# KB3 TOOL: FEEDBACK ON KNOWLEDGE BASES

Marc BOUISSOU, Sibylle HUMBERT, Sabine MUFFAT, and Nathalie VILLATTE
EDF R&D, 1 avenue du général de Gaulle 92141 Clamart cedex France

## Summary

This paper shows the diversity of the applications in which the KB3 program can be used, thanks to an approach that capitalises on modelling work in knowledge bases. Using concrete examples, it also highlights the advantages this approach has for applications, together with difficulties and prospects for development.

Feedback on the development of knowledge bases has served to identify a number of directions in which progress can be made and which orient future development. The principal orientations include development of a knowledge-base editor and implementation of an organisation set-up and tools to facilitate sharing of feedback by different developers of knowledge bases.

## INTRODUCTION

In order to improve the quality, rapidity, and accessibility of dependability studies, Electricité de France (EDF) developed the KB3 program [1]. KB3 automatically builds reliability models of the structural type (fault trees or systems of Boolean equations) or behavioural type (Markov graphs, Monte Carlo simulation models, etc.) for studying a system on the basis of a graphic description of the system layout, a description of system missions, and a generic knowledge base (kb).
To carry out dependability studies with KB3, a knowledge base [2] adapted to the problems involved in the studies to be carried out is needed. Such a knowledge base must contain a generic description of the different kinds of components that might be encountered in the studies (description of possible component failure modes and of their consequences on the system). This single generic description is independent of the topology of a given system and can therefore be used for all system studies involving the problems dealt with. The KB3 knowledge bases are written in Figaro, a special language [3] developed by EDF.
For 10 years now, KB3 has been used in a great many operational applications, and has consequently participated in the development of a very wide range of knowledge bases to meet the requirements of those applications.

**In this paper we will first give a general introduction to the Figaro language and its power, backing it up with examples. We will also indicate the range of possible applications. We will then present a list of questions that must be asked before a knowledge base is built. This list was drawn up on the strength of feedback from the use of KB3.**

### General presentation of the Figaro language

The Figaro language [3] is a very general modelling language with the following objectives:

- provide an appropriate formalism for developing knowledge bases (with generic descriptions of components)
- be more general than all the usual reliability models
- find the best trade-off between modelling power (or generality) and possibilities for the processing of models
- be as legible as possible
- be easily associated with graphic representations.

The following presentation is not intended to give all the details of the language syntax, but rather to give the minimum necessary to introduce the two levels of the language - order 1 and order 0 - and the relations between them. A **formal** definition of the semantics of the order 0 language is given in article [4], published in the same conference proceedings as this paper.
*NB: The keywords of the Figaro language are in upper-case letters. To help distinguish them from all other elements in the following examples, all other words will be in lower-case letters.*

Apart from some global information (such as the declaration of the order of steps - cf. [4]), a knowledge base contains generic models (introduced by the keyword CLASS) whose **instances** can be used in a wide variety of assemblies entered by the user by means of graphic interfaces. Each class contains the following set of fields, all of which - except for the class declaration - are optional:

```
CLASS  t  KIND_OF t1 t2 ;
INTERFACE  i1 KIND  t1 CARDINAL 1 TO INFINITY ;
           i2 KIND t2 ;
CONSTANT c1 DEFAULT { constant expression (Boolean,
           numeric, character string) };
DIST_PARAMETER p1 DEFAULT {constant numeric expression};
FAILURE  p1 LABEL "first failure mode of %OBJECT" ;
         p2 ;

EFFECT  e1 LABEL "first effect of %OBJECT";
        e2 ;

ATTRIBUTE  a1 DOMAIN BOOLEAN DEFAULT FALSE;
           a2 DOMAIN 'value1' 'value2' 'value3'
              DEFAULT  'value1';

OCCURRENCE
  { occurrence rules }
INTERACTION
  { interaction rules }
```

Each class may additionally contain a few utility fields for specifying the elements which can be viewed and/or modified via the graphic interface, but no more.

A class therefore clearly consists of two parts:

⇒ a purely **static** and declarative part :

- declaration of the name of the class and of the class(es) whose characteristics it has inherited,
- declaration of interfaces (classes with which the class concerned will interact, possibly with constraints on the number of objects authorised for each interface),
- constant characteristics,
- three categories of state variables, with their initial values. The possibility of declaring state variables as FAILURE or EFFECT serves two purposes: to provide a writing shortcut to avoid having rules written in a more complicated manner, and to use concepts that “say something” to reliability specialists (EFFECT = component failure mode associated with loss of one of the main functions of that component). There is no basic difference between these categories of state variables (which are Boolean variables) and ATTRIBUTES, which are the most general category. ATTRIBUTES can be of four types: BOOLEAN, INTEGER, REAL, or enumerated (like a2 in the example above) types.

⇒ a **dynamic** part : the occurrence and interaction rules describing the behaviour of the class. The occurrence rules describe elementary events with the conditions governing how they are triggered and the associated probability distributions. The purpose of the interaction rules is to propagate the effects that are the immediate and certain consequences of

an event in the system [4]. These rules often make use of quantifiers in order to be valid irrespective of the content of sets of objects defined by the interfaces; some examples are given below.

When the graphic interface associated with a knowledge base is defined, a class can be associated equally well with an icon as with a link. This means that a link could be a complex object with rules, and not a simple “tube” conveying information from one end to another.

Once the architecture of a system has been graphically entered, the man-machine interface translates it into a list of objects described in Figaro language.
For example, the diagram below (Figure 1) could correspond to the following description in Figaro language:

```
OBJECT v1 IS_A valve ;
OBJECT v2 IS_A  valve ;
OBJECT v3 IS_A  valve ;
 DIST_PARAMETER lambda = 1.E-3;
 INTERFACE
     upline = v1, v2 ;
```

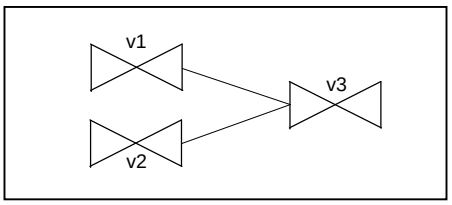



**Figure 1: Architecture of a system**

Nothing is specified for objects v1 and v2, which means that their upline interface is empty, and that their probability distribution parameter ‘lambda’ takes the value defined in the knowledge base.
On the other hand, v3 has two objects in its upline interface—v1 and v2—and the user has overwritten the ‘lambda’ value with 1.E-3.

The set "knowledge base + list of parametered objects" is a complete model in order 1 Figaro language for a given system. This model is very concise, but it would be very complex to use it directly, and not all the recommended checks could be run on it. For this reason, prior to any processing, this model is fully instantiated in order 0 Figaro, a very simple sub-language of order 1 Figaro which is particularly suitable for description of the behaviour of a particular system, and which enables all consistency checks to be run (cf. [4]) and effective processing to be carried out.

This instantiation procedure comprises the following operations:

- application of inheritance and overwriting rules to every object in the system,
- elimination of the quantifiers in the rules. This is made possible by the fact that quantifiers concern sets that are known by the list of their elements: the interfaces of objects. The rules obtained will generally be simpler (and in some cases they are simply eliminated!), but also more numerous since they will be repeated as many times as there are objects of a given class.

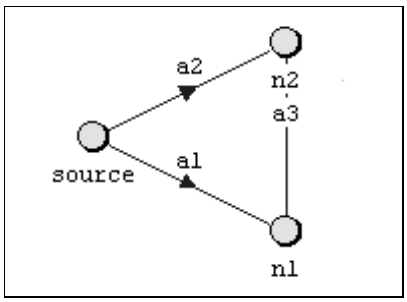
Here is a **complete** example of a micro knowledge base and of a system description, with the version in order 1 and extracts of it in order 0. This knowledge base is used to model the propagation of a flow initiated by sources in a network of any topology (Figure 2) whose nodes and edges (which can be mono-directional or bi-directional) have a *gamma_unavail* probability of being unavailable.

**Knowledge base:**

```
CLASS                   component ;
FAILURE                 unavailability ;
DIST_PARAMETER          gamma_unavail DEFAULT 0.00005;
EFFECT          linked
                LABEL "%OBJECT connected to a source" ;
ATTRIBUTE       tested_unavail DOMAIN BOOLEAN
                DEFAULT FALSE ;

OCCURRENCE
  IF NOT tested_unavail
  MAY_OCCUR
          FAULT unavailability
          DIST INS ( gamma_unavail )
          INDUCING tested_unavail  <-- TRUE
      OR_ELSE TRANSITION availability
          INDUCING tested_unavail  <-- TRUE ;
(*-----------------------------------------------------------*)
CLASS node  KIND_OF  component ;
CONSTANT
  function DOMAIN 'source' 'consumer' 'intermediary'
          DEFAULT 'intermediary' ;
INTERACTION
IF WORKING AND  function = 'source' THEN linked;
(*-----------------------------------------------------------*)
CLASS  mono_dir_edge KIND_OF component ;
INTERFACE
  origin       KIND  node CARDINAL 1 ;
  target       KIND   node CARDINAL 1 ;
INTERACTION
  IF WORKING AND linked OF origin THEN linked OF target;
(*-----------------------------------------------------------*)
CLASS  bi_dir_edge  KIND_OF  component ;
INTERFACE
  end  KIND  node CARDINAL 2 ;
INTERACTION
  IF WORKING
    AND ( THERE_EXISTS x AN end SUCH_THAT linked OF x )
    AND ( FOR_ANY x AN end WE_HAVE  WORKING OF x )
  THEN FOR_ALL y AN end DO linked OF y ;
```

**System description (graphical and in Figaro language):**




```
OBJECT   n1 IS_A node ;
OBJECT   n2 IS_A node ;
OBJECT   source IS_A node ;
  CONSTANT function = 'source';

OBJECT   a1 IS_A  mono_dir_edge ;
  INTERFACE    origin = source  ;
                target = n1  ;

OBJECT   a2 IS_A mono_dir_edge ;
  INTERFACE    origin = source  ;
               target = n2  ;

OBJECT   a3 IS_A  bi_dir_edge ;
  INTERFACE    end  = n1 n2  ;
```

**Figure 2: A network with 6 components**

In the order 1 description, the two classes of edges clearly show that depending on the cardinality of the sets that make up the interfaces, it is possible for a rule to refer directly to the sole object that is in the interface (case of a mono-directional edge) or to make use of a quantified expression which will be valid irrespective of the number of objects declared in the interface (case of a bi-directional edge). The latter case is interesting in that it illustrates the main basic constructs of the order 1 Figaro language: existential quantifier, universal quantifier, and "loop" on the conclusion side of the rule.

Below, to illustrate the instantiation procedure, is object a3 taken from the order 0 description:

```
OBJECT a3 IS_A bi_dir_edge ;
INTERFACE   end  = n1 n2 ;

ATTRIBUTE    tested_unavail   DOMAIN BOOLEAN  =  FALSE ;
(* FAILURE *) unavailability     DOMAIN BOOLEAN   = FALSE ;
EFFECT         linked LABEL "%OBJECT connected to a source" ;
DIST_PARAMETER
gamma_unavail = 5e-005 ;

OCCURRENCE
xx1
  IF (tested_unavail OF a3 = FALSE ) AND
     (unavailability    OF a3 = FALSE )
  MAY_OCCUR
    FAULT unavailability
       DIST INS ( gamma_unavail OF a3 )
       INDUCING unavailability OF a3   <--TRUE,
                    tested_unavail OF a3 <-- TRUE
    OR_ELSE
    TRANSITION availability
       INDUCING tested_unavail  OF a3 <-- TRUE ;

INTERACTION
xx4
  STEP default_step
  IF ( ( ( (unavailability OF a3 = FALSE ) ) ) AND ( ( linked OF n1 =
TRUE  )   OR ( linked OF n2 = TRUE ) ) )
    AND  ( ( ( (unavailability  OF  n1  =  FALSE  ) ) )  AND  ( (
(unavailability OF n2 = FALSE ) ) )  )
  THEN linked OF n1 <-- TRUE , linked OF n2 <-- TRUE  ;
```

To complete this quick overview of the Figaro language, here are two examples demonstrating how very flexible it makes modelling.

**Example 1: definition of the semantics of the "transition" object of a Petri net**

In a "knowledge base" whose classes are the graphic objects of a Petri net, the semantics of transitions (in the Petri-net sense) have to be defined in Figaro language. The purpose of this kb is to enable the user to work with Petri-nets while having access to all the generic functions of KB3 such as interactive simulation, the search for sequences leading to undesirable states by FigSeq, management of variants of a model, etc.
In this example, a state variable is accessed by double indexing through the interfaces. More precisely, in the following rule (it is part of the description of the transition), one speaks of marking of object x (a place) declared in the "origin" interface of an arc which is itself declared in the "input_arc" interface of the transition. This rule remains very legible despite the complexity of what it expresses.

```
IF( FOR_ANY x AN input_arc     WE_HAVE
            marking ( origin OF  x ) >= weight OF  x )  AND
   ( FOR_ANY y AN  inhibition_arc WE_HAVE
            marking (origin OF y ) < weight of y )
MAY_OCCUR
     TRANSITION firing  DIST EXP (lambda)
        INDUCING
     ( FOR_ALL  x AN input_arc DO
marking (origin OF x ) <--  marking (origin OF x )  - weight OF x ),
     ( FOR_ALL x AN output_arc DO
marking ( target OF x ) <--  marking (target OF x ) + weight OF x );
```

**Example 2: rule with nested quantifiers**

Constructions using quantifiers can be nested easily by using mute variables, making it possible to write expressions describing complex logics in a generic manner, as in the following example.

```
Rule_R1
IF THERE_EXISTS e1 AN input SUCH_THAT
    (FOR_ANY x AN input OF e1 WE_HAVE inhibited(e1))
AND THERE_EXISTS e2 AN input VERIFYING e2<>e1
    SUCH_THAT (FOR_ANY x AN input OF e2 WE_HAVE
    masked(e2))
THEN blockage ;
```

Once it has been instantiated at order 0, this single rule will generate rules whose complexity rapidly increases with the number of component inputs, as can be seen below (Figure 3).

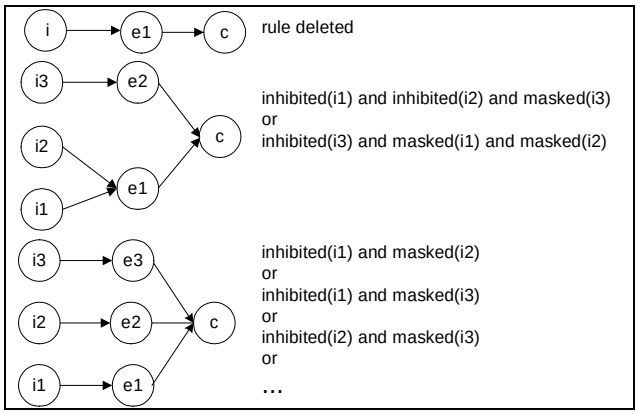



**Figure 3: Instanciation of Rule_R1 on various topologies**

In what follows we present the various categories of knowledge bases developed by EDF to meet the requirements of different applications, and also the "right" questions that should be asked when a knowledge base is being designed.

## Presentation of different knowledge bases

The purpose of this article is to make feedback from the knowledge bases developed by EDF since 1986 available to the world community of reliability specialists. The first work done in this respect consisted in surveying and grouping these knowledge bases in accordance with their source and purpose. We then drew up a number of pertinent questions to facilitate the work of knowledge base designers.

### Categories of knowledge bases

The answer to "**Why develop a knowledge base?**" helps determine the category that a kb belongs to. It may have to be developed because of a specific project requiring several studies of the same type (Probabilistic Safety Analyses (PSA) for nuclear power plants, for instance) to be carried out, or simply to have a conventional formalism that reliability specialists are familiar with and/or which is still close to the R&D stage. These knowledge bases are then necessarily one of the following three categories:

- **Abstract knowledge base** — These knowledge bases, in which classes represent abstract objects (such as the places and transitions of Petri nets), **are available to all KB3 users since they are independent of the domain of application.**

- **Dedicated kb** Their function is to meet the requirements of industrial applications.

- **R&D test kb** Their function is to serve as a support for thinking on unsolved problems.

The first category includes all the known reliability models in the field of dependability: an expert in this field can easily model his problem with Markov graphs, Petri nets, fault trees, or reliability diagrams … if the system and requirements so allow.

These standard formalisms can meet a great many of the requirements of system studies but require prior user knowledge of the modelling concepts.

As its name indicates, the second category is **dedicated** to the specific requirements of special applications which cannot be met by the first category. EDF has had plenty of opportunity to meet requirements concerning:

- within EDF:
  - Probabilistic Safety Analyses (electrical or hydraulic systems [2]),
  - Evaluation of instrumentation and control systems,
  - At the design stage, assessment of availability risks for power plants (combined-cycle plants, nuclear plants, etc.)
  - Substations and transmission lines,
  - Availability of telecoms networks, etc.

- outside the EDF
  - on-board electronics (in collaboration with Renault [7]),
  - assessment of the reliability of a plan for putting a power grid back on-line after a blackout, etc.

The third category includes the knowledge bases written in-house to answer basic questions which have not yet been solved. They are used to run modelling and feasibility tests. One of the problems raised, for example, is modelling of two-way flows with a view to producing fault trees.

## Operational feedback on knowledge bases

As a result of more than ten years of experience, a set of the "right questions" to be asked when writing a kb has been drawn up. This kb approach makes it easier to capitalise on reliability expertise and is a source of constant enrichment and progress in terms of productivity of studies, traceability, consistency of studies, etc.

To effectively benefit from this potential, three essential questions must be asked when a kb is being designed.

### Question 1: possibility of re-use?

When a kb is being designed, it is important to refer to the existing kb and assess to what extent it can be re-used.
If the future user of the KB3 program and its knowledge base is not a reliability specialist, it will certainly be necessary to work in the direction of a dedicated kb. In this case, a check must be carried out to determine whether an existing kb that meets the objectives perfectly can be re-used, or whether an existing kb can be adapted and converted.
This approach can make for time savings. However, one of the difficulties lies in writing knowledge bases that are **dedicated** while still remaining sufficiently **generic** for other applications.
If the users are reliability specialists, it will be necessary to consider using abstract knowledge bases, if the study allows it, or otherwise to adopt the above procedure. It is perfectly possible to model electrical systems with an abstract kb.
For example, a kb called BDMP (Boolean logic Driven Markov Processes), the concepts of which are a combination of those of fault trees and Petri nets, can perfectly well model electrical systems with their possibilities of reconfiguration. This requirement arose internally. The existing BDMP knowledge base, one of the category of abstract bases, was immediately used for the study in question.

### Question 2: Major objectives of a kb

Writing a kb requires definition of a number of main objectives: the type of processing, the quantities to be calculated, the level of detail, and the skill of users.

### Type of processing

**Processing** is an essential criterion in determining the possibilities a kb will bring to carry out a study. A **static** study is based on transformation of a model into a fault tree. Fault-tree generation is the main function of the KB3 tool. The processing of trees is then carried out using tools developed outside EDF (Aralia, Risk Spectrum, etc.). This type of study concerns above all PSAs and all the models which do not take account of repairs. The second type of processing for dependability studies concerns **dynamic** models. In the best of cases, the fault tree generated with a dynamic model gives a simplified and approximate view, but it is generally unusable. There are three categories of dynamic models: Markovian, non-Markovian using instantaneous laws[1] (INS) and constant-time laws (CT), or non-Markovian using laws other than INS and CT laws (Table 1). Changes in states, using a constant-time law, are applied after a deterministic time and provide a sure result. Table 1 gives the correspondences between the main typologies of models encountered, what they are used for, and the associated tools.

Markovian models are used to explore sequences, based on order 0 Figaro models generated by KB3. The exploration and quantification are carried out with a special program: FigSeq (FIGaro SEQuences) developed by EDF [3]. Non-Markovian models using only instantaneous and constant-time laws can also be processed by the FigSeq program, using a suitably adapted algorithm. The Topase tool used to design VHV substations was designed principally for this kind of model [5].
As for non-Markovian models using laws other than INS and CT laws, EDF currently has no operational tools but has done some work on the subject. A prototype on Monte-Carlo simulation exists and deserves to be taken up again and validated [6].
Starting from these types of models, the tools presented all enable both qualitative and quantitative studies to be carried out.

| Typology of models | Use | Potential / Operational |
|---|---|---|
| Static, unlooped | Generation and processing of fault trees | Operational (<br>- generation: KB3,<br>- processing: Aralia or Risk Spectrum) |
| Static, looped | Generation and processing of logic equations | Operational (<br>- generation: KB3,<br>- processing: Aralia) |
| Markovian | Generation and quantification of sequences | Operational (FigSeq) |
| Neither static nor Markovian, with Instantaneous and Constant-Time laws | Generation and quantification of sequences | Operational (FigSeq, Topase) |
| Neither static nor Markovian, with other laws than INS and CT laws | Monte Carlo simulation | Potential |

**Table 1: Typology of models and possible uses**

To illustrate the possible consequences the choice of processing can have on how knowledge bases are written, we give the

[1] We use "law" as a synonym for "probability distribution"

example of a normal/standby system (Figure 4) which can be modelled “exactly” dynamically, and approximately with a static approach.

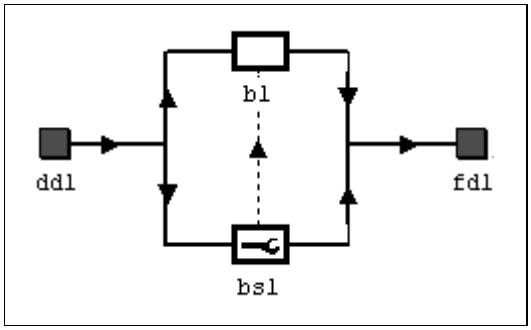



**Figure 4: Normal/Standby system**

If the user wants to be able to conduct two types of processing with a single model, he can use a global constant authorising or prohibiting certain rules, depending on its value. For example, for the type of system modelled above, the following order 1 description could be used for the standby blocks:

```
CLASS standby_block KIND_OF block ;
ATTRIBUTE
  state DEFAULT ‘off’;
INTERFACE
(* a standby_block is started up when all the blocks declared in
the interface " backed_up "  are broken down. *)
   backed_up  KIND block CARDINAL 1 TO INFINITY ;

INTERACTION
startup
   IF (state = 'off') AND  (class_of_model = ’dynamic’) AND
      (FOR_ANY x A backed_up WE_HAVE
      (NOT fulfilled_function(x)))
   THEN state <-- 'startup';

shutdown
   IF (class_of_model = ’dynamic’)     AND
     (state = 'on’ OR state = 'startup')  AND
     (THERE_EXISTS x A backed_up SUCH_THAT
     fulfilled_function(x))
   THEN state <-- 'off';
```

In this example, the two rules (startup and shutdown) describing the dependency of the standby block on the backed-up blocks it has to replace completely disappear at order 0 when the user chooses the value “static” for the class_of_model constant.

Consequently, when class_of_model = ‘static’, the standby block designated bs1 is either overlooked or deemed to be in a state of active redundancy, depending on whether its initial state is “off” or “on”. With a dynamic model, these two rules are involved in triggering changes in the state of the standby block, in accordance with the state of the normal block.

**Parameters to be calculated: reliability, availability, productivity**

The ultimate objective of any dependability study is to obtain reliability parameters such as reliability itself or availability. It is therefore important to inform the user of the calculation capacity of the kb. Concealed behind this criterion are some totally different orientations in the design of the kb. This difference will be illustrated by two knowledge bases, one leaning towards “reliability calculation” and one towards “availability calculation”, but both of them modelling electrical systems.
“Elec” is a kb for electrical systems that generates fault trees and can carry out reliability calculations. “ElecD” is a kb devoted to studying the availability of electrical systems. To enable availability to be studied, we began with the Elec base and made the necessary adaptations. The modifications chiefly concern introduction of:

- repair times,
- management of the startup time of a plant as a whole, after unscheduled outage,
- request rates per time unit, to calculate equivalent exponential failure rates for components that may fail when their startup is requested.

**Level of detail**

The design of a knowledge base is often linked beforehand to generic functional analysis and Failure Modes and Effects Analysis of systems. This serves to define the level of detail necessary for proper transcription of the behaviour of the system. The important thing is to set a level of analysis.
Experience shows that assessment of this level depends on two things: the complexity of generic models and the availability of the reliability data.
It is important to adapt the level of detail according to the processing capacities of the tools. In addition, modelling with a high level of detail does not necessarily ensure better results, but it does result in exponential growth in the volume of results to be analysed. This approach might not be pertinent in terms of analysis of the results and is likely to be less interesting than first thought.
Another constraint involved in the level of detail is the availability of reliability data. Some fields, such as the nuclear power industry, benefit from operational feedback and consequently from usable reliability data. On the other hand, the **electrical industry**, which is constantly evolving with new technologies, cannot benefit from operational feedback to determine reliability data. This factor alone therefore limits the level of detail to be used for some descriptions.

**Usable by people who are not reliability specialists**

Designing a kb that can be used by people other than reliability specialists implies the development of a “**dedicated**” kb enabling data to be entered graphically in a form similar to the diagram of the system to be studied. For example, with an electricity-oriented kb, there is no need to be knowledgeable about dependability theory in order to be able to enter a given electrical system and quantify its reliability. Therefore, using an “expert” knowledge base (with reliability expertise and system expertise) enables people who are not specialists in dependability to carry out studies themselves, using KB3.
It is advisable for a kb that is to be used by non-specialists to produce easily legible trees, and to have the possibility of simulating the system operation interactively. These objectives are examined next.

**Question 3: Secondary objectives of a kb**

**Level of legibility**

Generating fault trees implies carrying out static studies. PSA studies, for example, required a lot of work on generation in order to facilitate reading of the fault trees obtained. In fact, the requirement for good legibility concerns PSA studies above all, their requirement being simply to introduce pertinent information at the gates of the tree and thereby to facilitate reading. A new concept has therefore been introduced: that of the **obstruction block.**
An obstruction block groups the components whose obstruction has the same consequences on propagation of flow in the system. They are essential to the construction of the tree since it is through them that the kb propagates flow. Obstruction blocks are built automatically before each tree is generated. The algorithm which calculates them uses the interfaces of components and the nodes of the system.

To illustrate this concept of the obstruction block, we will take an example of modelling of a hydraulic system (Figure 5). In this

model, the tank feeds two lines, each of which has a valve and a pump. The undesirable event defined for generation of the tree is absence of fluid at node S.

The left-hand fault tree was generated from the hydraulic system using the algorithm for construction of obstruction blocks. In the right-hand fault tree, which represents the same logic, but without obstruction blocks, the same undesirable event is defined by a cascaded description of faults. In the first tree, the events explaining the obstruction of a block are presented as the tines of a rake, while without obstruction blocks the gates are cascaded, breaking up the information.

Using the different blocks defined by the algorithm, all the gates or events of the tree, except for the leaves corresponding to basic events, have information stating which block they belong to.

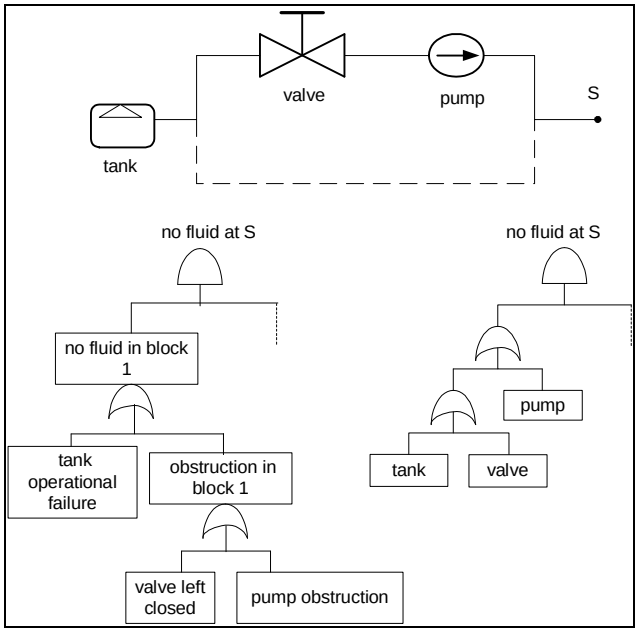



**Figure 5: Hydraulic flow diagram for a system with fault tree generated with, then without, obstruction blocks**

**Interactive simulation**

KB3 contains an interactive simulator. In any state of the system studied, it gives the user a list of all the possible events (faults or repairs) in a Simulation window.
To help the user follow the behaviour of the system when he chooses an event, display options (colour change for components or links, display of different states on components: open, closed, etc.) can be parametered in the kb right from the design stage. These options enable the user to observe directly (interactively) the consequences of the choices he has made regarding the state of components.
Interactive simulation corresponds to processing of the rules in the kb with forward chaining, i.e. according to the semantics defined in [4].
But a user does not always have to carry out dynamic studies. In the case of static studies involving generation of fault trees, the simulation should not be taken at face value.

The simulator could lead the user to believe that his tree takes account of dynamic aspects when that is in fact totally false. The greatest of care must therefore be taken when dealing with the information provided by the simulator in static studies.

## Conclusion and prospects

We have first shown that the Figaro modelling language developed by EDF can be used to write the knowledge bases that are essential for describing a physical system in the KB3 program, with a formalism close to spoken language.

Then, operating feedback on the design of knowledge bases has enabled us to define “the right questions” to be asked. This approach guarantees the relevance and satisfactory completion of the work of designers, but is not enough in itself.

Further progress must be made to transform this set of questions into methods, and then to formalise those methods of design and updating of knowledge bases more precisely, and to develop the associated tools.

The main objectives are to:
- help designers control information,
- facilitate updating of knowledge bases, some of which might have a very long service lifetime,
- facilitate management of versions of the kb, to ensure perfect consistency and traceability between the different knowledge bases written,
- facilitate access to knowledge bases by “intermediate” users who are able to make “slight” changes to them.

**Bibliography**

[1] M. PILLIERE, P.MOUTTAPA, N. VILLATTE, I. RENAULT, “KB3: Computer Program for Automatic Generation of Fault Trees”, RAMS’99, Washington (U.S.), January 99.

[2] N. VILLATTE, M. BANNELIER, “EXPRESS, a Knowledge-Based System for Automating Reliability Studies of Thermohydraulic Systems in Power Plants”, ISAP’94, Montpellier, September 97.

[3] M. BOUISSOU, N. VILLATTE, H. BOUHADANA, M. BANNELIER, “Knowledge Modeling and Reliability Processing: Presentation of the FIGARO Language and Associated Tools”, SAFECOMP’91, Trondheim, October 91.

[4] M. BOUISSOU, J.-C. HOUDEBINE, “Inconsistency detection in KB3 models”, ESREL’02, Lyon, March 2002.

[5] M. BULOT, I. RENAULT, “Reliability Studies for High Voltage Substations using a Knowledge Base: TOPASE Project Concepts and Applications”, EDF internal report 96NR00101, ISSN 1161-0581, 1996.

[6] M. BOUISSOU, J. COLLET, J.-P. SIGNORET, D. EHRET, “Jessica: a new tool for Monte-Carlo simulation”, European Safety and Reliability Conference, Copenhagen, June 1992.

[7] T. GAUDRE, M. BOUISSOU, P. CHAUSSIS, P. NONCLERCQ "Application de l’outil KB3-IFAST (ARALIA inside) de génération automatique d’Arbres de Défaillances à un système électronique embarqué automobile", 2ème Conférence Annuelle d’Ingénierie Système, Toulouse, Juin 2001.