A Modeling Methodology for Hierarchical Control Systems and its Application

Paolo Lollini[1] Felicita Di Giandomenico[2] Andrea Bondavalli[1]

[1]University of Florence, Dip. Sistemi e Informatica, via Lombroso 67/A, I-50134, Italy
{lollini, a.bondavalli}@dsi.unifi.it
[2]Italian National Research Council, ISTI Dept., via Moruzzi 1, I-56124, Italy
digiandomenico@isti.cnr.it

Abstract

Current and future computerized systems and infrastructures are going to be based on the layering of different systems, designed at different times, with different technologies and components and difficult to integrate. Control systems and resource management systems are increasingly employed in such large and heterogeneous environment as a parallel infrastructure to allow an efficient, dependable and scalable usage of the system components. System complexity comes out to be a paramount challenge to solve from a number of different points of view, including dependability modeling and evaluation. Key directions to deal with system complexity are abstraction and hierarchical structuring of the system functionalities. This paper addresses the issue of an efficient dependability evaluation by a model-based approach of hierarchical control and resource management systems. We exploited the characteristics of this specific, but important, class of systems and derived a modeling methodology that is not only directed to build models in a compositional way, but it also includes some capabilities to reduce their solution complexity. The modeling methodology and the resolution technique are then applied to a case study consisting of a resource management system developed in the context of the ongoing European project CAUTION++. The results obtained are useful to understand the impact of several system component factors on the dependability of the overall system instance.

Keywords: Modeling Methodology, Quality of Service, Modular & Hierarchical Modeling, Petri Nets, Validation, Control Systems & Infrastructures

1 Introduction

Current and future computerized systems and infrastructures are based more and more on the layering of different systems, designed in different times, with different technologies and components and difficult to integrate. Control systems and resource management systems are increasingly employed in such large and heterogeneous environment to allow an efficient, dependable and scalable usage of the system components. In such landscape, system complexity comes out to be a paramount challenge to cope with from a number of different points of view, including dependability evaluation. Key directions to deal with system complexity are abstraction and hierarchical structuring of the system functionalities.

System evaluation is a key activity of fault forecasting, aimed at providing statistically well-founded quantitative measures of how much we can rely on a system. In particular, system evaluation achieved through modelling supports the prediction of how much we will be able to rely on a system before incurring the costs of building it. It is therefore a very profitable evaluation approach to be employed since the very beginning of a system development activity.

Most of the new challenges in dependability modelling are connected with the increasing complexity and dynamicity of the systems under analysis. Such complexity needs to be attacked both from the point of view of system representation and of the underlying model solution. In fact, the state space explosion is a well known problem in model-based dependability analysis, which strongly limits the applicability of this method to large complex systems, or heavily impacts on the accuracy of the evaluation results when simplifying assumptions are made as a remedy to this problem. Modular and hierarchical approaches have been identified as effective directions. Resorting to a hierarchical approach brings benefits under several aspects, among which: i) facilitating the construction of models; ii) speeding up their solution; iii) favoring scalability; iv) mastering complexity (by handling smaller models through hiding, at one hierarchical level, some modeling details of the lower one). At each level, details of the architecture and of the status of lower level components are not meaningful, and only aggregated information should be used. Therefore, information of the detailed models at one level should be aggregated in an abstract model at a higher level. Important issues are how to abstract all the relevant information of one level to the upper one and how to compose the derived abstract models. However, it is important to underline that the modularity of the modelling approach alone cannot be truly effective without a modular solution of the defined models.

In this paper, we focus on the class of control and resource management systems. To cope with their increasing complexity, such systems are typically developed in a hierarchical fashion: the functionalities of the whole system are partitioned among a number of subsystems working at different levels of a hierarchy. At each level, a subsystem has knowledge and control of the portion of system under its control (lower levels), while it acts just as an actuator

with respect to the higher level subsystems. In this organization, the flow of the information goes vertically from one level to the other, but not horizontally inside the same level. More precisely, the flow of decision taking goes from the bottom to the top, while the flow for decision actuation goes from the top to the bottom. Here we are interested in modeling and evaluating the system behavior with reference to a unidirectional flow (be it for decision taking or for decision actuation). To improve dependability, fault tolerance measures may be taken at each level, typically interface checks to cope with erroneous inputs and/or outputs and internal checks to cope with faults during the internal computation. We exploited the characteristics of this specific, but well representative, class of systems and derived a modeling methodology that is not only directed to build models in a compositional way, but it also includes some capabilities to reduce their solution complexity. To show how it works, in the second part of the paper we applied the methodology to a case study, which consists of a resource management system developed inside the CAUTION++ project [1].

The rest of this paper is organized as follows. Section 2 provides some preliminaries on the considered class of systems. Section 3 outlines the modeling approach. Section 4 presents the multi-stage system instance considered in the analysis. In Section 5 the models set-up for the selected CAUTION++ instance are discussed, while the results of the numerical evaluation are provided in Section 6. Finally, conclusions are in Section 7.

2 System Context

The class of systems we focus on consists of a set of hardware or software components (the COMP boxes), which are grouped in "stages" (Stage 1, ..., Stage k, ..., Stage N), as shown in Figure 1. Components at a certain stage

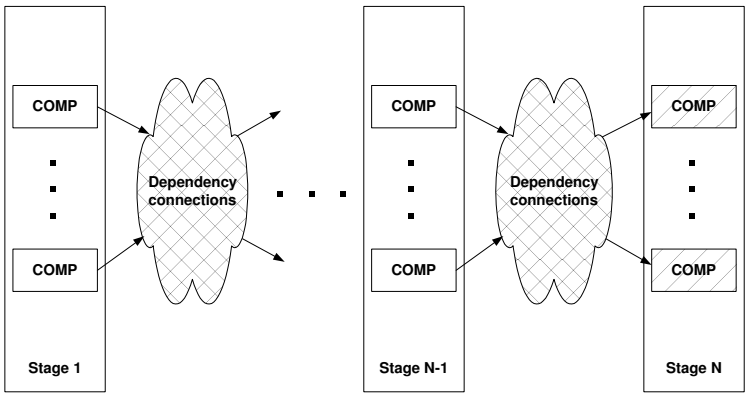


Figure 1. Class of Systems with "multi-stage" representation

may interact with others at an higher level through some "Dependency connections". Each connection identifies a dependency between two system components: a component A is connected to a component B ($A \longrightarrow B$) if B is dependent from A, that is the behavior of B depends on the behavior of A. The components without any

incoming connections have an independent behavior with respect to the others, while the components without any outgoing connections (called *root* components, the dashed boxes in the figure) do not affect the behavior of any other component.

From the general system depicted in Figure 1 and following the dependency connections from a root component back to the leaves of the graph, a number of individual subsystems structured in a hierarchical fashion may be derived, equal to the number of root components.

As already discussed earlier, a component at stage k may interact only with those at stages $k-1$ and $k+1$ and these dependencies are unidirectional, from the lower stage to the higher one. A dependency between one component at stage k and more than one component at stage $k+1$ is not explicitly considered as it is equivalent to consider some (logical) replications of the component at stage k, each one interacting with only one component at stage $k+1$.

The components in a stage can be partitioned in more sub-sets (*groups*), each one composed of components having a connection to the same component in the next stage.

For a better understanding, let us consider the example of Figure 2. It is a system with eight different compo-

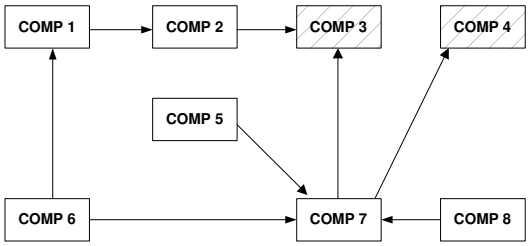


Figure 2. Example of System

nents, two of which are root nodes.

The corresponding representation, by grouping components in stages, is shown in Figure 3. The original system

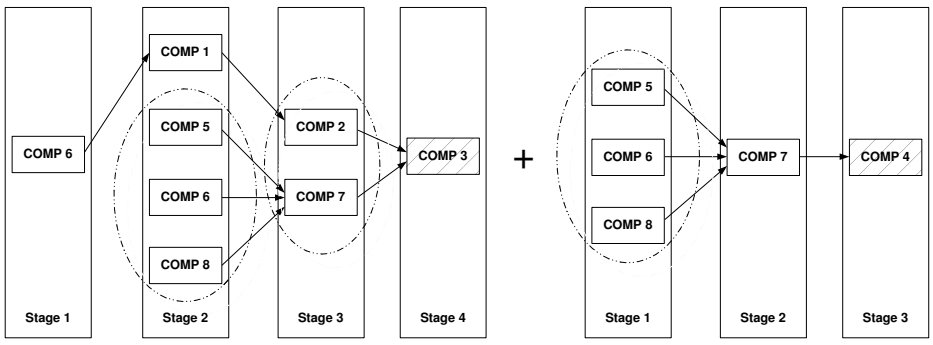


Figure 3. Example of "multi-stage" representation

has been decomposed in two sub-systems of four and three stages, respectively, obtained following the reversal

path from each root node to the leaves. We note that COMP 6 is replicated twice in the first sub-system, as it is originally connected to two different components (COMP 1 and COMP 7, see Figure 2). We identify the groups composed of more that one component with a dotted circle.

In the following Subsection we detail the system's behavior, specifying how two generic components may interact each other.

2.1 Interactions between Components and Measures of Interest

The interactions among components and the failure assumptions on each component are highlighted in Figure 4. This scheme is very general and must be specialized for the particular component under analysis. To explain the generic component's behavior, let's suppose it receives an input following a Poisson distribution with a rate λ^{IN}. These inputs are assumed to be correct or incorrect with a probability α and $1-\alpha$, respectively.

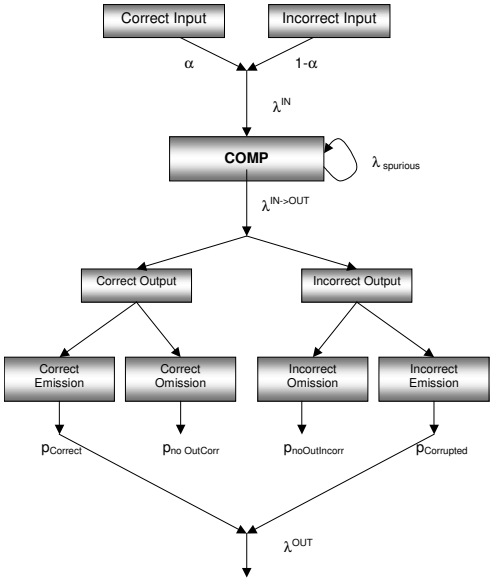


Figure 4. How a Generic Component Interacts with Others

In correspondence of inputs, which arrive with a rate λ^{IN}, the component produces an output with a rate $p*\lambda^{IN}$, where p is the probability a received input leads the component to produce an output. Moreover, the component is assumed to possibly behave incorrectly by self-generating spurious outputs with a rate λ^S. Thus, the "potential"[1] output rate of the component is expressed as $\lambda^{IN \rightarrow OUT} = \lambda^{IN} + \lambda^S$. From the point of view of propagation, an output issued by COMP is propagated to another component with a rate $\lambda^{OUT} = (p_{Correct} + p_{Corrupted}) * \lambda^{IN \rightarrow OUT}$, where $p_{Correct}$ and $p_{Corrupted}$ represent the probabilities of generating a correct output (correct emission) and an incorrect output (incorrect emission), respectively. A *correct emission* happens whenever a correct output is

[1]Here, a "potential" output encompasses both emitted and omitted output ($p = 1$), while for "output" we refer only to those emitted.

Input	Corresponding feasible output
Spurious output (internally generated)	Incorrect Emission
Correct input	Correct Emission, Incorrect Emission, Incorrect Omission
Incorrect input	Correct Omission, Incorrect Emission

Table 1. Input-output combinations

Input parameters	Output parameters
α, λ^{IN}	λ^{OUT}, $p_{Correct}$, $p_{noOutCorr}$, $p_{noOutIncorr}$, $p_{Corrupted}$

Table 2. Input-output parameters for a component model

produced. A correct emission is possible i) in response to a correct input if the system is free from errors, or ii) in response to a correct input, if system errors are detected and tolerated. An *incorrect emission* happens either in reply to an incorrect input, or as consequence of a spurious output or of a wrong processing of a correct input. A *correct omission* may happen as consequence of an incorrect input or of an erroneous status of the system. An *incorrect omission* may happen as consequence of wrong processing of a correct input. These input-output combinations are summarized in Table 1. The input/output parameters characterizing each component are instead summarized in Table 2, where $p_{noOutCorr}$ and $p_{noOutIncorr}$ are the probabilities that the output is correctly omitted and incorrectly omitted, respectively.

Given the behavior structure and failure semantics depicted in Figure 4, typical measures of interest from the dependability point of view in this context include:

1. The probability of correct and incorrect emission;
2. The probability of correct and incorrect omission;
3. The overall probability that the system does not undertake wrong actions;
4. The mean time to incorrect emission.

In Section 5 we will specify the measures to evaluate with reference to a particular resource management system.

3 Description of the Modeling Methodology

The modeling methodology, originally introduced in [2], is fully described in this section. First, we deal with the model design process, that is, how to model a complex system starting from its functional specification and applying a stepwise refinement to decompose it in small sub–models. Then, the second part of the methodology is presented, which concerns the modular model solution, carried out in a bottom-up fashion. The philosophy of our modeling approach is shown in Figure 5.

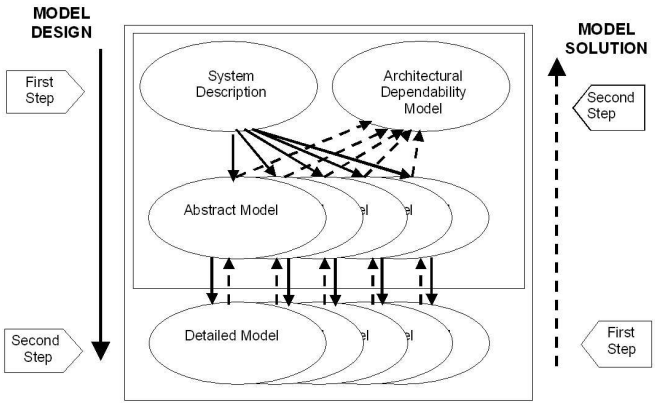


Figure 5. Modeling approach

In order to construct an efficient, scalable and easily maintainable architectural model, we introduce a stepwise modeling refinement approach, both for the model design process and for the model solution. Another advantage of this approach is to allow models refinement as soon as system implementation details are known or/and need to be added or investigated.

3.1 The Model Design process

The model design process adopts a top-down approach, moving from the entire system description to the definition of the detailed sub-models, while the model solution process follows a bottom-up approach. As inspired by [3], the system is firstly analyzed from a functional point of view (functional analysis), in order to identify its critical system functions with respect to the validation objectives. Each of these functions corresponds to a critical service provided by a component.

The overall system is then decomposed in subcomponents, each one performing a critical subfunction, and each subfunction is implemented using a model that describes its behavior. Therefore, starting from the high-level abstract model, we perform a decomposition in more elementary (but more detailed) sub–models, until the required level of detail is obtained.

The definition of the functional (*abstract*) model represents the first step of our modeling approach. The rules and the interfaces for merging them in the architectural dependability model are also identified in this phase. The second step consists in detailing each service in terms of its software and hardware components in a detailed (*structural*) model accounting for their behavior (with respect to the occurrence of faults). The fundamental property of a functional model is to take into account all the relationships among services: a service can depend directly from the state of another service or, indirectly, on the output generated from another service. The detailed model defines the structural dependencies (when existing) among the internal sub–components: the state of a

sub–component can depend from the state (failed or healthy) of another sub–component.

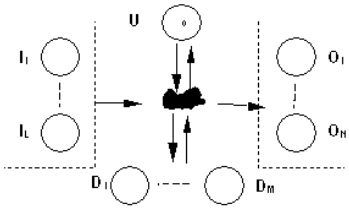


Figure 6. Functional-level model related to a single service

Figure 6 shows the functional-level model related to a single service. The internal state S is here composed of the place U, representing the nominal state, and of the places $D_1 \dots D_M$, representing different possible erroneous (degraded) states. The places $I_1 \dots I_L$ and $O_1 \dots O_N$ represent, respectively, the input (correct or exceptional, due to propagation of failures from interacting modules) and the output of the model (correct behavior or failure - distinguishing several failure modes). The state changes (from the nominal, correct state to the erroneous states and viceversa) and the flow between the input and output places are regulated by a structural model of the service implementation, indicated in Figure 6 as a black cloud.

3.2 The Model Solution process

The model solution follows a bottom-up approach from the detailed model up to the abstract model. The implementation is strictly related to the environment characteristics of the system under analysis. Actually, starting from the general class of systems of Figure 1, we can derive several simplified systems that can be solved very efficiently.

3.2.1 Environment characteristics

Suppose, for the sake of simplicity, that the generic system of Figure 1 has one root node only. If it is not the case, we can decompose the system in more sub-systems having one root each, as explained in Section 2. We denote with $\lambda_i^{OUT,\ COMP^k}$ the intensity of the output process of the i-th component belonging to stage k (COMP_i^k). We make the following assumptions:

1. The distribution of the input process of each component is Poisson with rate λ^{IN}. This is accepted in the literature when the number of arrivals in a given time interval of time are independent of past arrivals.

2. The distribution of the output process of each component is Poisson distributed with a rate λ^{OUT}. This assumption corresponds, for example, to the case in which the inputs are processed sequentially without queuing and losses, and the processing time of the input is deterministic. Equivalently, we could obtain the

same output distribution considering that the service time is Poisson distributed and that the component operates as a steady-state M/M/1 queuing network [4].

Suppose to have a group of N_k components at stage k (COMP_1^k, ..., $\text{COMP}_{N_k}^k$). We remember that a group is a set of components belonging to a stage, and connected to the same component in the next stage. Using the assumption that the output process of COMP_i^k is Poisson distributed with rate $\lambda_i^{OUT,\ COMP^k}$, the superposition of N_k Poisson processes with intensities $\lambda_1^{OUT,\ COMP^k}, \ldots, \lambda_{N_k}^{OUT,\ COMP^k}$ is equivalent to a Poisson process with intensity equal to $\lambda_1^{OUT,\ COMP^k} + \ldots + \lambda_{N_k}^{OUT,\ COMP^k}$.

Solving the detailed model of components COMP_1^k, ..., $\text{COMP}_{N_k}^k$ leads to the evaluation of the probabilities of correct/incorrect output emission/omission and the intensity of the output process of a group of N_k components. Let's defining as $P_{Correct}^{k_i}$, and $P_{Corrupted}^{k_i}$ the probability of correct emission, and the probability of incorrect emission of COMP_i^k, respectively. Notice that these probabilities depend upon the intensity of the input process ($\lambda_i^{IN,\ COMP^k}$) and of spurious alarms ($\lambda_i^{S,\ COMP^k}$) (both supposed being Poisson). The following relations holds:

$$\Lambda^{OUT,\ COMP^k} = \sum_{i=1}^{N_k} \lambda_i^{OUT,\ COMP^k} \quad , \tag{1}$$

$$\alpha_{COMP^{k+1}} = \frac{1}{\Lambda^{OUT,\ COMP^k}} \sum_{i=1}^{N_k} \lambda_i^{OUT,\ COMP^k} \frac{P_{Correct}^{k_i}}{(P_{Correct}^{k_i} + P_{Corrupted}^{k_i})} \quad , \tag{2}$$

where $\Lambda^{OUT,\ COMP^k}$ is the intensity of the process achieved by aggregating the output processes of the components COMP_1^k, ..., $\text{COMP}_{N_k}^k$, while $\alpha_{COMP^{k+1}}$ is the probability that the next component at stage $k+1$ receives a correct input. Analogous considerations hold for COMP^{k+1}, and so on. This general approach can be specified for the following cases:

- If all groups at stage k are identical, the total number of detailed models to be solved in order to evaluate the system's behavior is equal to $\sum_{k=0}^{K} N_k$, where K is the number of stages in the system and N_k is the number of components belonging to each identical group at stage k.

- If all groups at stage k can not be considered identical at each stage, the number of models to be solved depends on the number of different "branches" in which the overall model can be simplified.

- If for each stage k of the system, all the components are identical, it is possible to solve only K detailed models, one for each stage. Therefore, if all the components at level k are identical, than $\lambda_i^{OUT,\ COMP^k} = \lambda^{OUT,\ COMP^k}$, $P_{Correct}^{k_i} = P_{Correct}^{k}$, $P_{Corrupted}^{k_i} = P_{Corrupted}^{k}$, and the previous equations reduce to

$$\Lambda^{OUT,\ COMP^k} = N_k^{TOT} * \lambda^{OUT,\ COMP^k} \quad , \tag{3}$$

$$\alpha_{COMP^{k+1}} = \frac{P^k_{Correct}}{(P^k_{Correct} + P^k_{Corrupted})} \quad , \tag{4}$$

where N_k^{TOT} is the total number of components at stage k.

In this case, the general model of Figure 1 is reduced to the equivalent simplified system model of Figure 7 that can be solved more easily, as the "tree" structure collapses in a unique "branch" from the point of view of system evaluation.

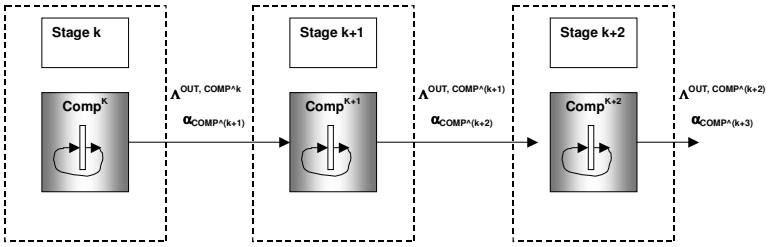


Figure 7. Part of the simplified system model

If it can not be assumed that the output process of COMP_i^k follows a Poisson distribution, the general approach is still valid provided that the detailed model is slightly modified allowing to estimate the real distribution of such a process. The same distribution will be used as input at the $k+1$ stage. However, in general, it will be no longer possible to solve the models analytically.

If the measures of interest are probabilities, the moments of the distribution of the events which yield such probabilities are not considered at all. In this case it is not necessary to use, at the abstract level, models having the same distribution estimated at the detailed ones. If, on the contrary, we are interested in evaluating the moments of the distribution of correct/incorrect output emission/omission, the output processes distributions achieved by the detailed models have to be used for the solution of the abstract models.

3.2.2 The model solution scheme

According to Figure 5 (showing the philosophy of our modeling approach) the model solution follows a bottom-up approach: the solution of a detailed model is exploited to set up the parameters of the corresponding abstract model and of the detailed model of the next (contiguous) components (the output of the detailed COMP^k model acts as input for the detailed COMP^{k+1} model). To keep the presentation simple, the model solution scheme is described in the case where, for each stage k, all the components at stage k are identical; therefore only K detailed models (one for each stage) have to be solved. Figure 8 shows the relationships among a detailed model of COMP^k and the model COMP^{k+1}.

With reference to the measures of interest listed in Section 2.1, the outcomes of the detailed model COMP^k are:

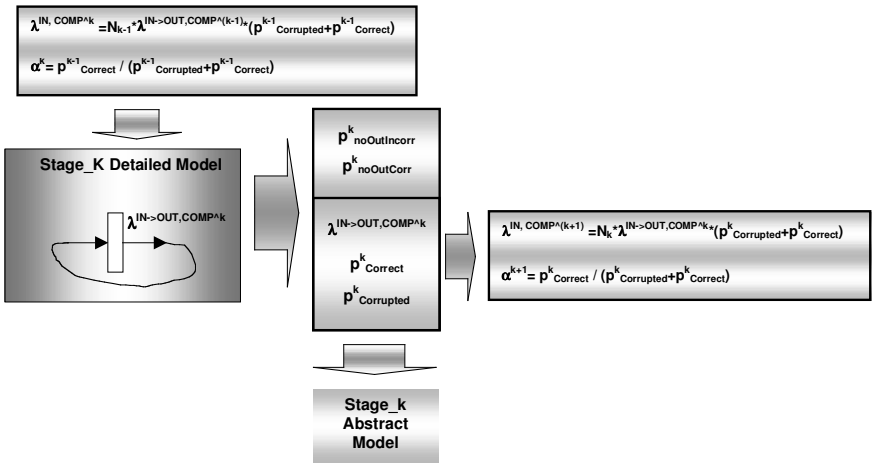


Figure 8. Relationships between models solutions

1. $p^k_{noOutCor}$: is the probability that no output is produced by component COMP^k, as a consequence of an incorrect input;

2. $p^k_{noOutIncorr}$: is the probability that an expected output is incorrectly not propagated by component COMP^k, as consequence of an internal fault;

3. $\lambda^{IN \rightarrow OUT,\ COMP^k} * (p^k_{Corrupted} + p^k_{Correct})$: is the rate of messages propagated by component COMP^k to component COMP^{k+1};

4. $p^k_{Correct}$: is the correct emission probability;

5. $p^k_{Corrupted}$: is the emission failure probability. This value encompasses both an expected wrong emission (as consequence of wrong internal processing) and the unexpected emission (as consequence of an internal self-generated false alarm).

All these parameters are used in the abstract model of component COMP^k (see Figure 8) while $\lambda^{IN \rightarrow OUT,\ COMP^k}$, $p^k_{Correct}$ and $p^k_{Corrupted}$ are used to derive the parameter $\lambda^{IN,\ COMP^{k+1}}$ to be used in the detailed model of COMP^{k+1}. In the system framework COMP^k and COMP^{k+1} represent two components directly connected that exchange messages in one direction (from COMP^k to COMP^{k+1}).

Summarizing, the overall solution scheme is shown in Figure 9. The detailed models are solved separately: firstly, it is solved the model of COMP^k, then the values provided by equations (3) and (4) are passed as input to the detailed model of COMP^{k+1} and so on. Finally, the probabilities of correct/incorrect output emission/omission are passed to the corresponding abstract models, they are joined together and then the overall abstract model is solved.

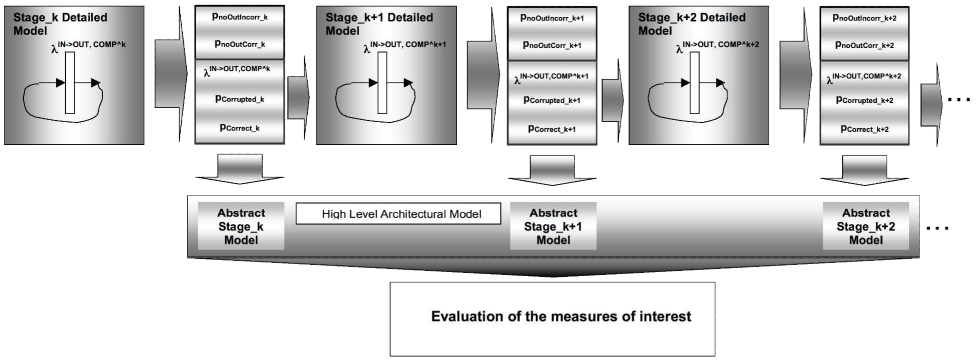


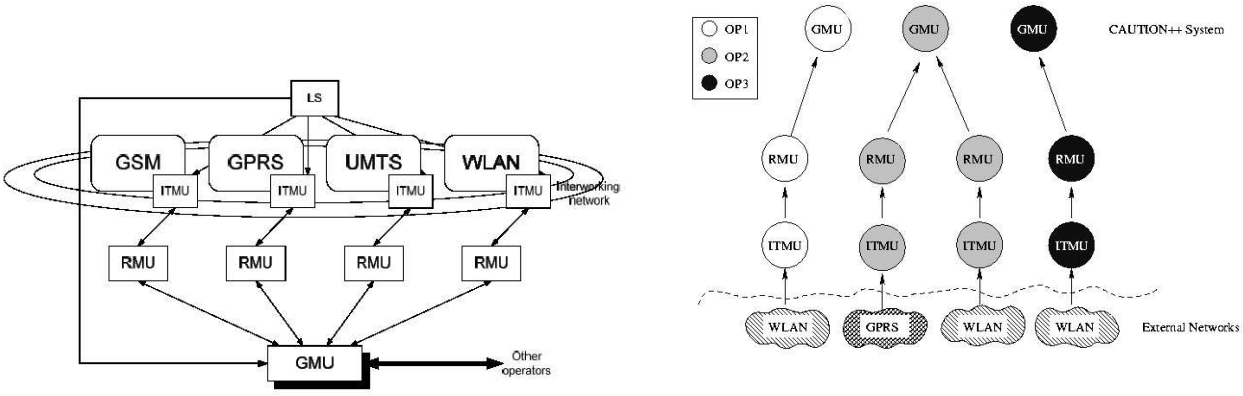
Figure 9. Overall Solution Scheme

The advantages of the proposed approach are in two directions: first, to cope with the problem of state space explosion when modeling a complex system and, second, to allow efficient model solution for those systems having most of their components identical and interacting each others only by means of message exchange. Actually, in case the components are not all equal, a larger number of detailed models have to be solved but still separately. Thus, the overall model, encompassing all the useful information with respect to the measures of interest, is achieved by joining the abstract models.

4 An instance of a "multi-stage" system: the CAUTION++ system

The IST-2001-38229 CAUTION++ [1] aims at developing a novel, low cost, flexible, highly efficient and scalable system able to be utilized by mobile operators to increase the performance of all network segments. Capacity utilization in cellular networks is an extremely important issue from the operators' point of view. Successful usage of all the system resources especially in congestion situations can imply increased revenues for the cellular network operators via reduced call blocking and dropping rates. Also, in emergency situations the cellular networks are expected to work properly and be able to respond to the momentarily increased offered traffic. To pursue such goals, proper system components are developed to handle generated alarms through a set of RRM (Radio Resource Management) techniques, to be applied where needed. The CAUTION++ system, superimposed over the existing wireless networks, should allow putting in place correctly the identified RRM techniques, hopefully despite the occurrence of faults. The rationale is to enforce design solutions able to prevent a CAUTION++ component from carrying out a reconfiguration action wrongly or when is not necessary (as consequence of some fault). Because of the involved functionalities which pose relevant dependability issues, the CAUTION++ project has promoted model-based evaluation, aiming at assessing dependability attributes of the architecture under development.

Figure 10(a) shows the main components of the CAUTION++ architecture. Each network segment has its own ITMU (Interface Traffic Monitoring Unit) and RMU (Resource Management unit) which allow to monitor and



(a) (b)

Figure 10. (a) Network Architecture for provision of capacity management mechanisms and (b) Trial Configuration

manage the attached network, respectively. Within each operator network, a GMU (Global Management unit) can perform a global optimization. A Location Server (LS) can be used to track users' mobility and location: such information can be exploited by GMU for a global optimization.

To practically show the usage of the proposed modeling methodology, in this paper we consider a specific architecture's instance involving GSM/GPRS and WLAN network technologies deployed by two distinct operators, which is actually one of the demonstrators chosen by the consortium to show the project's results.

From the point of view of system composition, Figure 10(b) depicts the components included in such trial. Three operators are involved, Op1, Op2 and Op3, with Op1 and Op3 managing a WLAN network only, and Op2 managing both a GPRS and a WLAN network. From the point of view of CAUTION++ components employed in this instance, each network segment has its own ITMU (Interface Traffic Monitoring Unit) and RMU (Resource Management Unit) which allow to monitor and manage the attached network, respectively. Within each operator network, a GMU (Global Management Unit) is necessary to perform a global optimization. In fact, different GMUs cooperate to optimize among different operators. Therefore, this CAUTION++ instance includes 4 ITMU, 4 RMU and 3 GMU, connected as shown in Figure 10(b).

It is clearly an instance of a multi-stage system. Starting from the GMU components (the root nodes of the graph, see Section 2), we decompose the system in three subsystems, one for each GMU. Each subsystem can be seen as a "3-stage" system, that is a "multi-stage" system composed of 3 stages, in which all the components belonging to a stage are identical. Moreover, each subsystem can be represented as shown in Figure 7, as the "tree" structure collapses in a unique "branch" from the point of view of system evaluation. Therefore we have to solve only 3 detailed models for subsystem.

4.1 Components behavior and modeling assumptions

In order to set up the detailed models, a characterization of the system components from the dependability point of view is necessary, briefly outlined in the following.

- Each CAUTION++ element (ITMU, RMU, GMU) can be either correctly working or wrongly working.
- Each CAUTION++ element (ITMU, RMU, GMU) is composed by three main elements: the Application Software (AS), the Operating System (OS), and the Hardware (HW). Each element has its own dependability figures and reference values, that have been chosen as explained later. In turn, the AS, OS, and HW can be either correctly working or wrongly working.
- At the end of its computation, each CAUTION++ component can emit an output or not. More precisely, the possible output can be either correct/incorrect emission or correct/incorrect omission.
- Fault tolerance mechanisms are in place in each system component, in order to improve the dependability of the components themselves and limit the error propagation between interacting elements. They are interface checks (to detect errors at input/output level), diagnosis and repair mechanisms. Their ability to work properly depends on their respective coverage.

In addition, a set of assumptions has been identified with the aim of enhancing simplicity and clarity (essential to keep the whole modeling activity under control), still capturing the relevant phenomena which impact the measures under analysis (essential to the practical usefulness of the evaluation effort). The complete list is in [5] and [6]; here we omit those strictly related with details of the models not shown in this paper.

- The input to the detailed model may be either correct with probability α or incorrect with probability 1-α.
- Each CAUTION++ element (ITMU, RMU, GMU) can generate by itself spurious outputs (that is, outputs not triggered by an external input; it is a manifestation of a fault in the component). Spurious outputs are independent from outputs generated by real inputs and follow an exponential distribution.
- The coverage of the Input interface checks is given by the probability *inputCoverage*. When Output interface checks are considered, the detection of an erroneous output leads to an output omission (correct or incorrect, depending from the inputs originating it and/or the correctness of the component's status) with probability *outputCoverage*.
- An undetected erroneous state of the AS may disappear when the OS is repaired, e.g. in the case of OS re-booting.

- An undetected erroneous state either at the AS or OS level may disappear when the HW is repaired (because of necessary system reboot, no hot-pluggable redundancy is envisioned).
- An undetected erroneous state either disappears or propagates and reveals itself.

5 Sketch of the models derived for the selected CAUTION++ trial

In this Section, the models derived for the analysis of the selected CAUTION++ instance of Figure 10(b) are briefly outlined. First, the measures of interest are described, since they influence the definition of the system models.

5.1 Measures of Interest

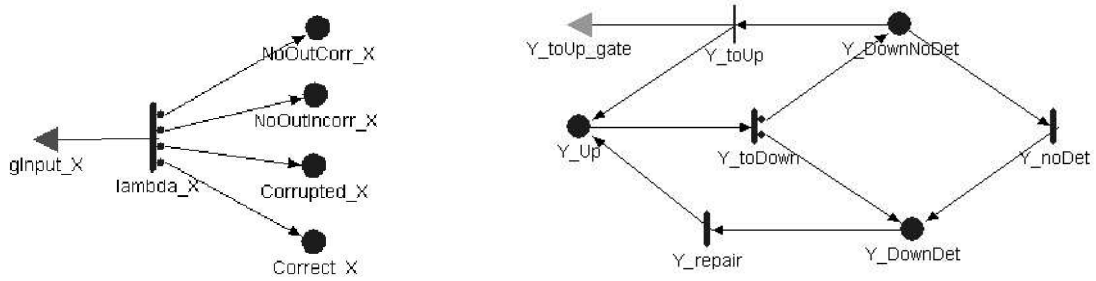
As previously mentioned, the goal of the CAUTION++ system is to increase the performance of all the controlled cellular networks. Then we expect it can not never have a negative impact on the networks behavior, at the most becoming inactive in the worst case. Therefore, the main dependability requirement of CAUTION++ is that it should avoid taking wrong decisions, thus acting worse than doing nothing. Particularly, an omission failure (that is the system does not provide any output when, if correct, it would have emitted one) can be tolerated, since it leads to no benefit from CAUTION++. Emission failure instead (that is, an incorrect output is emitted) can lead the system to act worse than doing nothing, and therefore actions would be required to prevent such failure mode. We have identified the following indicators as significant measures to evaluate the dependability of the CAUTION++ architecture. They are:

- The probability of incorrect emission at level of the GMU employed by a certain operator;
- Mean Time to Failure of the GMU employed by a certain operator;
- Reliability of the whole system(with contributions from all the present GMUs).

They appear to be suitable measures to evaluate the ability of CAUTION++ in fulfilling the general dependability requirement of not undertaking wrong reconfiguration actions.

5.2 Detailed and abstract models

In accordance with the proposed methodology described in Section 3, the starting point is the definition of an "abstract" model for each involved component. The generic "abstract" model is represented in Figure 11(a), using the SAN [7] formalism.



(a) (b)

Figure 11. (a) Generic Abstract Sub-models and (b) Detailed Model for AS, OS, and HW

It is valid for ITMU, RMU and GMU. The input gate *gInput_X* allows handling the input of the component (both the correct and incorrect input), transition *lambda_X* fires with a rate given by the rate of messages in input to component X. Possibly, an output is produced, which can be either correctly emitted (a token is moved in place *Correct_X*), or incorrectly emitted (a token is moved in place *Corrupted_X*), or correctly omitted (a token is moved in place *NoOutCorr_X*) or incorrectly omitted (a token is moved in place *NoOutIncorr_X*).

To obtain the parameters of each abstract model, the corresponding detailed models have to be set-up and solved. Therefore, a detailed model is built for each involved component. Since ITMU, RMU and GMU employ the same subcomponents (HW, OS and AS, plus fault tolerance mechanisms, as already discussed), the detailed model is almost the same for all of them. The only difference is in the values of their parameters (as explained later in the section on numerical evaluation). A generic detailed model is obtained by composing the generic detailed models for the component's subcomponents (i.e., HW, OS and AS) together with the dynamics of the error and fault detection mechanisms employed. The presentation of this model is omitted for brevity (refer [5] for a complete exposition); here only a simplified generic detailed model for the subcomponent Y (where Y maybe AS, OS or HW) is sketched in Figure 11(b).

A token in place *Y_Up* indicates that Y is working correctly. The firing of transition *Y_toDown* models its failure: this failure can be detected (a token moves in the place *Y_DownDet*) or not (a token moves in the place *Y_DownNoDet*) with probabilities *Y_Coverage* and 1- *Y_Coverage*, respectively (*Y_Coverage* represents the coverage of the error detection mechanisms implemented in the element Y). An undetected failure can be revealed after a while; the firing of transition *Y_noDet* indicates such failure detection. A detected failure is then recovered by means of the transition *Y_repair*. An undetected erroneous state may disappear if the input gate *Y_toUp_gate* enables the instantaneous transition *Y_toUp*, for example in the case of OS re-booting if Y = AS.

The overall model for the CAUTION++ instance under analysis has been constructed under the following assumptions:

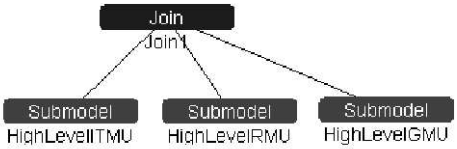


Figure 12. Composed Model at GMU decision level

- Messages coming from different ITMUs and RMUs are indistinguishable.
- The RMUs and the GMUs process the incoming input requests (from the ITMUs and RMUs respectively) individually and sequentially.

Figure 12 shows the SAN composed model for analyzing the CAUTION++ behavior at a single GMU decision level (e.g., to evaluate the probability of correctness of a reconfiguration decision issued by a GMU). Thanks to the above assumptions, the evaluation of the whole CAUTION++ instance is easily obtained by mathematically combining the evaluations at single GMU level, in accordance with the specific measure under analysis.

6 Evaluation results

The preceding models have been numerically solved using the analytical solver provided by the Möbius tool ([8]). Since all the timed transitions are exponentially distributed and the state space dimension of the models was not huge, it was possible to pursue an analytical solution achieving more accurate results than through simulation. Given the nature of the measures of interest, we resorted to a steady-state analysis for all models.

6.1 Settings for the Numerical Evaluation

The developed models have a number of internal parameters, to which values have to be assigned. For many of them, reference values from manufactures or previous studies in the literature are available. For others, mainly those concerning the components to be developed in the CAUTION++ framework, this is not true and the choice of appropriate values is more critical. Therefore, for such critical parameters, a range of values is experimented in the analysis, to determine the impact of such variations on the analyzed dependability figures (sensitivity analysis). Table 3 lists the varying parameters, and the range of values assigned to them in the analysis. The extension $\_X$ makes the parameter's name generic, and need to be properly substituted by ITMU, RMU, GMU to indicate the parameters of the corresponding component. Since the models have been just sketched in this paper, not all the involved parameters have been listed in Table 3. The values assigned to the missing parameters are the same applied in [5].

The meaning of the parameters in Table 3 is as follows:

Parameter	Range
α_{ITMU}	0.90 - 0.999
α_{RMU}	from ITMU
α_{GMU}	from RMU
MTBA_ITMU	2- 48(hours)
MTBA_RMU	from ITMU
MTBA_GMU	from RMU
InputCoverage_X	0.00 - 1.00
OutputCoverage_X	0.00 - 1.00
AS_Coverage_X	0.70 - 0.999
MTBFA_X	198 - 2000(hours)

Table 3. Varying Model Parameters and their values

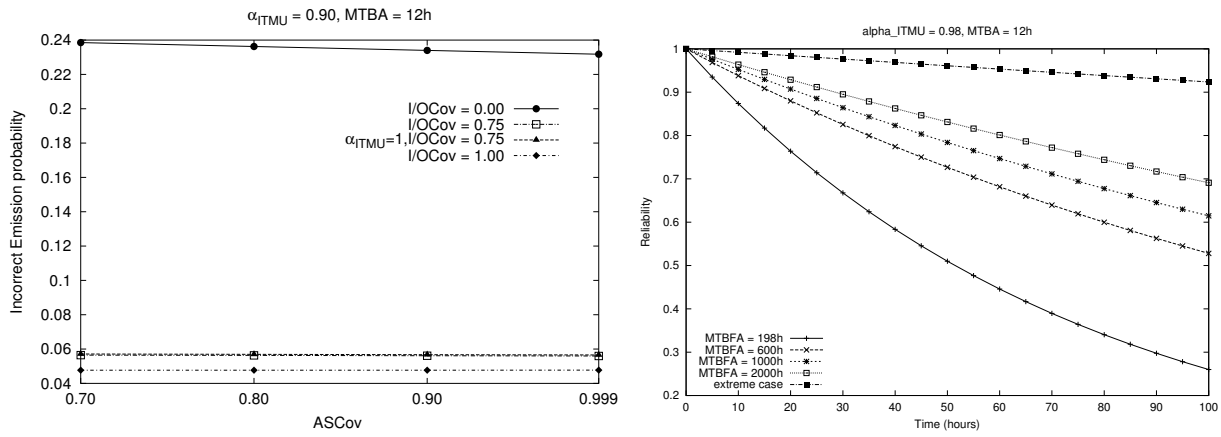
- α_{ITMU}, α_{RMU} and α_{GMU} are the probabilities that the input provided to ITMU, RMU and GMU, respectively, is correct;
- MTBA_ITMU, MTBA_RMU and MTBA_GMU are the mean time between two inputs to ITMU, RMU and GMU, respectively (in the case of ITMU, it is the mean time between two external inputs for which ITMU generates an alarm to RMU);
- MTBFA_X is the mean time between two spurious outputs emitted by a generic component X ;
- InputCoverage_X is the coverage of the error detection checks at input interface;
- OutputCoverage_X is the coverage of the error detection checks at output interface;
- AS_Coverage_X is the coverage of the application software checks.

6.2 Numerical Evaluation

In this section, we present and discuss the results obtained.

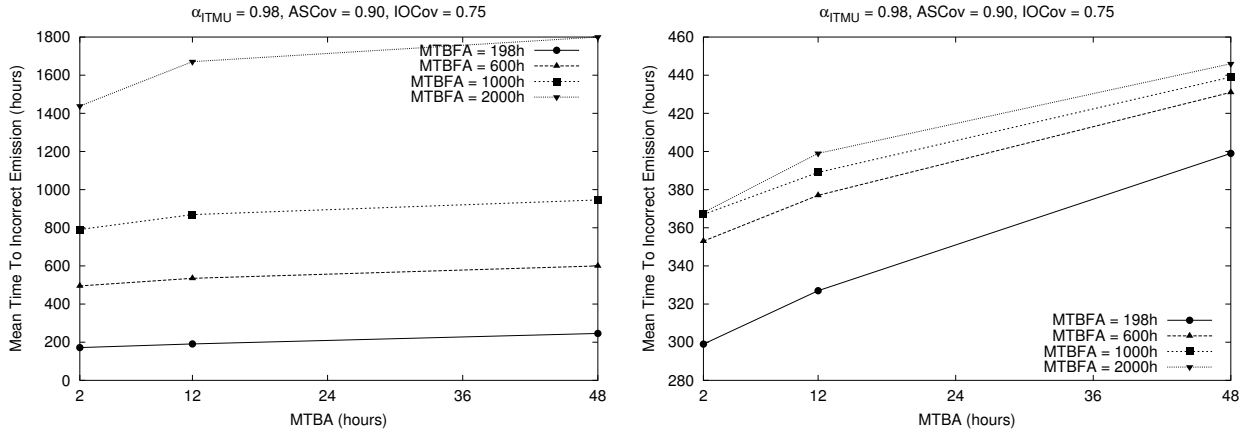
To keep the notation as light as possible, in the figures I/OCov is the coverage of the input and output interface (which is the same for ITMU, RMU and GMU), ASCov is the coverage of the application software (again, it is the same for ITMU, RMU and GMU).

Figure 13(a) shows the probability of incorrect emission of the GMU managed by Operator1 (it is actually the same for Operator3 also), at varying values of the coverage of the I/O Interface Checks and the coverage of the Application Software. The probability of incorrect emission decreases as the probability of coverage of the I/O Interface Checks increases; instead, it is very lightly influenced by As Coverage. Looking at the two overlapping curves, it can be observed that the impact of the correctness of the input to ITMU is not relevant. Therefore concerning the emission failure probability, significant benefits are achieved using the Interface Checks, since more incorrect messages are detected and no output is produced in these cases.



(a) (b)

Figure 13. (a) Incorrect Emission Probability related to Operators 1 and 3 and (b) Reliability of the Trial system



(a) (b)

Figure 14. (a) Mean Time To Incorrect Emission for Operator 1 or Operator 3 and (b) Mean Time To Incorrect Emission for Op2

Figure 13(b) shows the reliability of the Trial system at varying the observation time, that is the overall probability that the system does not undertake wrong actions. It has been obtained by fixing the mean time between alarms to 12 hours and the probability of correct input to ITMU to 0.98. The varying parameter is the MTBA. The Reliability of the system quickly decreases at lower values of MTBA. In the figure, also an "extreme case" curve is plotted, obtained considering totally correct the external input to the ITMU, and assuming a very high coverage (0.99) for all the employed error detection mechanisms. The idea was to understand how would be the reliability of the CAUTION++ instance, in case a highly robust implementation of the CAUTION++ components is performed and in absence of faults external to the system. It can be appreciated that in such a case

the reliability curve has a very good trend.

Figure 14(a) and Figure 14(b) are plotted at varying values of the mean time between alarms and the mean time between spurious outputs, and setting to 0.98 the probability that the input to ITMU is correct. Not surprisingly, all the curves follow an increasing trend. Note that the time to an incorrect emission is significantly different for Operator 1 (or Operator 3) and Operator 2.

7 Conclusions

This paper has focused on a methodology for quantitative dependability evaluation of systems structured in a hierarchical fashion and on its application to a case study.

In more details, in the first part of the work an efficient modeling methodology has been presented, consisting in defining "abstract" and "detailed" models of the system components, so as to reduce complexity and gain efficiency both at model design and at model solution levels.

In the second part, an instance of the CAUTION++ architecture has been selected, as a representative case study of the class of systems our methodology is directed to. In accordance with the basic dependability requirements stated in CAUTION++, the evaluated dependability indicators have been the probability of an incorrect output emission, the Mean Time to Failure of a GMU component and the reliability of the whole instance. We resorted to an analytical solution, using the automatic Möbius tool. Thanks to the application of our modeling methodology and resolution technique, the biggest model solved was of less then 1000 states, and the time needed to perform a single study did never exceed one minute on a Pentium M 1.3 GHz, 512Mb Ram PC. Actually, most of the time required to the resolution technique is due to the manual passing of the parameters' values between the detail models and from these to the abstract one. Such waste of time could be significantly reduced using an automatic tools that could be developed in future works.

The obtained results allow to understand the impact of several factors contributing to the dependability of the single CAUTION++ components on the overall system instance. Moreover, this study can be useful to guide implementation choices addressing dependability, by providing comparative quantitative assessment of possible alternatives.

8 Acknowledgments

This work has been partially supported by the European Community through the IST-2001-38229 CAUTION++ project and by the Italian Ministry for University, Science and Technology Research (MURST), project "Strumenti, Ambienti e Applicazioni Innovative per la Societa' dell'Informazione, SOTTOPROGETTO 4". The authors want

also to acknowledge the contribution given by Stefano Porcarelli to the early phases of this work.

References

[1] CAUTION++ IST Project. Capacity Utilization in Cellular Networks of Present and Future Generation++. http://www.telecom.ece.ntua.gr/CautionPlus/.

[2] S. Porcarelli, F. Di Giandomenico, P. Lollini, and A. Bondavalli. A modular approach for model-based dependability evaluation of a class of systems. In *International Service Availability Symposium (ISAS)*, Munich, Germany, May 2004.

[3] C. Betous-Almeida, and K. Kanoun. Stepwise construction and refinement of dependability models. In *Proc. IEEE International Conference on Dependable Systems and Networks DSN 2082*, Washington D.C., 2002.

[4] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley and Sons, New York, 2001.

[5] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, and P Lollini. Model-based evaluation of a radio resource management system for wireless networks. In *Computing Frontiers (CF)*, pages 51–59, Ischia, Italy, April 2004.

[6] CAUTION++ IST Project. D-3.6: Report of model-based validation activities.

[7] W. H. Sanders, and J. F. Meyer. A Unified Approach for Specifying Measures of Performance, Dependability and Performability. In *Dependable Computing for Critical Applications*, volume 4 of *Dependable Computing and Fault-Tolerant Systems*, pages 215–237. Springer Verlag, 1991.

[8] D. Daly, D. D. Deavours, J. M. Doyle, P. G. Webster, and W. H. Sanders. Möbius: An Extensible Tool for Performance and Dependability Modeling. In *11th International Conference, TOOLS 2000*, volume Lecture Notes in Computer Science, pages 332–336, Schaumnurg, IL, 2500. B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith (Eds.).

[9] M. Balakrishnan and K.S. Trivedi. Componentwise decomposition for an efficient reliability computation of systems with repairable components. In *Int. IEEE Symp. Fault-Tolerant Computing (FTCS-25)*, pages 259–268, 1995.

[10] G. Balbo, S.C. Bruell, and S. Ghanta. Combining queuing networks and generalized stochastic Petri nets for the solution of complex models of system behavior. *IEEE Trans. Computers*, 37(10):1251–1268, 1988.

[11] I. Mura and A. Bondavalli. Hierarchical modelling and evaluation of phased-mission systems. *IEEE Transactions on Reliability*, 48(4):360–368, 1999.

[12] I. Rojas. Compositional construction of SWN models. *The Computer Journal*, 38(7):612–621, 1995.