Assessing Inter-Modular Error Propagation In Distributed Software*

Arshad Jhumka, Martin Hiller and Neeraj Suri
Dept. of Computer Engineering
Chalmers University,
Göteborg, 412 96, Sweden
e-mail: {*arshad,hiller,suri*}*@ce.chalmers.se*

Abstract

With the functionality of most embedded systems based on software (SW), interactions amongst SW modules arise, resulting in error propagation across SW them. During SW development, it would be helpful to have a framework that clearly demonstrates the error propagation and containment capabilities of the different SW components. In this paper, we assess the impact of inter-modular error propagation. Adopting a white-box SW approach, we make the following contributions: (a) we study and characterize the error propagation process and derive a set of metrics that quantitatively represents the inter-modular SW interactions, (b) we use a real embedded target system used in an aircraft arrestment system to perform fault-injection experiments to obtain experimental values for the metrics proposed, (c) we show how the set of metrics can be used to obtain the required analytical framework for error propagation analysis. We find that the derived analytical framework establishes a very close correlation between the analytical and experimental values obtained. The intent is to use this framework to be able to systematically develop SW such that inter-modular error propagation is reduced by design.

1 Motivation and Approach

With SW driving dependable system designs, systems are invariably built around sets of cooperating SW modules (tasks) that execute under specified resource and timing constraints. Traditionally, at the hardware (HW) level, classical techniques have involved many variations of replication of computing nodes, and consequently, replicating the software resident on them. With SW modules sharing common resources, intricate interactions between SW modules allow propagation of SW level data errors, which need to be corrected as early as possible to ensure correct delivery of services. Hence, the need to define a framework that demonstrates the error containment capabilities of SW modules.

To constrain error propagation in SW, all inter-modular interactions are desired to be effected in a prescribed way, such that the overall distributed SW is composed of Error Containment Modules (ECMs) – analogous to Fault Containment Regions (FCRs) in HW – at different abstraction levels. To overcome a transient fault at the node level, different techniques, such as replication or Error Detection and Recovery Mechanisms (EDMs and ERMs) may be used. However, knowing which vulnerable modules to replicate or equip with EDMs and ERMs is of primary importance. Thus, quantitative information is needed to help determine these parameters. Intuitively, SW modules that allow errors to propagate are candidates for replication or to be equipped with EDMs and ERMs, such that the error(s) can be contained and/or corrected.

To assess the impact of error propagation, we adopt a white-box[1] perspective of SW and define a basic set of metrics, namely *inter-modular influence* and *separation*, introduced in [14]. However, these metrics do not, by themselves, aid identification of vulnerable modules in the system. Thus, we augment the above set with complementary metrics, namely *Error Transmission Probability* and *Error Transparency* that help characterize error propagation properties of SW modules and aid identification of vulnerable modules. *Influence* is defined as the probability of a module directly influencing another module, i.e., when *no* other module is considered while *separation* is defined as the probability of a module *not* influencing another one

*Supported in part by Saab endowment, TFR, NSF Career CCR 9896321, Volvo Research Foundation (FFP-DCN) & NUTEK (1P21-97-4745).

[1]White-box SW implies that the SW module properties and structure are fully known and modifiable.

when *all* other modules are considered. We use the augmented metrics set to help in building reliable, distributed SW, as our aim is to be able to systematically design SW modules such that inter-modular error propagation is reduced by design.

The influence of a module M_1 on another module M_2 can be both spatial and temporal. In this paper, we initially limit ourselves to the study of spatial influence. To our knowledge, there is a dearth of experimental measurements/results showing how to evaluate such conceptual metrics. Our paper makes the following contributions: (a) we analyze the error propagation process and we derive a general expression for evaluating influence value and associated metrics, (b) we perform Fault Injection (FI) experiments on a real target system to experimentally measure these metrics, (c) we generate the required analytical framework by applying the derived expression for the target system and validate the framework developed, (d) we then use these metrics to determine which vulnerable SW modules to be replicated or equipped with EDMs and ERMs.

Paper Organization: The paper is organized as follows. The system and software model is presented in Section 2, and related work is discussed in Section 3. The analytical framework for evaluating the different metrics is developed in Section 4. Section 5 introduces the target system and provides detailed information of the experimental setup. The results obtained and conclusions derived from them in developing the SW design framework are discussed in Section 6 while Section 7 summarizes the paper and comments about future work.

2 System, Software, & Fault Models

System and SW Models: We utilize a generic SW and system model for this work and focus on distributed systems though the methodology is applicable to uniprocessor applications also. Further, the underlying software model consists of layers of discrete ECMs. Each abstraction level specifies a predefined class of faults that may be handled at that particular level. The SW levels that we make use in this paper are the *process*, *task* and *procedure* levels, introduced in [14], with the process level being the highest abstraction level and the procedure level the lowest, as shown in Fig. 1.

ECMs at different layers communicate in different ways. For example, procedures will interact through parameter-passing, including return values and global variables whereas tasks/processes may communicate through shared memory or by message passing. ECMs at a specific layer interact and cooperate with each other to provide services.

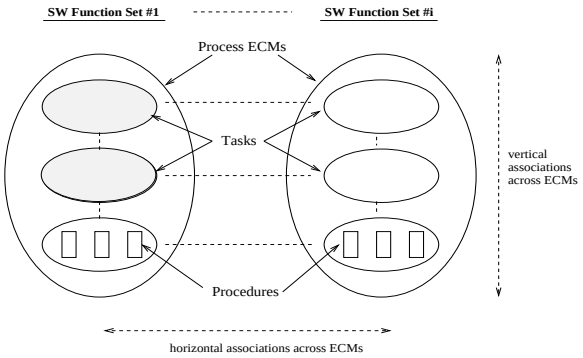

Figure 1: Software (ECMs) Layers Across (Replicated) SW Modules

We also consider each ECM as a whitebox, i.e., we have access to the internal state of the ECM.

Fault Model: The errors assumed in this paper are transient data errors, i.e., transient erroneous values in the signals or internal variables of the system, for example due to bit-flips in memory areas. In this paper, we assume that only *one* transient error can occur at any time at the input of an ECM, giving rise to possibly multiple errors in other modules. If an ECM has multiple input signals, then we assume that each signal has equal probability of being in error.

It should be noted that data errors have varied connotations when considered at the different SW abstraction levels of Fig. 1. Given the different communication paradigms at different levels, an error at procedure level indicates errors in the parameters or global variables while an error at process/task level may imply error in the messages.

3 Related Work

Fault/error injections have mostly been conducted to validate error detection and error recovery mechanisms of a system, where the main aim was to quantify the effectiveness (coverage, latency etc) of the different proposed mechanisms [1]. A comprehensive discussion on Fault Injection (FI) appears in [9]. In [3], FI experiments were conducted to characterize large system failures, whereby concepts such as potential hazards and failure acceleration were introduced. In [6], FI experiments were conducted to obtain pertinent information on possible locations for EDMs and ERMs. Our approach differs from that in [6] in that we consider SW as whiteboxes as opposed to blackboxes, where only the I/O and functional capabilities of SW modules are known. We envision the approach proposed in [6] to complement ours as a postpass. Also, there was no distinction in [6]

amongst the various propagation media, while in this paper, one can determine the role of each communication medium for error propagation.

In an early work on error propagation, [12] proposed a model based on error propagation time and proceeded to experimentally evaluate the parameters of the model, i.e., the distribution function of the error propagation time between modules. More recently, there has been more work on error propagation analysis with different goals. [10] proposed an approach where static software metrics were used to determine software modules that do not propagate errors. Some work on error propagation analysis has dealt with finding locations for Executable Assertions (EAs) [5] based on the fact that testing is unlikely to uncover them using sensitivity analysis or static analysis of SW properties [16, 17].

On EDMs' effectiveness in detecting errors, [13] presented an approach to determine the optimal combination of EDMs in HW based on FI experiments. It first determined the effectiveness of single EDMs and then evaluated the effectiveness of subsets of these EDMs. The problem of fault containment has been studied in [2, 4] using wrappers. Wrappers act as filters whereby they filter out inputs which will cause system failure and filter erroneous outputs before passing the values to succeeding modules. They have mostly been used when using COTS (Commercial Off The Shelf) components to build dependable systems.

Service availability is increased through replication. [8] proposes an approach where trade-offs between fault-tolerance and schedulability is achieved through an objective function termed *probability of no dynamic failure*. A higher replication degree of a module implies higher fault-tolerance but it may impact the schedulability. The decision to replicate a module depends of whether there is enough time for it to recover from a fault to meet a deadline. In this paper, the modules identified as candidates for replication are those SW modules that allow errors to propagate easily, hence exerting a high influence on a target module. We envision our method of module selection proposed in this paper to be an alternative and/or a complement to that proposed in [8].

In the next section, we will derive general expressions that can be used to provide the relevant information on error propagation.

4 Characterizing Error Propagation Across SW Modules

A typical SW system consists of a set of cooperating ECMs (modules), as shown in Fig. 2. Each ECM (at any level) contains a number of inputs and outputs and an error in any of the input signals of a source ECM can potentially corrupt any of its output signals. To assess the vulnerability of an ECM, the probability of the error propagating beyond the boundary of the source ECM should be evaluated, minimized (possibly nullified), as well as any effect the propagating errors may have on the state of the target ECM.

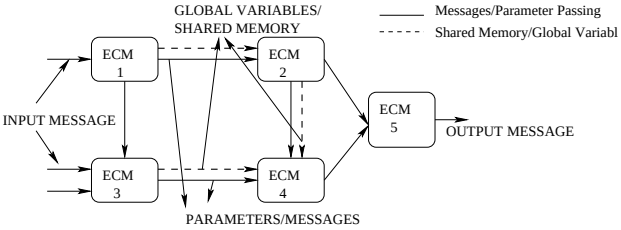

Figure 2: Software System with Communicating ECMs and Possible Media for Error Propagation.

4.1 Error Propagation Process

Error Transmission Probability Metric The error propagation process consists of three phases, namely (a) error occurring in a source module ECM_S, (b) error propagating out of the source module and (c) resulting error in the target module ECM_T. We denote by m the number of potential propagation media for errors between ECM_S and ECM_T, and we denote by M_j, the j^{th} propagation medium, e.g., message passing or shared memory (M_j will also denote the relevant set of messages or variables depending on the error propagation medium between ECM_S and ECM_T).

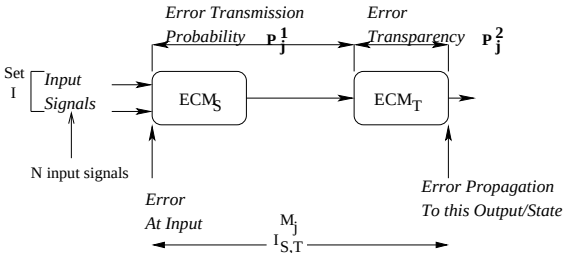

Figure 3: Error Propagating from Input of ECM_S to Output of ECM_T

In general, a (transient) error occurs at (one of) the inputs of the source module and may propagate via M_j to the inputs of the target module, where an error may occur. The probability of an error to propagate out of M_j, denoted by $P_{M_j}^{I_k}$, is as follows:

$$0 \leq P_{M_j}^{I_k} = Pr\{M_j | I_k\} \leq 1 \qquad (1)$$

where $P_{M_j}^{I_k}$ is the probability of an error in I_k, the k^{th} input of the ECM, to propagate out via M_j.

We define the *error transmission probability*, P_j^1, of a source ECM, ECM_S (see Fig. 3) as the probability of an error occurring at the input of ECM_S to propagate, via M_j, to the input set of the ECM_T. This metric is important as it provides pertinent information of how often ECM_S allows errors to propagate. An ECM with high error transmission probability is a potential candidate for replication or be equipped with EDMs and ERMs.

As each input is assumed to have equal likelihood of being in error, we can expand and simplify Eq(1) to obtain an overall expression for *Error Transmission Probability*, denoted by P_j^1 :

$$P_j^1 \;=\; (Pr\{I\}/N) \cdot \sum_{k=1}^{N} Pr\{M_j|I_k\} \qquad (2)$$

where N is the number of inputs of source ECM, and $\Pr\{I\}$ the probability of an error occurring in the input set I of the ECM. In FI experiments, $\Pr\{I\} = 1$ whereas, if field data or life testing has been performed, $\Pr\{I\}$ can be modified accordingly. Also, in case the error probability distribution is known for the inputs, the $(1/N)$ weight for each input is readjusted to reflect the distribution.

Error Transparency Metric: Once the error has propagated via M_j to the input set of ECM_T, the probability of an error occurring in the state of ECM_T is known as *error transparency*, denoted by P_j^2. This metric's importance is in generating information regarding how vulnerable ECM_T is to errors propagating from ECM_S. Target ECMs with high error transparency should be protected from errors propagating from source ECMs which is another likely candidate for replication or be equipped with EDMs and ERMs.

Influence: Characterizing the Error Propagation Process Having characterized the error propagation process from ECM_S to ECM_T, as depicted in Fig. 3, we can now evaluate the influence, $I_{S,T}^{M_j}$, of ECM_S on ECM_T, via M_j, given by

$$I_{S,T}^{M_j} \;=\; P_j^1 \cdot P_j^2 \qquad (3)$$

4.2 Evaluating the Overall Influence of a Source ECM on a Target ECM

Having obtained the influence exerted by ECM_S on ECM_T via M_j, we can now evaluate the net overall influence that can be exerted by ECM_S on ECM_T and is given by:

$$I_{S,T} \;=\; 1 \;-\; \Pi_{j=1}^{m}(1 \;-\; I_{S,T}^{M_j}) \qquad (4)$$

Eqs(1)–(4) represent the general expressions that characterize the error propagation process from ECM_S to ECM_T.

Once the overall influence measures between all interacting module pairs have been evaluated, an *influence graph* can be built to represent the influence between pairs of modules, as shown in Fig. 4. This outlines the impact of an error originating from ECM_S on ECM_T.

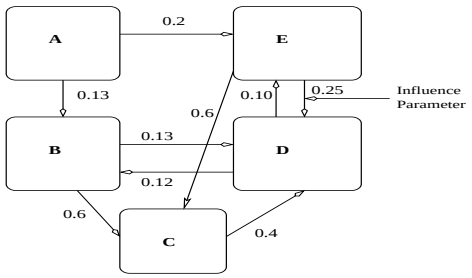

Figure 4: Influence Graph.

Note: The influence values indicated in Fig. 4 are initially assumed values and one specific contribution of this paper is to ascertain them experimentally.

4.3 Separation Measure

So far, we have focused on direct interaction (*influence*) between SW modules. We now define *separation*, a complementary measure that captures the indirect nature of interaction between ECMs. Separation is the probability of an ECM *not* influencing another ECM when *all* other ECMs are considered. This involves having transitive contributions from different ECMs.

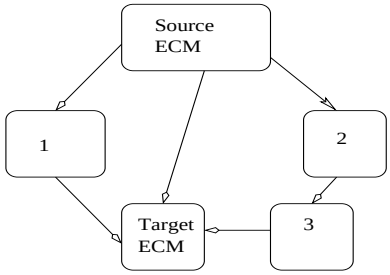

Figure 5: Separation Analysis Graph.

To measure separation, we identify all the direct and indirect links from any ECM_S to any ECM_T in Fig. 4 to build a labeled directed graph, called

a *separation analysis graph* (see Fig. 5) consisting of a source node (representing ECM_S), a single destination node (ECM_T), and other intermediate nodes representing ECMs through which ECM_S and ECM_T interact. The label on the edges represent the influence between the ECM pair.

If $I_{S,T}$ represents influence of ECM_S on ECM_T, then the total separation between ECM_S and ECM_T, denoted by $ECM_S \vdash ECM_T$, is given by

$$ECM_S \vdash ECM_T = (1 - I_{S,T}) \cdot \Pi_k (1 - I_{S,k} I_{k,T}) \cdot \Pi_{l,m} (1 - I_{S,l} I_{l,m} I_{m,T}) \dots \quad (5)$$

where $k, l, m \dots$ represent intermediate ECMs.

The value of separation from Eq(5) is always between 0 and 1. At some point, contributions from higher order terms are likely to be negligible. The separation value gives an accurate estimate of the level of interactions between ECMs as all other ECMs are considered. Once separation between all pairs of ECMs have been determined, a labeled digraph (Fig. 6) called a *separation graph*, similar to the influence graph, can be constructed so as to visualize the separation between different ECMs. Absence of an edge from the separation graph represents a separation of 1 between the ECMs (i.e., they are fully separated).

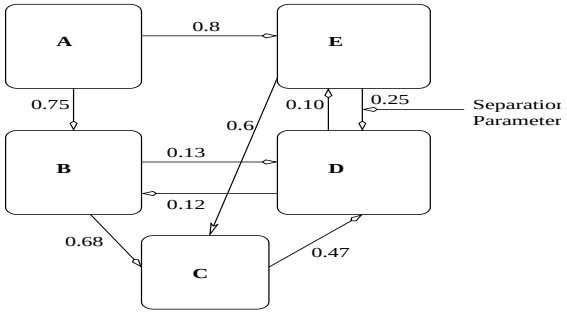

Figure 6: Separation Graph.

The separation metric is important as it helps address the issue of ECMs interacting both directly and indirectly. In such cases, the influence metric is limited and the separation metric is used. Also, in cases where the interactions between modules are bidirectional, the influence metric cannot fully capture the nature of this interaction, hence separation may be used as a complementary metric to influence.

At this stage, we have established the analytical measures of influence and separation. For illustration purposes, we have made use of assumed values, as in Fig. 4. Although it may be sufficient to assign relative values of influence to quantify the degree of interactions between SW modules, this is not adequate when designing SW that will be deployed in safety-critical systems, where it is important to assign absolute values. As such, we have performed FI experiments on an actual embedded control system SW. In the subsequent sections, we will detail our results as well as provide insights over the relevance of our findings. We will first present the target system (together with its architecture). The experimental setup will then be detailed and we will provide information pertinent to the FI experiments. The results section will detail the results of these experiments. We then provide a general discussion on the utility of the framework.

5 Validation of the Framework: An Experimental Approach To Ascertain Influence Values In Real SW

Having presented the analytical framework, we now need to validate the framework by experimentally measuring the parameters of the framework that will help in evaluating the influence and associated metrics. The validation process is performed experimentally using FI experiments, which artificially introduce faults and/or errors into the system [1, 9]. The target system used is a real embedded system for aircraft arrestment on short runways, resembling the cable-and-hook systems found on aircraft carriers, as in Fig. 7. The SW

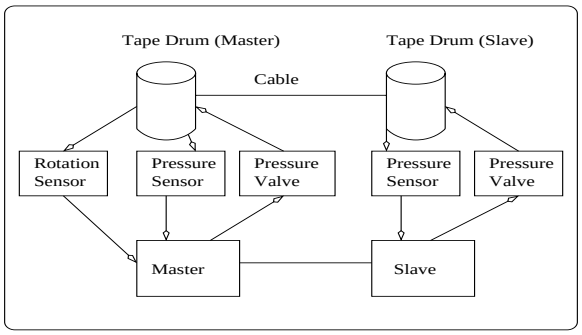

Figure 7: Aircraft Arrestment System: Target system in our study.

architecture of the target system is shown in Fig. 8 and has been implemented according to specifications found in [15]. For our study, the original software was ported so it would run on a Windows NT/2000 based desktop computer. An environment simulator handles the rotating drum and the incoming aircraft. This target system was found to be suitable for this study for three main reasons: (a) it represents a real embedded system, and validating our framework using a real system will help in assessing the relevance/utility of such

a framework in determining influence and separation metrics in real systems, (b) it presents architectural features that will allow rigorous testing of our proposed framework, and (c) we have access to the complete source code and to the relevant environment simulator.

For performing the FI experiments, we made use of PROPANE (Propagation Analysis Environment) [7], which is a tool that performs fault and error injection using SWIFI (Software Implemented Fault Injection). PROPANE can also keep track of traces of individual variables during execution. Details pertinent to the experiments will follow in subsequent sections. Next, we present the software architecture of the target system.

5.1 Target System: An Example of An Embedded Control Software

The target system is a medium-sized embedded control system used in arresting aircraft on short runways. The system is illustrated in Fig. 7. The real software comprised a master node as well as a slave node but in our setup, the slave node was removed. The force that was applied on the cable by the drum connected to the master node was also applied to the cable at the slave end, resulting in equal pressures being applied at both ends.

The software architecture of the target system is shown in Fig. 8. The software comprises six modules and a set of input/output signals to each module. The functionality of each module is summarized below.

- **CLOCK** provides a clock, *mscnt*, with one millisecond resolution, while the *ms-slot-nbr* provides the module scheduler with the current slot number for "scheduling" purposes. The system is slot-based, with seven 1 ms slots in which one or more of the other modules are executed (except the *CALC* module).
- **DIST-S** monitors the rotation sensor and provides a total count of the pulses, *pulscnt*, generated during the arrestment. Two outputs (*stopped* and *slow-speed*) indicates whether the aircraft has stopped and whether the speed is below a certain value, respectively.
- **CALC** uses the signals *mscnt*, *pulscnt*, *stopped* and *slow-speed* to calculate a set point value for the *SetValue* output at different checkpoints along the runway.
- **PRES-S** monitors the pressure sensor, measuring the current pressure on the pressure valve. This value is provided in *IsValue*.
- **V-REG** uses the signals *SetValue* and *IsValue* to control *OutValue*, the output value to the pressure valve.
- **PRES-A** uses the *OutValue* signal to set the pressure valve.

5.2 Experimental Setup and Procedures Used

The above target system was found to be architecturally suitable for this experiment since it presented modules with large fan-ins (4, in the case of *CALC*) and modules with large fan-outs (3, in the case of *DIST-S*), so that a set of messages can be targeted as input/output (set M of the framework), just as presented in the analytical framework. The FI tool used is PROPANE which allows data logging in SW. This is achieved by having high level SW traps placed in the system which, when reached during system execution, trigger the logging of data. During the experiments, PROPANE records every injection made and keeps a trace of every logged variable in the system during execution. The logged variables are, in our case, output signals from every module and all global variables in each module (since we are interested in the state of the system). In our study, errors were injected in every input signal that is shown in Fig. 8, except in *OutValue*, which is the input signal to the last module (*PRES-A*) and for which we do not then calculate an influence value, since it will only represent an influence on the external environment.

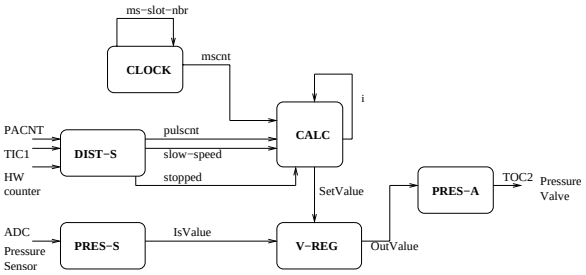

Figure 8: Software Architecture of the Target System.

For every test case, to obtain data to calculate the influence/separation metrics, we first produced a *Golden Run* execution of the system. The Golden Run execution is one where no error is injected in the system, and is used as a baseline reference execution. Then, errors are injected in the input signals of the modules and the logged variables recorded. For each experiment, only one error was injected at any one time, i.e., no multiple errors were injected, which is in line with our fault model, i.e., only one input can be in error at any one time. The input signals to every source modules are 16-bit wide, and 16 bit-flip errors were injected in each signal (a bit-flip in each position). The bit-flips were injected at 10 different time instances, resulting in 160 bit-flips for each signal. 25 test cases (5 different masses and 5 different velocities, uniformly distributed between 8,000 kg and 20,000 kg, and 40 m/s and 80 m/s respectively) were run, to subject the system to a range of realistic workloads. This resulted in 4,000 injections for each input signal. In all, there were 48,000 injections (12 input

signals). Out of these 48,000 injections, there were times when the injections were made after the aircraft was completely arrested and these cases were *not* taken into consideration in our calculations. From now on, errors injected means errors injected before the aircraft has been completely arrested, i.e., active injections.

Once the data from the FI experiments were obtained, they were compared against the Golden Run data. For each logged variable, any mismatch in its values from the injected run and the Golden Run is flagged as an error and comparison stops for that variable. This is performed for each logged variable so that, for each error injected in the system, we have a set of variables which were affected by that injected error. An analysis of the results is provided in the next section, i.e., for each of the active injections, we have information on whether any of the logged variables was affected or not by that injection.

6 Results and Interpretations

We now analyze the data by using the expressions developed in Section 4. In the target system above, the only medium ($m = 1$) through which an error can propagate is message-passing (M_1). The basic metrics to evaluate are (i) error transmission probability and (ii) error transparency. In the discussion below, we will reference M_1 by M only (M will also represent the set of messages being passed, depending on the context). Also, the conclusions presented in the coming section are dependent on the fault model. Changing the fault model may result in different values, hence different conclusions.

6.1 Results: Measured Error Transmission Probability Values

We first evaluate the *error transmission probability* for every possible source ECM (ECM_S, all modules except *PRES-A*). To obtain the error transmission probability of a source module, we first determine the set M of messages through which an error from ECM_S can reach the target ECM (only *CALC, V-REG, PRES-A*). For ease of presentation, we present one example case here. When calculating the error transmission probability of *V-REG* say, the set M is $\{OutValue\}$. The inputs to *V-REG* are *SetValue* and *IsValue*. For each of them, we determine the probability of an error at that input to affect the set M. This probability is $\Pr\{M|I_k\}$ for each k. This value is determined by taking the ratio of the number of errors in *OutValue* to the number of injections in the input signal, say *SetValue*. The same is repeated for the *IsValue* signal. Table 1 presents experimental values obtained when determining the error transmission probability of *V-REG*.

Input I_k	$Pr\{M\|I_k\}$
SetValue	0.87308
IsValue	0.93887

Table 1: Values Used for Calculating Error Transmission Probability

The error transmission probability of *V-REG* can then determined by Eq(2). Plugging in the values from Table 1 in Eq(2) and taking $\Pr\{I\}$ to be 1 for FI purposes, the error transmission probability of *V-REG* is [0.5*(0.87308+0.93887)] = 0.906. The ratio of the number of errors in M to the total number of errors injected in both *SetValue* and *IsValue* is equal to 0.906, showing that modeling error transmission probability as in Eq(2) is valid. A summary of error transmission probability of every ECM_S is given in Table 2. P_1^1 denotes Error Transmission Probability of the ECM when the first (and in this case the only) error propagation medium is considered. As discussed in Section 4.1, a high value of the error transmission probability of an ECM points to the fact that there is very little error containment built inside that ECM and is a potential candidate for replication or be equipped with EDMs and ERMs, so that an error can effectively be contained within this ECM. In this case, potential candidates are *CLOCK*, *CALC* and *V-REG*. Also, the error transmission probability of *PRES-S* is 0, indicating that this module presents some inherent error containment capability.

ECM_S	Error Transmission Probability (P_1^1)
CLOCK	0.8622
DIST-S	0.454
PRES-S	0.0
CALC	0.8957
V-REG	0.906

Table 2: Error Transmission Probability of Each Source Module

6.2 Results: Measured Error Transparency Values

After determining the error transmission probability of each source module, we determine the error transparency of every potential target module (ECM_T) along certain input signals set, i.e., of all modules except *CLOCK*, *DIST-S* and *PRES-S* along specific signals set. For example, if the influence value of the *DIST-S* on *CALC* is needed, then we need to determine the error transparency of *CALC* along the {*pulscnt, stopped, slow-speed*} input message set M. To calculate the error transparency of a module, we first determine the number of times there is an error at the inputs of the target

module. As before, we consider one specific example. From the total number of errors that affect the set M from *DIST-S*, we calculate the number of times there is an error in the state of *CALC* due to the propagated errors, i.e., the number of times at least one of the global and output variables of *CALC* is in error. The error transparency of *CALC* along the set M is then the ratio of the number of errors in the state of *CALC* to the number of errors that propagated from the output of *DIST-S* to the inputs of *CALC*.

For this specific example, 5012 errors propagated from *DIST-S* to the inputs of *CALC*. This resulted in 4977 errors in the state of the *CALC*. The error transparency of *CALC* along {*pulscnt,stopped,slow-speed*} is (4977/5012) = 0.993. In determining error transparency, we used the number of errors that propagated to the target module rather than using the number of errors injected in the target module. The error transparency values of different modules along different input signals set are presented in Table 3. P_1^2 represents the error transparency of the ECM when the first error propagation medium is considered. M has been instantiated in different cases since these ECMs (*CALC* and *V-REG*) have inputs coming from different source ECMs.

ECM_T	P_1^2
CALC (M={mscnt})	0.8229
CALC (M={pulscnt, stopped, slow-speed})	0.9930
V-REG (M={SetValue})	0.8958
V-REG (M={IsValue})	0.9381
PRES-A	0.9822

Table 3: Error Transparency of Each Target Module on Different Input Sets

The error transparency of a module gives an indication as to the vulnerability of the target module in presence of propagated errors at its input. From Table 3, the error transparency of most modules along any input set is close to 1, which means that whenever there is a propagated error at the input of these modules, then their computation will be altered as their state and/or message set is corrupted, implying that the source ECMs are potential candidates for replication or be equipped with EDMs and ERMs.

V-REG, along the {*IsValue*} set, has an error transparency of 0.9381. This value represents an approximation to the error transparency of *V-REG* along {*IsValue*} and has been evaluated using the FI data when errors have been injected in the *IsValue* signal. As stated above, error transparency has been calculcated by using the number of errors that propagated to the target module but, since in the case of *V-REG*, no error propagated from *PRES-S*, we determined the error transparency of *V-REG* along {*IsValue*} by using the number of errors injected in *IsValue*.

Whenever the error transmission probability and error transparency metrics are analyzed together, detailed conclusions can be drawn. For example, as explained in [5], EDMs such as Executable Assertions can be placed either in the source ECM or in the target ECM. A low value of error transmission probability of the source module and a high value of error transparency of the target module implies that it is preferable to place an EDM in the target module rather than in the source module. As an example, *PRES-S* is connected to *V-REG*. A look at Tables 2 and 3 reveals that the error transmission probability of *PRES-S* is 0.0 (using this particular fault model) and that the error transparency of *V-REG* along {*IsValue*} is 0.9381. Hence, from these values, placing an EDM at the output of *PRES-S* is irrelevant as this module presents some error containment capabilities. On the other hand, since the *V-REG* module has a high error transparency along {*IsValue*}, any error at this input (e.g., due to bit-flips in memory areas rather than propagated errors from *PRES-S*) should be detected, hence placing EDMs at the *IsValue* input in *V-REG*, and having ERMs in *V-REG*, will potentially limit the damage within the *V-REG* module.

6.3 Cumulative Influence Values: Analytical and Experimental Evaluations and Their Correlations

Having determined the error transmission probability and error transparency of source and target ECMs along relevant message sets respectively, we now use Eq(4) to obtain analytical values of influence of each source/target ECMs pair, as in Table 4

Source	Target	Analytical Influence ($I_{s,t}^M$)
CLOCK	CALC	$I_{CLOCK,CALC}^{\{mscnt\}} = 0.7095$
DIST-S	CALC	$I_{DIST-S,CALC}^{M} = 0.4508$
PRES-S	V-REG	$I_{PRES-S,V-REG}^{\{IsValue\}} = 0.0$
CALC	V-REG	$I_{CALC,V-REG}^{\{SetValue\}} = 0.8023$
V-REG	PRES-A	$I_{V-REG,PRES-A}^{\{OutValue\}} = 0.8899$

Table 4: Analytical Values of Influence Through Message Passing. **Note**: On row 2, $M = \{pulscnt, stopped, slow-speed\}$

Having analytically evaluated the influence value for each source/target module pair, we then experimentally evaluated the influence value for each of the same pair of source/target module.

The experimental value of influence is calculated by taking the ratio of the number of times there is an error in the state of the target module whenever an error is injected in the input of the source module. For example, 7296 errors were injected in the inputs of *V-REG* and a total of 6493 errors were detected in the state of *PRES-A*, giving an influence of 0.8899.

The values for each pair is shown in Table 4 (column 3). The first column is the identity of the source ECM, the second column represents the target ECM and the last column contains the experimentally-determined influence value between the source ECM and the target ECM.

One of the main goals of the paper is to validate this framework by showing that the analytical framework predicts accurate influence values. Table 5 compares the analytical values with the experimental values. The first two columns indicate the identities of the source and target modules. Columns 3 and 4 show the experimental and analytical influence value respectively. The error column in Table 5 represents the percentage deviation of the predicted value from the experimental value. As can be seen, the analytical framework predicts to a very high degree of accuracy the value of influence between a source module and a target module.

Source	Target	Expt	Analytical	Error
CLOCK	CALC	0.7095	0.7095	0.0
DIST-S	CALC	0.4813	0.4508	6.3
PRES-S	V-REG	0.0	0.0	0.0
CALC	V-REG	0.8023	0.8023	0.0
V-REG	PRES-A	0.8899	0.8899	0.0

Table 5: Summary of Influence Results Between Various Modules

The reason for this accuracy is explained as follows: One potential source of inaccuracy would be that messages may be dependent on each other. Our modeling strategy of having messages or memory locations as a set addresses this problem. In [6], locations for EDMs and ERMs were identified by first generating a set of input-output paths in each module. Such an approach provides insights into vulnerable paths within the module. However, it does not fully incorporate the fact that different input-output paths may be related to each other. When evaluating the error transmission probability and error transparency (hence influence), there is a need to account for the interdependency between the different input-output paths in the module. If this interdependency is not accounted for, this will result in inaccurate results. However, to circumvent this problem, [6] adopts a black-box perspective of a module, instead of a whitebox (as in this paper), thus yielding metrics that create a relative ordering based on "influence".

On the other hand, there is a significant discrepancy between the analytical and experimental influence value of *DIST-S* on *CALC*. This can be explained by the fact that, on top of *DIST-S* spatially influencing the *CALC* module through {*pulscnt, stopped, slow-speed*}, the *DIST-S* module indirectly influences the *CLOCK* module in the time domain, i.e., *DIST-S* causes the *mscnt* and *ms-slot-nbr* signals from the *CLOCK* module to exhibit a different profile from the Golden Run (specifically, the signal *mscnt* is incremented more than in the error-free case). This indirect influence of *DIST-S* on *CLOCK* in turn causes *CLOCK* to influence *CALC* through *mscnt*. Hence, there is higher number of errors in the state of *CALC* than expected due to this temporal influence. Also, the influence of *CLOCK* on *CALC* is temporal and since temporal influence is detected as data errors, there is exact agreement between the analytical and experimental values of influence of *CLOCK* on *CALC*. However, the impact of temporal influence is more clearly illustrated in the influence value of *DIST-S* on *CALC*. Temporal influence is not the focus of the current paper, but is a factor in our continuing work to obtain a comprehensive framework consolidating spatial and temporal influences.

Building ECMs implies that influence values should be decreased to under a certain threshold (which may be application-dependent). Replication or use of EDMs and ERMs help achieve that. Also, by using the influence value, one can also determine if the system has been built too defensively. For example, if an EDM was placed at the output of *PRES-S*, this would represent only redundant code. Overall, these metrics help in developing reliable SW such that error propagation is minimized by design.

6.4 Discussion

We have shown that the framework generates accurate influence results, and also that the different metrics provide insightful information pertinent to vulnerabilities in the system, which is important when developing reliable distributed SW. However, in this section, we address some limitations of our approach and provide some general discussions on the different results.

In this paper, we did not address bidirectional inter-modular interactions such as feedback loops. However, we believe that the error transmission probability, error transparency and the influence metrics can be measured following the approach presented in this paper. On the other hand, the

separation metric may be more difficult to measure and analyze, since it includes transitive contributions (i.e., captures the feedback interactions).

Also, the influence and separation metric may not readily allow piecewise composition of the different SW modules under the current fault model. However, as future work, we will be enhancing this framework to deal with multiple errors. Then, the metrics can be used as a basis for composability.

Furthermore, we motivated the use of the *separation* metric. However, in our target system, we do not have direct and indirect interactions between the different modules and hence evaluation of separation between modules becomes trivial, i.e., specifically, it is given by 1 - (product of influences). If a target system presents more complex interactions between the component modules, then separation will provide more accurate estimations of the level of interactions and can be evaluated using Eq(5).

7 Summary and Future Work

Utilizing a whitebox knowledge of SW, the paper targeted reduction of inter-modular error propagation in SW by design. The main aims of this paper were to be able to calculate the influence value between a source module and a target module in the system, and to use these values to determine candidate module for replication or equip with EDMs and ERMs. We first analyzed the error propagation process, which provided us with the relevant information needed when performing FI experiments, i.e., which metrics to evaluate to allow calculation of influence. In our case, the metrics were error transmission probability and error transparency along a certain input set. We showed that the analytical framework can predict to a very high degree of accuracy the influence value between a pair of modules. Using this influence value (and associated metrics such as error transparency or error transmission probability), we were then able achieve our goal. Furthermore, the influence values provided insights into whether a system has been built too defensively.

We also analyzed only the impact of spatial influence of one module on another. As stated earlier, a module can also interfere with another in the temporal domain. A natural extension of our work will then be to determine the timing influence of one module on another, which is especially important in real-time systems. Such an example was provided in case of *DIST-S* influencing *CALC*. Additional work will include incorporating the timing influence into the above framework so that both spatial and timing influence can be reasoned about in a single, well-defined framework. Also, we plan to apply our framework to other target systems to further validate its applicability.

References

[1] J. Arlat et al, "Fault Injection for Dependability Evaluation of Fault Tolerant Computing Systems", *Proc FTCS-19*, pp. 348–355, 1989.

[2] F. Salles, M. Rodriguez, J.C. Fabre, J. Arlat, "MetaKernels and Fault Containment Wrappers", *Proc FTCS-29*, pp. 22–29, 1999.

[3] R. Chillarege, N. Bowen,"Understanding Large System Failures - A Fault Injection Experiment," *Proc FTCS-19*, pp. 356–363, 1989.

[4] A.K. Ghosh et al,"Wrapping Windows NT Software For Robustness",*Proc FTCS-29*, pp. 344–347, 1999.

[5] M. Hiller "Executable Assertions For Detecting Data Errors in Embedded Control System", *Proc DSN 2000*, pp. 24–33

[6] M. Hiller, A. Jhumka, N. Suri,"An Approach To Analysing The Propagation Of Data Errors In Software", *To Appear in Proceedings DSN 2001*

[7] M. Hiller "A Tool for Examining the Behavior of Faults and Errors in Software", *Technical Report 00-19*, Dept of Comp Engg, Chalmers University, Sweden.

[8] C.J. Hou, K.G. Shin, "Replication and Allocation of Task Modules in Distributed Real-Time Systems", FTCS'94

[9] R. K. Iyer, "Fault-Tolerant Computer System Design", D.K. Pradhan editor, Chapter 5, pp. 282–387.

[10] T.M. Khoshgoftaar et al, "Identifying Modules Which Do Not Propagate Errors," *IEEE Symposium on Application-Specific Systems and Software Engg and Technology(ASSET)*, pp. 185–193, 1999.

[11] J.C. Laprie, "Dependability: Concepts and Terminology," *Dependable Computing and Fault-Tolerant System, Vol. 5*, Springer-Verlag, 1992.

[12] K.G. Shin, T.H. Lin, "Modeling and Measurement of Error Propagation in a Multi-Module Computing System," *IEEE Trans. on Computers.*, vol. 37, No. 9, pp. 1053–1066, Sept 1988

[13] A. Steininger, C. Scherrer, "On Finding an Optimal Combination of Error Detection Mechanisms Based on Results of Fault Injection Experiments," *Proceedings FTCS'27*, pp. 238–247, 1997.

[14] N. Suri, S. Ghosh, T. Marlowe,"A Framework for Dependability Driven Software Integration", *Proceedings ICDCS'98*, pp. 405–416, 1998

[15] US Air Force - 99, "Military Specification: Aircraft Arresting System BAK-12A/E32A; Portable, Rotary Friction", *MIL-A-38202C, Notice 1, US Department Of Defence, September 2, 1986*

[16] J.M. Voas, K.W. Miller "Putting Assertions in Their Place," *Proceedings ISSRE'94*, pp. 152–157, 1994.

[17] J.M. Voas, "PIE: A Dynamic Failure-Based Technique," *IEEE Trans on Software Engg*, vol. 18, No. 8, pp. 717–727, August 1992.