE: Hundreds of thousands of inconsistency clauses were generated.

Indeed, after a thought, this is not amazing. The RTSE answers are correct provided that data are correct, but nothing ensures in its design that it is still the case otherwise. So, the inconsistency clauses produced by Melodia are *constraints on the physical data*. If the clauses are satisfied by the data, then the data can be considered as being correct. If a contradiction then occurs during the RTSE inference process, there is probably an error in the structural description from which the RTSE was compiled. On the contrary, if some clauses are violated, then there is a problem in the data. In this case, diagnosis strategies can be used to isolate the faulty piece of data.

These ideas have not been implemented yet. However, we think that this could be possible, despite the size of the set of clauses to be considered. In particular, checking the data against 500,000 clauses should be possible in a few seconds: this is a problem very similar to zero-order rules compilation.

5 Conclusion

We think that this work can provide other AI practitioners with interesting knowledge compiling techniques, which can reconcile the model-based paradigm with real-time requirements.

[1] It happened once that a causal principle was incorrect. This has been readily fixed.