Safety Analysis and Evaluation of an Air Traffic Control Computing System

Nicolae Fota, Mohamed Kaâniche, Karama Kanoun, Alain Peytavin

Proc. of the 15th International Conference on Computer Safety, Reliability and Security (SAFECOMP96), Vienna, Austria,, Springer-Verlag, pp. 219-229, October 1996

Safety Analysis and Evaluation of an Air Traffic Control Computing System

Nicolae Fota[◊*], Mohamed Kaâniche[◊], Karama Kanoun[◊] and Alain Peytavin[◊◊]

[◊]LAAS-CNRS
7, Av. du Colonel Roche
31077 Toulouse—France
fota@laas.fr

[◊◊]CENA
7 Av. Edouard Belin, BP. 4005
31055 Toulouse—France
{kaaniche;kanoun}@laas.fr

[*]SOFREAVIA
3 Carrefour de Weiden
92441 Issy les Moulineaux—France
peytavin@cena.dgac.fr

1 Introduction

The French Air Traffic Control is based on an automated system referred to as CAUTRA (Coordinateur AUtomatisé du Trafic Aérien). The CAUTRA is implemented on a distributed fault-tolerant computing system installed on five en-route traffic control centers and one centralized operating center, that are connected through an aeronautical telecommunication network. The CAUTRA mission is to provide computerized means for the safe and efficient movement of aircrafts. The main services provided are flight plans processing, radar data processing and air traffic flow management.

However, the computing system failures could temporarily prevent the system from performing some or all of its required functions. The impacts of failures on the traffic safety depend on the criticality of the affected functions and the duration of the service interruption. In order to analyse and evaluate these impacts, we have defined a global approach that can be decomposed into two parts. The first part is aimed at a preliminary Failure Modes Effects and Criticality Analysis of the global CAUTRA: this study led us to identify the main subsystems that have a significant impact on the traffic safety. The second part of the approach focuses on the dependability modeling and evaluation of each subsystem and the combination of the dependability measures evaluated for each subsystem to obtain global measures characterizing the impact of the CAUTRA failures on the traffic safety. This approach is presented in [1]. In this paper, we focus on one subsystem of the CAUTRA centralized operating center, referred to as "STIP", which performs the centralized acquisition, processing and distribution of the flight plan information to the en-route traffic control centers.

This paper is decomposed into seven sections. Section 2 presents the STIP architecture. Section 3 outlines the failure and repair assumptions considered. Section 4 discusses the classification of the STIP failures according to their impact on the traffic safety. Section 5 summarizes the modeling approach used to describe the STIP behaviour and evaluate dependability measures. Section 6 comments some quantitative results. Finally, Section 7 concludes and outlines some directions for the future work.

2 The STIP Architecture

The STIP is implemented on a duplex architecture (Figure 1) composed of two redundant computers (Co the operational one and Cs the spare one), two redundant disks (Do and Ds where each one is composed of a couple of mirrored disks), two replicas of the application software (So and Ss), and an I/O board (B) for the connection of the communication lines and peripherals to the STIP operational configuration. So periodically sends a copy of its current state to Ss through a local data link. Do and Ds can be accessed by either Co or Cs. Two configuration modes are possible: "local disk association" when Co is connected to Do, and "remote disk association" when Co is connected to Ds. The computers — disks interconnections are implemented with coupler devices. The couplers and the dedicated data link are supposed to be failure-free since their failure rates are significantly small compared to the failure rates of the rest of components.

3 Failure-Repair Assumptions

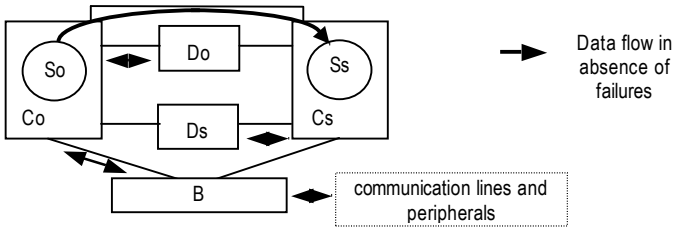


Figure 1 - Overview of the STIP Architecture

So & Ss. Two parameters are relevant to identify software failures and their impact on the air traffic control: the amount of dynamic data lost and the duration of service interruption. The dynamic data correspond to information about the flight plans processed by the system that are vital for the air traffic control services.

Most of the So failures can be recovered without a significant loss of data either by restarting the software, or by rebooting Co. However, some So failures may lead to the partial or total loss of the dynamic data. In this case, several recovery actions are attempted by the operators: 1) the reconfiguration of Ss from the spare mode to the operational mode and the switching of I/O links; 2) the reboot of Co and its association to the remote disk followed by the restart of So (this sequence of recovery actions is denoted as *"asso"*); 3) the restart of So after *partial* elimination of some dynamic data (denoted as *"rpl"*) ; 4) the restart of So after *total* elimination of the dynamic data (denoted as *"rtl"*). It is noteworthy that the data lost has to be manually recreated by the controllers after the service restoration.

Three failure modes of the So replica have been identified:

- *"So and Ss common mode failures"* which are due to some erroneous dynamic data processed by both replicas; they are immediately diagnosed by the operators as they lead to the simultaneous failure of both replicas; these failures may be recovered by an "rpl", followed by an "rpt" if the first recovery action fails;
- *"local failures - immediately diagnosed"*; these failures affect the dynamic data processed by So without affecting those processed by Ss and they are immediately diagnosed by the operators. The reconfiguration of Ss (from the spare to the

operational mode) and the switching of the I/O links are first attempted followed, in case of failure, by an *"asso"* recovery action;

- *"local failures - not immediately diagnosed"*: other software failures whose origin is not immediately identified by the operators. The four recovery procedures mentioned above are consecutively applied until the service restoration.

With regards to the Ss replica, it is assumed that any failure of Ss leads to the loss of the dynamic data processed by this replica. Moreover, Ss can be restarted only if So is operational.

Co and Cs. A computer failure leads to the stop of the software replica hosted. Moreover, when Co fails, Cs becomes the operational computer after the reconfiguration of the system.

Do and Ds. The failure of Do or Ds results from the unavailability of the associated couple of mirrored disks. The failure of a disk causes the associated software replica to stop and leads to the total loss of the dynamic data processed by that replica. When Do fails, the reconfiguration of Ss into So and the switching of the I/O links are first attempted followed, in case of failure, by an *"asso"* recovery action.

B. Two failure modes may affect the I/O board: 1) "a switching failure" when a system reconfiguration is attempted; this failure mode precludes the use of Cs, however Ds may still be accessed via a remote disk association; 2) "the interruption of I/O links" that may occur at any time and leads to the global system unavailability .

Repair resources. We assume the availability of two repairmen for hardware failures: one for the I/O board and another one for the disks and the computers.

4 Impact of the STIP Failures on the Traffic Safety

The STIP failures impact on the traffic safety is measured through the assessment of the degradation of the controllers capability in performing safe control. During the global CAUTRA analysis, we have identified five classes of service degradation that are summarized in Table 1. The level of degradation is dependent on the amount of data lost, the duration of service interruption and the availability of alternative facilities to continue the air traffic control service in the degraded operation mode (control services provided by systems outside the air traffic control computing system). Levels IS1 and IS5 correspond respectively to the most and the least critical failures.

Table 2 gives a classification of the STIP failures and their correspondence with the safety levels defined in Table 1. This classification, which has been validated by experts involved in the system design and operation, takes into account the criticality of the services provided by the STIP. It is noteworthy that the "swl" failures are not critical when the rest of the CAUTRA subsystems are not failed. However, their impact becomes more significant when some of these subsystems are in a degraded mode (See [2] for more details).

Safety levels	Definition
IS1	Failure conditions which lead to a long interruption of the service and which prevent continued safe control
IS2	Failure conditions which lead to a highly degraded service until the application of alternative recovery means and the service continues in a full degraded mode with these means
IS3	Failure conditions which lead to a degraded service; the service can be continued in a degraded mode with the alternative means
IS4	Failure conditions which have minor effects on the service
IS5	No impact

Table 1 - Failures classification according to their impact on the air traffic safety

Notation	Definition	IS
swl	short interruption (≤15 min), without loss of data	5
lwl	long interruption (> 15 min), without loss of data	4
pl	interruption (independent of the duration), with partial loss of the data	3
tl	interruption (independent of the duration), with total loss of data	2

Table 2 - Classification of the STIP failures and corresponding IS levels

Table 2 leads to the partitioning of the STIP states into five state classes. Several paths between these classes may be observed. The transition from one class to another one corresponding to a more degraded service are due to the failure of recovery actions or the occurrence of additional hardware or software failures. To evaluate quantitative measures characterizing the impact of the STIP failures on the traffic safety, we have further considered the following grouping of states:
E2 = {tl} ; E3 = {tl, pl} ; E4 = {tl, pl, lwl}. Therefore, when the STIP occupies Ei, the minimum level of degradation is equal to ISi. Two measures can be evaluated:

- MTFFi (Mean Time to First Failure): mean time to the first visit to Ei (i = 2, 3, 4);
- UAi (Unavailability): Ei sojourn probability in steady-state (i = 2, 3, 4).

To evaluate these measures, we model the system behaviour resulting from the failure-repair assumptions presented in Section 3.

5 Modeling Approach

5.1 Principles

The STIP and the other CAUTRA sub-systems have complex hardware and software fault-tolerant architectures, characterised by a large number of components with complex behaviour and multiple interactions. Over the years, several model types such as combinatorial models (reliability block diagrams, fault trees, etc.), and state-space models (homogeneous Markov chains and their extensions) have been used to model systems and to evaluate various dependability measures [3]. Because they are able to capture various functional and stochastic dependencies among components (such as shared repair facilities), state-space models, in particular homogeneous Markov chains, are commonly used to model the dependability of fault-tolerant systems [4]. To alleviate the problem of specification and generation of complex models, higher level model types such as stochastic Petri nets [5] are generally used since they a) allow the definition of a more compact representation of a model (closer to the real system behaviour), b) provide some model structural

verification facilities, and c) can be automatically converted to Markov models. In our study we use the SURF-2 dependability modeling software tool, developed at LAAS-CNRS [6] which allows the construction and processing of Generalized Stochastic Petri Nets (GSPN). Nevertheless, the specification and the construction of complex GSPN models is a fastidious and an error-prone task. To master the models complexity, we have defined a modular and an incremental modeling approach based on:

- a *specification formalism* allowing the structured high-level description of each component behaviour, accounting for its interactions with the other components;
- a *set of transformation rules* allowing the specification formalism to be directly converted to a GSPN model;

The model is specified, built and validated in an incremental manner. At the initial step, the behaviour of the system is described taking into account the failures of only one selected component, assuming that the rest of the components are in an operational nominal state. The failures of the other components are integrated progressively. At each integration step, the GSPN model is derived and validated. The validation is carried out at the GSPN level (structural verifications) and also at the Markov level in order to check the different scenario represented by the model. When the Markov chain size increases, the exhaustive analysis of the Markov chain is impractical. In this case, sensitivity analyses are used to check the validity of the model assumptions.

To our knowledge, most of the studies that addressed the direct generation of dependability models from the translation of a specification formalism focus on the construction of Markov models without considering the intermediate step of GSPN generation. This latter step offers some model verification facilities allowing the user to gain confidence in the models on which judgements about the system dependability will be based (see for instance [7-10]).

Because of the lack of space, this paper is restricted to a brief presentation of the specification formalism through an example taken from the STIP; only the elements necessary for understanding the example are outlined. The transformation rules for the conversion of the specification formalism to a GSPN model are not presented.

5.2 Presentation of the Specification Formalism

The aim of the specification formalism is to provide a structured description of the system behaviour allowing an optimal GSPN model to be directly derived from the specification by the automatic translation of the formalism elements. The optimality criteria we have considered are: 1) conciseness and ease of model specification to enhance readability, 2) flexibility allowing, for instance, an easy modification of the model when new assumptions are to be considered, and 3) reusability of some parts of the formalism to facilitate the validation of the model and the specification and modeling of alternative fault-tolerant architectures based on similar assumptions.

5.2.1 Main Concepts

Our formalism, called an *evolution diagram*, provides a high-level description of each system component behaviour. It is composed of a graphical representation with textual definitions of some elements of the graph. An evolution diagram describing the behaviour of a given component is a directed graph made up of two types of

vertices: *phases* (represented by circles) and *evolution functions* (represented by rectangles), which are connected by edges. An unique evolution function is associated to each phase. A phase designates a state or a class of states of the component having some common features, while the associated evolution function describes the possible state changes from that phase. The evolution of a given component state may be due to the occurrence of local events (generated by the component itself) or external events (generated by other components). It is noteworthy that some local events may be common to several components (e.g. a common-mode failure affecting two software replicas). An evolution function defines on one hand, the local events that may occur during the sojourn of the component in one of the states of the associated phase (activation of faults, local repair actions, etc.), the enabling conditions associated to these events and the stochastic parameters characterizing each event. On the other hand, it describes, in a decision diagram-like form, each event occurrence consequences on the component itself and on its environment (various consequences are possible depending on the state of the component environment).

Different states from which the same kinds of events may occur can be identified in a component behaviour. Because the graphical representation of these different states increases the number of vertices of the evolution diagram which, therefore, becomes more difficult to understand, we chose to represent such a state class using one phase and defined Boolean *variables* to distinguish a particular state in the class when needed. These Boolean variables are not graphically represented but they are defined in the textual description of the evolution functions.

Moreover, to enhance the readability of the evolution diagrams associated to each component, only phase evolutions due to the occurrence of local events are represented graphically. Component state changes resulting from interactions between components are described in the textual definition of the evolution functions. Bold characters are used to identify these interactions.

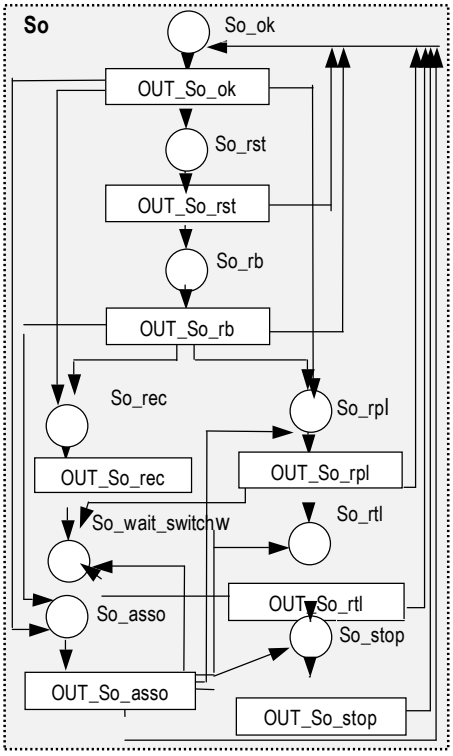
Figure 2 gives an example of an evolution diagram. The example describes the behaviour of the STIP So software replica that results from the failure-repair assumptions presented in Section 3. Only the evolution function associated to phase "So_ok" is presented and commented in the following.

5.2.2 Evolution Function Structuring

An evolution function attached to a phase P is denoted by OUT_P. Four levels can be distinguished in the specification of an evolution function.

– **level 1**, identified by the symbol "⇒", defines the local events that may occur from the states of P and the associated enabling conditions (if any). If an event is common to OUT_P and OUT_Q (Q being a phase of another component), it is identified by the symbol: "⇒ (OUT_Q)".

  Example: three events may occur from the phase S_ok corresponding to the three failure modes identified in Section 3. The first event is shared by phases So_ok and Ss_ok.



OUT_So_ok

⇒ (**OUT_Ss_ok**) Common mode failure of both replicas:
- DO (So_ok » So_rpl AND **Ss_ok » Ss_d**)
  [DO **(+Ss_M_der** AND +So_M_der AND +So_M_pl**)**]

⇒ So local failure - immediately diagnosed:
- IF **Ss_M_der** DO So_ok » Sp_rpl
  [DO (+So_M_der AND +So_M_pl]
- ELSEIF (**/Ss_M_der** AND **Cs_ok** AND **B_ok**) DO So_ok » So_rec
  [DO (*So_M_sw AND **+Ss_M_der** AND +So_M_der AND +So_M_swl AND **Stp_Ss** AND **Chg_D**]
- ELSE DO So_ok » So_asso
  [DO (+So_M_der AND **Stp_Ss** AND **Chg_D)**]

⇒ So local failure - not immediately diagnosed:
- DO So_ok » So_rst
  [DO +So_M_swl]

[p_1.m(Ss_ok),
p_2.(m(Ss_ok) + (1-m(Ss_ok)/(1-p_1)),

Notation	Phases and variables semantics
So_ok	So is in the "Up" state
So_rst	So is being restarted
So_rb	So while rebooting
So_rec	So while reconfiguration of Ss from spare to operational mode
So_wait_switch	So waiting for I/O switching
So_asso	So involved in a remote disk association
So_rpl	So is being restarted after partial dynamic data elimination
So_rtl	So is being restarted after total dynamic data elimination
So_stop	So is stopped (waiting for repair)
Ss_ok	Ss in the "Up" state
Ss_d	Ss is down (either stopped or under restart)
Cs_ok	Cs is in the "Up" state
B_ok	B is in the "Up" state
So_M_der	So variable characterizing the state of the data processed by So
So_M_sw	So variable used to memorise the need for an I/O switching
So_M_swl	So variable identifying an "swl" type service degradation
So_M_pl	So variable identifying a "pl" type service degradation
Ss_M_der	Ss variable characterizing the state of the data processed by Ss

Figure 2 - So evolution diagram and one of its evolution functions

The consequences of each event are defined at levels 2 and 3;

– **level 2**, identified by the symbol "•", describes the impact of each event on the component phase. The event occurrence may lead the component to move to another phase or keep it in the same phase. In the latter case, the event occurrence leads to the value modification of some internal variables. The phase change is either timed (represented by the symbol "»") or instantaneous (represented by the symbol "›"), depending on the corresponding event type. Several evolutions are possible from a given phase depending on the current state of the component environment. The test conditions are described as follows: IF (conditions) DO (consequences) ELSEIF DO() ELSE DO ().

  The sets of conditions are obtained by combining elementary conditions using AND, OR and "/" (for negation) logical operators. An elementary condition indicates the presence (or absence) of a component in one of its phases, or the logical value of a variable. It is required that the entire set of conditions of a decision diagram forms a *complete and exclusive system* in order to ensure that all the possible cases are considered in the specification.

  Example: the occurrence of a "local failure - immediately diagnosed" failure mode leads to three possible phase evolution depending on the state of the dynamic data processed by Ss (given by the value of the Ss_M_der variable) and the states of Cs and B. If the spare data are damaged, then So is restarted after the partial elimination of some dynamic data (So_ok » So_rpl);

– **level 3**, identified by the symbol "[ ]", describes the consequences of each event on the internal variables and on the states of the other components (i.e. on their internal variables and phases). Several groups of consequences are possible, depending on the current state of the system;

  Example: the failure mode "So local failure - immediately diagnosed" occurrence affects the dynamic data processed by So (+So_M_der) and leads the STIP to a "pl" degradation class (+So_M_pl). The "+" operator is used to set the value of a logical variable to "True";

– **level 4**, identified by the symbol "[ ] x [ ]^T", specifies in a matrix format the stochastic parameters associated to each event. The first array defines the probabilities associated to each event (1 is the default value) while the second one gives the occurrence rates for the timed events only (the symbol "∞" is used for the instantaneous events).

  Example: the parameters associated to the three So failure modes are specified as follows:
  $[p_1.m(Ss\_ok), p_2.(m(Ss\_ok)+(1-m(Ss\_ok)/(1-p_1)), (1-p_1-p_2).(m(Ss\_ok)+(1-m(Ss\_ok)/(1-p_1))]$ x [λ_so, λ_so, λ_so]

  λ_so is the failure rate of So. p_1 and p_2 are the probabilities of occurrence of a common mode failure and of a " local failure - immediately diagnosed" respectively when Ss is in the "ok" state. If Ss is in the "down" state, then the probabilities associated to the second and the third failure modes have to be updated. Function "m(Ss_ok)" returns the value 1 if Ss_ok is true and 0 if not. This function allows state dependent parameters to be specified. Therefore stochastic dependencies between component behaviours can also be specified in the evolution functions definition.

In addition to the concepts introduced above, the user can define *procedures* which can be reused by several evolution functions allowing the optimisation of the evolution functions description. The procedures are specified with a decision diagram-like format. The only condition imposed is that the associated set of

conditions should be complete and exclusive. For instance, Stp_Ss and Chg_D are two procedures used in the STIP specification. Stp_Ss is invoked when the occurrence of an event (for instance, the failure of Cs or Ds) causes the Ss replica to be stopped. Chg_D describes the switching of the disks roles (switch Do to Ds and vice-versa). Stp_Ss is defined as follows:

Stp_Ss: IF Ss_ok DO Ss_ok › Ss_d ELSE DO NIL

The NIL operator is introduced to satisfy the completeness property. It denotes that for the remaining set of conditions no action is performed.

6 Application to STIP and Evaluation Results

We have applied the incremental modeling approach presented in Section 5 in order to build a GSPN model describing the behavior of the STIP and evaluate quantitative measures characterizing the impact of the STIP failures on the traffic safety. Due to the complexity of the STIP, it is not possible to present the corresponding GSPN models. Table 3 outlines the different steps considered and the size of the corresponding Markov chains. Each step led to the validation of the model and the evaluation of MTTF and UA measures (see Section 4). At each step, we check that the assumptions considered at the previous step are also satisfied. Model construction and processing have been carried out with the tool SURF2.

Modeling steps	# Markov chain states
So	5
So - Ss	12
So - Ss - B	38
So - Ss - B - Co - Cs	104
So - Ss - B - Co - Cs - Do - Ds	212
So - Ss - B - Co - Cs - Do - Ds - Rep	256

Tableau 3 - The STIP modeling steps and size of the corresponding Markov chains

Two kinds of quantitative analyses have been carried out. Firstly we conducted sensitivity studies in order to identify the model parameters that have the most impact on the quantitative measures. These studies revealed the major impact of the software failure rates, compared to the rest of the parameters. Secondly, we analysed several operating configurations of the STIP in order to evaluate their impact on the air traffic safety. These configurations are obtained from the "reference" model corresponding to the assumptions presented in Section 3 by modifying some recovery scenarios or some model assumptions. The configurations studied are listed hereafter:

- "cold" spare instead of a "warm" spare;
- "hot" spare instead of a "warm" spare;
- try a "remote disk association" before the reconfiguration of the software replicas, whenever this is possible ("Rec->Asso");
- try a reconfiguration of the software replicas or a "remote disk association" instead of a reboot, each time the spare dynamic data are not damaged ("Shunt_Reboot") ;
- consider three repairmen instead of two: one for the I/O board and two for the computers and the disks.

Thanks to the specification and model construction method, the modification of the reference model to account for these alternative configurations was relatively easy. The MTFFi and UAi values corresponding to each configuration are listed in Table 4.

	MTFF 4	**MTFF3**	**MTFF2**	**UA4**		**UA3**		**UA2**	
				Pr. (E^{-6})	min/an	Pr. (E^{-6})	min/an	Pr. (E^{-6})	min/an
Reference	1324.9	3985.2	38422.1	165.9	87	41.5	21	4.0	2
Cold spare	743.0	1235.2	1885.1	304.4	159	159.5	83	112.9	59
Hot spare	1324.8	3985.1	38422.1	165.8	87	41.5	21	4.0	2
Rec->Asso	641.7	3986.7	38482.6	392.3	206	41.5	21	4.0	2
Shunt_Reboot	1324.9	3984.6	38332.6	108.1	56	41.6	21	4.0	2
3 Repairmen	1324.9	3985.2	38421.9	165.9	87	41.5	21	4.0	2

Table 4 - MTFF and UA for different STIP configurations

It can be noticed that the use of a warm spare instead of a cold one improves the safety measures: MTFF4, MTFF3 and MTFF2 increase by a factor of respectively 2, 3 and 20, while UA4, UA3 and UA2 decrease by a factor of respectively 2, 4 and 30. The improvement is more significant for class 2 which includes the most critical failures with respect to the traffic safety. It is noteworthy that the difference between these configurations decreases with the improvement of the disks reliability. Moreover, similar results are obtained for the "hot spare" and the "cold spare" configurations and also for the three and two repairmen cases. The configuration "Rec->Asso" leads to a small degradation of class 4 safety measures which concern the least critical failures. Finally, the configuration Shunt_Reboot leads to a significant improvement of UA4 (35%) against a weak decrease of MTFF2 (0.2%).

To conclude, the results given in Table 4 show the benefit of using a "warm spare" instead of a "cold spare". The other alternatives do not have a significant impact on safety. It is noteworthy that the STIP configuration that is currently operational is based on a cold spare. It is expected that the warm spare configuration will be introduced in the next release.

7 Conclusion and Future Work

In this paper, we analysed the failure impact of one subsystem of the French air traffic control computing system on the traffic safety. To master the complexity of this system, we presented a modeling approach that is based on a specification formalism allowing the structured description of the system behaviour and the automatic generation of a GSPN model from the specification. The aim of the formalism is to assist the modellers in the construction of dependability models. Moreover, we analysed several system operating configurations and evaluated the impact of each of them on the traffic safety. The results show the benefit of using a "warm spare" configuration instead of the "cold spare" configuration that is currently used. These results will be used to support the definition of the future architecture of the system. In the future, we will focus on the modeling and analysis of the CAUTRA subsystems implemented in the five en-route traffic control centers and the combination of the results obtained with those presented in this paper.

References

1. N. Fota, M. Kaâniche, K. Kanoun, and A. Peytavin, "The Air Traffic En route Control Computing System Dependability: Analysis and Modeling," Proc.10ème Colloque national de Fiabilité & Maintenabilité, Saint-Malo, France, 1996 (*in French*).

2. N. Fota, M. Kaâniche, K. Kanoun, and A. Peytavin, "Analysis of the Global Air Traffic en-route Control Computing System for Dependability Evaluation," LAAS-CNRS, LAAS Report n° 95280, june 1995 (*in French*).

3. A. L. Reibman and M. Veeraraghavan, "Reliability Modeling: An Overview for System Designers," *IEEE Computer*, vol. 24, pp. 49-57, 1991.

4. M. Malhotra and K. S. Trivedi, "Power-Hierarchy of Dependability-Model Types," *IEEE Transactions on Reliability*, vol. 43, pp. 493-502, 1994.

5. M. A. Marsan, G. Balbo, G. Franceschinis, and S. Donatelli, *Modelling with Generalized Stochastic Petri Nets*: John Wiley & Sons, 1995.

6. C. Béounes, M. Aguéra, J. Arlat et al. "SURF-2: A Program for Dependability Evaluation of Complex Hardware and Software Systems," Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23), Toulouse, France, 1993.

7. S. Berson, E. de Souza e Silva, and R. R. Muntz, "A Methodology for the Specification and Generation of Markov Models," in *Numerical Solution of Markov Chains*, W. .Stewart, Ed.: Marcel Dekker, 1991, pp. 11-36.

8. M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte, "Knowledge Modelling and Reliability Processing: Presentation of the FIGARO Language and Associated Tools," Proc. 10th Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP'91), Trondheim, Norway, 1991, pp. 69-75.

9. A. Goyal and S. S. Lavenberg, "Modeling and Analysis of Computer System Availability," *IBM Journal of Research and Development*, vol. 31, pp. 651-664, 1987.

10. J. A. Carrasco and J. Figueras, "METFAC: Design and Implementation of a Software Tool for Modeling and Evaluation of Complex Fault-Tolerant Computing Systems," Proc. 19th Int Symp. Fault-Tolerant Computing (FTCS-19), Vienna, Austria, 1986, pp. 424-429.