Dependability of computer control systems in power plants. Analytical and experimental evaluation

Claudia Betous-Almeida, Alberto Arazo, Yves Crouzet, Karama Kanoun

Dependability of Computer Control Systems in Power Plants

Analytical and Experimental Evaluation

Cláudia Almeida, Alberto Arazo, Yves Crouzet and Karama Kanoun

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

Abstract. The work presented in this paper is devoted to the evaluation of the dependability of computer control systems in power plants. Two complementary approaches are used to analyse and evaluate the dependability of such systems, based respectively on analytical modeling and experimental validation. Both approaches as well as examples of their mutual interactions are briefly illustrated on a subsystem of a specific computer control system. The analytical approach allows evaluation of dependability measures such as availability and identifies the most influential dependability parameters. Fault injection provides the numerical values of the above mentioned parameters and allows identification of specific behaviors that may not have been considered by the analytical model.

1 Introduction

The work presented in this paper is devoted to the evaluation of the dependability of *Computer Control Systems* (CCSs) in power plants. Our study focuses on a subset of the CCS that provides the necessary means for the operators to control the power production.

The considered system is partially composed of *Commercial-Off-the-Shelf* (COTS) hardware and software components. Hence, only partial information is available for system validation, and we cannot rely only on this information to characterize the system and to acquire enough confidence for its use in critical applications. More generally, the use of COTS components in such systems raises the question of their acceptance by the proper authorities.

Our study emphasizes the combined use of two approaches to analyze and evaluate the dependability of the CCS, based respectively on analytical and experimental evaluation.

The purpose of the analytical approach is to provide stochastic models that will be used to evaluate the dependability of the CCS. Modeling is done by means of *Generalized Stochastic Petri Nets* (GSPNs). One of the major problems in stochastic modeling is to have the most realistic values for the parameters. If for

rates like failure or repair, we can rely on statistical data obtained by feedback on similar components or systems, when it comes to the parameters directly related to the fault tolerance mechanisms (*e.g.*, coverage), a specific analysis has to be done in order to measure them. However, all such parameters may not have significant impact on system dependability. A sensitivity study enables the identification of the most influential one. Analytical evaluation helps to focus the experimental evaluation on the parameters with strong impact, thus reducing the number of experiments.

By forcing the system to react to undesirable inputs (the faults), fault injection aims at evaluating the efficiency of the fault tolerance algorithms and mechanisms, through measurement of some dependability parameters such as coverage factor, fault dormancy, error latency, etc [1]. On the other hand, fault injection, may identify specific failure modes of the system that have not been taken into account by the modeling approach. This may lead to introduce some modifications on the analytical model improving its representativity.

The above approaches are complementary and their combined use provides accurate dependability measures [2].

The remainder of the paper is organized as follows. Section 2 briefly presents the CCS. Section 3 is devoted to analytical modeling where examples of models are presented to illustrate the needs for experimentation. Section 4, addressing the experimental approach, discusses the way fault injection will be performed according to the features of the CCS. Finally, Section 5 concludes the paper.

2 CCS Presentation

For the sake of simplicity, in what follows, the subset of the CCS considered in our work will be referred to simply as the CCS. The CCS is a distributed system with five nodes connected by a *Local Area Network* (LAN). It performs five main functions: *Processing* (PR), *archiving* (AR), *management of configuration data* (MD), *man-machine interface* (MMI) and *interface with other parts of the CCS* (IP). The functions are not totally independent and they exchange information through the LAN. The mapping between the various nodes and the functions is given in figure 1. Note that while MMI 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, one of them acting as a primary (its outputs are the only considered) and the other as a backup (called secondary). The primary computer executes three primary software replicas (corresponding to AR, PR, and IP), while the secondary computer executes a subset of these functions. All nodes have several *Fault-Tolerance and Mechanisms* (FTMs).

The computers are commercial workstations running COTS operating systems and protocols. Indeed, the whole CCS is itself a COTS system constituted by COTS software and hardware components to which specific proprietary software has been added by the CCS provider. Roughly speaking, we can consider that each node has three layers. The lowest layer corresponds to the COTS hard-

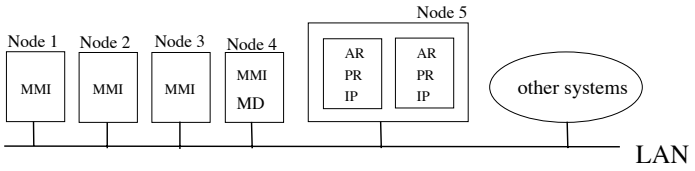

Fig. 1. CCS architecture

ware and software components. On top of it runs the second layer constituted by the distributed middleware (for system configuration, system fault-tolerance, real-time services, etc.), on top of which run the main functions sketched above.

From a dependability point of view, very little knowledge about the CCS development and integration is available. As a consequence, its use in critical applications may be questionable. However, we will see later that this lack of knowledge may be compensated by information resulting from their widespread use in other systems. Nevertheless, such systems require a lot of validation before being accepted in critical application systems, hence the need for a complete validation approach.

3 Analytical Approach

The purpose of the analytical approach is to provide stochastic models to evaluate measures of dependability such as availability, reliability and safety. The evaluation is based on Markov chain models.

The modeling approach is modular, taking benefit from the systems' modularity. It is an incremental approach, similar to the one described in [3]. Model construction is based on GSPNs (see *e.g.*, [4]): The Markov chain is derived from the processing of the GSPN. In a first step, we build the GSPNs of all nodes and of the LAN independently. The system's GSPN is obtained by composition of the above GSPNs, taking into account the dependencies between the nodes and between the nodes and the LAN. Indeed, two sources of dependencies are to be considered:

- functional dependencies due to the communication between system functions, and
- dependencies due to maintenance (when several nodes are in failure, repair may be performed in successive steps if not enough repairmen are available).

Due to all these dependencies, the resulting global model is very complex. This complexity is increased by the fact that, within each node, several dependencies resulting from the interactions between the hardware and software components have also to be considered. Examples of dependencies internal to each node are: Software stop following hardware failure, error propagation from hardware to software or from software to software, switching from the primary computer on to the secondary computer. Because of this complexity, it is not possible to

present in this paper the complete model of the whole CCS. We will thus consider one node, Node 5. In addition, as our purpose in this paper is to emphasize the combined use of the analytical and experimental approaches for system validation (more specifically for dependability evaluation), we have selected a small part of Node 5 to illustrate this objective.

In the rest of the section, we will first present the GSPNs of the two software replicas of function DP running on Node 5, then the GSPN of the switch from the primary computer on to the secondary computer. These examples have been chosen so as to introduce some significant parameters that can be evaluated by fault injection. An example of results is then commented before discussing how fault injection can help.

3.1 Examples of GSPN Models

Figure 2 gives the GSPNs of two software redundant replicas: The primary and secondary replicas running respectively on the two redundant computers. To model these replicas, the following assumptions and notations are used:

- the activation rate of a software fault is λ_p (Tr_1) in the primary and λ_s (Tr_4) in the secondary;
- a software fault is detected by the FTMs with probability d (tr_2 and tr_4). The detection rate is δ (Tr_2 and Tr_5);
- the effects of a non detected error are perceived with rate π (Tr_3 and Tr_6);
- errors detected on the secondary are of two categories: Those requiring to stop the primary (tr_5), and those necessitating only a secondary reset (tr_6);
- the reset rate is ρ (Tr_7) and the probability that an error induced by the activation of a permanent software fault disappearing 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 re-installation rate is σ (Tr_8).

A detected or perceived (tr_2 and Tr_3) software error in the primary induces a *switch* (P_{sw}) from the primary computer to the secondary. And, as mentioned above, certain detected errors in the secondary software (tr_5) component may also induce a switch between computers.

Note that an error in the primary software may propagate to the secondary (tr_{11}). Indeed tr_{11} models the common modes failures (probability p) of the primary and secondary (λ_p = common mode failure rate).

The second example of GSPN concerns the switch between the two redundant computers (figure 3). As seen in figure 2, a switch between the two software replicas may occur when either an error is detected or perceived in the primary or detected in the secondary software components. Indeed, switching of replicas involves switching the computers on which the replicas are running. In addition, an error detected or perceived in the primary hardware induces a switch.

The switch mechanisms in figure 3, are characterized by two parameters: The *switch rate* (β) and the *switch coverage factor* (c). c is the probability that the service is re-established when a fault affects a given component in an operational context.

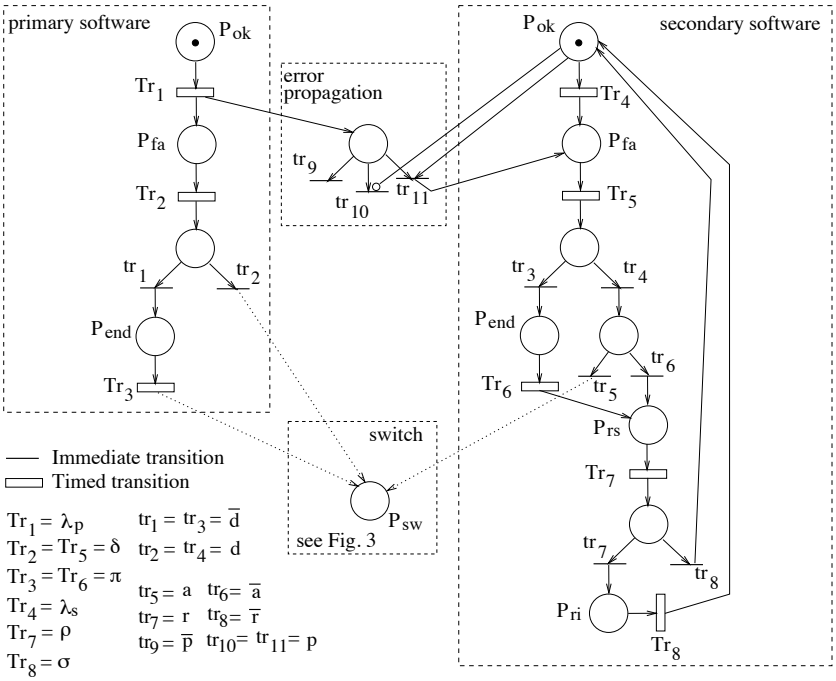

Fig. 2. GSPN of the activation of a software fault in Node 5

3.2 Model Parameters

In order to process the model to obtain the desired measures of dependability, numerical values should be given to the various model parameters (*i.e.*, event rates and probabilities). One of the major problems is to have the most realistic values for these parameters.

As mentioned earlier, the system is composed of COTS. In spite of all the drawbacks the use of COTS in critical systems may have, there are some advantages. Among them, there is one in particular that directly affects the analytical approach: *Feedback information*. Feedback information can help the analytical approach insofar as to give realistic values to some of the models' parameters.

If for rates like failure or repair, we can rely on statistical data obtained from feedback, when it comes to the parameters directly related to the fault tolerance mechanisms, a specific analysis has to be done in order to measure them. The experimental approach through fault injection is perfectly suitable to help the analytical method in this point.

Usually, we perform sensitivity analysis to point out the parameters that affect significantly the dependability measures evaluated. This means that the model is processed with different values of the parameters to check how strongly they affect these measures. Only the parameters with strong impact will be evaluated experimentally, to reduce the number of experiments.

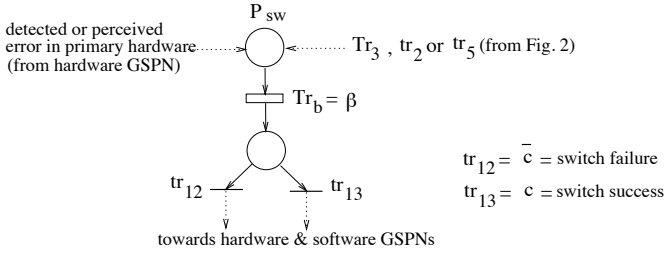

Fig. 3. GSPN of the computer switch

For the CCS, we have considered nominal values for the failure and repair rates obtained by analogy with other systems and we made sensitivity analysis for the other parameters. The objectives were twofold: Identify the most significant ones to measure them experimentally and, for those that cannot be measured experimentally, make sensitivity analysis to evaluate a range for the dependability measures. The model has been processed using the SURF-2 tool [5]. The unavailability, evaluated in hours per year (with a repair time of 10 hours) according to c and β is given in table 1.

It can be seen that both parameters have significant impact. It is obvious that the smallest unavailability is obtained for the shortest switch time and the best coverage factor. This table shows that an annual unavailability of approximately 2h may be obtained for (0.99; 5min) and (0.98; 1min or 30s). Since β an c are not known at this stage, we cannot make any conclusions. Fault injection will help us to identify the accurate result.

Table 1. Node 5 annual unavailability

$c \backslash 1/\beta$	30s	1min	5min	10min
0.99	**1h34min**	**1h38min**	**2h06min**	2h45min07s
0.98	**2h00min**	**2h04min**	2h33min	3h11min
0.95	3h20min	3h23min	3h52min	4h31min

4 Experimental Approach

We have developed an efficient fault-injection strategy taking advantage of the main basic characteristics of the CCS, which are: i) the modular and multilayer design and implementation paradigms, and ii) the use, at the lowest layer of the system's architecture, of standard COTS components.

In the rest of the section, we explain the features of our fault-injection strategy resulting from the early integration of the above characteristics. After that, we show how a prototype tool, supporting our strategy, can be used to obtain the set of expected measures.

4.1 A Modular and Multilevel Approach

To take into account the multilayer and modular design of the system, our fault injection strategy is *multilevel*. Also, the notion of *fault region* around both hardware and software components plays an important role. The integration of these two concepts enables us to a) design fault models emulating with a high accuracy the real faults affecting the CCS at any architectural layer, and b) refine progressively fault injection campaigns. Refinement is not only needed to improve analytical analysis results, but also to obtain other diagnostic information, increasing our knowledge about the CCS behavior in the presence of faults. The two following paragraphs explain these concepts and their relation with the CCS characteristics.

In a modular, distributed and multilevel computation approach, one can consider that each component is associated to a fault region. For example, a node can be considered as a fault region at a high level of abstraction. At a lower level of abstraction, a computer or a software replica can also be considered as a fault region. Each failure mode of each component is seen by the other interacting components, as being a fault model.

Our fault-injection approach is intended to be used at any level of abstraction. This enables us to refine the outcomes of fault injection campaigns. The same fault injection protocol is used for emulating both high-level faults affecting link communication between software replicas, and low-level real hardware faults affecting the memory or internal processor units. The templates emulating a real fault and the means to inject and to observe their consequences, are instantiated in a strong relationship with the granularity of the target injection and monitoring levels. For example, at the low system layer, a well-known bit-flip model is considered as a good representation of the consequences of real electromagnetic perturbations (hardware faults). By an error propagation analysis, this multilevel approach would give us the means to observe, at a higher system layer, the consequences of the same low-level fault. Such analysis is necessary for emulating, with a coarse, yet satisfactory, high-level software fault models, low-level real faults. The corollary of this schema is a good coverage of the real fault types considered by the analytical approach. Both the injection and the observation activities are carried out by means of agents placed in different places of the system.

4.2 Exploiting Standard COTS Component Features

As stated earlier, the CCS is composed of proprietary software running on top of the distributed middleware that is running on top of standard software and hardware COTS. The middleware provides, among other services, fault-tolerance mechanisms to protect the services required by the upper functions from errors propagated from the lowest layer. Because very little information of CCS's structure, specification, development and integration is available, sometimes we will be constrained to place our fault injector Agents out of the proprietary CCS software. Ideally, they will be placed between the low layer (constituted by COTS

components) and the CCS middleware services. Thus, the implementation of CCS on top of COTS components is an interesting benefit for the fault-injection point of view. Because services, features and structures of standard COTS are well documented, we can easily interface the target node and use them to generate "COTS-level" faults, emulating real hardware and software faults.

4.3 Implementation of the Fault-Injection Strategy

Fault injection campaigns give us an estimation of the parameters identified by the analytical approach, by means of readouts (R) obtained during the CCS operational activity (A), in presence of well selected sets of faults (F). Moreover, further unknown CCS operational behaviors under faulty conditions may be detected by a set of readouts correctly placed in the nodes. The desired measures (M) are obtained by processing the readouts.

Our fault injection strategy is built around a *Software Implemented Fault Injection* (SWIFI) approach. As recent works (see *e.g.*, [7, 8]) have shown, SWIFI approaches take full advantage of features provided by standard hardware and software COTS. Their main advantages are their ability for: i) emulating a wide variety of fault models (F), ii) programming, in time and space, the point of impact of the injections, iii) placing, almost wherever we want, the readouts (R) to observe the consequences of emulated faults, and iv) automating the experimental campaign by script file orders addressed to injector Agents.

Our tool platform is based on the SPARC^{TM} hardware architecture, running Solaris 2.5.1 Operating System^{TM}. It is depicted in figure 4 where the target node is assumed to be Node 5. Because a large degree of portability is necessary for covering the evaluation of a wide variety of CCSs, we have separated the tool into a portable high-level software, and a small system-dependent part (which depends on the target platform). For this purpose we have adopted a "*Host-Target*" paradigm, separating the *campaign computation tasks* (design, experiment control and measures) running on the host, from *fault-injection and monitoring tasks* achieved by *Agents* running on the target node (more precisely on the two workstations of the node, WS1 and WS2). The Agents have three main roles:

- program when and where the fault is to be injected according to the scripts received;
- perform the fault injection;
- record and send the results (through the readouts).

4.4 Contributions of Fault Injection to the Analytical Approach

In the model described in figure 3, $1/\beta$ corresponds to the switch duration following any failure origin (detected or perceived failure in the primary hardware, failure of the primary software or a subset of errors in the secondary software). Even though the three origins have been distinguished, it is assumed that the

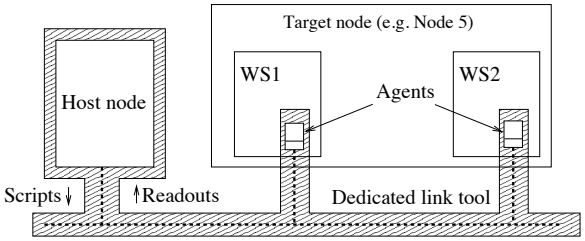

Fig. 4. Fault injection platform

switch rate and the coverage factor are independent from the origin. Fault injection may be performed separately, keeping record of the failure origin. If the results show significantly different switching times or coverage factors, this means that the analytical model should be modified to distinguish the three cases. Otherwise, if the switch duration and coverage factor are independent from the switch origin, there is no need to modify the model. This shows an example of how fault injection results can lead to modify the model or to strengthen our confidence on the assumptions supporting it. In the following, we comment the campaign aiming at evaluating $1/\beta$.

The first step is to select a satisfactory abstraction level for the experiments. This selection is done according to both our knowledge about the CCS components, and the degree of details required by the GSPN model. For the purpose of our example, the *middleware* layer is a satisfactory target level. This choice results from the fact a large part of the CCS activities (A) are instantiated on top of this layer, and also the hardware error detection and fault tolerance mechanisms (provoking the switch) are in this layer. The second step concerns the description of the aims of the experimental campaign. For this purpose, we rely on the approach described in [6] that proposes two domains for fault injection characterization: The *output domain* (measures, M and readouts, R) describing the objectives of the campaign (here the measure of $1/\beta$), and the *input domain* (faults, F and activities, A) describing the means employed to reach these objectives.

The campaign for measuring the switch time $1/\beta$ implies the definition of a set of commands to be sent to the injector and monitoring Agents. Here, three campaigns (taking into account different Fault sets (F)) must be programmed, since system switch may result from three cases as recalled above. However, the goal being the same, a subset of common class of readouts (R) can be defined for the three campaigns. Here, the first part of R is defined as a set of reads of the internal clock to compute the time spent between error detection and service restart. This time is computed and recorded by the Agents[1]. For this clock service we rely on the hardware and operating systems COTS features. The second part of R concerns the observation of the consequences of F (the

[1] We assume that nodes' clocks of the CCS architecture are synchronized by an external mechanism

switch) and, in this case, the objective is the evaluation of the coverage factor with respect to the sets F. This subset of R is closely related to the definition of F for each campaign.

In order to emulate the three sets F, we rely on some structural aspects according to the level of abstraction chosen and the R set. First of all, we identify the set of computational objects (a subset of the activities (A)) concerned by the experiments. Here, these computational objects are the set of tasks provoking the start of the switch mechanisms. In order to program a fault emulation, we define a series of actions using the resources provided by the COTS components: Breakpoint registers, clock ticks register, task control blocks, etc. Indeed, the first step of the fault injection experiment is to stop, for a very short of time, the nominal behavior of the WS at the application layer, by modifying the nominal behavior of the underlying layer (here the middleware layer). After this, we emulate the fault. After resuming the CCS activity, the Agents analyze the behavior of the computation objects under faulty assumptions and record, for a pre-defined period of time, the results of the fault-tolerance mechanisms and the time of switch in readouts. At the end of the three campaigns, the readouts recorded are used by the host node to calculate the three values of $1/\beta$ and c.

5 Conclusions

In this paper an evaluation of the dependability of computer control systems (using standard COTS components) in power plants has been presented. More precisely, a strategy combining the use of two complementary approaches based respectively on analytical and experimental evaluation has been discussed.

It is shown how an optimization of the fault injection campaign is obtained by making sensitivity analysis on the analytical models. Thus, only the parameters that affect the most the system's unavailability and for which no feedback information has been given, will be evaluated by fault injection campaigns. Also, we have discussed the way results of the experimental evaluation may influence the analytical models and how fault injection takes advantage of the use of standard COTS components in the sytem.

The work is still under progress. Particulary the fault injection is under implementation. Even though the paper was devoted to a specific CCS architecture, our work is intended to be more general. Its aim is to support the industrial company for which this work is performed, in several ways:

- select, based on dependability analysis and evaluation, among other things the most suitable system among those proposed by systems' integrators;
- characterize more precisely the selected system.

The modeling and the fault injection approaches can thus be applied with progressive knowledge of the system: For system selection, high level approaches are sufficient, whereas for accurate system characterisation, more precise, low level approaches are needed.

References

1. J.A. Clark and D.K. Pradham, "Fault Injection : A Method for Validating Computer-System Dependability", in *IEEE Computer*, vol. 28, pp. 47–56, 1995.
2. J. Arlat, "Informatique Sûre de Fonctionnement: défis et solutions", in *Journal Européen des Systèmes Automatisés*, vol. 30, n. 10, pp. 1433-1465, (in french), 1996.
3. K. Kanoun, M. Borrel, T. Morteveille and A. Peytavin, "Availability of CAUTRA, a Subset of the French Air Traffic Control System", in *IEEE Trans. on Computers*, vol. 48, n. 5, pp. 528–535, may 1999.
4. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franchescinis, *Modelling with Generalized Stochastic Petri Nets*, Series in Parallel Computing, Wiley, 1995.
5. C. Béounes et *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, pp. 668–673, 1993.
6. J. Arlat, Y. Crouzet and J.-C. Laprie, "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", in *Int. Symp. on Fault-Tolerant Computing (FTCS-19)*, Chicago, IL (USA), pp. 348–355, 1989.
7. J. Carreira, H. Madeira and J.G. Silva, "Xception : A Technique for the Experimental Evaluation of Dependability in Modern Computers", in *IEEE Trans. on Software Engineering*, vol. 24, no. 2, pp. 125–135, 1998.
8. M. Rodriguez, F. Salles, J. Arlat and J.-C. Fabre, "MAFALDA: Microkernel Assessment by Fault Injection and Design Aid", in *3rd European Dependable Computing Conference (EDDC-3)*, Prague, Tcheck Republic, pp. 143–160, 1999.