

Estimating Error Propagation Probabilities in Software Architectures¹

D.E Nassar, W.A. Rabie, M. Shereshevsky, N. Gradetsky, H.H. Ammar²
Lane Dept of Computer Science, West Virginia University
Morgantown WV 26506

BoYu, S. Bogazzi, A. Mili
College of Computer Science, New Jersey Institute of Technology
Newark NJ 07102

Abstract

The study of software architectures is emerging as an important discipline in software engineering, due to its emphasis on large scale composition of software products, and its support for emerging software engineering paradigms such as product line engineering, component based software engineering, and software evolution. Architectural attributes differ from code-level software attributes in that they focus on the level of components and connectors, and that they are meaningful for an architecture. In this paper, we focus on a specific architectural attribute, which is the *error propagation probability* throughout the architecture, i.e. the probability that an error that arises in one component propagates to other components. We introduce, analyze, and validate formulas for estimating these probabilities using architectural level information.

1. Introduction: Architectural Attributes

The study of software architectures is emerging as an important discipline in software engineering, because software architectures support many emerging paradigms of software development (product line engineering, component based software engineering, COTS-based software development) as well as the increasingly prevalent paradigm of *software evolution*. The shift from the traditional functional view of software development to the architectural view was first advocated by Shaw in [Shaw95], and has been widely recognized/ adopted since. As architectures emerge, so does the need to quantify them in a way that reflects their relevant quality attributes; relevant architectural attributes include features of the architecture that have an impact on the quality of software products that are instantiated from it.

¹ This work is supported by the National Science Foundation through grant No CCR 0296082, and by NASA through a grant from NASA IV&V.

² Correspondence author. Email: ammar@csee.wvu.edu

In this paper, we study the specific architecture-level attribute of *Error Propagation Probability*, which reflects the probability that an error that arises in one component of the architecture is propagated (rather than masked) to other components. This study is part of a larger project which investigates a wide range of attributes, including *Change Propagation Probability*, *Requirements Propagation Probability*, *Diagonality*, etc [Ammar et al, 2001]. In the spirit of the *Goal/ Question/ Metric* of Basili and Rombach [Basili, 1988], we map our attribute of interest onto a computable metric that can be evaluated on the basis of information that is available at the architectural level. At this level, we do not usually have the wealth of structural and semantic information that is available at the code level, but we do have information about the flow of control and data within components and between components. This precludes using traditional software metrics, which are based on such code-level features as tokens [Halstead77], flow graphs [McCabe76], data dependencies [Bieman, 1998] and control dependencies (to mention a few). Our approach can further be characterized by a combined *Bottom-Up/ Top Down* discipline, whereby we complement the top down approach advocated by Basili and Rombach's *Goal/ Quality/ Metric* paradigm with a bottom up approach that analyzes the architecture and derives a matrix that quantifies the flow of information within components and between components of the architecture. In the absence of structural and semantic information, we cannot analyze the flow of information within an architecture deterministically, hence we resort to a stochastic approach. This stochastic approach captures information flow within the architecture by means of random variables, and quantifies this flow by means of entropy functions applied to these random variables.

Software metrics has been a very active field for several decades, and it is impossible to do justice to all the relevant work in this area. We will briefly mention some of the research efforts that are most closely related to ours, highlighting in what way we are different. Many authors have used information theory to derive software metrics. Some, such as Allen and Khoshghoftar [Allen and Khoshghoftar, 1997], Chapin [Chapin02], and Harrison [Harrison, 1992] do so explicitly; others, such as Halstead [Halstead, 1977], do not invoke information theory explicitly, but their metrics can be interpreted in terms of this theory. Whereas all these authors define metrics by interpreting the source text of the program as the message, we derive metrics by interpreting information flow throughout the architecture as the message whose entropy we are calculating.

Measures of coupling and cohesion have also grabbed a great deal of researcher attention. Yourdon and Constantine [Yourdon, 1979] define a detailed classification of cohesion criteria, which Bieman [Bieman, 1998] and others use to define cohesion metrics for code-level and design-level artifacts. Briand, Morasca and Basili [Briand, 1996] define a detailed axiomatization of several property-based software metrics, including coupling and cohesion. Many of these measures of coupling and cohesion quantify the functional/ data dependencies between various components of a system, and are based on a structural analysis of

the system, viewed as an aggregate of processing elements and elementary data items.

Basili and Rombach [Basili and Rombach, 1988] introduce an analytical paradigm for the derivation of software metrics, called the *Goal/ Question/ Metric* paradigm. It is based on a systematic, goal-oriented, procedure for the derivation of software metrics. In [Fenton, 1994], Fenton presents a *Representational Theory of Measurement*, and argues that software metrics work must adhere to it. Also, he evaluates various metrics efforts with respect to the guidelines of this theory, and argues that Basili's *GQM* is but an instance of it. In our work we combine the top-down goal oriented approach advocated by Basili and Rombach [Basili and Rombach, 1988] and Fenton [Fenton, 1994] with a bottom-up approach, which analyzes the architecture and produces a matrix of entropies. These entropies are used for the purposes of estimating error propagation, but also for other purposes that are outside the scope of this paper.

Voas [Voas, 1997] analyzes error propagations between COTS components. His group developed a tool to simulate the error propagation behaviors using fault injection techniques. The users of the tool have to tell it what classes of internal states or outputs are undesirable. In another paper [Voas et al, 1998] of the same group, Voas et al apply fault injection techniques on part of a nuclear research program using another tool. They find that their results reflect the sensitivity of the software's tolerance to the tightening of the burnout hazard condition. Michael et al [Michael and Jones, 1997] present an empirical study of data state error propagation behavior. The authors argue that at a given location either all data state errors injected tend to propagate to the output, or else none of them does. They did their experiment on three programs sizing from around 400 lines to 75,000 lines. Their experiment indicates the homogeneous propagation behavior of error propagation that is for any n data-state errors either all of the errors propagate to the output, or none of the errors propagate to the output. In [Hiller et al, 2001] Hiller et al analyze error propagation conceptually. They define *error permeability* as the probability of an error in an input signal permeating to one of the output signals (there is one permeability value assigned between each pair of input/output signals). The value of error permeability is measured experimentally using fault injection techniques.

In section 2, we introduce the definition of *Error Propagation Probability* as it applies to software architectures, and discuss the importance of this feature in practice. In section 3, we discuss how to estimate error propagation probabilities throughout an architecture using the generic matrix of entropies, and document the approximations that we make in the process; also, we briefly discuss an automated system that we have developed and are currently evolving, which estimates the error propagation probabilities throughout an architecture from an analysis of its UML/ Rose RT representations. In section 4, we introduce a large scale system that performs a command and control function, and build its matrix of error propagation probabilities, using the automated tool to derive its entropy

matrix, then using the analytical formula to derive its error propagation matrix. In section 5 we discuss an empirical experiment whereby we simulated the behavior of the sample command and control system under fault injections, and observed how errors arising from these faults were propagated throughout the system. In section 6 we confront the results of the analytical study (section 4) against the results of the empirical study (section 5) to assess the validity of our analytical formula. Finally, in section 7 we summarize our findings and chart directions of future research.

2. Error Propagation Probabilities

In this section, we first introduce and discuss (in section 2.1) the feature of *error propagation* in a matrix. Then we review (in section 2.2) some derivatives of this feature, and discuss (in section 2.3) some applications thereof.

2.1 Error Propagation: Definition

Software architectures can be partitioned into *architectural styles*, which can be characterized by topological properties, message data types, and interaction protocols. Bass et al [Bass et al, 1998] distinguish between at least five distinct styles, the most prominent of which is the *Independent Components* architectural style. This style is characterized by its relative genericity: its topology is an arbitrary graph; its messages can be data or control signals; and its interactions can take place through a central message medium, or through one-to-one links. With limited loss of generality, we focus our subsequent discussions in this paper on the style of *Independent Components*. Specifically, the focus of our analysis of software architectures is the information flow within architectural components, and the information flow through architectural connectors. The absence of detailed semantic information at the architectural level precludes a deterministic approach to the analysis of data flow and control flow throughout an architecture; consequently, we resort to a stochastic approach, whereby control flow and data flow is modeled by a random variable, which is defined by a set of possible values, along with a probability distribution on this set.

Within this context we consider two components, say A and B, of an architecture, and we let X be the connector that carries information from A to B; for the purposes of our current discussion, the specific form of connector X is not important, we will merely model it as a set (of values that A may transmit to B). Also, the specific form of components A and B is not important for the purposes of our discussion; we will merely model them as functions that map an internal state and an input stimulus into a new state and an output.

Definition 1. The *Error Propagation Probability* from component A to component B is denoted by $EP(A,B)$ and defined by

$$EP(A,B) = \mathbf{P} ([B](x) \neq [B](x') \mid x \neq x'),$$

where $[B]$ denotes the function of component B , and x is an element of the connector X from A to B . We interpret $[B]$ to capture all the effects of executing component B , including the effect on the state of B as well as the effect on any outputs produced by B .

We interpret $EP(A,B)$ as the probability that an error in A is propagated by B (as opposed to being masked by B) because the outcome of executing B will be affected by the error in A . By extension of this definition, we let $EP(A,A)$ be equal to 1, which is the probability that an error in A causes an error in A . Given an architecture with N components, we let EP be an $N \times N$ matrix such that the entry at row A and column B be the error propagation probability from A to B .

2.2 Error Propagation Derivative: Unconditional Error Propagation

Note that the definition of the error propagation given in above uses the concept of *conditional* probability, i.e. we calculate the probability that an error propagates from A to B *under the condition that A actually transmits a message to B* . It is often useful, however, to use the *unconditional error propagation* which we will denote simply as $E(A,B)$, and define as the probability that an error propagates from A to B not conditioned upon the event that A sends a message to B . Function $E(A,B)$ is clearly dependent on $EP(A,B)$, but it further integrates the probability that A indeed sends a message to B .

In order to bridge the gap between the original (*conditional*) *error propagation* and the newly introduced *unconditional error propagation*, let us consider the *transmission probability matrix* T where the entry $T(A, B)$ reflects the probability with which the connector ($A \rightarrow B$) gets activated during a typical/ canonical execution. We want to distinguish between a connector that is invoked intensively in each execution and one that is invoked only exceptionally, under rare circumstances. We found no simple way to define matrix T so that we are completely satisfied with it. While the purpose of matrix T is perfectly clear in our mind (to reflect the variance in frequency of activations of different connectors during a typical execution), we have not been able to derive a generic formal definition; this matter is under investigation; this matter is under investigation. It has not precluded our progress so far because when we compare analytical results and experimental results (as we do in the sequel), we use the same matrix T (derived by simulation) on both sides.

By virtue of simple probabilistic identities, we argue that the *unconditional error propagation* is obtained as the product of the conditional error propagation probability with the probability that the connector over which the error propagates is activated, i.e.

$$E(A,B) = EP(A,B) \times T(A, B).$$

The concept of *unconditional error propagation* is useful when we discuss *cumulative error propagation probabilities*, which we do in the next subsection.

2.3 Error Propagation Derivative: Cumulative Error Propagation

Whereas so far we have focused our attention on single step error propagation from some component A to some component B , we want to consider, now, the probability that an error in some component A propagates to some component B in an arbitrary number of transmissions starting in A and ending in B . We call this the *cumulative error propagation probability* from A to B . We submit two premises pertaining to the analysis of cumulative error propagation:

- Cumulative error propagation probabilities must be derived, not from matrix EP (which represents conditional error propagation probabilities) but rather from matrix E (which represents unconditional error propagation probabilities). Indeed, the probability that an error propagates along some path depends first and foremost on the probability that the path is actually taken, combined with the probability that it is propagated through each arc of the path.
- Second, the matrix of cumulative error propagation probabilities cannot be derived as the traditional transitive closure of matrix E , because E is not a stochastic matrix (we have no basis to claim that if component A has an error, then it propagates to one component or another with probability 1). Hence we need to find a specific formula for this case, which we do in the sequel.

We denote by $E^*(A,B)$ the cumulative error propagation probability from A to B , and let E^* be the matrix of such probabilities. For arbitrary integer $s \geq 0$ let E_s be the *s-step error propagation* matrix, i.e. $E_s(A, B)$ is the probability that the error in A propagates to B in *exactly* s steps (s transmissions). For the trivial case when $B=A$ we accept the convention that for every component A error always propagates from A to itself in *zero* steps. Thus, it is reasonable to assume that E_0 is the unit matrix (i.e. has 1s on the diagonal and 0s elsewhere. Our convention also implies that for all $s > 0$ matrix E_s must have zero diagonal $E_s(A,A) = 0$ for all components i .

If an *error* in A is propagated to B after exactly n transmissions, it implies that the *message* from A that carries the error reaches B after exactly n transmissions. The reason for this is because once an error in a message is "corrected" (masked) at some point along its path, it is clear that the error does not propagate any further, and therefore does reach its destination in any number of steps. Therefore the *s-step error propagation* ($s > 1$) from A to B can be expressed by the formula

$$E_s(A, B) = \sum_{\substack{1 \leq C_1, C_2, \dots, C_{s-1} \leq N \\ C_r \neq B \text{ for } 1 \leq r \leq s-1}} E(A, C_1)E(C_1, C_2) \dots E(C_{s-1}, B) \quad \text{when } A \neq B; \quad (1)$$

and $E_s(A, A) = 0$ for all A .

In the sum above we count only the products of propagation probabilities along the paths in the architecture graph which *do not have a loop at their terminal node*, because a path where such loop exists (see Fig. 1) actually represents a propagation in *less than n* steps and, thus, should not be counted in $E_s(A, B)$.

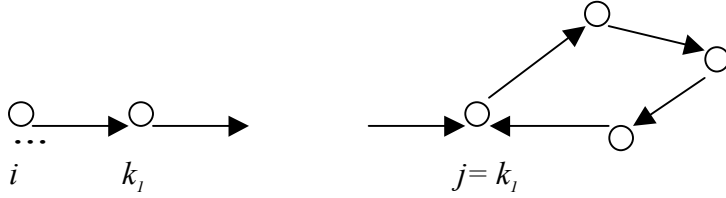


Fig. 1

Computing the s -step error propagation by formula (1) would evidently be very hard to automate. However, we can find an easy and rather tight upper estimate for $E(A, B)$ by replacing the right-hand side in (1) with the regular s th power of the matrix E , that is:

$$E(A, B) \leq \bar{E}(A, B), \text{ for all } s \geq 1 \text{ and every pair of components } A, B,$$

where $\bar{E}_1 = E_1 = E$ and for $s \geq 2$ we set:

$$\bar{E}_s(A, B) = \sum_{1 \leq C_1, C_2, \dots, C_{s-1} \leq N} E(A, C_1)E(C_1, C_2) \dots E(C_{s-1}, B), \text{ when } A \neq B;$$

(2)

$$\text{and } \bar{E}_s(A, A) = 0 \text{ for all } A.$$

The (non-diagonal) elements of the upper bound matrix $\bar{E}(A, B)$ can be computed recursively:

$$\bar{E}_{s+1}(A, B) = \sum_C \bar{E}_s(A, C)E(C, B) \text{ for } A \neq B$$

$$\text{and } \bar{E}(A, A) = 0 \text{ for all } A.$$

Basically, to obtain $\bar{E}_{s+1}(i, j)$ we just multiply the matrices \bar{E}_s and E , but we have to set all the diagonal elements at zero, therefore \bar{E} is **not** exactly the s -th power of the error propagation matrix E , but the following inequality always holds

$$\bar{E}(A, B) \leq E^s(A, B),$$

for all $s \geq 1$ and every pair of components A, B . (3)

Now we are prepared to derive the probability of an error propagating from A to B eventually, i.e. in *any* number of steps. We call this quantity the *cumulative error propagation from A to B* and denote it by $E^*(A, B)$ from A to B . From the definition of E_s one easily derives that:

$$E^*(A, B) = \sum_{s \geq 0} E_s(A, B).$$

(4)

Note that the above equation gives $E^*(A, A) = 1$ for each A which is exactly what one would expect. Using the inequality (3) we obtain the upper bound for the cumulative error propagation:

$$E^*(A, B) \leq \sum_{s \geq 0} \bar{E}_s(A, B). \quad (4)$$

2.4 The Error Isolation Coefficient

Whereas the two previous features are matrices (like the original EP), this feature is a scalar. Through this scalar, we aim to reflect the potential of an architecture to insulate its components from each other's errors. Obviously, the ideal *Error Isolation Coefficient (EIC)* corresponds to a matrix where only the diagonal cells are 1s and the rest of the matrix is 0, i.e. an identity matrix; in such a matrix, no component propagates errors to other components. On the other extreme, the *worst possible EIC* is one for which all cells are 1s; in such a matrix, whenever an error arises in some component, it propagates to all other components. We wish to define the error insulation coefficient in such a way as to reflect, using a value between 0 and 1, how close we are from the ideal matrix and how far we are to the worst possible matrix. Without dwelling into detailed mathematics, we submit that the following formula satisfies our criteria, and let it be our definition of the *Error Isolation Coefficient (EIC)*:

$$EIC = \frac{\sum_{A \neq B} EPM(A, B)}{N^2 - N},$$

where N is the number of components in the architecture. This formula can in principle be applied to any matrix, although we generally apply it to matrix E^* .

2.5 Error Propagation: Applications

By definition, the error isolation coefficient (EIC) represents, in a single scalar between zero and one, the potential of an architecture to facilitate (or resist) error propagation throughout a matrix. Even though we recognize the perils of reducing such a complex / multidimensional feature as an architecture's behavior vis-à-vis error propagation to a mere scalar, we argue that such a quantity can be useful as a criterion for comparison between two candidate architectures. In particular, it can be useful in the context of a superficial first-level comparison, or as a tie breaker if all other aspects fail to distinguish between candidates.

Furthermore, given a matrix E^* of cumulative error propagation probabilities through an architecture, we can use it in any one of the following arguments.

- If row A contains large values (close to 1), we infer that errors in A have a large likelihood to propagate to other components; one possible way to remedy this situation is to add *assert* statements to detect and correct errors in A ; another solution is to focus testing effort on A so as to reduce the likelihood of errors at run time.
- Likewise, if column B contains large values (close to 1), we infer that component B is likely to be affected by errors in other components; one possible way to remedy this situation is to enhance B with fault tolerant capabilities, so that it is more likely to mask incoming faults, and less likely to let incoming faults affect its behavior.
- A third possible interpretation involves a vector V of *error frequencies*, which we interpret to represent, at entry A , the estimated frequency of occurrence of errors (at run time). If we take the hypothesis that error frequency is a function of fault density and usage frequency of each component, we can derive vector V in two steps. We can infer usage frequency by analyzing expected usage patterns of the architecture [Mills et al, 1987]; then we infer fault density by analyzing the (relative) complexity of components, and relying on the extensive research linking complexity with fault density. The derivation of usage frequencies is similar to the discussion we had earlier about matrix T of transition probabilities: whereas usage frequencies focus on the frequency of invocation of components, transition probabilities focus on the frequency of activation of connectors. Given vector V and matrix E^* , we argue that the product $E^* \bullet V$ represents the error density of the architecture, taking into account error generated in-situ and errors propagated from other components.

3. Estimating Error Propagation Probability

In section 3.1, we introduce and analyze a formula for error propagation, which is discussed in detail in the appendix; in section 3.2, we discuss how to estimate this formula for architectures represented in UML.

3.1. An Information Theoretic Analysis

For the purposes of our analysis, we model architectural components using the state transition view, in which each component maintains an internal state that summarizes its invocation history. Upon receipt of a message from another component A, component B adjusts its internal state (eventually) and produces some outputs that are broadcast to other components. We are interested in considering under what condition (with what probability) does an error in the incoming message cause an error in the behaviour of the receiving component. Without significant loss of generality, we focus our attention on the effect of incoming messages on the internal state. In this framework, notions of state and message are not limited to a basic state machine, rather these notions are applied to the software system at different levels of abstraction, including the architectural level.

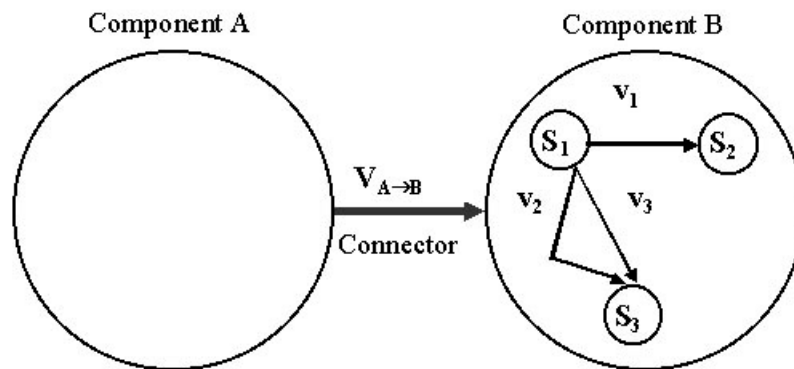


Figure 2. State transition view for two abstract components A and B.

In the framework of the *state transition view*, we consider two architectural components, say *A* and *B* communicating through a connector (see Figure 2). Every act of *A*-to-*B* communication consists of passing from *A* to *B* a message v selected from certain vocabulary $V_{A \rightarrow B}$. For example, if the connector is realized in the form of method invocations (for different types of architectural connectors see a survey by

Medvidovic [Medvedovic 00]) then set $V_{A \rightarrow B}$ can be thought of as the range of values of the “aggregate parameters” of all methods of B through which A may transmit information to B . Let S_B be the set of states of module B . In the appendix, we show that error propagation from A to B is given by the following equation:

$$\text{EP}(A \rightarrow B) = \frac{1 - \text{OR}(V_{A \rightarrow B}, S_B)}{1 - \text{IR}(V_{A \rightarrow B})}, \quad (*1)$$

where

- the term $\text{OR}(V_{A \rightarrow B}, S_B)$ measures the average amount of redundancy in messages that B receives from A as seen by component B ; i.e. based on the state transitions of component B , and hence we call this metric the *observed redundancy* (OR), and
- the term $\text{IR}(V_{A \rightarrow B})$ measures the intrinsic amount of redundancy in the messages that B receives from A ; i.e. the amount of redundancy in messages form A to B as seen by component A , and hence we call this metric the *intrinsic redundancy* (IR).

The equation (*1) implies, in particular, that the probability of error propagation decreases as more observed redundancy is embedded in messages from sender to receiver. For example, in Figure , if component B is in state S_1 and receives, by error, message v_3 instead of message v_2 , then the error will be masked off and will not propagate as both messages (at the given state S_1) cause a transition to state S_3 . It is worth noting here that: the redundancy in messages that counts towards decreasing error propagation is that redundancy observed by the receiving component (OR). However, the *observable redundancy* (OR) is a function of the *intrinsic redundancy* (IR) and the state-transition mapping. The detailed derivation of formula (*1) and expansion of its terms are given in the Appendix.

For a partial (but important) case when the nominal message and the erroneous one both have the same probability distribution $P_{A \rightarrow B}$ on the set of allowed messages $V_{A \rightarrow B}$, formula (*1) assumes the form:

$$\text{EP}(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)]^2}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2}. \quad (*2)$$

In the above formula the expression for the intrinsic redundancy $\text{IR}(V_{A \rightarrow B})$

$$IR(V_{A \rightarrow B}) = \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2$$

is easily recognized to be the so called *collision probability* of the ensemble $(V_{A \rightarrow B}, P_{A \rightarrow B})$. Clearly, $0 \leq IR(V_{A \rightarrow B}) \leq 1$, and $IR(V_{A \rightarrow B})$ reaches its minimum value of $1/|V_{A \rightarrow B}|$ if and only if $P_{A \rightarrow B}$ is the uniform distribution: $P_{A \rightarrow B}(v) = 1/|V_{A \rightarrow B}|$ (for all v), and its maximum value of 1 – when $P_{A \rightarrow B}$ is a “deterministic” distribution, i.e. one that assigns probability 1 to one element of the set $V_{A \rightarrow B}$ and probability 0 to all others. Now we note that the expression

$$- \log_2 IR(V_{A \rightarrow B}) = - \log_2 \left(\sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2 \right) \quad (*3)$$

behaves very much like the Shannon entropy [Shannon] of the probabilistic ensemble $(V_{A \rightarrow B}, P_{A \rightarrow B})$: the more homogeneous is the distribution – the larger is the value of (*3), which reaches its maximum of $\log_2 |V_{A \rightarrow B}|$ at the ensemble of highest uncertainty (all elements have equal probabilities), and its minimum of 0 at the ensemble of lowest uncertainty (all but one elements have probability zero). In fact, expression (4) turns out to be the so-called 2^{nd} order *Renyi entropy* (see e.g. [Renyi, 1961]) $H_2^R(V_{A \rightarrow B}, P_{A \rightarrow B})$ of the *A-to-B data flow ensemble* $(V_{A \rightarrow B}, P_{A \rightarrow B})$. Thus

$$IR(V_{A \rightarrow B}) = 2^{-H_2^R(V_{A \rightarrow B}, P_{A \rightarrow B})},$$

i.e. the intrinsic redundancy is a monotonically decreasing function of the 2^{nd} order *Renyi entropy* that can be viewed as a measure of the information content. Similarly, the observed redundancy term in equation (*2)

$$OR(V_{A \rightarrow B}, S_B) = \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)]^2 \quad (*4)$$

is the weighted sum (probabilistic *mean*) of redundancy terms which are inverse exponents of Renyi entropy of the emerging state of the receiving component B, considered as a random variable whose probability distribution is induced by the distribution $P_{A \rightarrow B}$ of incoming messages, once the current state x of B is fixed. The probabilities of various states of B play the role of weight coefficients in the weighted sum (*4). See appendix for the details of deriving these terms using the concepts of information theory. We observe here that OR is greater than or equal to IR . This is consistent with the argument that the state transitions that a receiving component B undergoes due to receiving messages from A can not contain more information than that is intrinsic in the messages themselves.

3.2 Computing Error Propagation

In this section, we illustrate how to compute the different quantities involved in the EP formula (*2) from a given architectural specifications of a software system. Our discussion focuses on UML-RT specifications [UML-RT], however, the approach is independent of the specification language used to describe a software architecture. Using the UML-RT notation, an architectural component is a *capsule* and an inter-component connector is a *connector* that connects a pair of *ports* (a *base port* and a *conjugate port*) which use a common communication *protocol* in two capsules (see Figure3 for illustration).

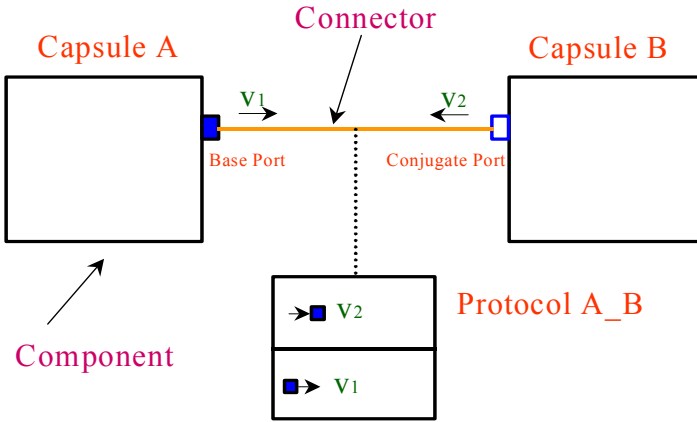


Figure3: Identifying components and connectors in UML-RT.

The EP formula (*2) that we presented in section 3.1 assumes that the error-free message transmissions and the erroneous message transmissions have the same probability distribution $P_{A \rightarrow B}$ on the set of allowed messages $V_{A \rightarrow B}$. For example, if the set of messages from component A to component B consists of 3 messages: $\{v_1, v_2, v_3\}$, (see Figure 2), and message v_1 gets corrupted, then the probability that message v_2 gets send instead on message v_1 is $P_{A \rightarrow B}(v_2)$, while the probability that message v_3 gets send instead on message v_1 is $P_{A \rightarrow B}(v_3)$.

The formula that governs EP under the corruption pattern deccribed above is given by (see appendix for formula derivation):

$$EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)]^2}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2},$$

Where,

- $P_B(x)$: is the probability of observing component B in state x, and
- $P_{A \rightarrow B}[F_x^{-1}(y)]$: is the probolity of transmisssion of a message (from A to B) that causes component B to transit from state x to state y.
- $P_{A \rightarrow B}[v]$: is the probability that message v is sent over the connector from A to B,

The probability of observing component B in state x , $P_B(x)$, is a dynamic quantity that depends on how the system is used. Hence, we employ all available *sequence diagrams* and the *state diagrams* (of component B) in computing this quantity using a component/ state traversal technique. Starting from the top most sequence diagram

that describes the usage of the system, we traverse all paths in which component B interacts with the components of the system (sending or receiving messages), these interactions also count for self-loops of B itself. We also traverse the paths in which B interacts with the surrounding environment (responding to stimuli or producing output), surrounding environment means all external entities to the software system. We count all incidents in which B is in action (N_B), out of this number of incidences, there are N_{Bx} incidences corresponding to B being in action while in state x . The probability of observing component B in state x is thus estimated to be:

$$P_B(x) = (N_{Bx} / N_B).$$

The probability of transmission of message v (from A to B), $P_{A \rightarrow B}[v]$, is also a dynamic quantity. In order to find an estimate of $P_{A \rightarrow B}[v]$, we traverse all available *sequence diagrams* along all paths in which messages are sent from component A to component B. We count number of times each of the messages in $V_{A \rightarrow B}$ is transmitted. We denote the number of times a message v_i is sent from A to B as Nv_i , then:

$$P_{A \rightarrow B}[v_i] = (Nv_i / (\sum_{<j> Nv_j}),$$

Where $j = 1 \dots M$ and M is the number of unique messages ($V_{A \rightarrow B}$) that are sent from A to B.

The probability of transmission of a message (from A to B) that causes component B to transit from state x to state y , $P_{A \rightarrow B}[F_x^{-1}(y)]$, is also a dynamic quantity. We recognize two necessary steps to compute $P_{A \rightarrow B}[F_x^{-1}(y)]$:

- The first step is to determine the set of messages ($V_{AB}^{x \rightarrow y}$) that cause component B to transit from state x to state y , this step is achieved by static analysis of the *state diagrams* of component B. For each pair of states (x, y) we determine the messages ($V_{AB}^{x \rightarrow y}$) that may cause B to transit from state x to state y . For the simple system shown in Figure , $(V_{AB}^{S1 \rightarrow S2}) = \{v_1\}$, and $(V_{AB}^{S1 \rightarrow S3}) = \{v_2, v_3\}$. Thus, the first step requires no dynamic analysis.
- The second step counts for the dynamic part of $P_{A \rightarrow B}[F_x^{-1}(y)]$, that is determining the probability of messages sent from A to B that causes B to transit from state x to state y . $P_{A \rightarrow B}[F_x^{-1}(y)]$ is the sum of probabilities of messages $V_{AB}^{x \rightarrow y}$, the probability of each message in $V_{AB}^{x \rightarrow y}$ is obtained from $P[V_{A \rightarrow B}]$.

For example, consider the simple architecture shown in Figure , let the probability distribution for the messages ($V_{A \rightarrow B}$) be given by: $P[V_{A \rightarrow B}] = \{p_{v1}, p_{v2}, p_{v3}\}$, then

$$P_{A \rightarrow B}[F_{S1}^{-1}(S3)] = p_{v2} + p_{v3}, \text{ while } P_{A \rightarrow B}[F_{S1}^{-1}(S2)] = p_{v1}.$$

An important quantity that we also compute from the dynamic specifications of a given system is the transition probability matrix \mathbf{T} (see section 2.2 for the definition). By traversing all available *sequence diagrams* we count the number of export messages from a component (say A) to all other components, let this be presented as:

$$\{n_{iA} : i = 1, \dots, N \wedge i \neq i(A)\},$$

where N is the number of components in the system and $i(A)$ is the index of component A. We then define:

$$T(A, B) = (n_{i(B)A} / (\sum_{<i> n_{iA})),$$

where $i = 1 \dots N$. All entries of the T matrix are populated using this relation. The transition probability matrix **T** is used in obtaining the unconditional error propagation matrix (**E**) from the conditional error propagation matrix (**EP**).

4. Sample Example: A Command and Control System

4.1. System Specification

The example we use to illustrate our work is a large command and control system that is used in a life-critical, mission-critical application. This system was modeled using the Rational Rose Realtime CASE tool. Due to restrictions from the organization that provided us with this sample case study, we are only authorized to present this system in abstract terms. It is a Computer Software Configuration Item (CSCI) that provides the following functions:

- Facilitating Communication, Control, Cautions and Warnings including subsystem Configuration Management, C&DH (Communication and Data Handling) Communications Control, Processing, Memory Transfer, C&DH Failure Detection, Isolation, and Recovery and Time Management,
- Controlling a Secondary Electrical Power System, and
- Environmental Control, which provides Temperature and Humidity Control.

We will concentrate on the Thermal Control part of the system, which is a rather complex system with operations setting controller, fault recovery procedures, and pump control functionalities. Figure 4 shows the system's main use case diagram and all the relationships among the use cases and the actors.

The system has a hierarchical architecture. The top-level software architecture of this system and other subcomponents that interact with it to achieve its mission is shown in Figure 5. Also, the internal architecture of SubSystem Z is shown in Figure 6.

The system is responsible for providing overall management of the pumps as well as performing the necessary monitoring and response to the sensors data. Also, it is responsible for performing automated startup, and controlling Thermal System reconfigurations. During each execution cycle, a check is performed for incoming commands. All commands processed by the Thermal System are received in this application, however, the commands processed by System Control are for the startup of the system and for system mode changes. Received commands are validated in the same execution cycle. Mode change commands, which will reconfigure the Internal Thermal System, are also accepted from other components of Thermal System to compensate for system component failures or coolant leaks. A failure recovery system detects failure conditions and performs recovery operations in response to the

detected failures. Failure conditions include combinations of Pump failures and Shutoff Valve failures.

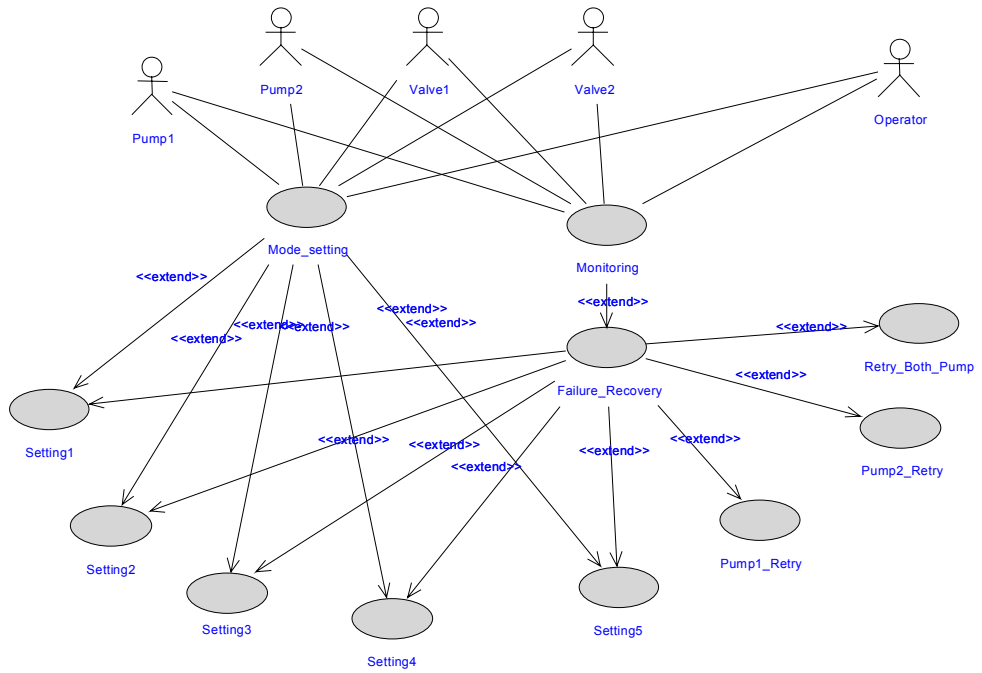


Figure 4 The Use Case Diagram of the Thermal Control

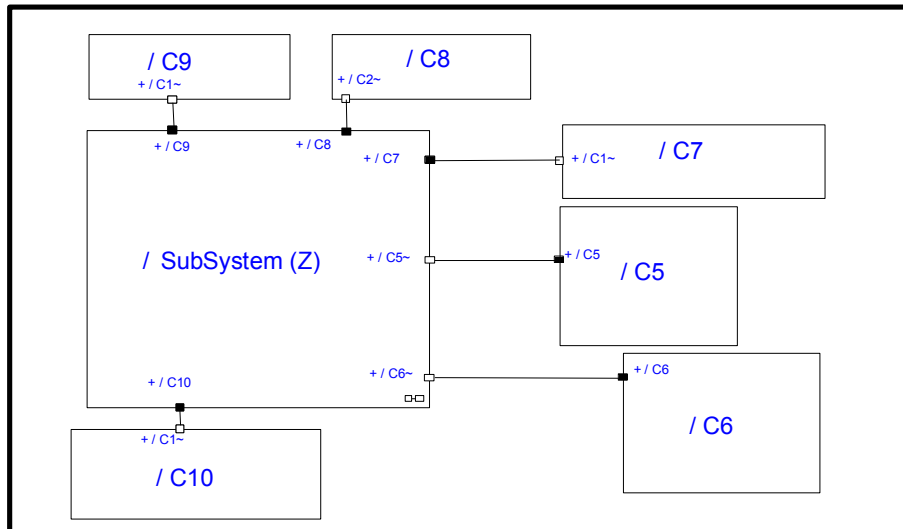


Figure 5: Top-level software architecture of system (Z).

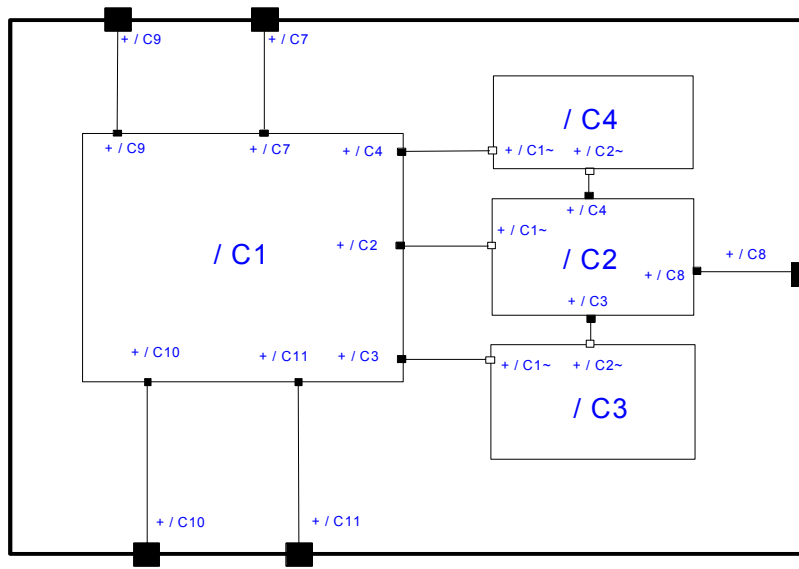


Figure 6: The architecture of SubSystem (Z).

Figures 7 and figure 8 show the sanitized sequence diagram and state diagram that describe the operation of the system (these have been sanitized at the request of the organization that has provided us the system for study).

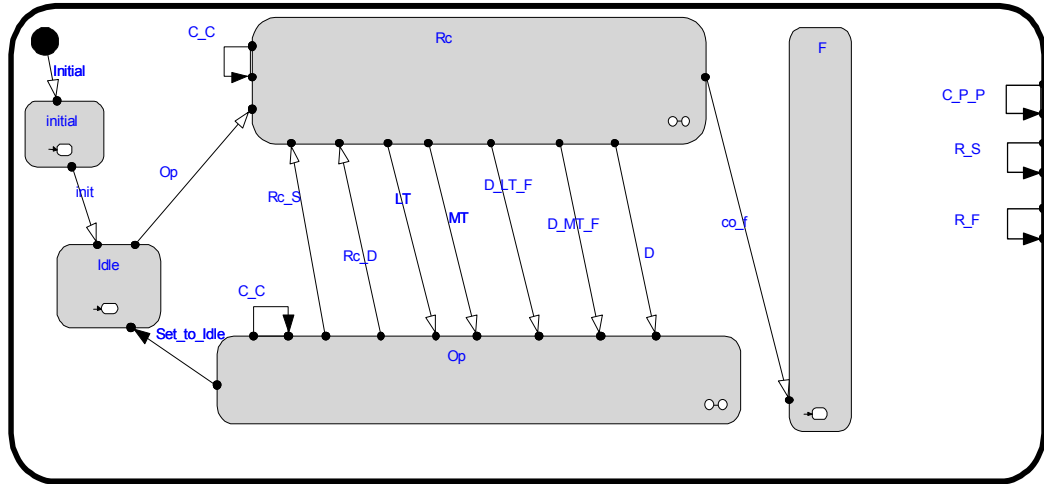


Figure 8: A sample of a sanitized state diagram

4.2. Error Propagation Matrix: Analytical Results

We consider the CSCI system discussed above, and we derive the matrix EP of (conditional) error propagation probabilities of this system, using the formula discussed in section 3. This formula of EP is estimated/ approximated from a static analysis of the UML-RT representation of this system, using exclusively information that is available at this level. Although we have developed and are currently refining an automated tool that derives matrix EP (as well as other metrics) from the UML model (and other input formats), this particular case study was done by hand.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	1.00	0.11	0.42	0.34	0.45	0.46	0.00	0.00	0.00	0.00
	C2	0.20	1.00	0.00	0.00	0.00	0.00	0.00	0.52	0.00	0.00
	C3	0.01	0.47	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C4	0.02	0.23	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
	C5	0.00	0.28	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
	C6	0.00	0.13	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	C7	0.38	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
	C8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
	C9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
	C10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Table 1: Conditional Error Propagation Matrix: Analytical Results

For this particular case study, we have derived the connector activation matrix T as a stochastic probability matrix that contains for each entry (A,B), the probability that connector (A,B) is activated, given that component A is broadcasting a message. Matrix T was derived by dynamic analysis of the UML-RT model. It is given in Table 1.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	0.00	0.01	0.03	0.03	0.03	0.03	0.86	0.00	0.00	0.00
	C2	0.55	0.00	0.10	0.10	0.00	0.00	0.00	0.24	0.00	0.00
	C3	0.57	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C4	0.56	0.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C5	0.64	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C6	0.60	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C7	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C10	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 2: Transmission Matrix

Using this connector activation matrix, we derive the *unconditional error propagation matrix* E_A , also referred to as the 1-step error propagation matrix of the system; this is given in Table 2.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	0.0000	0.0012	0.0132	0.0102	0.0146	0.0145	0.0000	0.0000	0.0000	0.0000
	C2	0.1104	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.1264	0.0000	0.0000
	C3	0.0060	0.2024	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C4	0.0107	0.1026	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C5	0.0000	0.1005	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C6	0.0000	0.0506	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C7	0.3761	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C9	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C10	0.0014	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 3: Unconditional Error Propagation Matrix: Analytical Results

Using the unconditional error propagation matrix, say E_A , given above, we derive the matrix of cumulative error propagation probabilities, which we call E^*_A . We find the following matrix.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	1.00	1.16E-03	1.32E-02	1.02E-02	1.46E-02	1.45E-02	0.00	1.46E-04	0.00	0.00
	C2	1.10E-01	1.00	1.46E-03	1.12E-03	1.61E-03	1.60E-03	0.00	1.26E-01	0.00	0.00
	C3	6.04E-03	2.02E-01	1.00	6.15E-05	8.82E-05	8.78E-05	0.00	2.56E-02	0.00	0.00
	C4	1.07E-02	1.03E-01	1.41E-04	1.00	1.56E-04	1.55E-04	0.00	1.30E-02	0.00	0.00
	C5	1.11E-02	1.01E-01	1.47E-04	1.13E-04	1.00	1.61E-04	0.00	1.27E-02	0.00	0.00
	C6	5.59E-03	5.06E-02	7.39E-05	5.69E-05	8.15E-05	1.00	0.00	6.40E-03	0.00	0.00
	C7	3.76E-01	4.35E-04	4.98E-03	3.83E-03	5.49E-03	5.47E-03	1.00	5.49E-05	0.00	0.00
	C8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
	C9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
	C10	1.41E-03	1.62E-06	1.86E-05	1.43E-05	2.05E-05	2.04E-05	0.00	2.05E-07	0.00	1.00

Table 4: Cumulative Error Propagation Matrix: Analytical Results

Except for possible round-off errors, matrix E^*A is greater than matrix E_A , entry by entry. The error isolation coefficient of this architecture, computed on the cumulative error propagation matrix, is found to be

$$EIC = 0.0128.$$

On the original (conditional) error propagation matrix, we find

$$EIC = 0.0447.$$

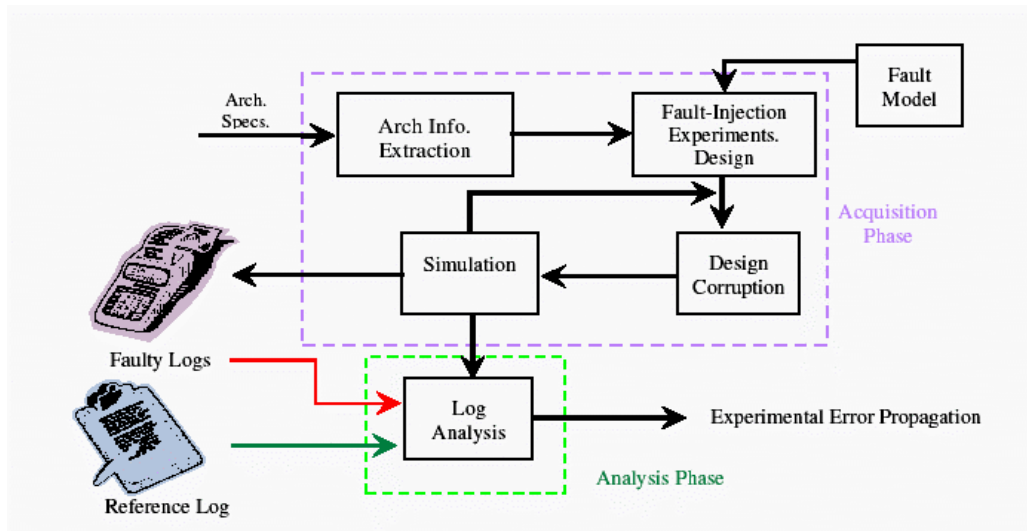
5. Empirical Fault Injection Experiment

5.1. Experiment

In order to validate our analytical study, we developed a framework for experimental error propagation analysis. Our empirical validation effort is based on fault injection, simulation-trace recording, and then comparing the obtained “faulty-run” logs against simulation trace of a fault-free “golden-run” [Michael 97, Ammar 01, Voas 97, Voas 98].

We perform the simulation-based error-propagation analysis in two phases (as shown in Figure 9): an acquisition phase and an analysis phase. In the acquisition phase we extract architecture information about the components and connectors that make up the software system (the underlying finite state machines in the components, and the inter-component messages). We generate faults based on message swapping fault model. In each experiment, we inject one of the generated faulty messages. Finally, we simulate the given system in the presence of injected faults and record simulation traces for the

Figure 9: A block diagram of the two-phase error-propagation analysis



different experiments. The faulty-run logs also contain markers that indicate when during the simulation faults are injected.

In the analysis phase, we conduct a post-simulation comparison between the faulty-run logs and the reference, fault-free, log. Any encountered discrepancy is counted as an error propagation incident. We compute the experimental error propagation factor between components A and B as the percentage of the faults injected into component A that developed into errors in component A and propagated to component B.

Figure 10 shows an example of how we compare a faulty-run log (dashed line) with a reference log (solid line). The indices on the vertical axis correspond to the states of the receiving component, while the horizontal axis indicates elapsed time during the simulation. The triangular marks (Δ) correspond to fault injection markers. The star symbols (*) correspond to log-alignment markers, which we use to compensate for any timing skews between faulty logs and the reference log. In this example, three faults were injected into one of the system components, two of them caused error to propagate to the component whose state-transition log is plotted in Figure 10.

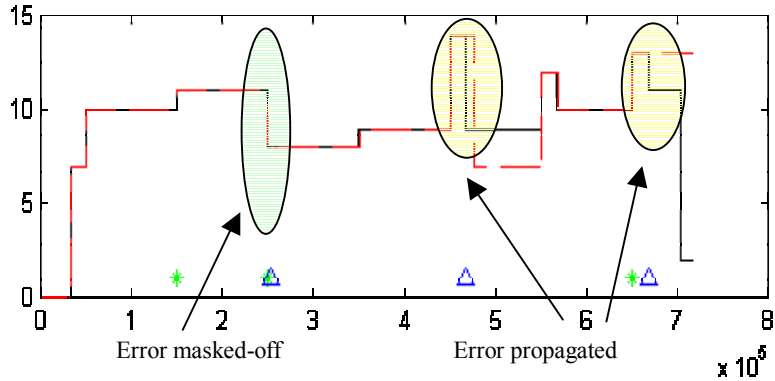


Figure 10: An illustration of log comparison.

5.2. Results of the Experiment

Table 5 shows the experimentally obtained error propagation matrix for the same mode of operation (k) whose analytical error propagation matrix is given in Table 1. As in any empirical effort for probability estimation, the larger the number of experiments we carry out the more confidence we have in the results. In the sequel we present a correlation study between the analytical and the experimental results of error propagation.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	1.00	0.11	0.15	0.30	0.76	0.73	0.00	0.00	0.00	0.00
	C2	0.37	1.00	0.00	0.00	0.00	0.00	0.00	0.52	0.00	0.00
	C3	0.03	0.32	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	C4	0.03	0.19	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
	C5	0.00	0.67	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
	C6	0.00	0.28	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
	C7	0.32	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
	C8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
	C9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
	C10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Table 2: Conditional Error Propagation Matrix: Empirical Results, EP_E

From EP_E , the conditional error propagation matrix obtained empirically, we derive matrix EE , the unconditional error propagation matrix, using the transition matrix T . Of course, for the sake of the comparison, we use the same transition matrix T as we did for the analytical study. Matrix EE is shown in Table 6.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	0.0000	0.5000	0.0557	0.4912	0.1331	0.1280	0.0000	0.0000	0.0000	0.0000

	C2	0.7838	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.4286	0.0000	0.0000
	C3	0.0161	0.7083	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C4	0.7917	0.6429	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C5	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C6	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C7	0.7500	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C9	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	C10	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 6: Unconditional Error Propagation Matrix E_E : Empirical Results

Using matrix E_E , we derive matrix E^*_E of cumulative error propagation probabilities, and find the result shown in Table 7. Note that except for round-off errors, this matrix is greater than the matrix of unconditional probabilities, entry by entry.

		B									
		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
A	C1	1.00	5.00E-01	5.57E-02	4.91E-01	1.33E-01	1.28E-01	0.00	2.14E-01	0.00	0.00
	C2	7.84E-01	1.00	4.37E-02	3.85E-01	1.04E-01	1.00E-01	0.00	4.29E-01	0.00	0.00
	C3	1.61E-02	7.08E-01	1.00	7.91E-03	2.14E-03	2.06E-03	0.00	3.04E-01	0.00	0.00
	C4	7.92E-01	6.43E-01	4.41E-02	1.00	1.05E-01	1.01E-01	0.00	2.76E-01	0.00	0.00
	C5	7.84E-01	1.00E+00	4.37E-02	3.85E-01	1.00	1.00E-01	0.00	4.29E-01	0.00	0.00
	C6	7.84E-01	1.00E+00	4.37E-02	3.85E-01	1.04E-01	1.00	0.00	4.29E-01	0.00	0.00
	C7	7.50E-01	3.75E-01	4.18E-02	3.68E-01	9.98E-02	9.60E-02	1.00	1.61E-01	0.00	0.00
	C8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
	C9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
	C10	1.00E+00	5.00E-01	5.57E-02	4.91E-01	1.33E-01	1.28E-01	0.00	2.14E-01	0.00	1.00

Table 1: Cumulative Error Propagation: Experimental Results

6. Validation: Correlating Analytical and Empirical Results

In this section, we confront the results computed by the analytical formula of section 4 against the results derived from the fault injection experiment of section 5 to assess the validity of our analytical formulas. We let E_A and E_E be (respectively) the analytical matrix and the empirical matrix of (unconditional) error propagation for our sample architecture; these are both 10×10 matrices. We use a number of criteria to this effect, all of them involving correlations between a set of values obtained analytically and an equivalent set obtained empirically.

- The first possible criterion is simply the correlation between the entries of the two matrices; because these matrices contain 100 values each, the correlations do bear some significance.

- The second possible criterion is to correlate, not all the values of the matrices, but rather the non-trivial values (other than 0 and 1); the rationale behind this criterion is that trivial values do really test our analytical formula.
- The third criterion discriminates between empirical values that were derived from a small number of fault injections and those that were derived from a large number of fault injections. If our analytical results are accurate, we should find empirical values that stem from large numbers of fault injections to be closer to analytical values than those that stem from small numbers of fault injections.

Orthogonally, we find it useful to compare not only E_A and E_E , which represent single step propagations, but also cumulative versions of these matrices, which represent probabilities of error propagations that may have taken more than one step through the architecture. In other words, whereas $E(A,B)$ represents the probability that an error propagates from A to B in a single exchange from A to B, we want $E^*(A,B)$ to represent the probability that an error propagates from A to B in an arbitrary number of hops through intermediate components. There are two reasons why we may want to consider the cumulative matrix E^* in addition to the single-step matrix E :

- In practice, if we are interested in the probability that an error in A propagates to B, we usually do not care in how many steps the propagation takes place; hence EP^* is a better reflection of what we want to measure than EP .
- Also, if there is any discrepancy between what the analytical study defines as a single step and what the empirical study does, this discrepancy will be smoothed out once we consider propagations of arbitrary length.

In the remainder of section 6, we apply these criteria to matrices E and E^* in order to determine to what extent the values obtained empirically are consistent with those found analytically.

6.1. Correlating One Step Matrices

In this section, we present the results of the study that we conducted to explore the correlation between the analytically estimated single step error propagation matrix and its experimentally computed counterpart. The correlation coefficient between the entire cells of the analytical EP matrix and the experimental EP matrix is

$$\text{Cor}(E_A, E_E) = 0.628.$$

We note, however, that there are only 15 non-trivial entries in each of the two matrices. Trivial entries correspond to self-loops from a component to itself (with error propagation probability of 1 by definition) and to the non-directly connected components (with error propagation probability of 0 by definition). It may be useful

to evaluate the correlation between the set of non-trivial values of matrices E_A and E_E . We find

$$\text{Cor} = 0.3240.$$

Table 8 contains the 15 non-trivial entries corresponding to the 15 connectors over which faults were injected during the controlled experiment. Note that the number of injected faults over connectors highly varies considerably across the entries. The connectors in the table follow a descending order with respect to the number faults injected over each connector.

We start computing the correlation coefficient by considering only the two connectors with the highest number of injected faults (entries 1 and 2 in Table 8). As we consider more connectors, with appreciable number of fault injections, we notice that a high correlation exists between the analytical and the experimental results. For example when the first four entries are considered (number of fault injections greater than 500), the correlation coefficient is greater than 0.8. Figure gives a bar chart representation of the correlation information presented in Table 8.

Entry	Connector		EA	EE	# Fault Injections	Correlation Coefficient	
	From	To					
1	1	5	0.0146	0.1331	1067	N/A	N/A
2	1	6	0.0145	0.1280	1055	1.0000	Entries 1 through 2
3	1	3	0.0132	0.0557	592	0.9999	Entries 1 through 3
4	3	1	0.0060	0.0161	559	0.8708	Entries 1 through 4
5	7	1	0.3761	0.7500	64	0.9894	Entries 1 through 5
6	1	4	0.0102	0.4912	57	0.8160	Entries 1 through 6
7	2	1	0.1104	0.7838	37	0.7153	Entries 1 through 7
8	1	2	0.0012	0.5000	36	0.6488	Entries 1 through 8
9	4	2	0.1026	0.6429	28	0.6433	Entries 1 through 9
10	3	2	0.2024	0.7083	24	0.6829	Entries 1 through 10
11	4	1	0.0107	0.7917	24	0.5576	Entries 1 through 11
12	2	8	0.1264	0.4286	7	0.5501	Entries 1 through 12
13	5	2	0.1005	1.0000	4	0.5068	Entries 1 through 13
14	6	2	0.0506	1.0000	4	0.4291	Entries 1 through 14
15	10	1	0.0014	1.0000	4	0.3240	Entries 1 through 15

Table 8: Correlation between analytical and experimental EP probabilities

The results in this table are interesting, in that they show a nearly perfect correlation between experimental results and analytical results in those cases where the experimental result is based on a large number of fault injections. Also, predictably, the correlation drops as the number of fault injections drop (though not monotonically).

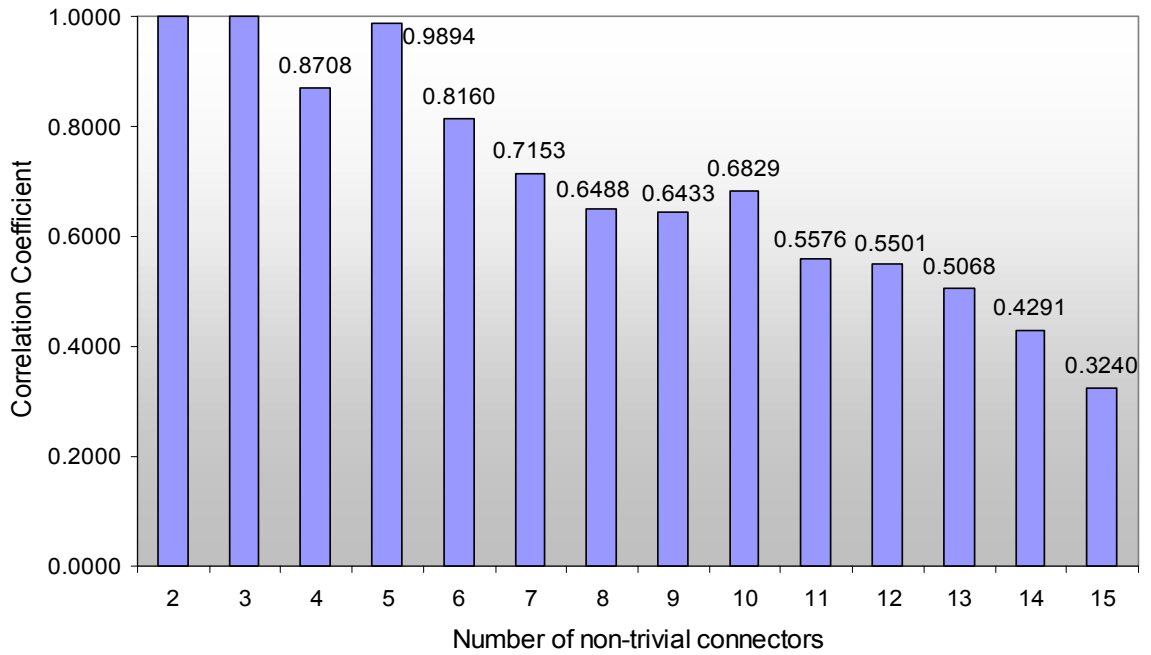


Figure 11: Variations in the correlation coefficient between analytical 1-step error propagation and experimental 1-step error propagation.

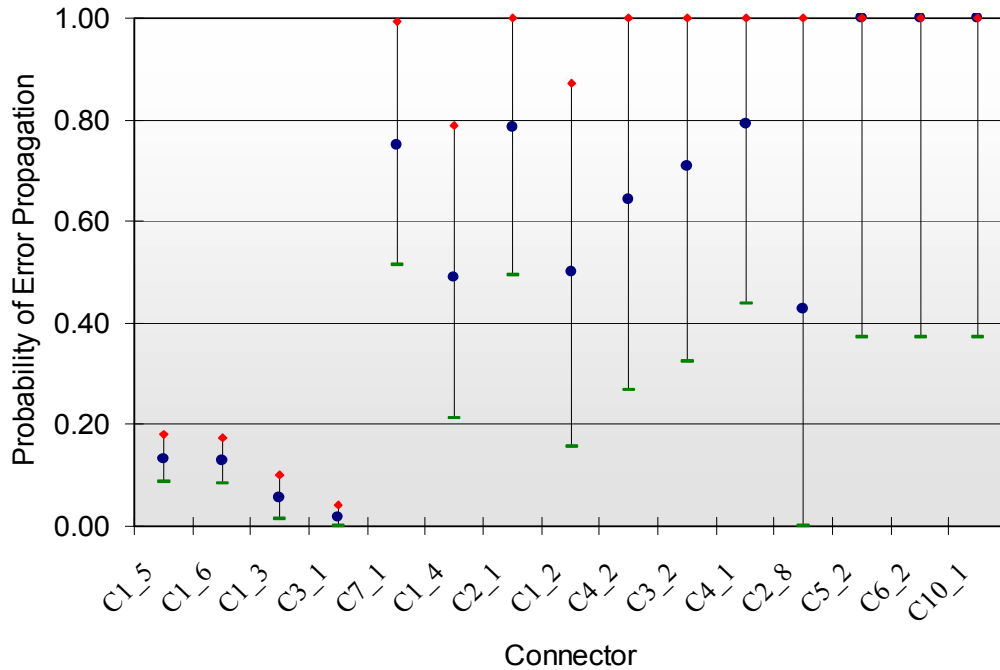


Figure 11: Confidence intervals (95 %) for experimental error propagation – mode k

Figure 12 illustrates the confidence intervals for error propagation based on the fault injection experiments we conducted on this system. The labels on the horizontal axis indicate the connectors between components of the system (Z). The presented

intervals are for a confidence level of 95%, based on the fault injection and error tracing experiments we performed. For instance, the true value for the probability of error propagation from component C1 to component C3 lies between 0.015 and 0.1, we can make this conclusion with a confidence of 95%. We notice that the width of confidence intervals (given a desired level of confidence) varies across the different connectors of the system. Connectors with high number of fault injections have smaller confidence intervals than those with low number of fault injections (see Figure 2).

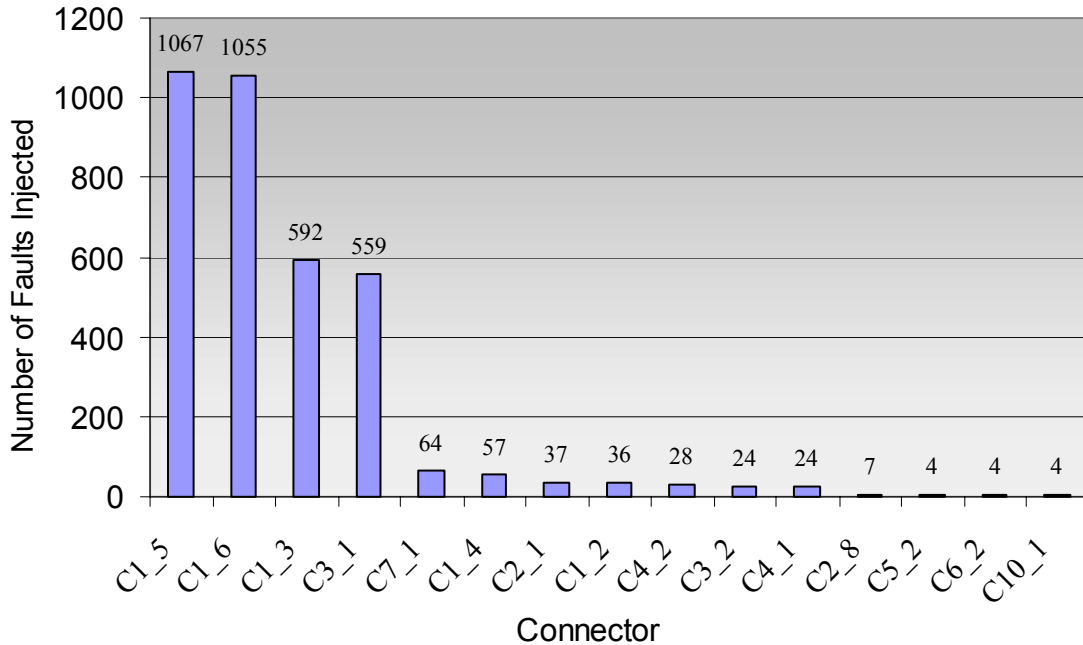


Figure 13: Number of injected faults per connector

Figure 13 shows the density of injected faults over each of the connectors between components of the system (Z) in operation mode (k). The number of faults that were injected over any connector is constrained by the capacity of the application. In other words, if we wish to further increase the number of injected faults over any connector (in order to establish higher confidence in experimental results), we would find it difficult to identify new faults that were not injected during the controlled experiment that we designed for experimental estimation of error propagation.

6.2. Correlating Cumulative Matrices

In addition to analyzing the correlation between matrices E_A and E_E (which represent the one-step unconditional error propagation probabilities, estimated analytically and experimentally), we are interested in analyzing the correlations between matrices E^*_A and E^*_E (which represent the multi-step versions of these matrices). We do expect that overall correlations between the matrices are better for the multi-step propagation, since the latter will smooth out any discrepancy in what we consider to be a single step. The results of our study, shown in Figure 14, bear us out. We find,

$$\text{Cor}(E^*_A, E^*_E) = 0.737,$$

which is an improvement over the figure of 0.628, found for single step error propagation. Also, we find that the correlation for multi-step error propagation, is

$$\text{Cor} = 0.460,$$

What is also an improvement over the same value for single step. Fortunately, the multi-step error propagation is more useful in practice, because more meaningful. Hence our analytical formula can be used to predict multi-step error propagation probabilities throughout an architecture with a significant positive correlation, at least in this sample case study ---all the while using nothing more than the UML-RT description of the system.

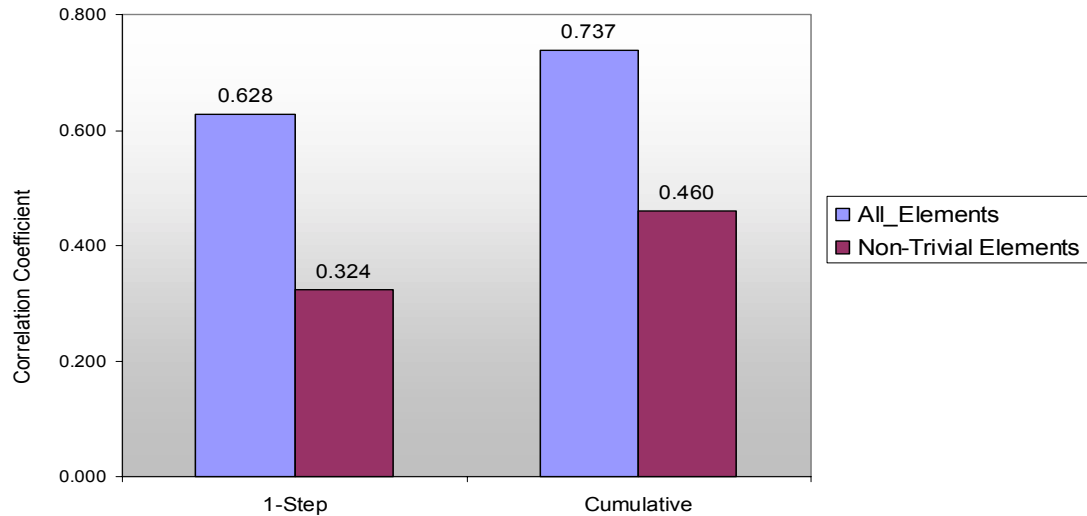


Figure 12: Correlation between analytically estimated error propagation and experimentally computed error propagation

7. Conclusion

In this paper, we have derived an analytical formula, supported by an automated tool, to estimate the probability of error propagation between components in a software architecture. Further, we have validated our proposed formula by means of a fault injection experiment, applied on a large command and control system, and found a fairly strong correlation between our analytical estimates and our experimental observations. Given that our analytical formula is based on a UML-RT description, and uses exclusively information that is typically available at an architectural level, we submit that our result can be used to estimate the error propagation behavior of an architecture, at a time when relatively little is known about the actual execution of products that instantiate the architecture. In addition to providing the basic conditional probability of error propagation over a given connector (conditioned on the activation of the connector), we have also provided analytical formulas for unconditional error propagation (which incorporate the probability of connector activation) as well as the cumulative error propagation probability, which quantifies

the probability that an error propagates from one component to another in an arbitrary number of connector activations. Finally we have briefly explored ways for a software architect to analyze and use the information provided by our analytical estimates.

Among the venues of further research we are considering to carry out more experiments on validating our analytical formulas, as well as further investigating candidate definitions of the transition probability matrix, which allows us to turn the conditional error propagation probabilities into unconditional cumulative probabilities.

This work is part of a wider effort to analyze and quantify attributes of software architecture. Attributes of interest include, among others, *change propagation probabilities* (likelihood that a change in one component affects other components), and *requirements propagation probabilities* (likelihood that a change in the requirements of one component mandate a change in the requirements of other components).

8. Bibliography

H. Ammar, S. M. Yacoub, A. Ibrahim, "A Fault Model for Fault Injection Analysis of Dynamic UML Specifications," *International Symposium on Software Reliability Engineering*, IEEE Computer Society, November 2001.

V.R. Basili and H.D. Rombach. *The tame project: Towards improvement oriented software environments*. IEEE Transactions on Software Engineering, 14(6):758--773, June 1988.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 1998.

J.M. Bieman and B.K. Kang. *Measuring design level cohesion*. IEEE Transactions on Software Engineering, 24(2):111--124, February 1998.

L.C. Briand, S. Morasca, and V.R. Basili. *Property based software engineering measurement*. IEEE Transactions on Software Engineering, 22(1), January 1996.

N. Chapin. *Entropy metric for systems with cots software*. Proceedings, Metrics 2002, Ottawa, Ont, Canada, 2002.

N. Fenton. *Software measurement: A necessary scientific basis*. IEEE Transactions on Software Engineering, 20(3):199--206, March 1994.

M.H. Halstead. *Elements of Software Science*. North Holland, Amsterdam, 1977.

W.~Harrison. *An entropy-based measure of software complexity.* IEEE Transactions on Software Engineering, 18(11):1025--1029, November 1992.

M. Hiller, A. Jhumka, and N. Suri, "An Approach for Analyzing the Propagation of Data Errors in Software," *Dependable Systems and Networks*, pp. 161 -170, 2001.

T.M. Khoshgoftaar, K.~Ganesan, E.B. Allen, F.D. Ross, R.~Munikoti, N.~Goel, and A.~Nandi. *Predicting fault prone modules with case-based reasoning.* In 8th International Symposium on Software Reliability Engineering, Albuquerque, NM, November 1997.

D.~Luckham, J.~Kenney, L.~Augustin, J.~Vera, D.~Bryan, and W.~Mann. *Specification and analysis of system architecture using rapide.* IEEE Transactions on Software Engineering, Special Issue on Software Architecture, 21(4), April 1995.

D.~Luckham, J.~Kenney, L.~Augustin, J.~Vera, D.~Bryan, and W.~Mann. *Specification and analysis of system architecture using rapide.* IEEE Transactions on Software Engineering, 21(4), April 1995.

D.~C. Luckham, J.~Vera, and S.~Meldal. *Three concepts of system architecture.* Technical Report Technical Report CSL-TR-95-674, Stanford University, July 1995.

T.J. McCabe. *A complexity measure.* IEEE Transactions on Software Engineering, SE-2(4):308--320, 1976.

C Michael C. C., and Jones R. C., "On the Uniformity of Error Propagation in Software," *Proc. of the 12th Annual Conference on Computer Assurance (COMPASS'97)*, pp. 68-76, 1997.

D. M. Nassar, W. A. Rabie, M. Shereshevsky, N. Gradetsky, H.H. Ammar, Bo Yu, S. Bogazzi, and A. Mili. *Estimating Error Propagation Probabilities in Software Architectures.* Technical Report, College of Computer Science, New Jersey Institute of Technology. <http://www.ccs.njit.edu/swarch/ep.pdf>

Rational Rose Realtime, Rational Software Corporation, <http://www.rational.com>.

A. Renyi. "On Measures of Entropy and Information", in N. Neyman (editor), *Proceedings of the Fourth Symposium on Mathematics, Statistics, and Probability.* San Francisco, CA 1961, pages 547-561.

C.~Shannon. *A mathematical theory of communication.* Bell Syst. Tech. Journal, 27:379--423, 623--656, 1948.

M.~Shaw. *Architectural issues in software reuse: It's not just the functionality, it's the packaging.* Proceedings, Symposium on Software Reusability}, Seattle, WA, April 1995. Association for Computing Machinery.

M.~Shereshevsky, H.~Ammari, N.~Gradetsky, A.~Mili, and H.~Ammar. *Information theoretic metrics for software architectures.* In Proceedings, COMPSAC 2001: Computer Software and Applications, Chicago, IL, 2001.

J. Voas, "Error propagation analysis for COTS system," *Journal of Computing & Control Engineering*, vol. 8, no. 6, pp. 269 –272, Dec. 1997.

J. Voas, F. Charron, and L. Beltracchi, "Error Propagation Analysis Studies in a Nuclear Research Code," *Aerospace Conference*, 1998 IEEE, vol. 4 , pp. 115 -121, 1998.

E.~Yourdon and L.L. Constantine. *Structured design: fundamentals of a discipline of computer program and systems design.* Prentice Hall, Englewood Cliffs, N.J., 1979.

Appendices

A. Analytical Formula for Error Propagation

A.1. The General Case

Consider two architectural components, say A and B communicating through some sort of connector. Every act of A -to- B communication consists of passing from A to B a message selected from certain vocabulary $V_{A \rightarrow B}$ (for example, if the connector is realized in the form of method invocations, then set $V_{A \rightarrow B}$ can be thought of as the range of values of the "aggregate parameters" of all methods of B through which A may transmit information to B). Let S_A be the set of states of module A , and S_B be the set of states of module B . When A transmits to B a message $v \in V_{A \rightarrow B}$ it, in general, results in B changing its state, thus defining the *state transition mapping* $F : S_B \times V_{A \rightarrow B} \rightarrow S_B$. Assuming that the system in question operates deterministically, $F(x, v)$ represents the state of module B after receiving message from A if the state of B before the transmission occurred was x . For the sake of convenience, we will sometimes write $F(x, v)$ as a function of one variable v only, i.e. as $F_x(v)$, assuming the pre-transmission state x to be fixed. Now suppose an error occurs in A and instead of transmitting v to B it sends v' . The error propagates into B if the (post-transmission) state of B resulting from receiving the corrupted message v' differs from the state which would result from receiving the correct message v , i.e. if $F_x(v) \neq F_x(v')$.

Definition 1: We define the error propagation as the probability that a random error in the data transmitted from A to B results in an erroneous state of B (assuming the pre-transmission state to be random as well), i.e.

$$EP(A \rightarrow B) = \text{Prob}(F_x(v) \neq F_x(v') | x \in S_B; v, v' \in V_{A \rightarrow B}, v' \neq v). \quad (\text{A1})$$

It easy to see that:

$$EP(A \rightarrow B) = \frac{\text{Prob}\{(x, v, v') \in S_B \times V_{A \rightarrow B} \times V_{A \rightarrow B} | F_x(v) \neq F_x(v')\}}{\text{Prob}\{(x, v, v') \in S_B \times V_{A \rightarrow B} \times V_{A \rightarrow B} | v \neq v'\}},$$

by virtue of the definition of conditional probability and of the fact that $F_x(v) \neq F_x(v')$ implies $v' \neq v$ for any x . The last fraction can be further rewritten in terms of probabilities of individual messages and states:

$$\frac{1 - \text{Prob}\{(x, v, v') \in S_B \times V_{A \rightarrow B} \times V_{A \rightarrow B} | F_x(v) = F_x(v')\}}{1 - \text{Prob}\{(x, v, v') \in S_B \times V_{A \rightarrow B} \times V_{A \rightarrow B} | v = v'\}} =$$

$$\frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)] P'_{A \rightarrow B}[F_x^{-1}(y)]}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v] P'_{A \rightarrow B}[v]},$$

where $F_x^{-1}(y) = \{v \in V_{A \rightarrow B} | F_x(v) = y\}$. Note that in the above formula we assume the following probability distributions.

- We assume that at every moment the state $x \in S_B$ is distributed according to probability distribution P_B on the set of states S_B .
- We assume that at every moment the correct (nominal) message $v \in V_{A \rightarrow B}$ is distributed according to probability distribution $P_{A \rightarrow B}$ on $V_{A \rightarrow B}$ (the set of messages A may transmit to B).
- We assume that at every moment the corrupted (actual) message $v' \in V_{A \rightarrow B}$ is statistically independent of the nominal message v and is distributed according to probability distribution $P'_{A \rightarrow B}$ on $V_{A \rightarrow B}$. In general $P_{A \rightarrow B} \neq P'_{A \rightarrow B}$.

Thus, we have

$$EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)] P'_{A \rightarrow B}[F_x^{-1}(y)]}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v] P'_{A \rightarrow B}[v]}.$$

(A2)

A.2. The Case of Identical Distributions

Consider formula (A2) under the assumption that the probability distribution $P'_{A \rightarrow B}$ of a corrupted message is the same as distribution $P_{A \rightarrow B}$ of the nominal message: $P'_{A \rightarrow B}(v) = P_{A \rightarrow B}(v)$ for every message $v \in V_{A \rightarrow B}$.

It is easy to see that in this case formula (A2) assumes the following form:

$$EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \rightarrow B}[F_x^{-1}(y)]^2}{1 - \sum_{v \in V_{A \rightarrow B}} P_{A \rightarrow B}[v]^2}. \quad (\text{A3})$$

A.3. The Case of Uniformly Distributed Error

Now let us consider a partial case when the probability distribution $P'_{A \rightarrow B}$ of a corrupted message is uniform, i.e. that (any) correct message is equally likely to be corrupted into *any* other message: $P'_{A \rightarrow B}(v') = M^{-1}$, where $M = |V_{A \rightarrow B}|$ is the number of possible messages from A to B .

It is easy to see that formula (A2) in this case can be rewritten as follows:

$$EP(A \rightarrow B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} |F_x^{-1}(y)| M^{-1} P_{A \rightarrow B}[F_x^{-1}(y)]}{1 - \sum_{v \in V_{A \rightarrow B}} M^{-1} P_{A \rightarrow B}[v]} = \frac{1 - M^{-1} \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} \tau(x, y) P_{A \rightarrow B}[F_x^{-1}(y)]}{1 - M^{-1}} \quad (\text{A3})$$

By $\tau(x, y)$ we denoted $|F_x^{-1}(y)|$, which is the number of messages from A that make the component B change its state from x to y , i.e. the number of arcs from node x to node y in the transition graph of component B .

Formula (3) gives the expression for the error propagation probability when we assume the uniform distribution of erroneous messages. This is the case in our experiments described in Section 5.

B. Derivation of the Matrix of Transition Probabilities

In this section, we illustrate how to compute the different quantities involved in the EP formula from a given architectural specifications of a software system. Our discussion focuses on UML-RT specifications [UML-RT], however, the approach is independent of the specification language used to describe a software architecture.

Using the UML-RT notation, an architectural component is a *capsule* and an inter-component connector is a *connector* that connects a pair of *ports* (a *base port* and a *conjugate port*) which use a common communication *protocol* in two capsules (see Figure3 for illustration).

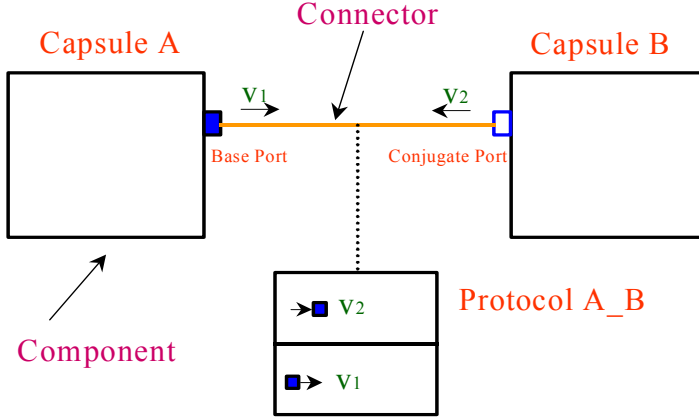


Figure B.3: Identifying components and connectors in UML-RT.

The EP formula that we will use through out the rest of this paper assumes uniform probability distribution for message corruptions. For example, if the set of messages from component A to component B consists of 3 messages: $\{v_1, v_2, v_3\}$, (see **Figure 2**), and message v_1 gets corrupted, then it is equally likely that either one of v_2 or v_3 will be sent (from A to B) instead of v_1 . This assumption leads to a less complicated design of the fault injection experiments that we conduct to validate our analytical framework.

The formula that governs EP under uniform message corruptions is given by:

$$EP(A \rightarrow B) = \frac{1 - M^{-1} \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} \tau(x, y) P_{A \rightarrow B}[F_x^{-1}(y)]}{1 - M^{-1}},$$

Where,

- M : is the number of unique messages sent over the connector from A to B,
- $\tau(x, y)$: is the number of unique messages that causes component B to transit from state x to state y ,
- $P_B(x)$: is the probability of observing component B in state x , and
- $P_{A \rightarrow B}[F_x^{-1}(y)]$: is the probability of transmission of a message (from A to B) that causes component B to transit from state x to state y .

The number of unique messages sent from A to B, (M), is a static quantity, which does not depend on how the system is used. So, it is directly counted from *class diagrams* and *protocol specifications*. Using class diagrams, we first identify connectors, then for each of the identified connectors, we search in the specifications of the protocol, used over that connector, for messages that originate from the connector port in A and terminate on the connector port in B ($V_{A \rightarrow B}$ in **Figure 2**). M is the number of messages in the set of messages $V_{A \rightarrow B}$. For the simple system shown in **Figure 2**, $M = 3$.

The number of unique messages sent from A to B that will cause component B to transit from state x to state y , ($\tau(x,y)$), is also a static quantity. $\tau(x,y)$ is directly counted from the *state diagrams* of component B. For each pair of states (x,y) we determine how many of the messages ($V_{A \rightarrow B}$) cause B to transit from state x to state y . For the simple system shown in **Figure**, $\tau(S_1, S_2) = 1$, and $\tau(S_1, S_3) = 2$.

The probability of observing component B in state x , $P_B(x)$, is a dynamic quantity that depends on how the system is used. Hence, we employ all available *sequence diagrams* and the *state diagrams* (of component B) in computing this quantity using a component/state traversal technique. Starting from the top most sequence diagram that describes the usage of the system, we traverse all paths in which component B interacts with the components of the system (sending or receiving messages), these interactions also count for self-loops of B itself. We also traverse the paths in which B interacts with the surrounding environment (responding to stimuli or producing output), surrounding environment means all external entities to the software system. We count all incidents in which B is in action (N_B), out of this number of incidences, there are N_{Bx} incidences corresponding to B being in action while in state x . The probability of observing component B in state x is thus estimated to be (N_{Bx} / N_B).

The probability of transmission of a message (from A to B) that causes component B to transit from state x to state y , $P_{A \rightarrow B}[F_x^{-1}(y)]$, is also a dynamic quantity. We recognize two necessary steps to compute $P_{A \rightarrow B}[F_x^{-1}(y)]$: The first step is to determine the set of messages that cause component B to transit from state x to state y , this step is part of the static analysis that we carry out in order to compute ($\tau(x,y)$) as we discussed earlier. Thus, the first step requires no dynamic analysis. The second step counts for the pure dynamic part of $P_{A \rightarrow B}[F_x^{-1}(y)]$, that is determining the probability distribution of messages sent from A to B. To estimate the probability distribution $P[V_{A \rightarrow B}]$, we again employ all available *sequence diagrams* to trace incidences of message transmissions from A to B. Assume that in a given use of the system (specified in sequence diagrams), the number of times message v_i is passed from A to B is Nv_i , then $P_{A \rightarrow B}[v_i] = (Nv_i / (\sum_{<j>} Nv_j))$, where $j = 1..M$ and M is the number of unique messages sent over the connector from A to B as we discussed earlier. For example, consider the simple architecture shown in **Figure 2**, let the probability distribution for the messages ($V_{A \rightarrow B}$) be given by $P[V_{A \rightarrow B}] = \{p_{v1}, p_{v2}, p_{v3}\}$, then $P_{A \rightarrow B}[F_{S1}^{-1}(S3)] = p_{v2} + p_{v3}$, while $P_{A \rightarrow B}[F_{S1}^{-1}(S2)] = p_{v1}$.

An important quantity that we also compute from the dynamic specifications of a given system is the transition probability matrix **T**. By traversing all available *sequence diagrams* we count the number of export messages from a component (say A) to all other components, let this be presented as $\{n_{iA} : i = 1, \dots, N \wedge i \neq i(A)\}$, where N is the number of components in the system and $i(A)$ is the index of component A. We then define $T(A,B) = (n_{i(B)A} / (\sum_{<i>} n_{iA}))$, where $i = 1 \dots N$. All entries of the **T** matrix are populated using this relation. The transition probability matrix **T** is used in obtaining the unconditional error propagation matrix (**E**) from the conditional error propagation matrix (**EP**).