KNOWLEDGE MODELLING AND RELIABILITY PROCESSING:

PRESENTATION OF THE FIGARO LANGUAGE AND ASSOCIATED TOOLS

Marc Bouissou, Henri Bouhadana, Marc Bannelier, Nathalie Villatte

Electricité de France, DER/ESF Section,
1 av. du Général de Gaulle
92141 Clamart cedex. FRANCE
Tel. (1) 47 65 58 22

Abstract. EDF has been developing for several years an integrated set of knowledge-based and algorithmic tools for automation of reliability assessment of complex (especially sequential) systems.
In this environment, the reliability expert has at his disposal all the powerful classic tools for qualitative and quantitative processing and besides he gets various means to generate automatically the entries for these tools, through the acquisition of graphical data.
The development of these tools has been based on FIGARO, a language for system modelling, which plays an important unifying role.
A variety of compilers and translators transform a FIGARO model into conventional models, such as fault-trees, Markov chains, Petri Nets...
In this paper, we present the main ideas which determined the FIGARO language, and we illustrate these general ideas by examples.

Keywords. Knowledge representation, Modeling, Simulation, Stochastic systems, Reliability, Performance, Monte Carlo methods, Markov chain.

I. INTRODUCTION

In the framework of the probabilistic safety analysis of the Paluel nuclear power plant (EPS 1300), EDF has developed software packages allowing the automation of reliability models' construction and assessment.

These tools were used to develop new concepts, original and highly performing algorithms /1/, but they lacked generality and user friendliness. The main problem lay in the fact that the expert systems being applied for generation of reliability models were too specific of the fields being dealt with and difficult to maintain.

EDF has therefore developed a second generation of these software packages. This version, which is available on a workstation (under UNIX/Xwindow) with user friendly, graphical interfaces, is no longer dedicated to nuclear applications.

Our concern for unification of the software packages, explanation of the reliability expert's modelling choices, and generality has led us to design a unique system modelling language (the FIGARO language) which is independent from the processing method used afterwards. This language has been worked out in order /4/:

- *to give a suitable formalism for setting up knowledge bases* (with generic component descriptions),
- to be *more general* than all conventional reliability models,
- to make the best possible *compromise* between *modelling power* (or generality) and processing *tractability*,
- to be as *readable* as possible,
- to be easily associated with graphic representations.

In fact the setting up of knowledge-based systems is the only way to reduce significantly the necessary outlay for the reliability studies.

On the basis of a FIGARO language modelling, different compilers and translators allow to deduce automatically the data which are necessary for the classical reliability model processing codes: fault trees, Markov chains, Petri nets, etc.

II. THE FIGARO LANGUAGE MAIN OBJECTIVE: TO MODEL DISCRETE STATE SPACE SYSTEMS

Let's take a physical system. We can define three probability model categories. Starting from the most detailed (and most complex) models to the least detailed ones, the specified categories are as follows:

A. Continuous state space dynamic simulation models,
B. Discrete state space dynamic simulation models,
C. Abstract models.

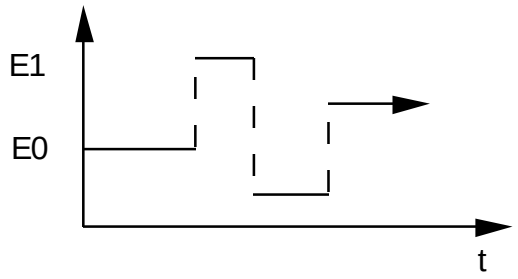
A model of type A is made up of:

- The deterministic differential equations which rule the system physical quantities: temperature, pressure, mass, etc...
- The discontinuities due to sudden component state changes induced by random phenomena (faults,

repairs...) or deterministic (timer triggered action...).

A model of type B does not imply differential equations: the system can only have a finite or countable number of states and runs over from a state to another following a random or deterministic phenomenon.

The evolution of the system is therefore a *continuous time stochastic process* which can be represented schematically as follows:



The models of type C (for example : fault trees) are thoroughly different from the two preceding ones as they have only a remote relation with the physical phenomena which rule the system life. The time is not introduced explicitly. That's why we have called them "abstract" as opposed to the simulation models.

We wished to find, in the definition of a language which describes probabilistic problems, a fair equilibrium between "full" simulation, which is quite detailed, and a too abstract model.

It is obvious that the first type of model, which is ideal in the absolute, is in practice too rich to be tractable (in most cases).

On the contrary, the choice of the C-type model (for instance a fault tree), or even of a too restrictive B-type model (for instance a Markov chain) may lead to unacceptable simplifications.

Therefore we have tried to work out a language which could describe unambiguously a B-type model, keeping in mind the objectives given in part I.

An analysis of the existing modelling and computer languages has shown that none of them provided the required set of characteristics; therefore we have created the FIGARO language **with specific object-oriented type syntax and semantics**. FIGARO is part of the so-called "hybrid" languages, that is to say it takes some of its features from the object-oriented languages and models the behavior of an object through order 1 production rules.

The use of rules offers two advantages:

- The rules are close to the natural language if their syntax is selected appropriately: their use will improve the model readability,

- EDF has got the mastery of different tools in this field and, in particular, worked on the validation of 0 order production rule bases.

The characteristics which make of FIGARO an object-oriented language bring some decisive advantages in *knowledge storage* and among them:

- Easy knowledge classification,

- No repetition (due to the "factorization" allowed by the heritage mechanism) which is a source of maintenance errors.

A FIGARO description is made of two kinds of rules:

- interaction rules: they model the propagation of instantaneous effects.

```
Ex : IF ( flow(pump1) + flow(pump3) ) <
   threshold2 THEN state(alarm) <- 'on';
```

- occurrence rules: they yield the list of events that may happen in a state of the system. These rules have a particular semantics, related to time; this is why they include a distribution law of the time after which the event will happen:

```
Ex1 : IF state(timer) = 'on'
   THEN-MAY-HAPPEN
     EVENT down
     EFFECT state(timer) <-- 'off'
     LAW FIXED_TIME(delay) ;
     (* deterministic law *)

Ex2 : IF state(engine) = 'working'
   THEN-MAY-HAPPEN
     FAILURE breakdown
     EFFECT state(engine) <-- 'breakdown'
     LAW EXPONENTIAL(lambda) ;
     (* constant failure rate *)
```

Important note: it can be easily proved /4/ (and that will appear in the below-mentioned examples) that FIGARO has the very important property to be a generalization of most of the modelling types used by the reliability experts; it generalizes in particular:

- the fault trees,
- the state graphs (including Markov chains),
- the stochastic Petri nets,
- the queuing models.

This property of FIGARO ensures that any conventional model has its FIGARO equivalent and therefore the exclusive use of FIGARO as representation formalism is not restrictive whatsoever.

III. FIGARO MODELS PROCESSING METHODS

The processing operations take place in two stages in order to master the combinatorial explosion problems at best:

*The first is only intended to pass over from the FIGARO representation of order 1, made up of the knowledge base types and objects describing a particular system, to an exactly equivalent FIGARO representation of order 0 which is obtained through application of the heritage mechanism to the objects and by instanciation of the first order (generic) rules in the form of zero order (specific) rules.

For example, the following first order rule (in the type "circuit-breaker"):

```
IF position = 'open' OR (FOR_ANY x AN
   upstream_component, energised(x) = FALSE)
THEN FOR_ALL y A downstream_component DO
   energised(y) <-- FALSE ;
```

will produce this much simpler zero order rule, for a circuit breaker cb1, having the two u1 and u2 upstream components, and the d1 downstream component:

```
IF position(cb1) = 'open' OR
  (energised(u1) = FALSE AND energised(u2) =
  FALSE)
THEN energised(d1) <-- FALSE ;
```

The transformation of first order rules into zero order rules is interesting for two reasons:

- direct processing from the first order would be too difficult as it would require numerous evaluations of first order rules which are time consuming,
- the rule base coherence checking tools exist only for zero order rules.

*The second is the choice (more or less automatized) of the processing method and the application of this method.

In order to choose the method, one has to determine the more or less static character of the system, in other words a more or less great independence between different parts of the model.

When it is possible to identify independent parts, one should take advantage of this feature to study these parts separately.

The breakdown of a big model into sub-models is a fundamental heuristic which offers decisive advantages:

- the sub-models are simpler to understand, to validate,

- they imply easier processing (the memory and the CPU time required for a study are exponential functions of the model size).

- due to their characteristics, the sub-models allow processing operations which are impossible for global models (example: a sub-model can be Markovian but that is not the case of the global model).

In order to achieve that, we have developed a program which elaborates the influence graph between the state variables of the FIGARO 0 model and allows different processing operations on the basis of this graph.

With this software, the user works in an interactive way: he can display, on request, different data on the dependence graph and order the output of sub-models (extracts from the global model in FIGARO 0) on files which will be reinserted at the start of the processing chain in order to be translated into quantifiable models.

When the user has identified the FIGARO O pertinent sub-models, he can choose the optimum method for each of them and generate the input data of one of the available evaluation tools by the adequate translator.

The evaluation tools now used at ESF and fully integrated in the FIGARO environment are as follows:

- for any FIGARO model: GSI, a software which has been developed since 1985 at ESF. GSI offers a wide variety of processings, all available from a single description (in zero order production rules). These processings are listed in part IV.

- for fault trees: PHAMISS /6/, a Dutch software developed by ECN. This software is remarkable through the variety of the quantifications which it can carry out: calculation of availability, reliability, importance of components and minimal cut sets, uncertainty and this is for components being repairable or not, periodically checked or not. The automatically generated trees have characteristics (such as many repeated leaves) which cause PHAMISS to lose much time: that's why we have developed a tree optimizer which carries out a pre-processing /2/.

- for stochastic Petri nets: either GSI, or MOCA-RP /5/, a code developed by ELF-Aquitaine which allows to carry out a Monte Carlo simulation.

IV THE GSI EVALUATION TOOL

The input of GSI is a rule model, including (like FIGARO) occurrence and interaction rules. All the rules are specific (like in FIGARO 0).

GSI allows three main types of treatments, corresponding to different methods:

- **Monte Carlo simulation**: this method is the most general, and can be used even for non Markovian models, including various lifetime distributions for the components (lognormal, Weibull...), and deterministic phenomena (fixed time type laws). The simulation can yield virtually any kind of result about the behavior of the system: reliability and availability, of course, but also average performance, number of events, sojourn times... The only limit of this method is well known : it may be very time consuming.

- **Markov chain generation and quantification:** this method is applicable when the model contains only exponential and instantaneous probability laws, in other words, when the model describes a Markov stochastic process, with a finite number of states. The (very severe) limit of this method is the exponential growth of the number of states when the size of the system increases.

- **Sequence generation and quantification**: this last method is the most original, and has given its name to GSI: "Generation of Sequences by Inferences" /3/, /7/.

  This approach does not require graph production and allows to deal with models which are equivalent to huge, **even infinite**, Markov chains.
  As a matter of fact, GSI, thanks to different heuristics, allows to limit the number of sequences to explore: to each type of exploration corresponds an approximation level.

  In 89-90, theoretical works run in cooperation with the ORSAY university have permitted, besides showing that the GSI quantification techniques remain valid on infinite graphs, to get better estimations of the errors due to the approximations.

Another advantage of the sequence generation method is that it gives the most probable sequences, i.e., the weak points of the system.

Since all the treatments of GSI may be very time consuming for big models, this software works in two steps: the input model is first translated into PASCAL procedures, which are compiled and linked with the fixed procedures containing the algorithms; then the execution itself takes place.
This technique is necessary to be able to run models containing thousands of rules in reasonable times.

V. APPLICATION FOR A USER FRIENDLY WORK ON CONVENTIONAL RELIABILITY MODELS

To carry out quick and simple studies or such studies which do not justify the development of reusable FIGARO component descriptions, the reliability expert may want to use a conventional model, such as a fault tree, a reliability diagram, a Petri net...

Such a model will be built more rapidly than a FIGARO based model, which obliges to structure and formalize the concepts being handled more or less consciously in the production of the specific model. In return, it won't be at all reusable for carrying out a second study of the same type.

FIGARO gives a very satisfying answer to this request through the possibility it offers to create "knowledge micro-bases" corresponding to the classical models. These bases allow *graphic model acquisition.*

Their development is quite fast (in general one day).

Therefore this allowed us to create easily a coherent set of graphic interfaces (as it rests on a single tool) for all the conventional models.

More generally, it is important to notice that any simple graphic language can be supported by means of a small, quickly written FIGARO knowledge base.
Besides, the FIGARO based modelling allows to access the full available processing set: for example it is possible to assess the ***reliability*** of a system represented through a fault tree by a Monte Carlo simulation, which is feasible whatever the FIGARO model, or by the analytical calculations of GSI whereas most of the fault tree codes do not permit such a calculation (for a repairable system).

Fig. 1 and Fig. 2 at the end of this paper show the aspect of graphic interfaces, which are set up through FIGARO for an example of reliability diagram, and of Petri net.

The reliability diagram can be calculated on request by PHAMISS after transformation into a fault tree, or directly by GSI.

The Petri net, as far as it is concerned, can be assessed on request by GSI, or by MOCA-RP.

VI. APPLICATION FOR STUDY OF SEQUENTIAL ELECTRICAL SYSTEMS

A knowledge base has been developed in order to model and quickly evaluate the reliability of nuclear power plant electrical distribution systems.

This knowledge base includes components such as: diesel-generators, busbars, circuit-breakers, transformers...
All the sequential aspects of this kind of systems (automatic reconfigurations after failures and repairs) are explicitly modelled in this knowledge base.
Fig. 4 gives examples of sequences which lead to the failure of the system of Fig. 3. GSI automatically found these sequences, *without building the underlying Markov chain.*

VII. APPLICATION FOR COMMUNICATION NETWORKS

A knowledge base was written to allow the quick comparison of different topologies of a communication network. The network is supposed to be made of nodes and links, which both may have failures; all the components are independent.
After the acquisition of the topology of a network, it is possible to study it either by generating a fault-tree, or by running GSI.

In a second knowledge base, a refinement has been introduced in the modelling: it is possible to declare that several components share the same repairmen. This introduces dependencies into the system, and the fault tree study remains no longer valid, whereas GSI can still be used.

VIII. DEVELOPMENT TOOLS, TARGET MACHINES

The Table 1 sums up, on a practical basis, the main features of the tools which have been quoted in this paper.

IX. CONCLUSION

We have provided the main characteristics of the FIGARO application prototype by illustrating through examples its high generality degree and user friendliness.

This application makes up a complete environment in which:

- a user of knowledge bases can carry out "fully mouse-controlled" system studies rapidly enough (10 to 20 times faster than without knowledge base) to compare different system design solutions,
- the developer of knowledge bases has at his disposal a high level formalism (the FIGARO language) to express the (functional or material) knowledge he wants to formalize.

For the time being, the user can still take many initiatives in processing selection from a FIGARO model and the developer is totally free in writing a knowledge base. In particular he can choose the modelling fineness: FIGARO allows reliability processing from a very simplified component modelling. In this way, the inexperienced user can gradually pass to the knowledge base developer's stage through gradual improvement of his models.

The following stage consists in obtaining, through an intensive use of these tools in fields as different as possible, more directing guides in order to help:

- the developer to structure and formalize his knowledge when he builds a knowledge base,

- the user to choose the most pertinent processing, in particular by controlling the validity of the choices he makes.

REFERENCES

/1/ Ancelin, C., Bannelier, M., Bouhadana, H., Bouissou, M., Lucas, J.Y., Magne, L., Villatte, N. (1990). Poste de travail basé sur l'intelligence artificielle pour les études de fiabilité.
Revue Française de Mécanique, Numéro spécial : les systèmes experts et la mécanique N° 1990-2 . ISSN 0373-6601.

/2/ Bouhadana, H. (1989). Méthodes d'amélioration qualitative d'un modèle de fiabilité.
Mémoire de DEA. EDF/Paris XIII. Sept. 1989.

/3/ Bouissou, M. (1986). Recherche et quantification automatiques de séquences accidentelles pour un système réparable.
5ème Congrès de Fiabilité et Maintenabilité de Biarritz (France). Oct. 1986.

/4/ Bouissou, M., Bouhadana, H.,Bannelier, M. (1990). Un moyen d'unifier diverses modélisation pour les études probabilistes :
le langage FIGARO.
HT-53/90-42A. EDF internal report.

/5/ Signoret, J.P. (1989). MOCA-RP batch, utilisation du logiciel, révision 0.
Internal report SNEA(P) DEA-SES-ARF, JPS/cl/n°89-71.

/6/ Terpstra, K., Dekker, N.H., Van Driel,G. (1986). PHAMISS, A Reliability Computer Program for phased Mission Analysis and Risk Analysis. User's Manual.
ECN-83. Netherlands Energy Research Foundation.

/7/ Villemeur, A., Bouissou, M., Dubreuil-Chambardel, A. (1987). Accident sequences: methods to compute probabilities.
International topical conference on PSA and Risk Management. (Zurich, Switzerland).

TABLE 1 : development tools and target machines

	Programming language	Machines and systems	Developer	AI techniques.
FIGARO language definition	----------	----------	EDF	Knowledge representation. Object language. Order 1 production rules.
Graphical Interface	LELISP AIDA	SUN 3 and 4 (UNIX/X11)	EDF	Object language.
Compilers (FIG0, GSI, fault tree)	LEX/YACC C LELISP	SUN 3 and 4 (UNIX)	EDF	Backward chaining (fault trees). Order 1 to order 0 rule transformation.
GSI V5.3	PASCAL VS PASCAL ISO	IBM 3090 (MVS), SUN 4, HP 9000 (UNIX), IBM PS (AIX)	EDF	Inference engine in forward chaining. Heuristic rules. Compilation of 0 order rules.
Dependence analysis	LEX/YACC C	SUN 4	EDF	----------
PHAMISS	FORTRAN 77	IBM 3090 (MVS), SUN 4 (UNIX), IBM PS (AIX)	ECN	----------
MOCA-RP	FORTRAN 77	IBM 3090 (MVS), SUN 4 (UNIX), IBM PC (MS.DOS)	ELF Aquitaine	----------

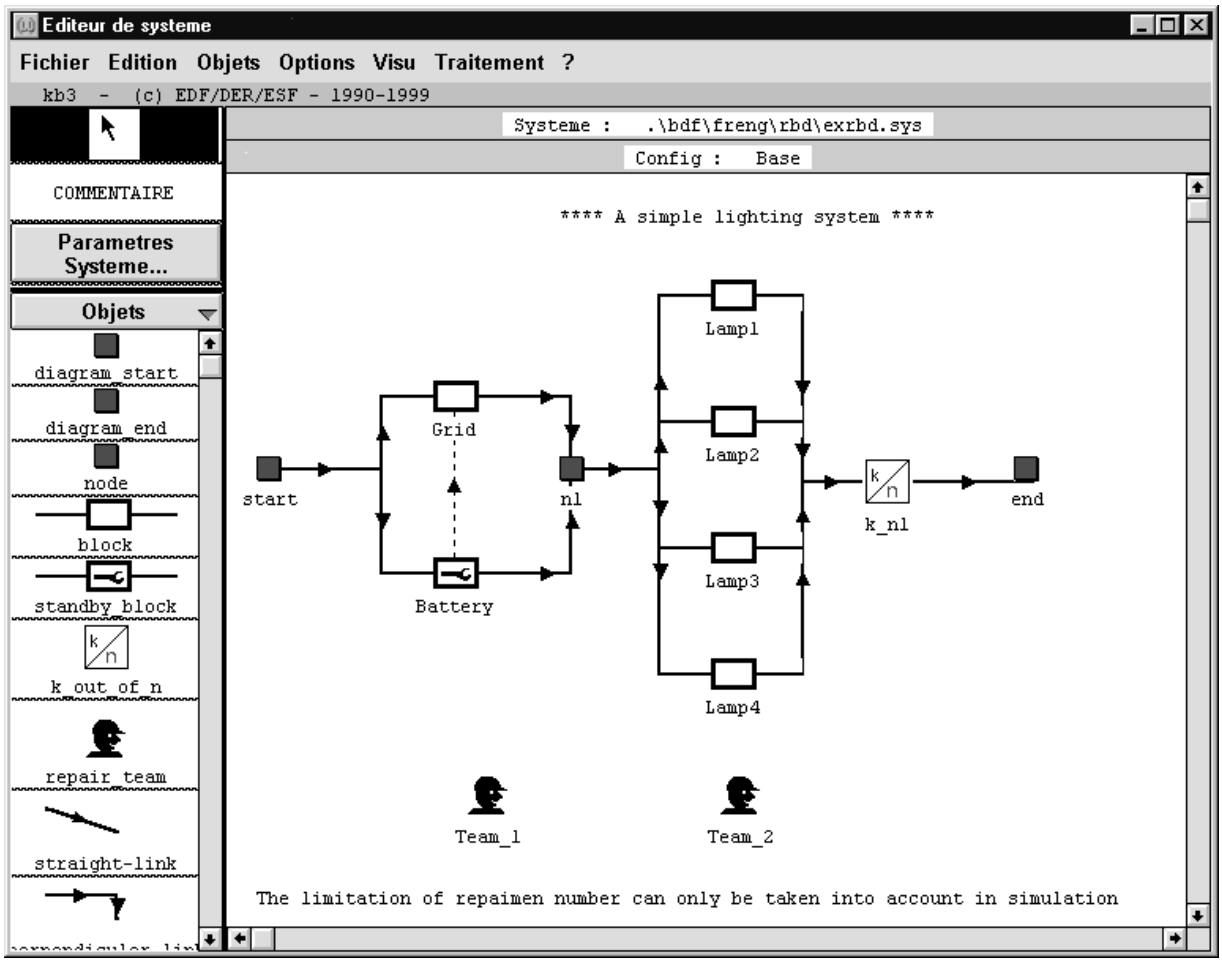


Fig. 1 : Graphic data acquisition for a reliability diagram

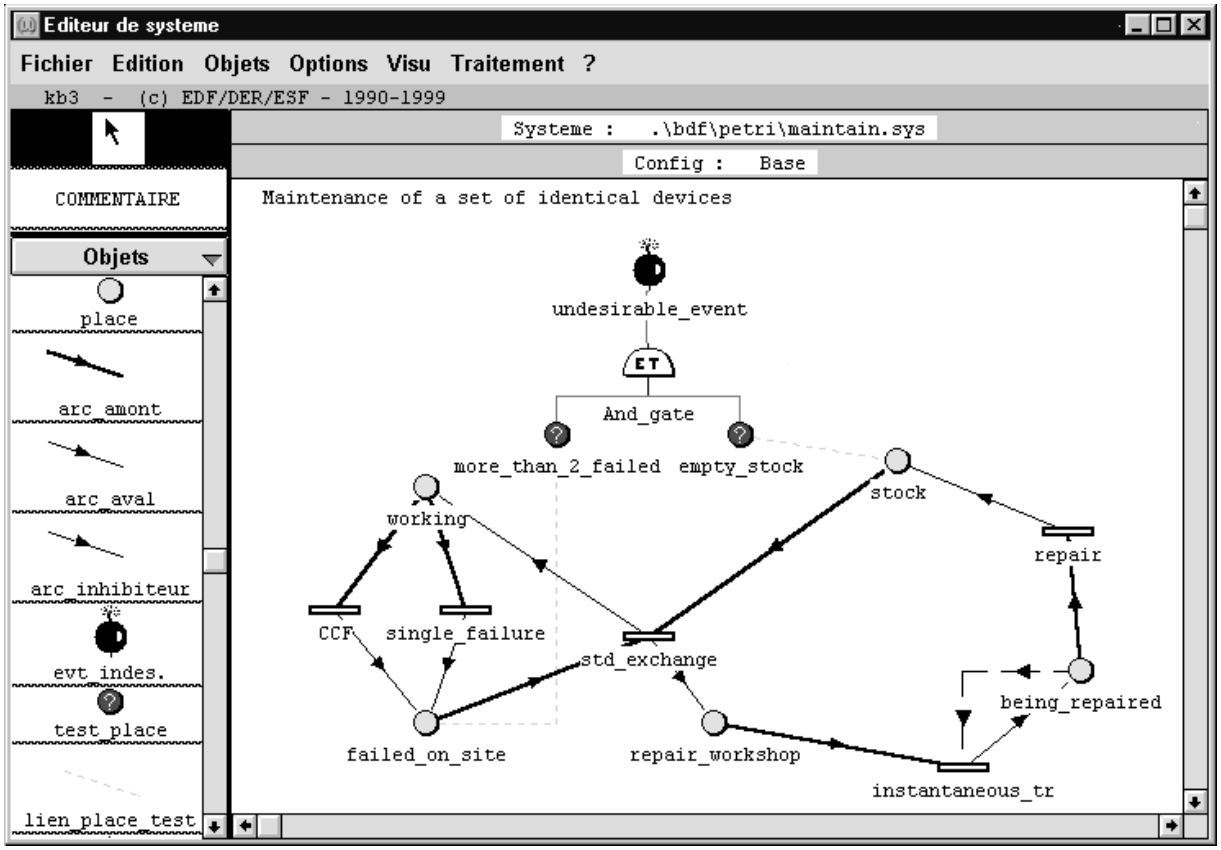


Fig. 2 : Graphic data acquisition for a Petri net

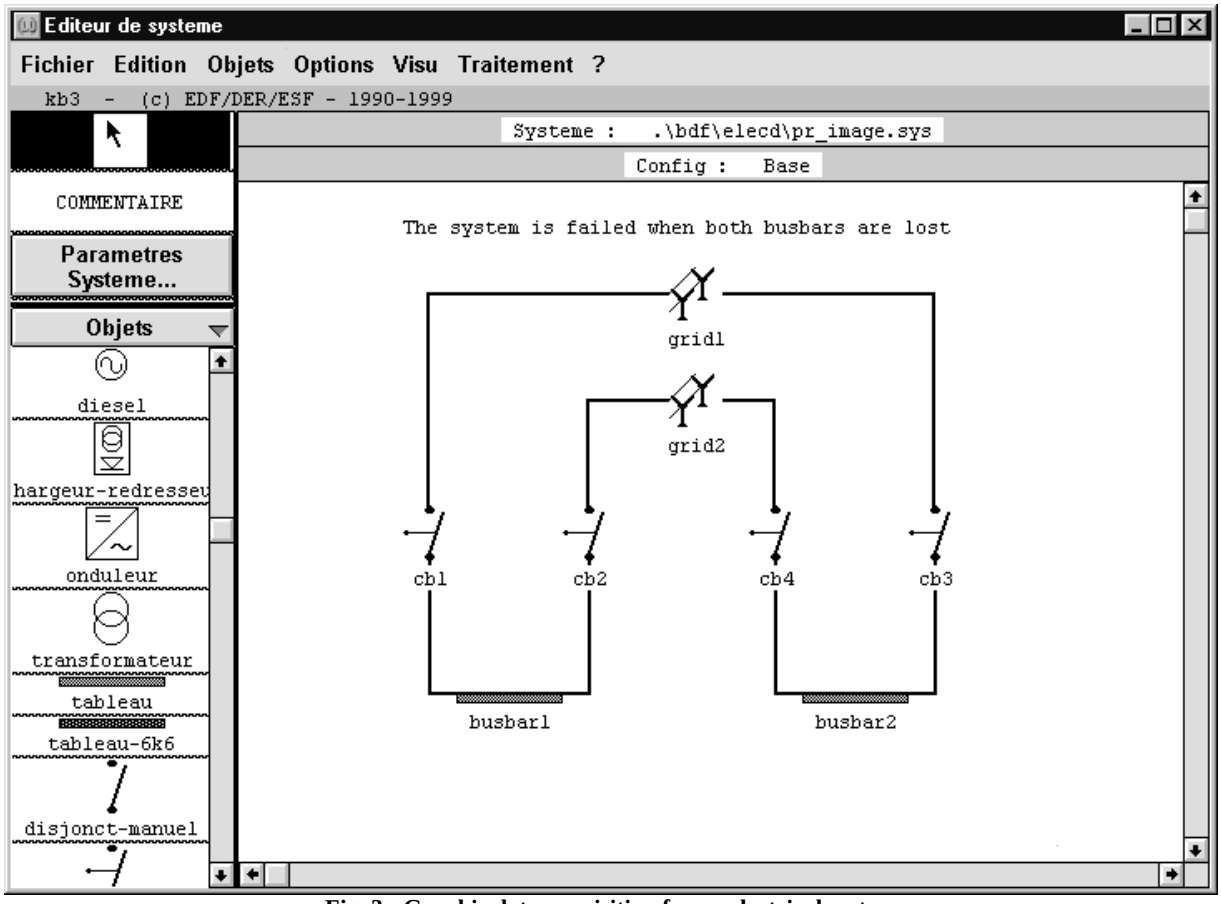


Fig. 3 : Graphic data acquisition for an electrical system

Seq. Number	Events	Ditrib. Parameter	Type	Asympt. Proba.	Mean duration	Contrib.
658	[grid1 fails]	1.0000e-04	EXP			
	[cb1 opens ,	9.9900e-01	INS			
	cb3 opens]	9.9900e-01	INS			
	[cb2 closes ,	9.9900e-01	INS			
	cb4 closes]	9.9900e-01	INS			
	[grid2 fails]	1.0000e-04	EXP	3.3101e-04	9.9701e+00	9.8230e-01
798	[grid1 fails]	1.0000e-04	EXP			
	[cb1 fails op.,	1.0000e-03	INS			
	cb3 fails op.]	1.0000e-03	INS	3.3333e-07	0.0000e+00	9.8919e-04
760	[grid1 fails]	1.0000e-04	EXP			
	[cb1 fails op.,	1.0000e-03	INS			
	cb3 opens]	9.9900e-01	INS			
	[cb4 fails cl.]	1.0000e-03	INS	3.3300e-07	0.0000e+00	9.8820e-04
797	[grid1 fails]	1.0000e-04	EXP			
	[cb1 opens ,	9.9900e-01	INS			
	cb3 fails op.]	1.0000e-03	INS			
	[cb2 fails cl.]	1.0000e-03	INS	3.3300e-07	0.0000e+00	9.8820e-04
723	[grid1 fails]	1.0000e-04	EXP			
	[cb1 opens ,	9.9900e-01	INS			
	cb3 opens]	9.9900e-01	INS			
	[cb2 fails cl.,	1.0000e-03	INS			
	cb4 fails cl.]	1.0000e-03	INS	3.3267e-07	0.0000e+00	9.8722e-04
759	[grid1 fails]	1.0000e-04	EXP			
	[cb1 fails op.,	1.0000e-03	INS			
	cb3 opens]	9.9900e-01	INS			
	[cb4 closes]	9.9900e-01	INS			
	[grid2 fails]	1.0000e-04	EXP	1.6617e-07	4.9950e+00	4.9312e-04

Fig. 4 : Sequences found by GSI for the system of Fig. 3.