Reliability Analysis of Component-Based Systems with Multiple Failure Modes

Antonio Filieri[1], Carlo Ghezzi[1], Vincenzo Grassi[2], and Raffaela Mirandola[1]

[1] Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
{filieri,ghezzi,mirandola}@elet.polimi.it
[2] Università di Roma "Tor Vergata", Viale del Politecnico 1, 00133 Roma, Italy
vgrassi@info.uniroma2.it

Abstract. This paper presents a novel approach to the reliability modeling and analysis of a component-based system that allows dealing with multiple failure modes and studying the error propagation among components. The proposed model permits to specify the components attitude to produce, propagate, transform or mask different failure modes. These component-level reliability specifications together with information about systems global structure allow precise estimation of reliability properties by means of analytical closed formulas, probabilistic model-checking or simulation methods. To support the rapid identification of components that could heavily affect systems reliability, we also show how our modeling approach easily support the automated estimation of the system sensitivity to variations in the reliability properties of its components. The results of this analysis allow system designers and developers to identify critical components where it is worth spending additional improvement efforts.

1 Introduction

In component-based (CB) systems it became quickly evident that the whole is more than the sum of its parts. Each component of the system can affect global, perceivable properties of the entire system. A crucial issue in CB development is the assessment of the quality properties of the whole system starting from the properties of its components. Methodologies to quickly predict these global properties, before the actual components integration and system release, can be used to drive the development process, by supporting architectural decisions about components assembly and giving indications about critical components that could deserve customized development efforts. In this paper, we focus on CB software systems that operate in safety-critical environments, where a relevant quality factor is the system *reliability*, defined as a probabilistic measure of the system ability to successfully carry out its own task. To support reliability engineering of such systems, we provide a methodology to analyze their reliability, starting from information about the reliability properties of their components and architectural information about how they are assembled. Using this information, we show how to get an estimate of the overall system reliability and of its sensitivity with respect to variations in the reliability properties of its components.

Avizienis et al. [1] clearly described the need to deal with *multiple different failure modes*. A single Boolean domain (failure/no failure) is not expressive enough to represent important pathological behaviors. Moreover, in the same paper the authors also stress the importance of considering the *error propagation* process among the system components. Nonetheless, few modeling approaches deal with error propagation across a component-based system (*e.g.*, [2–4]), and, to the best of our knowledge, none deals with multiple failure modes. On the contrary, to get a complete figure of the possible failure pathology of the whole system, our methodology takes into account that components can experience a number of different failure modes and those failures can propagate in different ways across the execution flow, possibly spreading up to the application interface. In particular, we also consider the transformation of failure modes across components. Indeed, a component invoked with an incoming failure mode, could possibly output a different failure mode. The modeling approach we propose is expressive enough to represent the failure mode transformation during its propagation through a component.

The proposed approach can be applied at early stages of software design, and it can provide a fine prediction model which can drive decisions about both architectural and behavioral aspects. The underlying model is also suitable for sensitivity analysis that establishes how much the global system reliability (for each failure mode) depends upon each model parameter. Specifically, not all the components have the same importance with respect to global reliability, and improvements in the reliability of certain components produce a larger impact on the improvement of the global systems reliability.

Besides estimating the system sensitivity, we propose a method to find the optimal combination of component reliability properties values, that maximizes, for example, the system reliability (possibly, under constraints related to the cost of reliability improvements for different components). This gives an additional support to design and development decisions: for example, it could lead to prefer slightly less reliable but cheaper components, with respect to more reliable versions. Furthermore, using the optimization results it is possible to obtain the best combination of values to look for in component selection.

The paper is organized as follows. In Section 2 we introduce the component and CB architecture models, suitable to describe multiple failure modes and error propagation. In Section 3 we show how to build a Markov model from these component and architecture models, while in Section 4 we sketch some useful analysis techniques, in order to make the most of the information just modeled. In Section 5, we show through a simple example the practical application of the presented ideas. In Section 6 we briefly review related work and finally, Section 7 concludes the paper.

2 System Model

2.1 Basic concepts

According to [1], a *failure* is defined as a deviation of the service delivered by a system from the correct service. An *error* is the part of the system state that could lead to the occurrence of a failure, and is caused by the activation of a *fault.* The deviation from correct service can be manifested in different ways, corresponding to different *failure modes* of the system.

Characterizing the failure modes that may occur in a system is, in general, a system-dependent activity. Two basic failure modes that can be identified are, for example: *content* and *timing failures* (where, respectively, content of system's output and delivery time deviate from the correct ones).

Other failure modes could be defined, for example, by grading basic modes' severity, or by combining them (*e.g.*, content and timing simoultaneously). Special failure modes, when both timing and content are incorrect, are *halt* failures; these make system activity, if any, no longer perceptible at the system interface.

Errors can arise both because of silent internal fault, or because of an erroneus input received through its interface. However, errors in a component not necessarily manifest themselves as component failures. In turn, component failures not necessarily imply system failures. A component failure occurs only when an error propagates within the component up to its interface, and a system failure occurs only when an error propagates through components up to the system interface. In this propagation path, an error can get masked, *e.g.*, when an erroneous value is overwritten before being delivered to the interface. An error can also get transformed, *e.g.*, content failure received from another component may cause additional computations, leading to the occurrence of a timing failure.

To analyze the reliability of a component-based system, we should take into account the whole set of factors outlined above, that can lead to the manifestation of a failure. At component level, this requires to estimate the likelihood that a given failure mode manifests itself at the component interface because of an internal fault, or by the propagation of the same (or different) failure mode received at the component input interface. At system level, we should consider the possible propagation paths through components, and their respective likelihood.

In the next subsection, we present a reliability model for component-based systems that provides a representation of this information.

2.2 Reliability Model of Component-Based System

It is well understood that, to support component-based development, each component should be equipped with information about its functional properties that permit it to interact with other components. This information includes, for example, a specification of the services required or provided by the component, and this is often referred to as the component *constructive interface* [5]. Several component models have been proposed in the recent past [6], characterized by slightly different definitions of the constructive interface.

To support reasoning about nonfunctional properties like reliability, additional information should be provided, expressed through a suitable *analytic interface*. The model presented in this section defines a reliability-oriented analytic interface. In this model, we assume that each component (and hence the system built from those components) is characterized by N different failure modes. Each mode r, $(1 \leq r \leq N)$ could be one of the basic modes outlined in the previous subsection, or a combination of some of them, or any other special purpose failure mode. For the sake of uniformity, we also introduce an additional failure mode (failure mode 0), which corresponds to the delivery of a correct service.

Component model. A component C_i is modeled as:

- an *input port* ip_i;
- a set of *output ports* O_i =$\{op_{ik}\}$;
- an *operational model* defined by the probabilities: $p_i(k)$ $(\forall op_{ik} \in O_i)$, where each $p_i(k)$ is defined as:
  $p_i(k) = Pr\{C_i$ produces an output on port $op_{ik} \in O_i | C_i$ received an input on its input port$\}$
  It holds: $\sum_{op_{ik} \in O_i} p_i(k) = 1$;
- a *failure model* defined by the probabilities:
  $f_i(r, s)(0 \leq r \leq N, 0 \leq s \leq N)$, where each $f_i(r, s)$ is defined as:
  $f_i(r, s) = Pr\{C_i$ produces an output with failure mode $s | C_i$ received an input with failure mode $r\}$
  It holds: $\sum_{s=0}^{N} \ f_i(r, s) = 1$

In this model, it is intended that transfer of both data and control takes place through the input and output ports: C_i receives data and control (activation stimuli) through its input port, and produces data and activation stimuli (control transfers) towards other components through its output ports. The operational model gives a stochastic characterization of the usage profile of other components when C_i is active. Each $p_i(k)$ probability can be interpreted as the fraction of all the data and control transfers that take place through the output port op_{ik} of C_i, over all the data and control transfers generated by C_i.

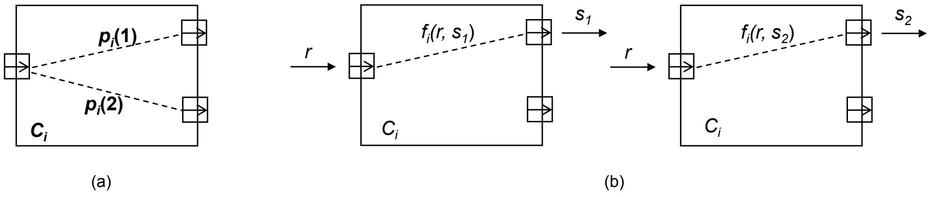

Fig. 1. Component model: (a) probabilistic transfer of control from input port to output port, and (b) probabilistic propagation of failure mode r from input port to output port.

Analogously, the failure model gives a stochastic characterization of the failure pathology of C_i. Figure 1 presents a graphical representation of component model's parameters.

The $f_i(r, s)$ probabilities can be used as a basis to define interesting reliability properties of a software component. Some examples of these properties are proposed in the following:

- **Internal failure probability** with respect to failure mode s, $s > 0$, is the probability $f_i(0, s)$.
- **Robustness** with respect to error mode r ($r > 0$, i.e. not correct) is the probability $f_i(r, 0)$.
- **Susceptibility** with respect to error mode r ($r > 0$) is the probability $1 - f_i(r, 0)$.
- **Proclivity** with respect to failure mode s is the probability $\sum_r \beta_r \cdot f_i(r, s)$, where β_r is the probability of receiving an input mode r.

These formal, human-understandable properties allow the easy formalization of requirements on single components together with easy-to-understand feedbacks to developers.

Finally, we point out that, with respect to component models whose constructive interface specifies multiple input ports for each component, this analytic interface definition can be interpreted in two ways:

- It abstracts from the actual presence of multiple input ports in a real component, by collapsing them into a single port. From this viewpoint, both the operational and failure model of this abstract component represents a sort of average profile of the real component behavior.
- It actually corresponds to a projection of a real component with respect to one of its input ports. From this viewpoint, a real component is modeled by a set of these abstract components, where each element of the set models the real component behavior conditioned on having received control and data through a specific input port.

For the sake of simplicity, in the following we will always use the term "component", without specifying according to which of these two viewpoints it should be interpreted.

Architecture model. An architecture A is modeled as:

- a set of components: $C = \{C_0, C_1, ...C_M\}$ with their analytic interfaces;
- a mapping : $map_A : \bigcup_{i=0}^{M} O_i \rightarrow \bigcup_{i=0}^{M} \{ip_i\}$.

Given an output port op_{ik} of a component C_i, $map_A(op_{ik})$ defines which input port the output port op_{ik} is attached to.

In this architecture definition, $C_1, C_2, \ldots C_{M-1}$ are intended to correspond to components used to build the application modeled by A. C_0 and C_M play instead a special role. They are fictitious components used to model the start of the application modeled by A and the final result produced by the application. C_0 has as many output ports as the possible entry points of the application modeled by A. Moreover, the C_0 input port is not connected to any of the output ports of the A components. C_M has only one input port, and no output port. Given an output port $op_{ok} \in O_0$, $map_A(op_{0k}) = ip_i$ means that C_i is a possible component from which the application starts its execution. Analogously, given an output port $op_{ik} \in O_i (1 \leq i \leq M-1)$, $map_A(op_{0k}) = ip_M$ means that C_i is a possible component that produces the final result of the application.

The operational model associated with C_0, (given by the probabilities $p_0(k)'s$) can thus be used to model the stochastic uncertainty about the application entry point and the user failure profile. The application termination is instead modeled by the occurrence of a transfer of control to C_M. Given the special nature of C_0 and C_M, their failure model is defined as:

$f_0(r,r) = f_M(r,r) = 1 \ (0 \leq r \leq N)$,

$f_0(r,s) = f_M(r,s) = 0 \ (0 \leq r \leq N, 0 \leq s \leq N, r \neq s)$,

which means that C_0 and C_M do not modify the failure modes they receive.

Let us define the following architecture level probabilities:

$F_A(r,s)(0 \leq r \leq N, 0 \leq s \leq N)$, where each $A_A(r,s)$ is defined as:

$F_A(r,s) = Pr\{A$ terminates with failure mode $s|A$ has been activated with failure mode $r\}$

Similar to the component-level properties defined above, we can use the $F_A(r,s)$ probabilities as a basis to define application-level reliability properties, such as:

- **Reliability**, is the probability $F_A(0,0)$.
- **Robustness** with respect to error mode r ($r > 0$, i.e. not correct) is the probability $F_A(r,0)$.
- **Susceptibility** with respect to error mode r ($r > 0$) is the probability $1 - F_A(r,0)$.
- **Proclivity** with respect to failure mode s is the probability $\sum_r \beta_r \cdot F_A(r,s)$, where β_r is the probability of receiving an input mode r

The component and architecture models presented above allow the definition of a reliability-oriented abstract view of a CB system that provides the starting point for our analysis methodology. This view corresponds to what is referred to in [5] as a *analytic assembly* of components. To carry out reliability analysis, a *constructive assembly* of actual software components should be mapped to this analytic assembly though a suitable *analytic interpretation.* It is our opinion that defining this analytic interpretation is quite easy for most of the existing component models. However, explicitly defining it for some component model is beyond the scope of this paper.

3 Markov Model of System Behavior

In this section, we show how, given a set of components $C = C_0, C_1, ...C_M$ with their respective analytic interfaces and an architecture model A as defined in the previous section, we can build a discrete time Markov process (DTMC) modeling the overall system behavior. This model can then be used to analyze reliability properties of the system. For the sake of clarity, we split in two steps the DTMC construction: as first step, we build a DTMC providing an abstract model of the system execution process. Then we expand this DTMC into another DTMC that also includes information about the failure occurrence and propagation.

Execution process model. We build a DTMC G_A with state space:

$$N_{G_A} = \{c_0, c_1, \dots c_M\}$$

where each state c_i corresponds to a components C_i used in the A definition, and represents the system state where component C_i has received control and data from some other component. A state transition from c_i models a transfer of data and control to some other component. The transition probabilities $q_{G_A}(i,j)$ from state c_i to state c_j $(0 \leq i < M)$, $(0 < j \leq M)$ are defined as follows.

Let us define the subset $O_i(A,j) \subseteq O_i$: $O_i(A,j) = \{op_{ik} | map_A(op_{ik}) \in I_j\}$. Hence, $O_i(A,j)$ is the subset of the output ports of C_i connected to the input port of C_j in the architecture A. Given the definition of $O_i(A,j)$, we calculate the probabilities $q_A(i,j)$ as follows:

$q_A(i,j) = \sum_{op_{ik} \in O_i(j)} p_i(k)$

Each $q_A(i,j)$ represents the total fraction of data and control transfers through any port from C_i to C_j, over the total number of data and control transfers from C_i. We point out that $q_A(i,j) > 0$ if and only if $O_i(A,j) \neq \emptyset$.

Given these definitions, c_0 is the initial state of G_A since, by construction, there is no transition entering this state, and c_M is the final absorbing state of G_A.

Finally, we point out that, for the same set of components with their associated failure and operational models, different ways of connecting them (*i.e.*, different architectures) lead to the definition of different G_A's (see figure 2 for two different architectures $A1$ and $A2$).

Failure and execution process model. G_A only models the execution process as a sequence of data and control transfers among the system components, without considering the possible occurrence and propagation of errors (in other words, it models the behavior of a failure-free system). To include also this aspect, we build from G_A a new DTMC H_A. The state space of H_A consists of a set N_{H_A}, defined as follows.

First, for each $c_i \in N_{G_A}$ we build two sets: $IM_i = \{im_{i0}, im_{i1}, \cdots, im_{iN}\}$ and $OM_i = \{om_{i0}, om_{i1}, \cdots, om_{iN}\}$. An element $im_{ir} \in IM_i$ represents the system state where component C_i has received data and control from another component with error mode r $(0 \leq r \leq N)$. An element $om_{ir} \in OM_i$ represents the system state where component C_i is going to transfer data and control to other components with failure mode s $(0 \leq s \leq N)$. Then, H_A'state space is:

$N_{H_A} = \bigcup_{i=0}^{M} (IM_i \bigcup OM_i)$

Hence, each state $c_i \in N_{G_A}$ can be seen as a "macro-state" that is expanded into $2(N+1)$ states in N_{H_A} to take into account both the execution and the error propagation processes (see fig. 3 for the case $N = 2$).

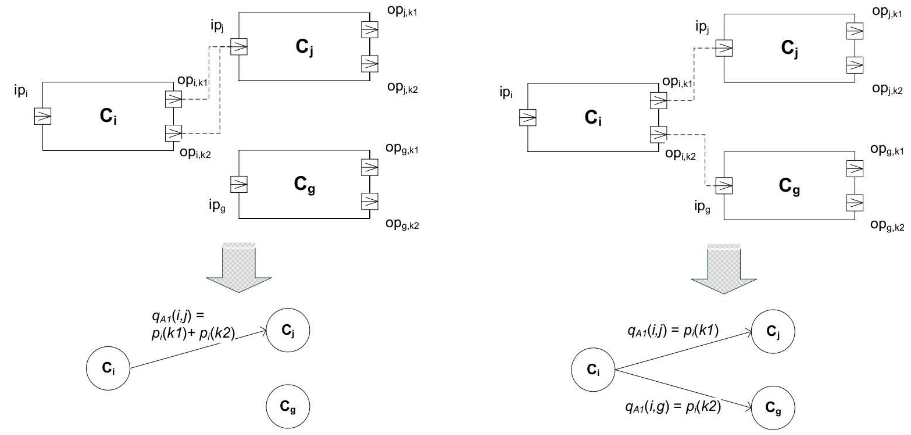

Fig. 2. Mapping from the execution process model to DTMC.

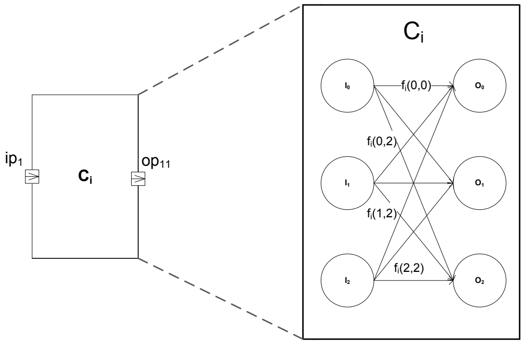

Fig. 3. Component's macro-state expansion.

The transition probabilities $p_A(x, y)(x \in N_{H_A}, y \in N_{H_A})$ are defined as follows:

1. $p_A(x, y) = f_i(r, s)$ if $x = im_{ir}, y = om_{is}(im_{ir} \in IM_i, om_{is} \in OM_i, 0 \leq i \leq M, 0 \leq r, s \leq N)$;
2. $p_A(x, y) = q_A(i, j)$ if $x = om_{is}, y = im_{js}(om_{is} \in OM_i, im_{js} \in IM_j, 0 \leq i, j \leq M, 0 \leq s \leq N)$;
3. $p_A(x, x) = 1$ if if $x = im_{Ms}(im_{Ms} \in IM_M)$;
4. $p_A(x, y) = 0$ otherwise.

Case 1 above corresponds to a transition occurring within the same macro-state, and models how a given component C_i propagates to its output a failure mode received at its input, according to the failure model specified in the C_i analytic interface. We point out that we are able to represent both the case where a failure mode r is propagated "as is" (when $f_i(r, r) = 1$), and the case where it is transformed into another mode (when $f_i(r, r) < 1$). Case 2 corresponds to a transition occurring between two different macro-states, and models the control and error transfer process from a component C_i to a component C_j: if a failure mode s manifests itself at the C_i output interface, it is transferred to the input interface of C_j according to the $q_A(i, j)$ probability (which has been calculated from the operational model specified in the C_i analytic interface, and the definition of the architecture A). Case 3 corresponds to the definition of the set of states IM_M as the set of final absorbing states of the DTMC H_A: thus, entering a state $im_{Ms} \in IM_M$ models the application termination with failure mode s.

We also remark that the set $IM_0 = \{im_{00}, im_{01}, \cdots, im_{0N}\}$ corresponds to the set of the initial states of H_A as, by construction, no transition is possible towards these states: starting from a state $im_{0r} \in IM_0$ models the application activation with failure mode r.

3.1 Modeling issues

Both the operational and failure models are based on the assumption that system's execution respects the Markov property (which in turn implies that components' failures are independent). In practical terms this means that each time control and data are transferred to C_i from other components, the C_i operational and failure behaviors are independent of its past history. The Markovian assumption is a limitation to the application scope of our approach. Nevertheless, many real-life applications have been proved to respect the Markov assumption, from business transaction and telephone switching systems, to the basis memory management strategies [7]. The Markovian issue is deeper treated in [8], where it is also recalled that an higher order Markov chain, the one in which the next execution step depends non only on the last but on the previous n steps, can be mapped to a first order Markov chain. Thus, our approach can be adapted to any finite order Markov chain, increasing the applicability horizon of the methodology to a large number of real-life systems.

Another interesting issue related to the defined reliability properties concerns how to estimate $f_i(r, s)$. The problem can be splitted in two phases. The first phase concerns observability: to measure e $f_i(r, s)$, we must be able to identify error modes r and s. Identification can be based both on code instrumentation or on communication channels monitoring or in any other effective ad-hoc way [8]. The second phase concerns how to obtain parameters estimation. There is not an always valid methodology to face the problem. Most of the approaches are based on setting up tests in order to obtain a statistically significant amount of measurements upon which parameters estimation can be based [9, 10]. In some case it is possible to shorten the testing time by adopting accelerated testing procedures, which are able to stress the most interesting parts and aspects of a system execution in order to obtain a large amount of data in a short testing time [11]. After getting measurements, the next step is the estimation of reliability parameters. This issue is more related to Statistics, even if the ability of embedding some Software Engineering knowledge in the process of sampling and estimation produces better results [12]. For the sake of completeness, there are also cases where it could be infeasible to quantify software reliability properties because of the nature of system runs which, for example, may be too long to allow the execution of a large enough test set [13]. As a final remark, components reuse may allow the exploitation of historical data upon which reliability properties estimation can be based.

4 Analysis

The transition matrix $P_A = [p_A(x, y)]$ associated with the DTMC H_A constructed in Section 2 represents the probability of moving in one step from state $x \in N_{H_A}$ to state $y \in N_{H_A}$. It has the following canonical form:

$$P_A = \begin{bmatrix} Q_A & R_A \\ 0 & I \end{bmatrix}$$

The submatrix Q_A is a square matrix representing one-step transitions among the H_A transient states only. The submatrix R_A represents instead the one-step transitions from these transient states to the set of absorbing states $IM_M = \{im_{M0}, im_{M1}, \cdots, im_{MN}\}$. I is an identity matrix of size $(N + 1) \times (N + 1)$.
Let us now define the matrix $V_A = [v_A(x, y)]$ $(x \in (N_{G_A} - IM_M, y \in IM_M)$, whose $v_A(x, y)$ entry is defined as:

$v_A(x, y) = Pr\{H_A \text{ is absorbed in state } y|H_A \text{ starts in state } x\}$.

Given the meaning of the states in H_A, we can readily see that the application-level reliability properties defined in Section 2 can be expressed in terms of the V_A matrix, since it holds:

$F_A(r, s) = v_A(x, y)$, with $x = im_{0r}$ and $y = im_{Ms}$.

In the following, we show how to calculate the V_A matrix from P_A matrix, which allows to calculate the application-level reliability properties. Moreover, we also show how to carry out further sensitivity and optimization analysis.

¿From the DTMC theory and matrix calculus [14], we know that matrix V_A can be calculated as:

$$V_A = W_A R_A \qquad \text{where } W_A = (I - Q_A)^{-1}.$$

We remark that component-level reliability properties (*i.e.*, $f_i(r, s)$ probabilities) correspond to specific entries of the Q_A matrix, and hence directly affect W_A values. Also, by construction, the matrix R_A is independent from these component-level properties, thus in order to establish a relation between application-level and component-level properties it suffices to study the matrix W_A.
W_A is the inverse of the matrix $I - Q_A$, hence from linear algebra [15]:

$$w_A(x, y) = (-1)^{x+y} \frac{|M_{y,x}|}{|I - Q_A|}$$

where $|M_{y,x}|$ is the minor of $(I - Q_A)(y, x)$ and $|I - Q_A|$ is the determinant of $I - Q_A$.

Let us give a look at matrices $I - Q_A$ and $M_{y,x}$. First of all, $M_{y,x}$ is obtained from $I - Q_A$ by removing the y-th row and the x-th column. Thus its structure is quite similar to the one of $I - Q_A$, just omitting one row and one column. An entry $(I - Q_A)(i, j)$ (as well as $M_{y,x}(i, j)$ when $i \neq y$ and $j \neq x$) represents the probability of moving from state i to state j of the DTMC. We recall that each state models either entering or leaving a component with a given failure mode. Let i correspond to the state when the system enters component C_k with incoming failure mode r). Entries on row i of $I - Q_A$ (and $M_{y,x}$, when applicable) will be all zeros, but a small set of them. Namely, each entry $(I - Q_A)(i, j)$ will possibly be not null if and only if either $i = j$ (because of the identity matrix I) or j corresponds to the state where component C_k, invoked with incoming failure mode r, is producing an output failure mode s: $(I - Q_A)(i, j) = f_k(r, s)$.

We want to exploit this structure of the matrices $I - Q_A$ and $M_{y,x}$ to make explicit the relation between elements $(I - Q_A)(i, j)$ and component-level reliability properties $f_i(r, s)$'s. Due to the previously mentioned fact that matrix R_A is independent of the $f_i(r, s)$'s, we will be able to extend this relation to the system-level properties $F_A(r, s)$ easily.

To this end we compute the numerator and denominator determinants through Laplace expansion with respect to the row corresponding to the set of properties $f_k(r, s)$. For matrix $I - Q_A$ we obtain an expressions like the following one:

$$det(I - Q_A) = \sum_j (I - Q_A)(i, j) \cdot \alpha_{ij}$$

where α_{ij} represents the cofactor of the entry $(I - Q_A)(i, j)$.

The same procedure can be applied to matrix $M_{y,x}$. Due to the $I - Q_A$'s and $M_{y,x}$'s structure discussed above, we get: $det(I - Q_A) = func(f_k(r, s_1), f_k(r, s_2)...)$ and $det(M_{y,x}) = function(f_k(r, s_1), f_k(r, s_2)...)$. Thus the elements of W_A can be redefined as well as function of the parameters $f_k(r, s)$'s.

Thanks to the fact that $F_A(r, s)$ is a function of W_A, we are able to formalize the system level reliability properties as function of any set of component-level properties. In the following, we focus on the system *reliability*, defined as $F_A(0, 0)$

Sensitivity Analysis and Optimization. To determine which property most heavily affect the global reliability, we compute its *sensitivity*. It corresponds to the partial derivative of the function $F_A(0, 0)$ with respect to each local property of each component C_i

$$\frac{\partial F_A(0, 0)}{\partial f_i(r, s)}$$

Besides estimating a sensitivity index, expressing explicitly $F_A(0, 0)$ as a function of the $f_k(r, s)$ parameters, allows finding the optimal combination of components reliability properties' values. Indeed, it could happen that under given design and development constraints, it could be better to set some $f_i(r, s)$ to a value less than the trivial one (*i.e.*, $f_i(0, 0) = 1$). In this respect, we remark that, due to the geometry of the transition matrix representing system's behavior, it is possible to reiterate the Laplace expansion for the computation of more cofactors. This leads to a representation of the reliability function where the dependency on interesting component-level properties is made explicit. Such a function has the shape of a fraction of polynomials and can be considered as the objective function to be maximized in a non-linear constrained optimization.

The set of constraints to be applied has to include, but not to be limited to, making all the complementary probabilities sum up to 1. Any other special purpose constraint can be added, *e.g.* expressing the fact that over certain thresholds some properties could be too expensive to be obtained. Also, the objective function can be extended to cope with other design goals. For example development costs can be used as coefficient to be applied to certain properties in order to make their growth more or less likely. Alternatively, the optimization problem can be restated as a multiobjective optimization, maximizing reliability and minimizing a related cost function. Trade-off between reliability improvement and development costs is an open problem. Special situations may require ad-hoc estimations for costs and their fitness to reality is mainly based on architects' experience. To see some examples of generalized cost functions related to software reliability issues refer to [16].

A typical optimization problem to find the best combination of reliability property values looks like this:

$$\begin{cases} \textit{max} & F_A(0,0) \\ \textit{subject to} & \text{probability constraints} \\ \textit{subject to} & \text{design constraints} \end{cases}$$

Performing sensitivity analysis and determining optimal property set can be very valuable. The former can be mainly applied to produce developer feedbacks: if the largest value of $\frac{\partial F_A(0,0)}{\partial f_i(r,s)}$ is referred to the property $f_1(1,0)$ of component C_1, the feedback to the developer is the advise:

"Increase C_1's Robustness with respect to incoming error mode 1".

Thanks to the user-friendly formalization of reliability properties proposed in Section 2, any advice can be explained to the developer without the need to significantly improve her/his skills in mathematical probability, thus improving the learning curve of the development team.

By using the optimization results, the best combination of reliability property values can be used as a target to look for in component selection by both a human designer or a self-adaptive system. For this purpose, the introduction of a proper distance metrics based on the set of interesting local properties can allow an automatic ranking of all the possible alternatives, making fast and easier picking the best available choice.

5 Example Application

In this section we present a short proof of concept to show an application of our novel methodology. We use for this purpose a small system consisting of three components. The first component (C_1) plays the role of dispatcher for all the incoming requests, and is the only entry point of the system. The dispatcher analyzes the incoming requests and sends them to the server (C_2). Depending on the specific operation requested, the server can accomplish the job by itself or it can issue some requests for more operations. In the first case, server's outputs have to pass through a service guard (C_3) before reaching the user. In the second case, server's requests get forwarded again to the dispatcher in order to be scheduled for execution. The server guard has the task to analyze server's output to ensure it does not carry any confidential detail: if the guard notices something illegal, it sends back the job to the dispatcher to be processed again, otherwise it delivers the result to the user.

The architecture of the system is sketched in Figure 4. We recall that in our model a transition stands for the transfer of control and data from a component to another. The probabilities expressing the operational model of each component are shown in bold face font in correspondence of the directional arrow of each connection.

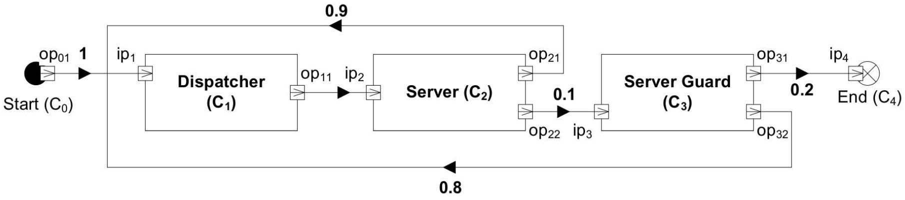

Fig. 4. Example application's architecture.

In this example we consider only two failure modes (denoted by 1 and 2, respectively), beside the correct execution (failure mode 0), in order to keep it simple. A partial view (without all the parameters) of the derived Markov process is represented in Figure 5. Vertical dashed lines were added as virtual delimiters between local and global execution, that is, the area entitled C_x represents the failure process ongoing while C_x holds the control. Arrows across delimiters represents the transfer of control between components. To keep it readable, only some parameters discussed in Section 3 are placed on the graph coherently with the mapping procedure explained in Section 2.

The special node before C_0 added to the DTMC H_A models the expected profile of starting failure modes for the system activation. The probability given to each starting failure mode is also shown in figure.

In order to analyze the impact of components' reliability properties on the global system reliability, let us firstly specify the component-level properties which are retained relevant for the system under exam:

- **Reliability** (R): the probability that the component C_i does not produce any erroneous output, given it was invoked without incoming errors ($f_i(0,0)$).
- **Robustness** (with respect to error mode r, B_r): the probability that the component C_i, invoked with incoming failure mode r, will produce a correct output ($f_i(r,0)$).

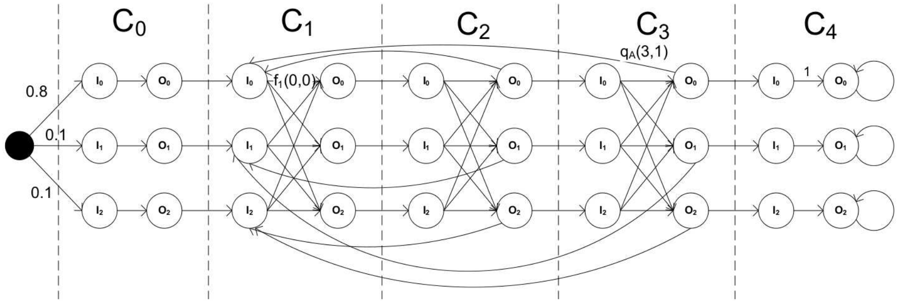

Fig. 5. Markov process derived from the example application.

- **Internal Failure** (with mode s, F_s): the probability that the component C_i, invoked without any erroneous incoming pattern, will produce an erroneous output pattern with mode s ($f_i(0,s)$).
- **Switching** (from mode r to mode s with $r, s > 0$, S_{rs}): the probability that the component C_i will produce an outgoing failure mode s, given an incoming error mode s ($f_i(r,s)$).

Reliability estimation. In table 1 we show an estimation of the overall reliability (last row of the table), for three different set of values assigned to the reliability properties of each component of the system (corresponding to columns *Initial, Cost 1000, Cost 1200*). Column *Initial* refers to an initial attribution of values to the component parameters. The other two columns report values calculated by optimization analysis, as explained below. The reliability values were computed by means of the formula explained in section 4 and validated by simulation.

Sensitivity analysis. As explained in Section 4, sensitivity analysis is a good mean to identify where to enforce improvement effort in order to obtain the larger growth of the overall reliability. Sensitivity analysis is an established method to evaluate the impact of local properties on the global system [17, 2]. Nevertheless, thanks to the fact that our methodology can deal with multiple error-modes and their propagation and transformation, a designer can obtain finer information not only about where to operate refinement, but also what and how he/she has to improve. Indeed, our methodology allows to estimate the sensitivity to a specific failure transformation or propagation as well as the the internal failure probability, thus obtaining accurate results on every aspect of a CB system.

However, we point out that results of sensitivity analysis should be carefully considered, when we differentiate with respect to multiple correlated parameters $f_i(r,s)$ (we recall that $\sum_s f_i(r,s) = 1$). In this case suggested variations of a parameter imply a change in the correlated ones, and this could affect the final result in different ways. Thus sensitivity results have to be considered as useful advices to be considered by a human expert.

The column labeled by *Sensitivity* in table 1 shows results of sensitivity analysis in our example, calculated around the initial set of parameter values (column

Initial). We note that the only component that directly delivers outputs to the user is the service guard C_3. The overall system reliability has the highest sensitivity with respect to $f_3(r, 0)$ for all the possible r. This is not at all surprisingly in this small example, but it could be very useful in large systems. Also notice that the 80% of the incoming requests reaches the dispatcher without any error pattern, thus the set of properties related to this situation was expected to have an high sensitivity index as it is.

Optimal configuration. A novel contribution of this paper is the possibility to identify in a automatic way the best values for component-level reliability properties in order to improve the overall reliability. These values can be useful in setting design plans and goals, or even in self-adaptive component-based systems where components must be selected to maximize the system QoS, in this case reliability. In this latter case, it could be effective to have referential goal values during the selection process.

To make not trivial the optimization of the model parameters, some constraints must be introduced. They could come, for example, from component availability on the market, as it could be unreasonable, in general, to ask for a component that will never fail. Other constraints could come from cost considerations. Component quality is expensive and component price could be a function more or less steep of some of its parameters. Other special purpose constraints could be in place for specific contexts.

In this example we introduce the following two constraints:

- **Cost**. Each property has its own improvement cost function. Let x represents any component's property, in this example we set a "price" of $800x^3$ for all the component's reliability properties, $500x^5$ for all the robustness properties and $200x^3$ for all the switching properties. This means that, for example, robustness is cheaper than reliability in the worst case, but the effective cost of robustness grows quite faster approaching higher values of x.
- **Quality of service**. We require that the system will not terminate with failure mode 1 for more than the 5% of the requests for the considered profile of starting failure modes (*i.e.*, we require that for failure mode 1 the *proclivity* property defined in Section 2 takes a value less than 5%).

Even for the small system considered in this example, optimization results are not trivial to guess. In table 1 we show optimization results for global cost up to 1000 and 1200 cost units.

6 Related Work

To the best of our knowledge, no other paper have tackled the issue of stochastic analysis of reliability for CB systems, taking into account multiple failure modes and their propagation inside the system. Nevertheless there are a number of works strongly related to this. In the following we present a selection of related works to show on what our solution stands, and which is the starting point of this

Component	**Property**	**Initial**	**Sensitivity**	**Cost 1000**	**Cost 1200**
C_1	R	0.94	3.3089	0.2595	0.2647
C_1	F_1	0.03	3.1958	0.3014	0.3002
C_1	F_2	0.03	3.0325	0.4390	0.4351
C_1	B_1	0.05	2.5692	0.4306	0.4261
C_1	S_{11}	0.92	2.4813	0.2851	0.2873
C_1	S_{12}	0.03	2.3546	0.2842	0.2866
C_1	B_2	0.70	1.8937	0.2955	0.2936
C_1	S_{21}	0.00	1.8290	0.3527	0.3535
C_1	S_{22}	0.30	1.7356	0.3518	0.3529
C_2	R	0.64	4.4781	0.2020	0.2041
C_2	F_1	0.13	4.4404	0.4350	0.4320
C_2	F_2	0.23	4.1232	0.3630	0.3639
C_2	B_1	0.03	2.5620	0.2609	0.2350
C_2	S_{11}	0.84	2.3960	0.3696	0.3825
C_2	S_{12}	0.13	2.2248	0.3694	0.3824
C_2	B_2	0.03	0.7744	0.2961	0.2730
C_2	S_{21}	0.13	0.7242	0.3147	0.3288
C_2	S_{22}	0.84	0.6725	0.3891	0.3983
C_3	R	0.96	0.4484	0.7727	0.8240
C_3	F_1	0.02	0.0583	0.0883	0.0834
C_3	F_2	0.02	0.0588	0.1391	0.0927
C_3	B_1	0.65	0.4100	0.8649	0.9035
C_3	S_{11}	0.169	0.0533	0.0364	0.0391
C_3	S_{12}	0.181	0.0538	0.0988	0.0575
C_3	B_2	0.05	0.2965	0.8660	0.9043
C_3	S_{21}	0.336	0.0386	0.0358	0.0386
C_3	S_{22}	0.614	0.0389	0.0982	0.0570
Reliability		0.6163		0.8419	0.8849

Table 1. Analysis Results.

research. We classify the papers according to their topic as: architecture-based reliability analysis and error propagation analysis.

Architecture-based reliability analysis. Surveys on architecture-based reliability analysis can be found in [18, 8]. However, albeit error propagation is an important element in the chain that leads to a system failure, all existing approaches ignore it. In these approaches, the only considered parameters are the internal failure probability of each component and the interaction probabilities, with the underlying assumption that any error that arises in a component immediately manifest itself as an application failure, or equivalently that it always propagates (i.e. with probability one) up to the application outputs. Hereafter, we shortly describe some of the works that mostly influenced the proposed/adopted solution. One of the first approaches to reliability that takes distance from debugging has been proposed in 1980 [7]. The approach got named from *user-oriented reliability*, which is defined as the probability that the program will give the correct output with a typical set of input data from the execution environment. The user-oriented approach is now the more widely adopted and it justifies the adoption of probabilistic methods as long as the system reliability depends on the probability that a fault gets activated during a run. The reliability of a system is computed as a function of both the reliability of its components and their frequency distribution of utilization, where the system is described by as a set of interacting modules which evolves as a stochastic Markov Process and the usage frequencies can be obtained from the structural description. In [19] the authors explore the possibility of transforming architecture expressed in three popular architectural styles into discrete Markov chains to be then analyzed by means of the approach proposed in [7]. Parameterized specification contracts, usage profile and reliability of required components as constituent factors for reliability analysis have been presented in [20]. Specifically, they consider components reliability as a combination of internal constant factors, such as reliability of the method body code, and variable factors, such as the reliability of external method calls. An approach for automatic reliability estimation in the context of self-assembling service-oriented computing taking into account relevant issues like compositionality and dependency on external resources has been proposed in [21].

Error propagation analysis. The concept of error propagation probability as the probability that an error, arising somewhere in the system, propagates through components, possibly up to the user interfaces has been introduced in [2]. The methodology in [2] assumes a single failure mode and provides tools to analyze how sensible the system is with respect to both failure and error propagation probability of each of its components. In [3], the authors proposed a notion of error permeability for modules as a basic characterization of modules' attitude to propagate errors. Also in this case, a single, non-halting failure mode is considered. Moreover, it is proposed a method for the identification of which modules are more likely exposed to propagated errors and which modules more likely produce severe consequences on the global system, considering the propagation path of their own failure. In [22–24] approaches based on fault injection to estimate the error propagation characteristics of a software system during testing are presented. In the context of safety some works exist dealing with multiple failure modes, see for example [25]. However they don't present any kind of stochastic analysis but only an examination of their possible propagation patterns.

With regard to the estimate of the propagation path probabilities, the basic information exploited by all the architecture-based methodologies is the probability that component i directly interacts with component j. At early design stages, where only models of the system are available, this information can be derived from software artifacts (e.g. UML interaction diagrams), possibly annotated with probabilistic data about the possible execution and interaction patterns [26]. A review and discussion of methodologies for the interaction probability estimate can be found in [8]. A more recent method has been discussed in [27], where a Hidden Markov model is used to cope with the imperfect knowledge about the component behavior. Once the interaction probabilities are known, the probability of the different error propagation paths can be estimated under the assumption that errors propagate through component interactions.

An important advantage of architectural analysis of reliability is the possibility of studying the sensitivity of the system reliability to the reliability of each component, as said in the Introduction. Although this advantage is widely recognized (e.g., [28]), few model-based approaches for computing the sensitivity of the system reliability with respect to each component reliability have been developed [7, 17]. A basic work for the sensitivity analysis of the reliability with respect to some system parameter was presented in [29], but it does not address specifically architectural issues. Moreover, all these models do not take into account the error propagation attribute and different failure modes.

7 Conclusions

In this paper we presented a novel approach to the reliability modeling and analysis of a component-based system that allows dealing with multiple failure modes and studying the error propagation among components. To support the rapid identification of components that could heavily affect system's reliability, we have also shown how our modeling approach can easily support the automated estimation of the system sensitivity to variations in the reliability properties of its components. Furthermore, we proposed a method to find the optimal combination of component reliability properties' values, that maximizes, for example, the system reliability. The results of these analyses support the design and development decisions in the identification of the critical components for the overall system reliability.

The methodology proposed in this paper can be extended along several directions. analytic interfaces possibly using model drive techniques. We also plan to extend our model to be able to deal with both black-box and white-box components.

A second direction concerns overcoming some of the modeling limitations of our methodology. A first limitation comes from the underlying assumption of a sequential execution model, where control and data can be transferred from one component to another one, but not to many components. Currently, this does not allow modeling applications with parallel execution patterns. We are working towards an extension of our approach to deal also with this kind of patterns. Another possible limitation comes from the assumption of the Markov property for the component behavior. We have discussed this issue in section 3.1. Anyway, we are planning to investigate in real experiments the degree of approximation introduced by this assumption. We are also planning to investigate to what extent the approximation can be improved by introducing in our model some kind of dependence on past history, as outlined in section 3.1.

Finally, we are aware that the effectiveness of the proposed approach should be assessed by an empirical validation, and we are planning for this purpose a comprehensive set of real experiments.

Acknowledgments

Work partially supported by the Italian PRIN project D-ASAP and by the EU projects Q-ImPrESS (FP7 215013) and SMScom (IDEAS 227077).

References

1. A. Avizienis et al.: Basic concepts and taxonomy of dependable and secure computing. IEEE JDSC **1**(1) (2004) 11–33

2. Cortellessa, V., Grassi, V.: A modeling approach to analyze the impact of error propagation on reliability of component-based systems. LNCS **4608** (2007) 140
3. M. Hiller et al.: Epic: Profiling the propagation and effect of data errors in software. IEEE Trans. Comput. **53**(5) (2004) 512–530
4. Ammar, H., Nassar, D., Abdelmoez, W., Shereshevsky, M., Mili, A.: A framework for experimental error propagation analysis of software architecture specifications. In: Proc. of International Symposium on Software Reliability Engineering, IEEE (2002)
5. Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Enabling predictable assembly. Journal of Systems and Software **65**(3) (2003) 185–198
6. Lau, K.K., Wang, Z.: Software component models. IEEE Trans. Software Eng. **33**(10) (2007) 709–724
7. Cheung, R.C.: A user-oriented software reliability model. IEEE Trans. Softw. Eng. **6**(2) (1980) 118–125
8. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Perform. Eval. **45**(2-3) (2001) 179–204
9. Nelson, E.: Estimating software reliability from test data. Microelectronics Reliability **17**(1) (1978) 67–73
10. Horgan, J., Mathur, A.: Software testing and reliability. The Handbook of Software Reliability Engineering (1996) 531–565
11. Meeker, W., Escobar, L.: A review of recent research and current issues in accelerated testing. International Statistical Review/Revue Internationale de Statistique **61**(1) (1993) 147–168
12. Podgurski, A., Masri, W., McCleese, Y., Wolff, F.G., Yang, C.: Estimation of software reliability by stratified sampling. ACM Trans. Softw. Eng. Methodol. **8**(3) (1999) 263–283
13. Butler, R.W., Finelli, G.B.: The infeasibility of experimental quantification of life-critical software reliability. In: SIGSOFT '91: Proceedings of the conference on Software for citical systems, New York, NY, USA, ACM (1991) 66–76
14. Cinlar, E.: Introduction to stochastic processes. Englewood Cliffs (1975)
15. Katsumi, N.: Fundamentals of linear algebra. McGraw-Hill (1966)
16. Pham, H.: Software reliability and cost models: Perspectives, comparison, and practice. European Journal of operational research **149**(3) (2003) 475–489
17. Gokhale, S.S., Trivedi, K.S.: Reliability prediction and sensitivity analysis based on software architecture. In: ISSRE, IEEE Computer Society (2002) 64–78
18. Immonen, A., Niemel, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and Systems Modeling **7**(1) (2008) 49–65
19. Wang, W.L., Wu, Y., Chen, M.H.: An architecture-based software reliability model. Pacific Rim International Symposium on Dependable Computing, IEEE **0** (1999) 143
20. Reussner, R., Schmidt, H.W., Poernomo, I.: Reliability prediction for component-based software architectures. Journal of Systems and Software **66**(3) (2003) 241–252
21. Grassi, V.: Architecture-based dependability prediction for service-oriented computing. In: Proceedings of the WADS workshop, Citeseer (2004)
22. W. Abdelmoez et al.: Error propagation in software architectures. In: METRICS '04, Washington, DC, USA, IEEE Computer Society (2004) 384–393
23. Voas, J.M.: Error propagation analysis for cots systems. Computing and Control Engineering Journal **8**(6) (1997) 269–272

24. Voas, J.M.: Pie: A dynamic failure-based technique. IEEE Trans. Software Eng. **18**(8) (1992) 717–727
25. Grunske, L., Han, J.: A comparative study into architecture-based safety evaluation methodologies using aadl's error annex and failure propagation models. In: HASE, IEEE Computer Society (2008) 283–292
26. Cortellessa, V., Singh, H., Cukic, B.: Early reliability assessment of uml based software models. In: Workshop on Software and Performance. (2002) 302–309
27. Roshandel, R.: Calculating architectural reliability via modeling and analysis. In: ICSE, IEEE Computer Society (2004) 69–71
28. Gokhale, S.S., Wong, W.E., Horgan, J.R., Trivedi, K.S.: An analytical approach to architecture-based software performance and reliability prediction. Perform. Eval. **58**(4) (2004)
29. Blake, J.T., Reibman, A.L., Trivedi, K.S.: Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: SIGMETRICS. (1988) 177–186