Dependability evaluation. From functional to structural modelling

Claudia Betous-Almeida, Karama Kanoun

DEPENDABILITY EVALUATION : FROM FUNCTIONAL TO STRUCTURAL MODELLING

C. BETOUS-ALMEIDA, K. KANOUN

LAAS REPORT 01088

FEBRUARY 2001

Dependability Evaluation
From Functional to Structural Modelling

Cláudia Betous-Almeida and Karama Kanoun

LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4 - France
{almeida,kanoun}@laas.fr

Abstract. The work presented in this paper is devoted to the definition of a dependability modelling approach for the selection process of instrumentation and control systems (I&C) in power plants. We show how starting from functional specifications, a functional-level model can be transformed into a dependability model taking into account the system's architecture, following a progressive and hierarchical approach. This approach is illustrated on simple examples related to a specific architecture of an I&C system.

1 Introduction

Dependability evaluation plays an important role in critical systems' definition, design and development. Modelling can start as early as system functional specifications, from which a high-level model can be derived to help analysing dependancies between the various functions. However the information that can be obtained from dependability modelling and evaluation becomes more accurate as more knowledge about system implementation is integrated into the models. The aim of this paper is to show how starting from functional specifications, a functional-level model can be transformed into a dependability model taking into account the system's architecture, using a progressive modelling approach. The modelling approach has been applied to three different *instrumentation and control systems* (I&C) in power plants, to help selecting the most appropriate one. Due to space limitations, in this paper we illustrate it on a small part of one of them.

The remainder of the paper is organised as follows. Section 2 gives the context of our work. Section 3 is devoted to the presentation of the modelling approach. Section 4 presents a small example of application of the proposed approach to an I&C system and Section 5 concludes the paper.

2 Context of our Work

The process of defining and implementing an I&C system can be viewed as a multi-phase process (as illustrated in Figure 1) starting from the issue of a *Call*

for Tenders by the stakeholder. The call for tenders gives the functional and non-functional (i.e., dependability) requirements of the system and asks for candidate contractors to make offers proposing possible systems/architectures satisfying the specified requirements. A preliminary analysis of the numerous responses by the stakeholder, according to specific criteria, allows the pre-selection of two or three candidate systems. At this stage, the candidate systems are defined at a high level. They are usually based on *Commercial-off-the-Shelf* (COTS) components and the application software is not entirely written. The comparative analysis of the pre-selected candidate systems, in a second step, allows the selection of the most appropriate one. Finally, the retained system is refined and thoroughly analysed to go through the qualification process. Dependability modelling and evaluation constitute a good support for both the selection and the refinement processes, thorough analysis and preparation of the final system's qualification. The main purpose of our work is to help the stakeholder in this modelling process. To this end, we have defined a rigorous, systematic and hierarchical modelling approach that can be easily used to select an appropriate architecture and to model it thoroughly. Thus this approach can be used by any system's developer.

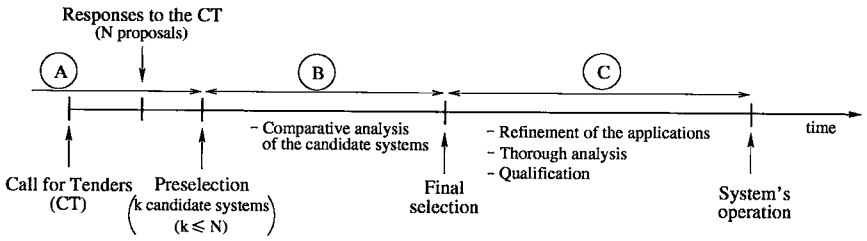


Fig. 1. Various steps of I&C definition process

3 Modelling Approach

Our modelling approach follows the same steps as the development process: It is also performed in three steps as described in Figures 1 and 2:

Step A. Construction of a functional-level model based on the system's specifications;

Step B. Transformation of the functional-level model into a high-level dependability model, based on the system's architecture. There is one for each pre-selected candidate system;

Step C. Refinement of the dependability model, based on the detailed architecture of the retained system.

Modelling is based on *Generalised Stochastic Petri Nets* (GSPN) due to their ability to cope with modularity and model refinement [1]. The GSPN model is processed to obtain the corresponding Markov chain. Dependability measures (i.e., availability, reliability, safety, ...) are obtained through the processing of the Markov chain, using an evaluation tool such as SURF-2 [3].

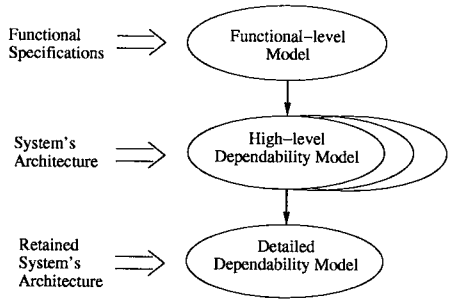


Fig. 2. Main steps of our modelling approach

3.1 Functional-Level Model

The system's functional-level model is the starting point of our method. This model is independent from the underlying system's architecture. Hence it can be done even before the call for tenders, by the stakeholder.

The system's functional-level model is formed by places which represent the possible states of functions. For each function, the minimal number of places is two (Fig. 3): One which represents the function's nominal state (F) and the other its failure state ($\overline{\mathrm{F}}$). Between these two states, we have the events that manage changes from F to $\overline{\mathrm{F}}$ and vice-versa. These events are inherent to the system's structure that is not specified in this step as it is not known yet. We call the model that contains these events and the corresponding places, the *link model* ($\mathrm{M_L}$). Note that the set $\{\mathrm{F}, \mathrm{M_L}, \overline{\mathrm{F}}\}$ that constitutes the system's GSPN model, will be completed once the architecture system is known[1].

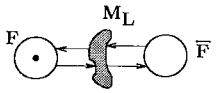
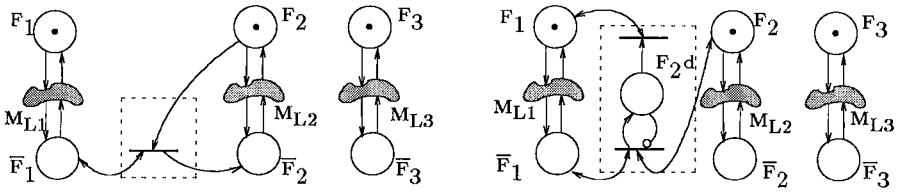


Fig. 3. Functional-level model related to a single function

Most of the times though, systems perform more than one function. In this case we have to look for dependancies between these functions due to the communication between them. We distinguish two degrees of dependancy. Figure 4 illustrates the two types of functional dependancy between two functions $\mathrm{F_1}$ and $\mathrm{F_2}$. $\mathrm{F_3}$ is independent of both $\mathrm{F_1}$ and $\mathrm{F_2}$.

Case (a) *Total dependancy* – $\mathrm{F_2}$ depends totally on $\mathrm{F_1}$, noted $\mathrm{F_2} \leftarrow\!\!\!\rho\ \mathrm{F_1}$. In this case, if $\mathrm{F_1}$ fails, $\mathrm{F_2}$ also fails, i.e. $(\mathcal{M}(\overline{\mathrm{F}_1}) = 1) \Rightarrow (\mathcal{M}(\overline{\mathrm{F}_2}) = 1)$, where $\mathcal{M}(\mathrm{F})$ represents the marking of place F;

[1] This modelling approach is applicable in the same manner when there are several failure modes per function.



(a) $F_2 \nleftrightarrow F_1$ (total)

(b) $F_2 \hookleftarrow F_1$ (partial)

Fig. 4. Types of functional dependancies

Case (b) *Partial dependancy* – F_2 depends partially on F_1, noted $F_2 \hookleftarrow F_1$. In this case, although F_1's failure does not induce F_2's failure, i.e. $(\mathcal{M}(\overline{F}_1) = 1) \not\Rightarrow (\mathcal{M}(\overline{F}_2) = 1)$, F_2 is affected. In fact, F_1's failure puts F_2 in a degraded state that is represented by place F_{2d}. F_{2d} will be marked whenever F_1 is in its failure state and F_2 in its nominal one, i.e. $\mathcal{M}(F_{2d}) = 1 \Leftrightarrow (\mathcal{M}(\overline{F}_1) = 1) \wedge (\mathcal{M}(F_2) = 1)$.

3.2 Link Model

The link model gathers the set of states and events related to the architectural behaviour of the system. The first step in constructing this model consists on the identification of the components associated with the system's functions. For modelling purposes, consider the following complete set of cases:

Case A. One function: In this case, several situations may be taken into account. A function can be done by:
- A.1. A *single* software component on a *single* hardware component;
- A.2. *Several* software components on a *single* hardware component;
- A.3. A *single* software component on *several* hardware components;
- A.4. *Several* software components on *several* hardware components;

Case B. Several functions: Again two situations can take place:
- B.1. The functions have no common components;
- B.2. The functions have some common components.

To illustrate the given situations, we will consider a simple example for each case. Here we give only an overview of the structure of the link model. Note that the structural models presented in this section are not complet. More information is given in sections 3.3 et 3.4.

Case A. Case of a single function.

A.1. Let us suppose function F carried out by a software component S and a hardware component H – Figure 5. Then, F and $\overline{F}$ markings depend upon the markings of the hardware and software component models. More specifically:

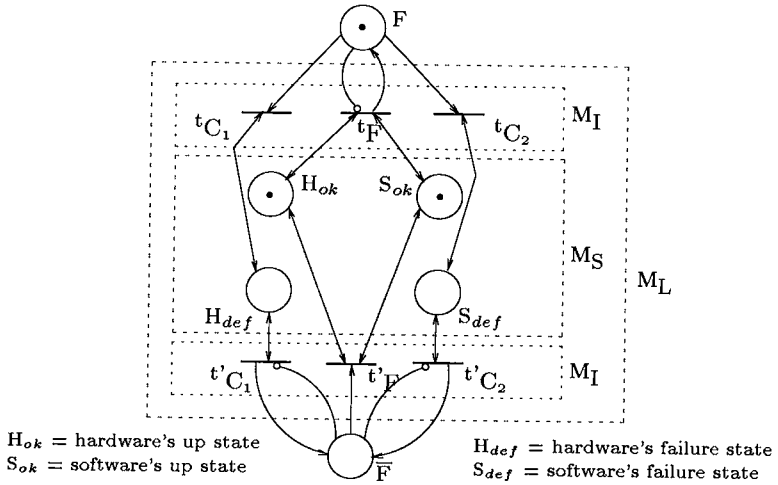


Fig. 5. Interface model of a function executed by 2 components

- F's up state is the combined result of H's up state and S's up state.
- F's failure state is the result of H's failure or S's failure.

The behaviour of H and S is modelled by the so-called *structural model* (M_S) and then it is connected to F and $\overline{F}$ through an *interface model* referred to as M_I. The link model (M_L) is thus made up of the structural model (M_S) and of the interface model (M_I): $M_L = M_S + M_I$. This interface model connects hardware and software components with their functions by a set of immediate transitions. Note that there is only one interface model but to make its representation easier, we split it into two parts: An upstream part and a downstream part.

Case A.2. Consider function F done by two software components S_1 and S_2 on a hardware component H, in which case we have to consider two situations:

- S_1 and S_2 *redundant* (Fig. 6(a))
  i. F's up state is the combined result of H's up state and S_1 or S_2's up states:

$$\mathcal{M}(F) = 1 \equiv (\mathcal{M}(H_{ok}) = 1 \ \wedge \ [\mathcal{M}(S_{1ok}) = 1 \ \dot{\vee} \ \mathcal{M}(S_{2ok}) = 1])$$

  ii. F's failure state is the result of H's failure or S_1's failure and S_2's failure:

$$\mathcal{M}(\overline{F}) = 1 \equiv (\mathcal{M}(H_{def}) = 1 \ \dot{\vee} \ [\mathcal{M}(S_{1def}) = 1 \ \wedge \ \mathcal{M}(S_{2def}) = 1])$$

- S_1 in *series* with S_2 (Fig. 6(b))
  i. F's up state is the combined result of H, S_1 and S_2's up states:

$$\mathcal{M}(F) = 1 \equiv (\mathcal{M}(H_{ok}) = 1 \ \wedge \ \mathcal{M}(S_{1ok}) = 1 \ \wedge \ \mathcal{M}(S_{2ok}) = 1)$$

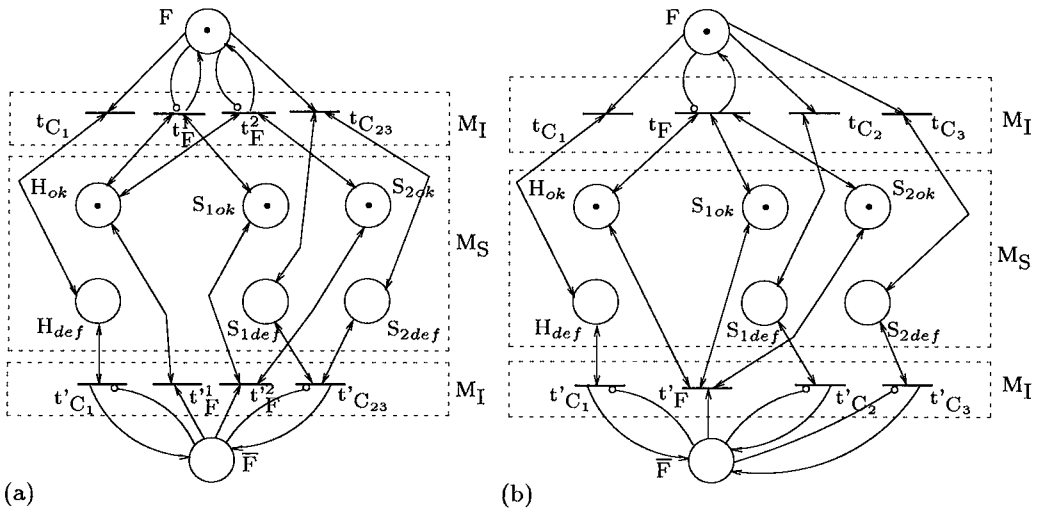


Fig. 6. Link model of F done by two software components on a hardware component

ii. F's failure state is the result of H's failure or S_1 or S_2's failure:

$$\mathcal{M}(\overline{F}) = 1 \equiv (\mathcal{M}(H_{def}) = 1 \dot{\vee} \mathcal{M}(S_{1def}) = 1 \dot{\vee} \mathcal{M}(S_{2def}) = 1)$$

A.3. The case of function F done by a single software on several hardware components, is essentially similar to the previous case;

A.4. Suppose function F done by a set of N components:

i. If all components, under the same conditions, have different behaviours, then the structural model will have N initial places. This case corresponds to a generalisation of Case A.1.

ii. If some of the N components, under the same conditions, have *exactly* the same behaviour, their structural models are grouped. In this case, the structural model will have Q initial places (Q < N).

Case B. Consider two functions (the generalisation is straight forward) and let $\{C_{1i}\}$ (resp. $\{C_{2j}\}$) be the set of components associated to F_1 (resp. F_2).

B.1. F_1 and F_2 have no common components, $\{C_{1i}\} \cap \{C_{2j}\} = \emptyset$. The interface models related to F_1 and F_2 are built separately in the same way as explained for a single function.

B.2. F_1 and F_2 have some common components, $\{C_{1i}\} \cap \{C_{2j}\} \neq \emptyset$. This case is illustrated on a simple example:

- F_1 done by three components: A hardware component H and two software components S_{11} and S_{12}. F_1 corresponds to case (a) of Figure 6.
- F_2 done by two components: The same hardware component H as for F_1 and a software component S_{21}. F_2 corresponds to Case A.1. of Figure 5.

Their model is given in Figure 7. It can be seen that i) both interface models (M_{I1} and M_{I2}) are built separatly in the same way as done before, and ii) in the global model, the common hardware component H is represented only once by a common component model.

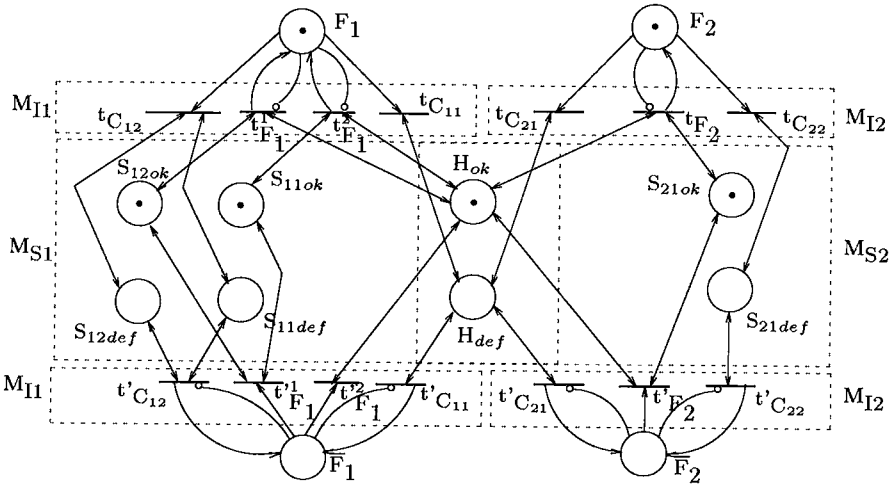


Fig. 7. Example of two functions with a structural dependancy

3.3 Interface Model

The interface model M_I connects the system components with their functions by a set of transitions. This model is a key element in our approach. It has been defined to be constructed in a systematic way in order to make the approach re-usable and to facilitate the construction of several models related to various architectures. Moreover, it has been defined in formal terms. The main rules are stated in an informal manner in this paper.

Both parts of the M_I have the same number of immediate transitions and the arcs that are connected to these transitions are built in a systematical way:

- Upstream M_I: It contains one function transition t_F for each series (set of) component(s) to mark the function's up state place and one component transition t_{Cx} for each series, distinct component that has a direct impact on the functional model, to unmark the function's up state place.
  - Each t_F is linked by an inhibitor arc to the function's up state place, by an arc to the function's up state place and by one bidirectional arc to each initial (ok) component place;
  - Each t_{Cx} is linked by an arc to the function's up state place and by one bidirectional arc to each failure component place.
- Downstream M_I: It contains one function transition t'_F for each series (set of) component(s) to unmark the function's failure state place and one component transition t'_{Cx} for each series, distinct component that has a direct impact on the functional model, to mark the function's failure state place.

- Each t'_F is linked by an arc to the function's failure state place and by one bidirectional arc to each initial (ok) component place;
- Each t'_{Cx} is linked by an inhibitor arc to the function's failure state place, by an arc to the function's failure state place and by one bidirectional arc to each component failure place.

3.4 Structural Model

In order to build the interface between the functional and the structural models, we need to identify the components implementing each function, and thus the initial places as well as their failure state places of the structural model.

The structural model can be built by applying one of the many existing modular modelling approaches (see e.g., [4–7]).

To complete the above examples, let us consider the simple case of Figure 5. The associated structural model is given in Figure 8 in which the S_{def} place of Figure 5, corresponds to either place S_{ed} or S_{ri}. The following assumptions and notations are used:

- The activation rate of a hardware fault is λ_h (Tr_1) and of a software fault is λ_s (Tr_6);
- The probability that a hardware fault is temporary is t (tr_1). A temporary fault will disappear with rate ε (Tr_2);
- A permanent hardware fault (resp. software) is detected by the *fault-tolerance mechanisms* with probability d_h (resp.d_s for software faults). The detection rate is δ_h (Tr_3) for the hardware and δ_s (Tr_7) for the software;
- The effects of a non detected error are perceived with rate π_h (Tr_4) for the hardware and rate π_s (Tr_8) for the software;
- Errors detected in the hardware component require its repair: repair rate is μ (Tr_5);
- Permanent errors in the software may necessitate only a reset. The reset rate is ρ (Tr_9) and the probability that an error induced by the activation of a permanent software fault disappears with a reset is r (tr_7);
- If the error does not disappear with the software reset, a re-installation of the software is done. The software's re-installation rate is σ (Tr_{10}).

Note that a temporary fault in the hardware may propagate to the software (tr_{11}) with probability p. We stress that when the software component is in place S_{ed} or S_{ri}, it is in fact not available, i.e., in a failure state.

Also when the hardware is in the repair state, the software is on hold. The software will be reset or re-installed as soon as the hardware repair is finished. Due to the size of the subsequent model, this case is not represented here.

4 Application to I&C Systems

An I&C system performs five main functions: *Human-machine interface* (HMI), *processing* (PR), *archiving* (AR), *management of configuration data* (MD), and

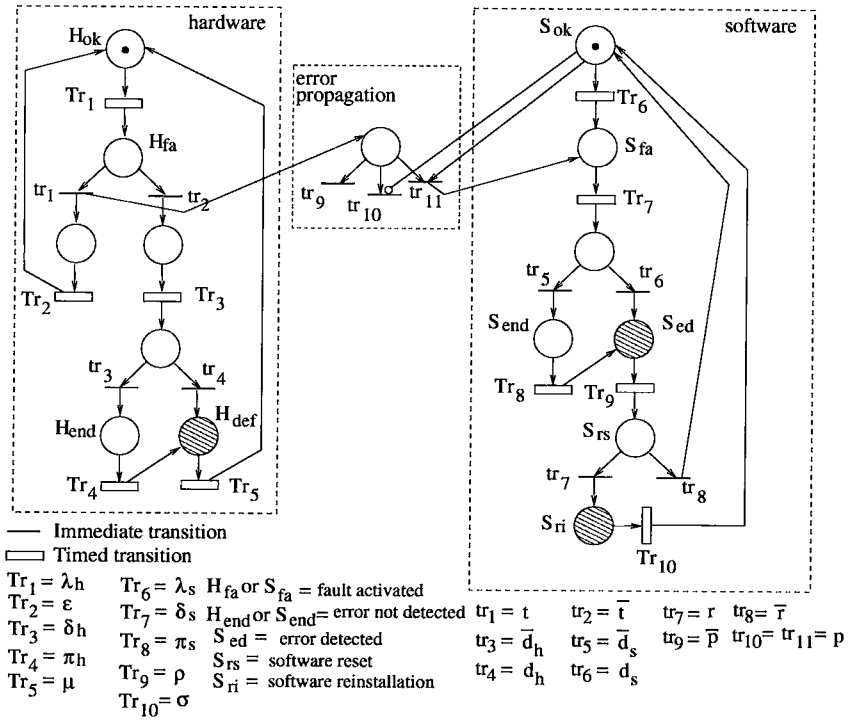


Fig. 8. Structural model of a software and a hardware components

interface with other parts of the I&C system (IP). The functions are linked by the partial dependencies given in column 1 of Table 1.

Taking into account the fact that a system's failure is defined by:

$$\mathcal{M}(\overline{\text{HMI}}) = 1 \ \vee \ \mathcal{M}(\overline{\text{PR}}) = 1 \ \vee \ \mathcal{M}(\overline{\text{IP}}) = 1$$

the above dependancies can be simplified as given in column 2 of Table 1.

Table 1. Functional dependancies of I&C systems

Function dependancies	Simplified funct. dependancies
HMI $\leftrightarrow$ {PR, AR, MD}	HMI $\leftrightarrow$ {AR, MD}
PR $\leftrightarrow$ {HMI, MD, IP}	PR $\leftrightarrow$ MD
AR $\leftrightarrow$ {HMI, MD}	AR $\leftrightarrow$ MD
IP $\leftrightarrow$ {PR, MD}	IP $\leftrightarrow$ MD

These relations are translated by the functional model depicted in Figure 9.

To illustrate the second step of our modelling approach, we consider the example of the I&C system used in [2]. This system is composed of five nodes connected by a *Local Area Network* (LAN). The mapping between the various nodes and functions is given in Figure 10. Note that while HMI is executed on

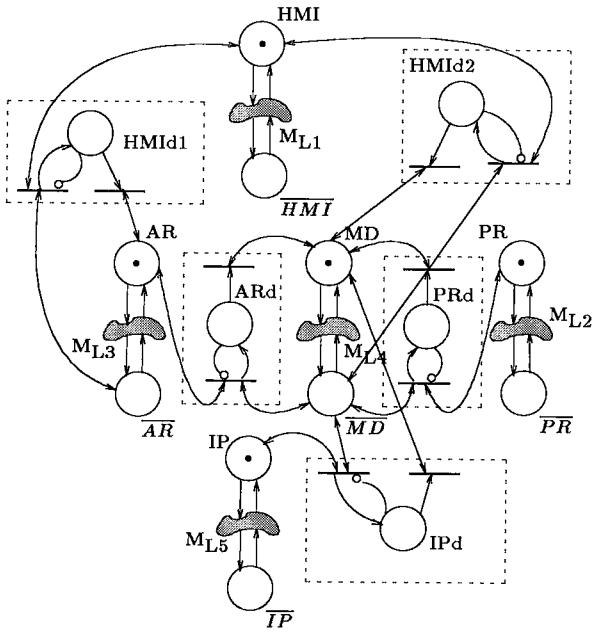


Fig. 9. Functional model for I&C systems

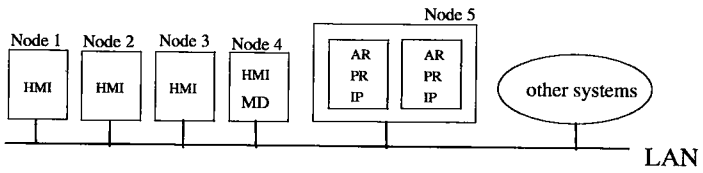


Fig. 10. I&C architecture

four nodes, node 5 runs three functions. Nodes 1 to 4 are composed of one computer each. Node 5 is fault-tolerant: It is composed of two redundant computers. The structural model of this I&C is built as follows:

- Node 1 to Node 3 – in each node, a single function is achieved by one software component on a hardware component. Its model is similar to the one presented in Figures 5 and 8;
- Node 4 – has two independent functions. Its structural model will be similar to the one depicted in Figure 7, followed by a model slightly more complex than the one of Figure 8;
- Node 5 – is composed of two hardware components with three independent functions each. Its structural model is more complex than the previous one due to the redundancy. A part of this model has been presented in [2].
- LAN – the LAN is modelled at the structural level by the new structural dependencies that it creates.

5 Conclusions

In this paper a three step modelling approach has been presented. This approach is progressive and hierarchical and can easily be used to select and thoroughly model an appropriate architecture. The functional-level and the structural models are linked by an interface model that is constructed in a formal way. This interface model plays a central role in our modelling approach.

Although we have presented in this paper the application of our approach to a small part of an I&C system, the approach has been applied to two other I&C systems to identify their strong and weak points.

The work is still in progress. In particular, the refinement of the dependability model with the formal definition of refinement rules is under study. This will help in the third step of the modelling approach for thorough analysis of the retained system.

References

1. Ajmone Marsan, M., Balbo,G., Conte,G., Donatelli,S., and Franchescinis,G., *Modelling with Generalized Stochastic Petri Nets*, Series in Parallel Computing, Wiley (1995).
2. Almeida, C., Arazo, A., Crouzet, Y., and Kanoun, K., "Dependability of Computer Control Systems in Power Plants: Analytical and Experimental Evaluation", in *Lecture Notes in Computer Science*, vol. 1943, Springer Verlag (2000) 165–175.
3. Béounes, C., and *al.* "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems", in *Proc. 23rd. Int. Symp. on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France (1993) 668–673.
4. Bondavalli, A., Mura, I., and Trivedi, K.S., "Dependability Modelling and Sensitivity Analysis of Scheduled Maintenance Systems", in *Proc. 3rd European Dependable Computing Conf. (EDCC-3)*, Lecture Notes in Computer Science, vol. 1667, Springer Verlag (1999) 7–23.
5. Fota, N., Kaâniche, M., Kanoun, K., and Peytavin, P., "Safety Analysis and Evaluation of an Air Traffic Control System", in *Proc. 15th Int. Conf. on Computer Safety, Reliability and Security SAFECOMP'96*, Vienna, Austria, (1996), 219–229.
6. Kanoun, K., Borrel, M., Morteveille, T., and Peytavin,A., "Availability of CAUTRA, a Subset of the French Air Traffic Control System", in *IEEE Trans. on Computers*, vol. 48, n. 5, May (1999), 528–535.
7. Rabah, M., and Kanoun, K., "Dependability Evaluation of a Distributed Shared Memory Multiprocessor System", in *Proc. 3rd European Dependable Computing Conf. (EDCC-3)*, Lecture Notes in Computer Science, vol. 1667, Springer Verlag (1999) 42–59.