Dependability modelling of instrumentation and control systems: a comaprison of competing architectures

Claudia Betous-Almeida, Karama Kanoun

Dependability Modelling of Instrumentation and Control Systems

A Comparison of Competing Architectures

Cláudia Betous-Almeida and Karama Kanoun

LAAS-CNRS

7, Avenue du Colonel Roche

31077 Toulouse Cedex 4 - France

{almeida,kanoun}@laas.fr

Abstract. The purpose of this paper is to present a framework for comparing different candidate architectures for the same system. To this end, we propose a rigorous approach for homogeneously modelling different architectures. Starting with the functional specifications of the system, we derive a functional-level model that is used to construct a high-level dependability model for each architecture, using well-defined, formal construction rules. Our modelling approach is then applied to three possible architectures of an instrumentation and control system, and an example of a comparative analysis of these systems is provided.

1 Introduction

The process of defining and implementing an *instrumentation and control* (I&C) system can be viewed as a multi-phase process, 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. Several systems' suppliers respond by proposing potential systems satisfying the specified requirements. In a first step, a pre-selection process, according to particular criteria, allows the stakeholder to keep two or three candidate systems identified as the most suitable ones. The comparative analysis of the pre-selected candidate systems, in a second step, allows the selection of the most appropriate one, referred to as the *retained system*. The latter 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 of the retained system. The main purpose of our work is to help the

stakeholder in this modelling task. Without a well-defined approach, modelling can be tedious and error prone. To this end, we have defined a rigorous, systematic and progressive modelling approach that can be easily used to select the most appropriate system and to model it thoroughly. Thus, this approach can also be used by any system's developer, even in a different system design and implementation process, based on comparison of potential systems.

Modelling can start as early as system functional specifications, from which a functional-level model is derived. A high-level dependability model is then constructed for each candidate system, based on the functional-level model and on the knowledge of the system's structure. The model of the retained system can be refined to include more details about the system, in order to obtain accurate results of the dependability evaluation.

This paper concentrates on the construction of the dependability models of the candidate systems, to select the most appropriate one. Three different I&C systems are used to illustrate our approach. In order to simplify the construction of each high-level dependability model, a library of basic models is developed. This library allows modelling of all components of the three candidate systems, with minimal modifications. Finally, an example of a comparative analysis of the systems is presented.

This paper elaborates on our previous work [3, 5]. [3] is devoted only to the construction of the high-level dependability model from the functional-level model, and [5] puts more emphasis on the refinement of the high-level dependability model into a detailed model. Neither of them refers to the comparison of the three I&C systems, particularly to the library of basic models and the results of the comparative analysis. The complete study is presented in [4].

The remainder of the paper is organised as follows. Section 2 is devoted to the presentation of the modelling approach. Section 3 presents the three I&C systems of our study. The application of the proposed approach to these systems is given in Section 4, and Section 5 concludes the paper.

2 Modelling Approach

Our modelling approach follows the same steps as the I&C system design and implementation process: It is also performed in three steps as described in Figure 1.

Step 1. Derivation of a functional-level model based on the system's specifications.

Step 2. Construction a high-level dependability model, based on the functional-level model and on the knowledge of the system's structure. There is one for each pre-selected candidate system. The aim of this step is to compare the pre-selected candidate systems, based on dependability evaluation.

Step 3. Refinement of the high-level dependability model, based on the detailed architecture of the retained system. The aim is to evaluate dependability figures as accurate as possible and to make sensitivity studies with respect to detailed system implementation.

As highlighted in Figure 1, this paper specifically addresses the first and second steps.

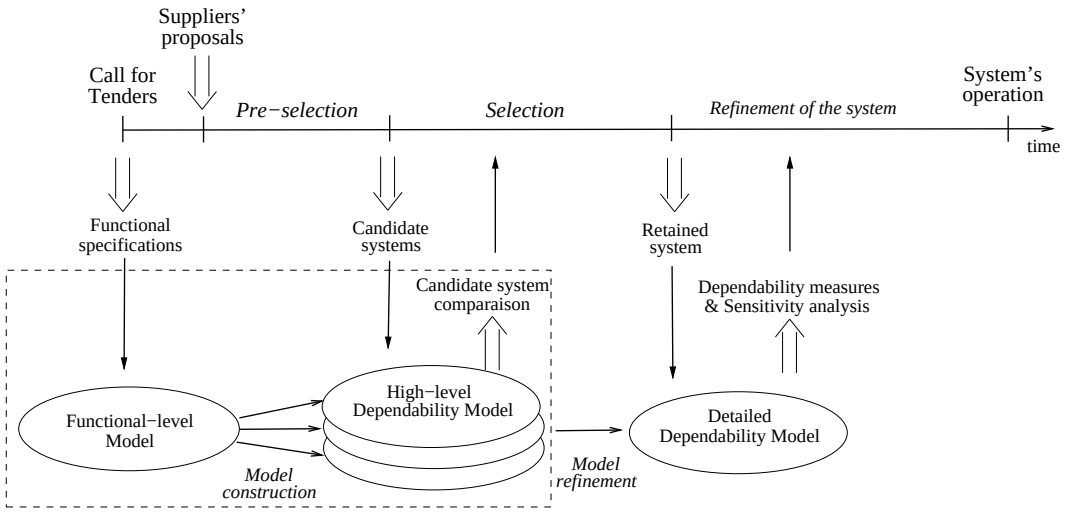


Fig. 1. Various steps of I&C definition and implementation, and modelling

The functional-level model gives a global overview of system's functions and helps analysing dependencies between the various functions. It is composed of a set of states corresponding to the functions' accomplishment states and a link model that is not known nor specified at this stage. The link model can only be built once the structure of the system is specified. Once it is built, the functional-level model becomes a dependability model. The link model is composed of the structural model (modelling the system's architectural behaviour in presence of faults) and an interface model between the functions' states (functional-level model) and the structural model.

The functional-level model cannot be processed *per-se*. Only the dependability models can be processed to obtain dependability measures. Dependability models are 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 associated dependability measures (i.e., availability, reliability, safety, ...) using an evaluation tool such as SURF-2 [2], DEEM [7], SPN [8] or UltraSAN [14].

In the rest of this section, we will first describe the functional-level model, then give a global overview of the link model before addressing successively the interface and structural models.

2.1 Functional-level Model

The system's functional-level model is the starting point of our approach. This model is independent from the underlying system's structure. Hence, it can be built as early as the call for tenders.

The system's functional-level model is formed by places representing the possible states of functions. For each function F[1], the minimal number of places is two (Figure 2): One representing the function's nominal state (F) and the other its failure state ($\overline{F}$). Between these two states, we have the events that manage changes from F to $\overline{F}$ 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* (M_L). Note that the set {F, M_L, $\overline{F}$} that constitutes the system's GSPN model, will be completed once the architecture system is known[2].

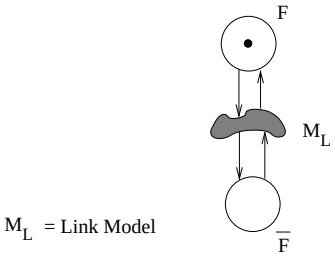


Fig. 2. Functional-level model related to a single function

Most of the times though, systems perform more than one function. If this is the case, one has to look for dependencies between these functions due to the communication between them. We distinguish two degrees of dependency: Total dependency and partial dependency.

[1] For the sake of simplicity, the nominal place of the model, associated with a function, has the same name as the function itself.

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

Case (a) *Total dependency* – F_2 totally depends on F_1, noted $F_2 \leftsquigarrow F_1$. In this case, if F_1 fails, F_2 also fails, i.e. $\left(\mathcal{M}\left(\overline{F}_1\right) = 1\right) \Rightarrow \left(\mathcal{M}\left(\overline{F}_2\right) = 1\right)$, where $\mathcal{M}(F)$ represents the marking of place F. This means that the probability that F_2 fails equals the probability that F_1 fails, times the probability that F_2 fails due to the failure of its components;

Case (b) *Partial dependency* – 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. $\left(\mathcal{M}\left(\overline{F}_1\right) = 1\right) \not\Rightarrow \left(\mathcal{M}\left(\overline{F}_2\right) = 1\right)$, 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. $\mathcal{M}(F_{2d}) = 1 \Leftrightarrow \left(\mathcal{M}\left(\overline{F}_1\right) = 1\right) \wedge \left(\mathcal{M}(F_2) = 1\right)$. This case is illustrated in Figure 3.

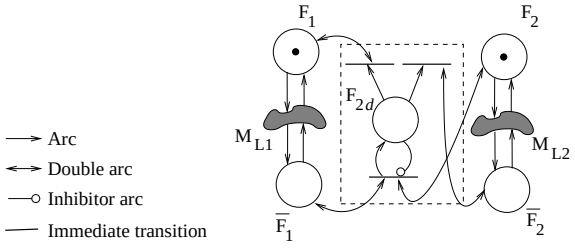


Fig. 3. Partial functional dependency ($F_2 \hookleftarrow F_1$)

2.2 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 in the identification of the components associated with the system's functions. Thus, we identify the nominal and the unavailability states of each component. We then built the interface between the functional-level and structural models. To illustrate the transition from the functional-level model to the structural one, we consider the following complete set of cases:

Case A. Single function: In this case, several situations may be taken into account. A function can be performed 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;

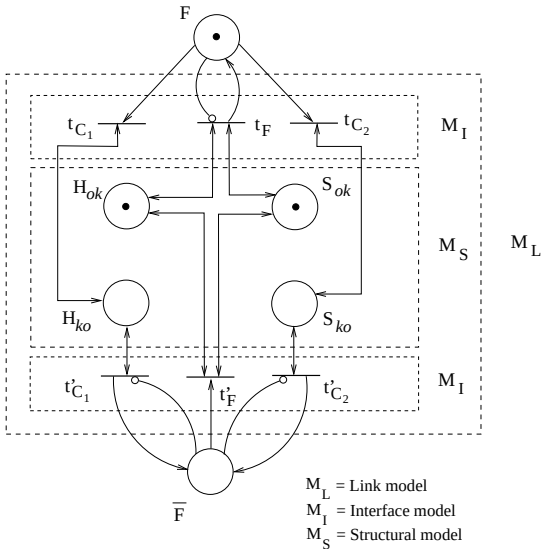
B.2. The functions have some common components.

Note that the models presented in this section are not complete. Rather, we put emphasis on the interface between the functional-level and the structural models. Examples of complete models are given in Section 4.

Case A. Single function.

A.1. Let us suppose function F carried out by a software component S and a hardware component H – Figure 4. Then, F and $\overline{F}$ markings depend upon the markings of the hardware and software component models. More specifically:

- 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 unavailability or S's unavailability.



State	Definition
H_{ok}	hardware's up state
H_{ko}	hardware's unavailability state
S_{ok}	software's up state
S_{ko}	software's unavailability state

Fig. 4. Interface model of a function executed by 2 components

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.

A.2. Consider function F performed 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* (Figure 5(a))

  i. F's up state is the combined result of H's up state and S_1 or S_2's up states:

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

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

$$\mathcal{M}(\overline{F}) = 1 \Leftrightarrow (\mathcal{M}(H_{ko}) = 1 \ \vee \ [\mathcal{M}(S_{1ko}) = 1 \ \wedge \ \mathcal{M}(S_{2ko}) = 1])$$

- S_1 in *series* with S_2 (Figure 5(b))

  i. F's up state is the combined result of H, S_1 and S_2's up states:

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

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

$$\mathcal{M}(\overline{F}) = 1 \Leftrightarrow (\mathcal{M}(H_{ko}) = 1 \ \vee \ \mathcal{M}(S_{1ko}) = 1 \ \vee \ \mathcal{M}(S_{2ko}) = 1)$$

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

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

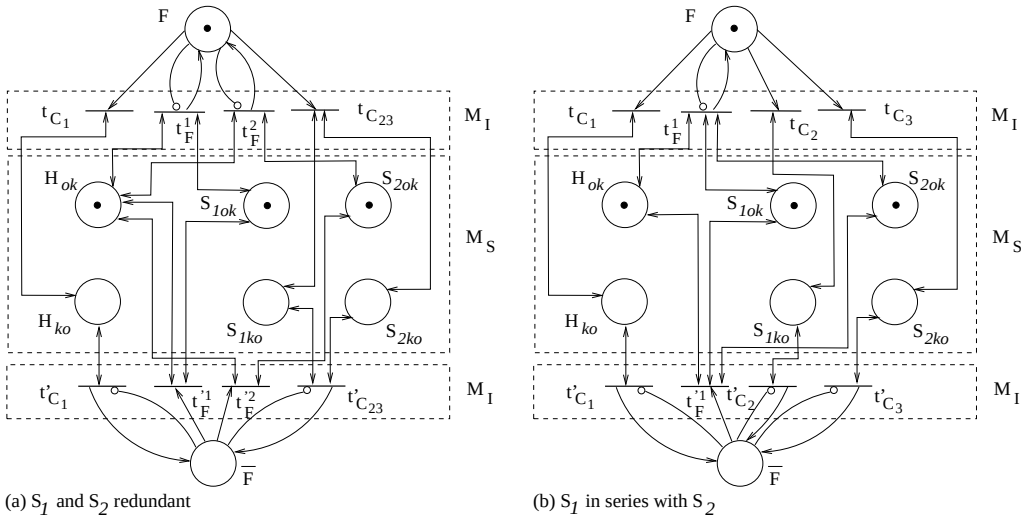


Fig. 5. Link model of F performed by two software components on a hardware component

i. If all components, under the same conditions, have different behaviours, then the structural model will have N nominal 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 nominal places (Q < N).

Case B. Consider two functions (the generalisation is straightforward) 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 performed by three components: A hardware component H and two redundant software components S_{11} and S_{12}. F_1 corresponds to case (a) – Figure 5.
- F_2 performed 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 4.

Their model is given in Figure 6. It can be seen that i) both interface models (M_{I1} and M_{I2}) are built separately in the same way as before, and ii) in the global model, the common hardware component H is represented only once by a common component model.

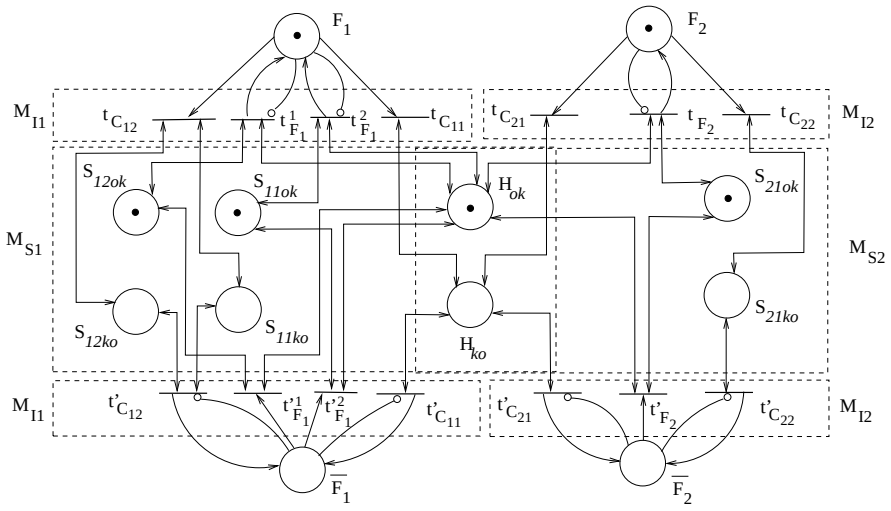


Fig. 6. Example of two functions executed on the same hardware component

2.3 Interface Model

The interface model M_I connects the system's component states with their function states by a set of transitions. This model is a key element in our approach. It can be built in a systematic way in order to make the approach re-usable and to facilitate the construction of several models related to various systems.

In this section the general organisation of the interface model is presented. Interfacing rules have been defined in formal terms [4]. However, the main rules are stated, in an informal manner, hereafter.

Upstream and downstream M_I have the same number of immediate transitions and the arcs that are connected to these transitions are built in a rational way:

- **Upstream M_I:** It contains one function transition t_F^i 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^i 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's 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's place.

- **Downstream $\mathbf{M_I}$:** It contains one function transition t'^{i}_{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'^{i}_{F} is linked by an arc to the function's failure state place and by one bidirectional arc to each initial (ok) component's place;
  - Each t'_{Cx} is linked by an inhibitor arc to the function's failure state place, by an arc from the function's failure state place and by one bidirectional arc to each component's failure place.

2.4 Structural Model

In order to build the interface between the functional-level and the structural models, we have identified the components implementing each function, and thus the components' nominal and unavailability state places in the structural model.

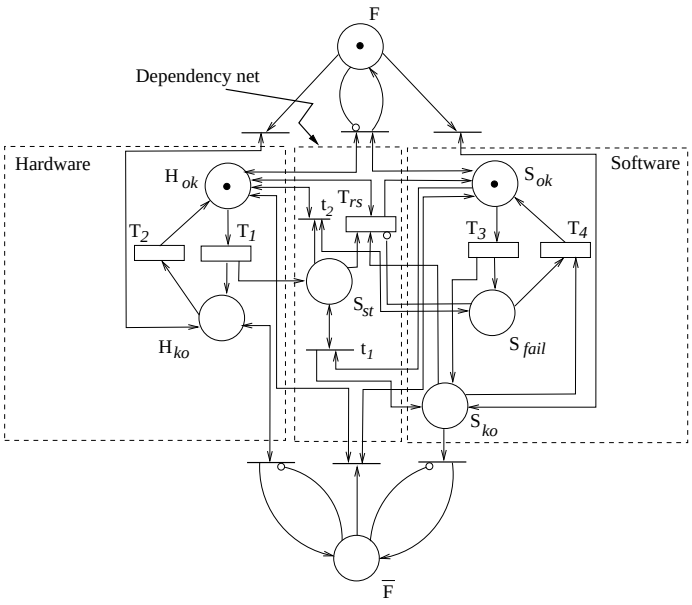
For several reasons, the first structural model that is built, starting from the functional-level model, may not be very detailed. One of these reasons could be the lack of information in the early system's definition and implementation phases. Another reason could be the complexity of the system to be modelled. Indeed, to master this complexity, the structural model is built at a high-level of detail using any of the many existing modular modelling approaches (see e.g., [6, 9, 10, 13]), and then refined progressively.

Figure 7 represents a simple example of a high-level dependability model composed of two components: A hardware component and a software component. It corresponds to the complete model of Figure 5. For each of these components, we considered two states: Nominal and unavailability. Transitions between these two states are ruled by events of failure (transitions T_1 and T_3) and restoration (transitions T_2 and T_4). These are *timed* transitions. It is worth noting that there is a dependency between these two components. Indeed, when a hardware component's failure occurs, the software component is stopped (immediate transition t_1 and place S_{st}). This last one will be restarted once the restoration of the hardware component is completed. Also, we took into account the possible hardware's failure after the failure of the software. In this case:

- If the software component is restored before the hardware's restoration is completed, it will be put on hold until the hardware is up again. Then, and just then, the software component will be restarted;

- If the hardware component is restored before the software component, then the token from place S_{st} will be removed. This is modelled by immediate transition t_2.

Note that we consider two unavailable states for the software component: The fail state, corresponding to place S_{fail}, and the software's stop after a hardware's failure, corresponding to place S_{st}. The interface between the structural and the functional-level model stays unchanged due to place S_{ko} that will be marked on both cases. All state and transition definitions are presented on the figure's tables.



Places	Definition
H_{ok}	hardware's up state
H_{ko}	hardware's unavailability state
S_{ok}	software's up state
S_{ko}	software's unavailability state
S_{fail}	software's failure state
S_{st}	software's standby state

Transition	Rate	Definition
T_1	λ_h	failure of the hardware component
T_2	ν_h	repair of the hardware component
T_3	λ_s	failure of the software component
T_4	ν_s	restoration of the software component
T_{rs}	ρ	restart of the software component

Fig. 7. Example of a high-level dependability model

Once the high-level model is built, it can be refined in order to have more accurate dependability measures. In [5] we define rules for model refinement according to three perspectives: a) Component decomposition, b) State and event fine-tuning and c) Distribution adjustment. They are not specifically addressed in this paper.

3 Presentation of the Considered I&C Systems

In this section, we present the main functions of an I&C system and its functional-level model. Then, we present examples of three very different candidate systems to show how, with our modelling approach, they can be easily modelled in an homogeneous way. We have defined these three hypothetical systems based on real-life I&C systems in order to form a set of different possible realisations for the considered I&C application. The dependability of these systems is modelled and compared in Section 4.

3.1 Functional Description

An I&C system performs five main functions: *Human-machine interface* (HMI), *processing* (PR), *archiving* (AR), *management of configuration data* (MD), and *interface with other parts of the I&C system* (IP) Figure 8. The arrows in the figure represent the interactions between these functions.

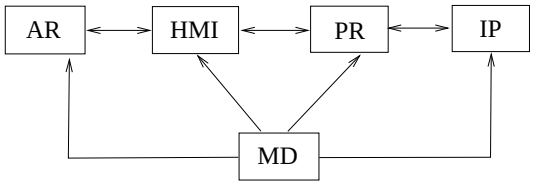


Fig. 8. Five main functions of an I&C system

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:

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

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

Table 1. Functional dependencies of I&C systems

Functional dependencies	Simplified dependencies
HMI $\hookleftarrow$ {PR, AR, MD}	HMI $\hookleftarrow$ {AR, MD}
PR $\hookleftarrow$ {HMI, MD, IP}	PR $\hookleftarrow$ MD
AR $\hookleftarrow$ {HMI, MD}	AR $\hookleftarrow$ MD
IP $\hookleftarrow$ {PR, MD}	IP $\hookleftarrow$ MD

These relations are translated by the functional model depicted in Figure 9, where the dotted rectangles identify the five partial dependencies of Table 1's column 2. The model for each function corresponds to the one given in Figure 2.

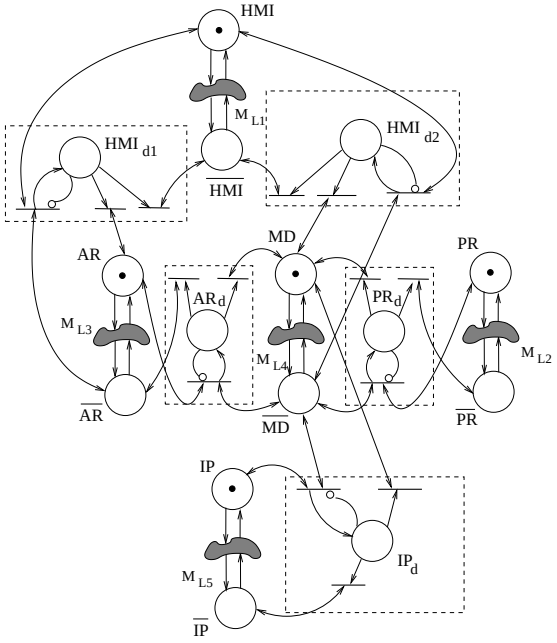


Fig. 9. Functional-level model for I&C systems

3.2 Three Examples of I&C Systems

The three I&C systems' architectures considered in our study are depicted in Figure 10.

System_1 (Figure 10(a)) is composed of thirteen nodes connected by a *Local Area Network* (LAN). Note that each node executes a single function. Nodes 1 to 10 are composed of a computer each. Nodes 11, 12 and 13 are fault-tolerant: They are composed of two redundant computers each. Also, nodes 12 and 13 are complementary (i.e., they interface complementary parts of the I&C system).

System_2 is composed of five nodes connected by a LAN. The mapping between the various nodes and functions is given in Figure 10(b). Note that while HMI is executed on 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.

Nodes 1 to 3 of *System_3* (Figure 10(c)) are composed of a single computer each running a single function. Node 4 is fault-tolerant: It is composed of three redundant computers (usually referred to as a TMR – *Triple-Modular Redundancy*).

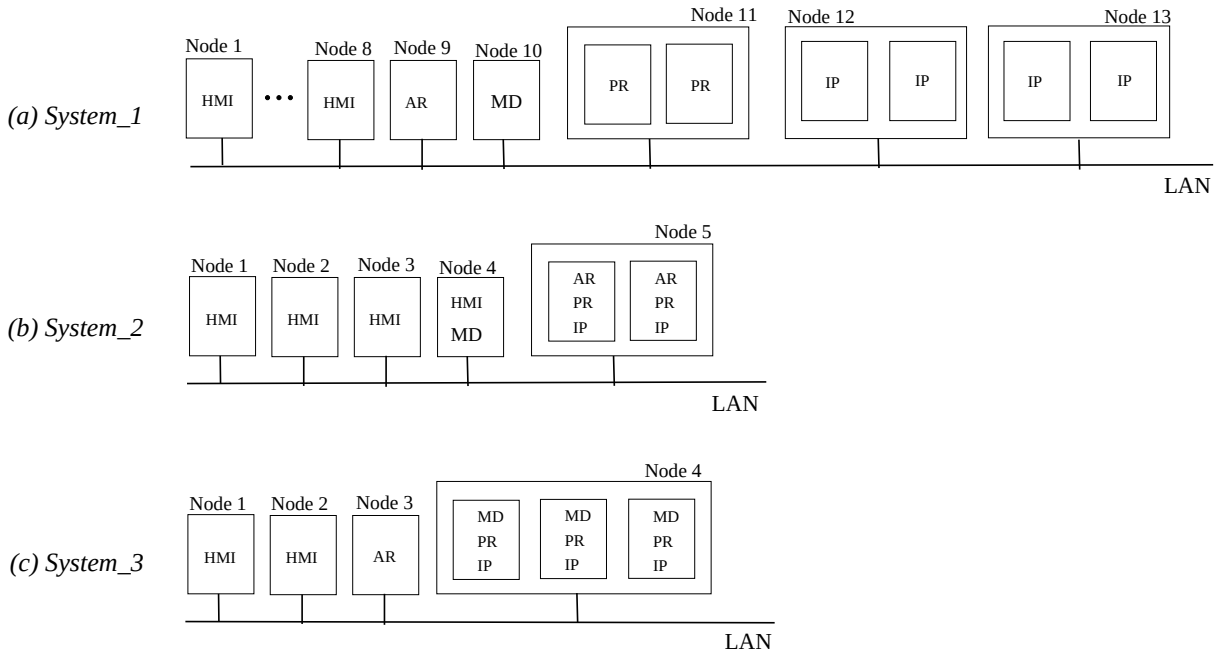


Fig. 10. Three examples of I&Cs' systems

These systems were chosen for their diversity of architectures and of redundancy techniques. For example in *System_1*, every component executes a single function, while in *System_2* and *System_3*, some components execute more than one function.

The three techniques of replication used in the above architectures are [12]:

- Passive replication: Only one of the n parallel replicas ($n \geq 2$) processes the input messages and provides output messages (active replica). The other (passive) replicas do not process the input messages. In case of unavailability of the active replica, one of the passive replicas becomes active;
- Semi-active replication: Only one of the two replicas processes all input messages and provides output messages (primary replica). The other replica (secondary) is active since it also processes the input messages even though it does not provide any output messages. In case of fault occurrence or activation in the primary, a switch from the primary to the secondary replica is performed;
- Active replication: The three replicas process all input messages concurrently so that their internal states are closely synchronised – in the absence of faults, outputs can be taken from any replica as long as at least two replicas are in the nominal state. In case of fault occurrence or activation, the error is masked.

For the two later cases, the switch/masking is done with a given probability of success, referred to as coverage factor.

Table 2 summarises the redundancy type (if existing) of each node. Non-redundant nodes are referred to as *single* nodes. We assume that each hardware component hosts a single software component. A hardware component with its associated software replica is referred to as a *unit*. To conclude, at this level of detail all redundancy cases are modelled by one of these three cases of replication/error recovery. Thus, we will refer to them respectively as: Passive redundancy, semi-active redundancy, and active redundancy or TMR.

Table 2. Node and redundancy type for the three studied architectures

System	Node	Redundancy	
		Replication	Recovery
System_1	#1-8	passive	–
	#9	single	–
	#10	single	–
	#11	semi-active	switch
	#12	semi-active	switch
System_2	#1-3	passive	–
	#4	single	–
	#5	semi-active	switch
System_3	#1-2	passive	–
	#3	passive	–
	#4	active	masking

4 Modelling of the I&C systems

To simplify the model's construction for the three systems, we have built a library of basic models which, with minimal modifications, allow the efficient modelling of the given systems.

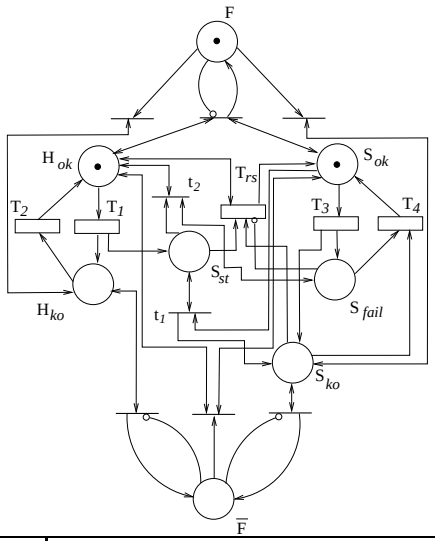
Before giving an overview of the global model of the three defined systems, we present the models' library.

4.1 Basic Models

The basic models are given for a single function at a high-level. They can be refined according to the rules given in [5]. These models correspond to the four redundancy types identified in

Table 2; respectively: i) A single unit, ii) passive redundancy, iii) semi-active redundancy, and iv) active redundancy. These models are presented in the rest of this section. For a complete description of the replica management see e.g. [12].

Single Unit This first model is given in Figure 11.



Transition	Rate	Definition
T_1	λ_h	hardware's fault occurrence
T_2	ν_h	hardware's repair
T_3	λ_s	software's fault activation
T_4	ν_s	software's reinitialisation after software's failure
T_{rs}	ρ_s	software's restart after hardware's repair

Fig. 11. High-level dependability model of a single unit

The rates, corresponding to the timed transitions managing the state changes, are given in the figure's table.

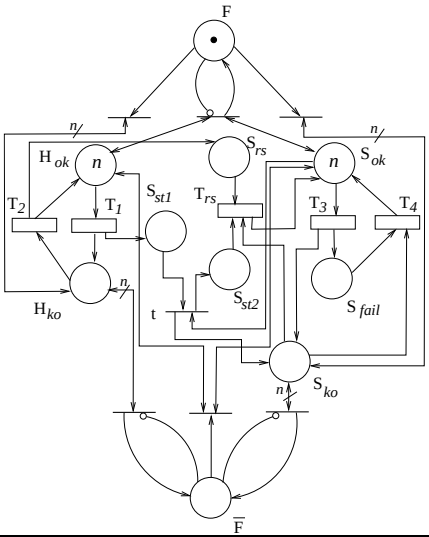
In this case, when a hardware component becomes unavailable (timed transition T_1), the associated software component is stopped. This is modelled by place S_{st} and immediate transition t. The software will be restarted (timed transition T_{rs}) once the hardware's repair has finished.

It is worth noting that we distinguish the case where the software component fails (corresponding to place S_{fail}), from the one where the software is stopped following a hardware's unavailability (corresponding to place S_{st}). Indeed in the first case, the software's failure may need a longer maintenance intervention, compared to the software's restart needed after a hard-

ware failure. Thus, place S_{ko}, representing the unavailability state of the software component, is marked either when the software component is stopped or when it fails.

This model completes the one of Figure 4. In particular, places H_{ok}, S_{ok}, H_{ko} and S_{ko} are the same.

Passive Redundancy Function F is performed as long as at least one unit is up – Figure 12.



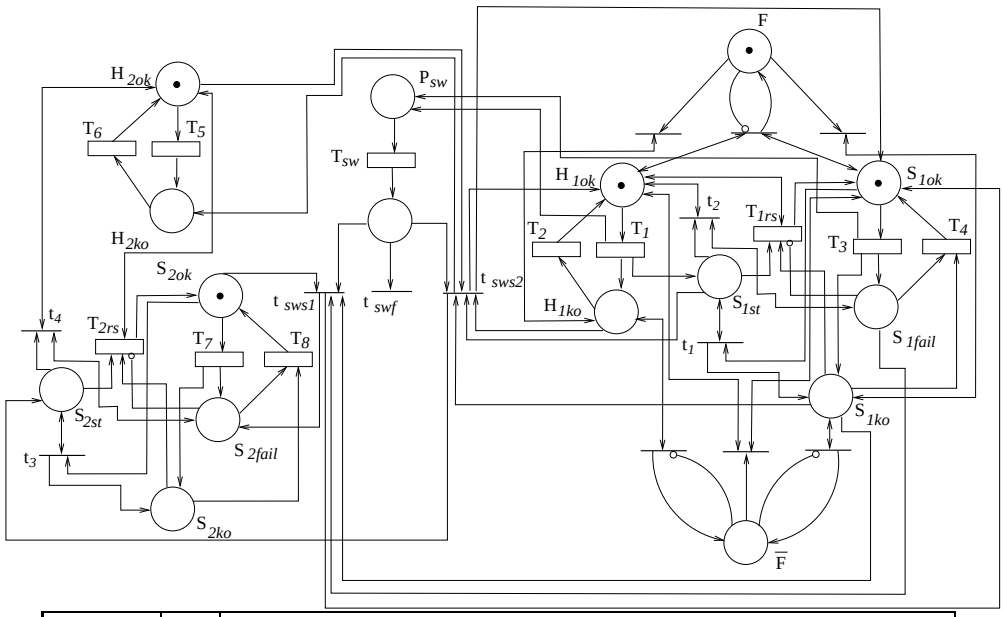
Transition	Rate	Definition
T_1	$\lambda_h\ \mathcal{M}(H_{ok})$	hardware's fault occurrence
T_2	ν_h	hardware's repair
T_3	$\lambda_s\ \mathcal{M}(S_{ok})$	software's fault activation
T_4	ν_s	software's reinitialisation after software's failure
T_{rs}	ρ_s	software's restart after hardware's repair

Fig. 12. High-level dependability model for passive redundancy

As in the previous case, when a hardware component becomes unavailable, the associated software component is stopped.

The main difference between this model and the previous one is that, in this case, we need a buffer to collect the software components that are stopped following a hardware unavailability (place S_{st1}) and another one to allow software's restart (place S_{rs}). Note that this model is a generalisation of the one depicted in Figure 11.

Semi-Active Redundancy There is a primary unit and a secondary one, their model is presented in Figure 13. This model is obtained by composition of two models of Figure 11 and a switch model.



Transition	Rate	Definition
T_1/T_5	λ_h	primary/secondary hardware's fault occurrence
T_2/T_6	ν_h	primary/secondary hardware's repair
T_3/T_7	λ_s	primary/secondary software's fault activation
T_4/T_8	ν_s	primary/secondary software's reinitialisation after software's failure
T_{rs1}/T_{rs2}	ρ_s	primary/secondary software's restart after hardware's repair
T_{sw}	β	switch from the primary to the secondary

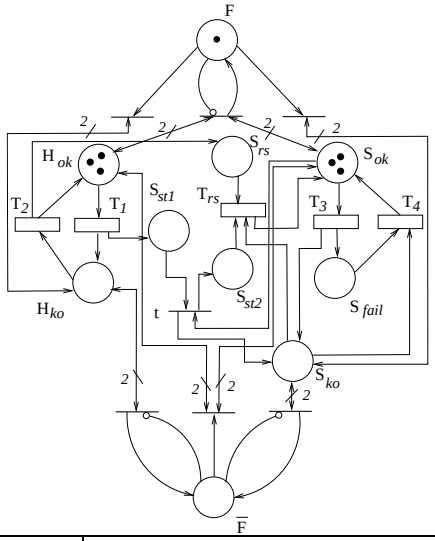
Transition	Probability	Definition
t_3	c	switch success following a primary hardware failure
t_4	c	switch success following a primary software failure
t_5	$1-c$	switch failure

Fig. 13. High-level dependability model for semi-active redundancy

If the primary unit fails due to the failure of one of its components, the internal fault tolerance mechanisms switch over to the secondary unit that becomes primary. The switching time is $1/\beta$ and the associated transition is T_{sw}. The coverage factor (i.e., the conditional probability that the switch succeeds given the failure of the primary) is c (immediate transitions t_3 and t_4). Thus, the switch fails and the function is lost with probability $1-c$ (immediate transition t_5).

Active Redundancy The last basic dependability model considered in our library corresponds to the active replication technique with masking error compensation (TMR) – Figure 14.

In this case, the function is performed as long as there are at least two units in the *ok* state. The model of Figure 14 is an adaptation of the one in Figure 12. The main difference between this model and the one depicted in Figure 12 is the token number in places H_{ok} and S_{ok} and the weight of the arcs that manage the failure and restoration of function F.



Transition	Rate	Definition
T_1	$\lambda_h\ \mathcal{M}(H_{ok})$	hardware's fault occurrence
T_2	ν_h	hardware's repair
T_3	$\lambda_s\ \mathcal{M}(S_{ok})$	software's fault activation
T_4	ν_s	software's reinitialisation after software's failure
T_{rs}	ρ_s	software's restart after hardware's repair

Fig. 14. High-level dependability model for a TMR

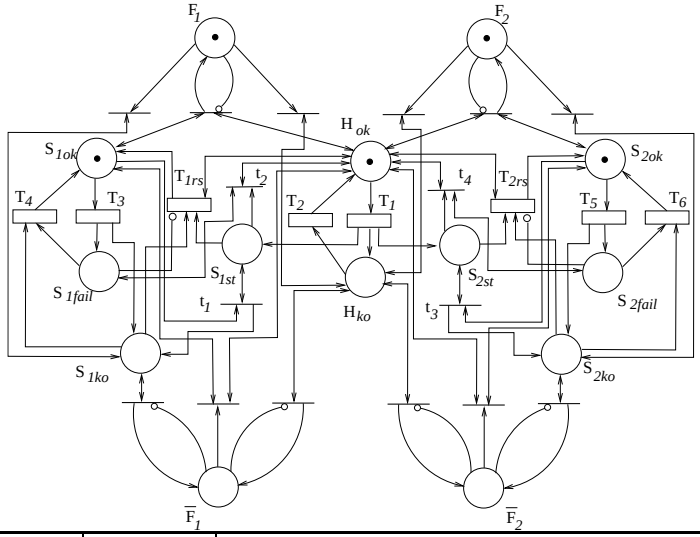
4.2 Adaptation of the Basic Models

It is worth mentioning that the model given in Figure 12 is a generalisation of Figure 11, the one of Figure 13 is a composition of two models of Figure 11, and finally, the model of Figure 14 corresponds to the one given in Figure 12 with a slight modification.

Before giving the respective high-level dependability models of the three systems, we present an example of the modifications that can be made to the basic models to take into account more than one function (the basic models are given for a single function). Let us consider two functions implemented on the same hardware computer.

Two Functions on the Same Hardware Component The general model for this case is given in Figure 15. This model is obtained by composition of two models of Figure 11 but the hardware model is not duplicated. To built this model we follow the interface rules given in Section 2.3.

In this model, we consider each function executed by a single software component.



Transition	Rate	Definition
T_1	$\lambda_h\ \mathcal{M}(H_{ok})$	hardware's fault occurrence
T_2	ν_h	hardware's repair
T_3	$\lambda_s\ \mathcal{M}(S_{ok})$	software's fault activation
T_4	ν_s	software's reinitialisation after software's failure
T_{rs}	ρ_s	software's restart after hardware's repair

Fig. 15. High-level dependability model for two functions on the same hardware component

4.3 Models of the Three Systems

Using the above presented basic models (with or without modification), we built the complete high-level dependability model for each architecture. Table 3 recalls the composition of these architectures as given in Table 2, and shows the associated basic model's figures. *System_2*'s node 5 and *System_3*'s node 4 models are adaptations of Figure 13 and Figure 14 respectively. Their adaptations are done in a similar way as the one presented in Figure 15.

Note that we have considered the LAN as a single net. If it is redundant, its model is either similar to the one of Figure 12 with $n = 2$ (if it is in passive redundancy) or to the one of Figure 13 (if it is in semi-active redundancy).

Table 3. Corresponding models for the three studied architectures

System	Node	Model
System_1	#1-8	Figure 12 with n=8
	#9,#10	Figure 11
	#11, #12, #13	Figure 13
	LAN	Figure 11
System_2	#1-3	Figure 12 with n=3
	#4	Figure 15
	#5	Figure 13, with 3 functions
	LAN	Figure 11
System_3	#1-2	Figure 12 with n=2
	#3	Figure 11
	#4	Figure 14, with 3 functions
	LAN	Figure 11

4.4 Comparison of the Three Systems

Once the high level dependability models are built, several comparative analysis can be performed. For example, the systems' dependability measures (like availability or reliability) or the choice of redundancy (like the number of HMI nodes) can be compared.

Regarding the comparison of dependability measures, appropriate values should be given to the dependability models' parameters. We consider two classes of parameters:

- Those referring to the components' failure and restoration rates;
- Those referring to the fault tolerance mechanisms (*e.g.*, coverage factors).

For rates like failure or restoration we can rely on statistical data obtained by feedback information on similar components or systems. Indeed, most of the time, the proposed systems are mainly composed of Components-Off-The-Shelf. For instance, this is the case for the current *System_1*, *System_2* and *System_3*.

When it comes to parameters directly related to the fault tolerant mechanisms, a sensitivity study enables the identification of the most impacting ones. Specific analyses are required to measure them. Such analyses might involve experimentation on the real or a prototype system, using fault injection, if necessary.

We have performed a comparative analysis of the systems, regarding two parameters: i) The switching time between the primary and the secondary units (parameter $1/\beta$) and ii) the coverage

factor (parameter c) (both parameters of Figure 13). Note that the analysis concerns *System_1* and *System_2* given that *System_3* has no components with semi-active redundancy.

Our goal is only to show the kind of results that can be provided to the stakeholder when using the modelling approach presented in this paper. To this end, we used classical values for the models' parameters. Results have where obtained using the SURF-2 tool [2]

Figure 16 presents the annual unavailability of *System_1* and *System_2*, as a function of the switching time ($1/\beta$) for $c = 0.95$. It is expressed in hours *per* year, to make the interpretation easier that when it is expressed in terms of probabilities.

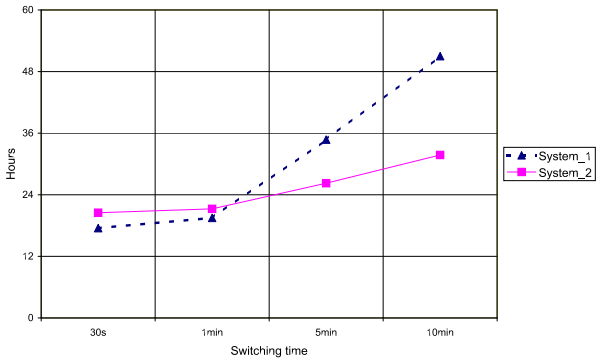


Fig. 16. Annual unavailability of *System_1* and *System_2* with c = 0.95

We notice that when the switching time is increased, *System_1* is more sensitive to this variation than *System_2*. This can be explained by the fact that *System_1* has three components dependent on these two parameters, whilst *System_2* has only one.

Also, we notice that for a small switching time ($1/\beta$ = 30s or 1min), the value of *System_1*'s annual unavailability is smaller than *System_2*'s. However, when we increase the switching time ($1/\beta$ = 5min or 10min), this trend is reversed.

When we increase the coverage ($c = 0.98$, Figure 17),

- The difference between the two systems when the switching time is small ($1/\beta$ = 30s) is more significant, 8h against 2h when $c = 0.95$;
- The trend is reversed for a longer switching time, $1/\beta$ = 5min instead of $1/\beta$ = 1 min when $c = 0.95$;
- Also, after the trend's reverse, the gap between the annual unavailability values of both systems is shorter (14h against 20h when $c = 0.95$).

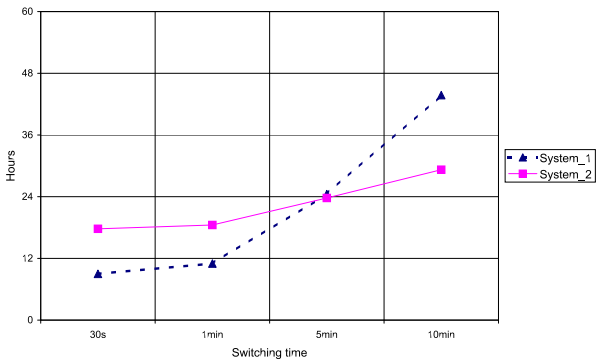


Fig. 17. Annual unavailability of *System_1* and *System_2* with c = 0.98

Finally, considering the same system for the two values of c, it can be seen that *System_1* is more sensitive to c than *System_2*. Improving c from 0.95 to 0;98, the unavailability is reduced by at least 10 hours *per* year for *System_1*, while it is only reduced by 2 to 3 hours for *System_2*. Hence the values of c and $1/\beta$ impacts more *System_1* than *System_2*.

All these analyses show that it is really important to have the most realistic values for the parameters, to allow a fair and relevant comparison of the systems.

Assuming that the considered failure and repair rates are not far from reality, the kind of conclusions the stakeholder can make are, for example:

- If $1/\beta = 10$ mn, then for all values of the c parameter, the safest case is obtained for *System_2*;
- If $1/\beta = 5$ mn, the value of parameter c has an important role in the conclusions. The safest case is still *System_2*;
- If $1/\beta \leq 1$mn, then for all values of the c parameter, the safest case is obtained for *System_1*;

However, the above results should be consolidated by sensitivity analyses to i) Identify the most significant parameters and ii) evaluate the impact of these parameters.

These examples of results also show the kind of analysis we can provide to the stakeholder and the conclusions he/she can make based on the results as well as their knowledge about their specific system. Indeed, the stakeholder usually has more information about the specific class of systems he/she is interested in and most of the time he/she can obtain complementary information from the possible contractors. Thus, more comparative analyses could be made. They might suggest contradictory choices according to the parameters considered. The stakeholder might have to make some choices based on some trade-offs.

5 Conclusions

In this paper we have briefly presented a modelling approach for comparing the dependability of candidate systems for an I&C system and illustrated it on three I&C systems. To make system's modelling efficient, a library of basic models is created, allowing us to model all the components of the considered systems', with minimal modifications. Some results concerning the comparison of the three systems are given.

Even though the starting point of our work was a particular I&C system, the modelling approach presented in this paper is applicable to other categories of systems. One of the main features of this approach is its ability for modelling a set of candidate systems, for a given application, in a homogeneous way to make the comparison as fair as possible. The first step consists in building a functional-level model based on the system's specifications that are common to all candidate systems. This functional-level model is thus independent from any system's realisation. Then this model is transformed into a high-level dependability model, based on the integration of information related to the system's structure. Hence, all candidate systems are modelled starting from the same functional-level model, at the same level of details and based on harmonised assumptions. This does not mean that the assumptions should be identical. Indeed, they should be as realistic as possible with respect to each candidate system. Also, a fair comparison requires realistic parameter values. This is made possible by the fact that most of the systems are based on Off-the-Shelf components (commercial or not), for which some field feedback could be available.

Our modelling approach follows in the footsteps of the existing work on dependability modelling. Its innovation concerns the add of functional specifications into the dependability model. We tried to make modelling as systematic as possible to allow experimented, but not necessarily specially-trained, modellers to analyse several systems and compare their dependability at the same level of modelling abstraction, if required.

Incidentally, this modelling approach can be integrated into a framework for dependability benchmarking, based on dependability evaluation [11]. In which case, modelling should be supported by experimentation for evaluating model parameters, mainly those related to fault tolerance.

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. 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.
3. Betous-Almeida,C., and Kanoun,K., "Dependability Evaluation: From Functional to Structural Modelling", in *Proc. 20th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP 2001)*, Lecture Notes in Computer Science, vol. 2187, Springer Verlag (2001) 227–237.
4. Betous-Almeida, C., "Construction and refinement of dependability models – Application to instrumentation and control systems", in French, PhD Thesis, LAAS-CNRS, France, June 2002.
5. Betous-Almeida,C., and Kanoun,K., "Stepwise Construction and Refinement of Dependability Models", in *Proc. International Conference on Dependable Systems and Networks*, IEEE Computer Society Press, Washington, D.C., USA (2002) 515 – 524.
6. 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.
7. Bondavalli,A., Mura,I., Chiaradonna,S., Filippini,R., Poli,S., and Sandrini,F.,"DEEM: A Tool for the Dependability Modeling and Evaluation of Multiple Phased Systems", in *Proc. 8th Int. Conf. on Dependable Systems and Networks (FTCS-30, DCCA-8)*, New York, USA (2000) 231–236.
8. Ciardo,G., Muppala,J., and Trivedi,K.S., "SPNP: Stochastic Petri Net Package", in *Proc. of the 3rd Int. Workshop on Petri Nets and Performance Models (PNPM'89)*, IEEE Computer Society Press, Kyoto, Japan (1989) 142–151.
9. 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.
10. 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, no. 5 (1999) 528–535.
11. Kanoun, K., Madeira, H. and Arlat, J., "Framework for Dependability Benchmarking", in *Supplement of the 2002 Int. Conf. on Dependable Systems and Networks (DSN-2002)*, Washington D.C., USA (2002) F7–F8.
12. Poledna,S., *Fault Tolerant Real-Time Systems: The Problem of Replica Determinism*, Kluwer Academic Publishers, (1996).
13. 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.
14. Sanders,W.H., Obal II, W.D., Qureshi,M.A., and Widjanarko,F.K., "The UltraSAN Modeling Environment", in *Performance Evaluation*, vol. 24, no. 1–2 (1995) 89–115.