Andrey Morozov

Dual-graph Model for Error Propagation Analysis of Mechatronic Systems

Beiträge aus der Automatisierungstechnik

Andrey Morozov

Dual-graph Model for Error Propagation Analysis of Mechatronic Systems

VOGT

Dresden 2012

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Bibliographic Information published by the Deutsche Bibliothek
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibiograpic data is available in the internet at http://dnb.ddb.de.

Zugl.: Dresden, Techn. Univ., Diss., 2012

Die vorliegende Arbeit stimmt mit dem Original der Dissertation „Dual-graph Model for Error Propagation Analysis of Mechatronic Systems“ von Andrey Morozov überein.

Gesetzt vom Autor

ISBN 978-3-938860-56-4

Jörg Vogt Verlag
Niederwaldstr. 36
01277 Dresden
Germany

Phone: +49-(0)351-31403921
Telefax: +49-(0)351-31403918
e-mail: info@vogtverlag.de
Internet : www.vogtverlag.de

Technische Universität Dresden

Dual-graph Model for Error Propagation Analysis of Mechatronic Systems

Andrey Morozov

von der Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs

(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender:	Prof. Dr.-Ing. habil. Dipl.-Math.Röbenack		
Gutachter:	Prof. Dr. techn. Janschek	Tag der Einreichung	02.03.2012
	Prof. Dr. Fetzer	Tag der Verteidigung	04.07.2012

This work has been supported by the
Erasmus Mundus External Co-operation Window Programme
of the European Union.

Abstract

Error propagation analysis is an important part of a system development process. This thesis addresses a probabilistic description of the spreading of data errors through a mechatronic system. An error propagation model for these types of systems must use a high abstraction layer that allows the proper mapping of the mutual interaction of heterogeneous system components such as software, hardware, and physical parts.

A literature overview reveals the most appropriate error propagation model that is based on Markovian representation of control flow. However, despite the strong probabilistic background, this model has a significant disadvantage. It implies that data errors always propagate through the control flow. This assumption limits model application to the systems, in which components can be triggered in arbitrary order with non-sequential data flow.

A motivational example, discussed in this thesis, shows that control and data flows must be considered separately for an accurate description of an error propagation process. For this reason, a new concept of system analysis is introduced. The central idea is a synchronous examination of two directed graphs: a control flow graph and a data flow graph. The structures of these graphs can be derived systematically during system development. The knowledge about an operational profile and properties of individual system components allow the definition of additional parameters of the error propagation model.

A discrete time Markov chain is applied for the modeling of faults activation, errors propagation, and errors detection during operation of the system. A state graph of this Markov chain can be generated automatically using the discussed dual-graph representation. A specific approach to computation of this Markov chain makes it possible to obtain the probabilities of all erroneous and error-free system execution scenarios. This information plays a valuable role in development of dependable systems. For instance, it can help to define an effective testing strategy, to perform accurate reliability estimation, and to speed up error detection and fault localization processes.

This thesis contains a comprehensive description of a mathematical framework of the new dual-graph error propagation model, several methods for error propagation analysis, and a case study that demonstrates key features of the application of the presented error propagation model to a typical mechatronic system. A numerical evaluation of the mechatronic system in question proves applicability of the introduced concept.

Acknowledgements

This thesis was written during my research activity at Institute of Automation at the Dresden University of Technology. It would not have been accomplished without a contribution of the following persons.

First of all, I would like to express my sincere gratitude to my mentor and advisor, *Professor Dr. Techn. Klaus Janschek*. His wise guidance, detailed analysis of my work, scientific and emotional support helped me to proceed through the doctoral program and complete this dissertation.

This work would have never been possible without *Professor Dr. Nafissa Yussupova*, my long-time advisor from the Ufa State Aviation Technical University. She provided the possibility to obtain the funding for this research project and guided me during these years.

I would like to specially thank *Professor Dr. Christof Fetzer* from the Computer Science Department of TU Dresden for his interest in my work and a number of valuable discussions.

Last, but not least, I would like to thank my *wife and parents*. I'm thankful to be the son of two loving and supportive people. While working on this thesis, I've spent two very long years apart from my wife. I deeply appreciate her patience and believe that I will make up the leeway.

Contents

List of Abbreviations

ASR	Architectural Service Route
CFG	Control Flow Graph
CFT	Component Fault Tree
COTS	Commercial of the Shelf
CTMC	Continuous Time Markov Chain
DFG	Data Flow Graph
DTMC	Discrete Time Markov Chain
EC	Error Correction
ECM	Error Containment Module
EDB	Error Detection Behavior
EDM	Error Detection Mechanism
EDP	Error Detection Probability
EM	Error Message
EPA	Error Propagation Analysis
EPF	Error Propagation Framework
EPG	Error Propagation Graph
EPM	Error Propagation Model
EPP	Error Propagation Probability
ERM	Error Recovery Mechanism
ET	Event Tree
FAP	Fault Activation Probability
FMEA	Failure Modes and Effect Analysis

FPTC	Fault Propagation and Transformation Calculus
FPTN	Failure Propagation Transformation Notation
FS	Fail-stop
FTA	Fault Trees Analysis
HAZOP	Hazard and Operability Studies
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
IPA	Interface Propagation Analysis
MNE	Mean Number of Executions
MTBF	Mean Time between Failures
MTTF	Mean Time to Failure
PIE	Propagation, Injection, and Execution Techniques
PTP	Probabilistic Table of Properties
SOA	Service-oriented Architecture
SSR	State Space Reduction
SWIFI	Software Implemented Fault Injection
SysML	Systems Modeling Language
TPM	Transition Probability Matrix
UML	Unified Modeling Language

List of Figures

List of Tables

List of Algorithms

1. Introduction

The research results presented in this thesis belong to a rather young scientific domain - *system dependability*. By this reason, in various papers devoted to error propagation analysis, different terms can describe similar entities.

The word "error" has many definitions. Many more meanings of this term exist in common use. In everyday life, "error" acts as a synonym for the word "mistake." It usually means that someone has performed an action, which has led to unintended consequences, or that the result of this action differs from the expected one. In science, particularly in the applied mathematics, the term "error" does not mean a mistake. It describes uncertainty in imperfect empirical measurement or data processing. In this case, an error estimates how close this measurement is to a real value.

However, in this work, the term "error" is used in a more general context that fits for the engineering domain better. This thesis adheres to the definition proposed by J.C. Laprie in his book "Dependability: Basic Concepts and Terminology" [LAK92]. The original English text of this book is translated into French, German, Italian, and Japanese languages. Computer science and engineering communities all over the world accept this concept. A brief overview of the dependability research domain helps to distinguish the term "error" from other similar terms.

Dependability is the ability of a system to deliver a service that can be justifiably trusted. The service, delivered by a system, is its behavior as it is perceived by its user. A user is defined as another system (physical, human) that interacts with the former. J.C. Laprie describes dependability from three points of view: the *attributes* of dependability, the *means* by which dependability is attained, and the *threats* to dependability.

This description is represented by the so-called "dependability tree," shown in Figure 1.1. The presented research is focused on the threats to the system: *faults*, *errors*, and *failures*. Formal definitions of these terms follow.

Fault is a *defect* in the system that can be activated and cause an error.

Error is an *incorrect internal state* of the system, or a discrepancy between the intended behavior of a system and its actual behavior.

Failure is an instance in time when the system displays behavior that is contrary to its specification.

Faults, errors, and failures operate according to the chain, shown in Figure 1.2. A broken wire, an electrical short, and a software bug are all different faults. Activation of a fault leads to the occurrence of an error. Execution of the line of code that contains a bug, an attempt to send a signal via a corrupted connector, or utilization of a broken hardware

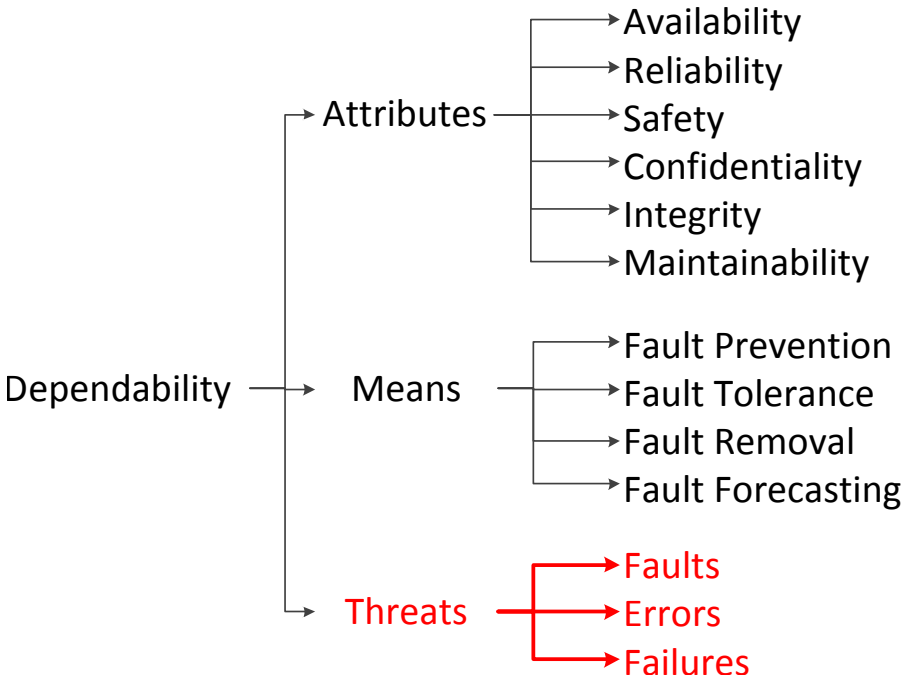

Figure 1.1.: Three main aspects of the dependability research domain: attributes, means, and threats.

part are the examples of faults activation. An error may act in the same way as a fault; it can create further error conditions. An incorrect physical state of a system, a wrong value of a software variable, or a false signal are different errors that can occur during system operation. The invalid internal system state, generated by an error, may lead to another error or to a failure. Failures are defined according to the system boundary. If an error propagates outside the system, a failure is said to occur. Since output data from one system may be transferred to another, a failure of the first system may propagate into another system as a fault.

Figure 1.2.: The fault-error-failure chain: Fault activation leads to error occurrence, error propagation out of system boundaries results in a failure, the failure can be the cause of further errors.

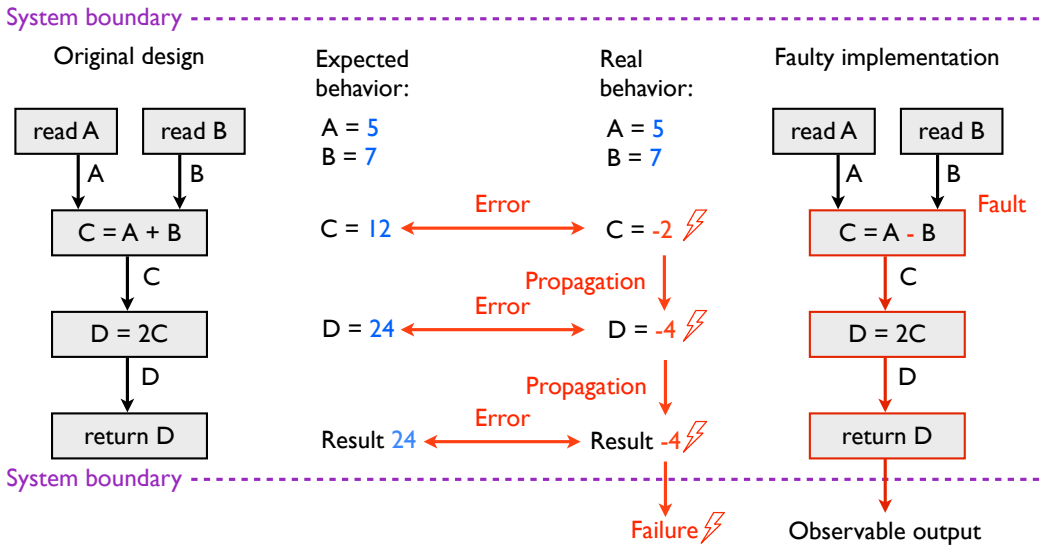

Figure 1.3.: An example that describes a system fault, activation of this fault, occurrence of a data error, and the propagation of this error that results in a system failure.

Figure 1.3 gives an example of fault activation and error propagation. It describes a simple system that reads two variables, A and B, processes them, and returns a result variable, D. An intended design of this system is shown in the left side of the figure. The right side of this figure shows a faulty implementation that contains a block $C = A - B$ instead of $C = A + B$. This particular bug is a typical fault within the system design. Activation of this fault happens with execution of this block. It results in an incorrect value of the variable C: C equals -2 instead of the expected 12. This deviation is an error that occurred during system operation. The variable D also takes a wrong value, because of incorrect C. It demonstrates propagation of the first error that leads to the occurrence of another error. Assume that an output of the system is observable by a user or another system. In this case, the incorrect value of D is visible and can be considered a failure of the system. However, it is not necessary that an error always results in a system failure. It might be masked, because of a specific system design, or detected by a user or an error detection mechanism. After error detection, system operation can be stopped to prevent further error propagation, or the error can even be corrected.

Analysis of fault activation, error propagation, and error (or failure) detection is defined in this thesis as *error propagation analysis*. The results of this analysis is extremely helpful in a wide range of analytical tasks associated with dependable systems development. For example, the error propagation analysis gives sound support for reliability evaluation, because error propagation has significant influence on the system behavior in critical situations. The error propagation analysis is a necessary activity for safety system design. It helps to estimate the likelihood of error propagation to hazardous parts of the system

and identify parts of the system that should be protected with error detection or error recovery mechanisms more strongly than the others. Another possible application area is system testing and debugging. An accurate error propagation analysis assists selecting an appropriate testing strategy. It helps to identify the most critical parts of the system (from either reliability or safety points of view) and to generate such a set of test-cases that will stimulate fault activation in these particular parts and allow the detection of occurred errors. Probabilistic error propagation analysis can be used for system diagnostics. In the case of error detection in observable system outputs, it helps to trace back an error propagation path up to an error-source. It speeds up the error localization process, system testing, and debugging.

In real systems, fault activation and further error propagation are very complex processes. This causes the need for a strong mathematical framework to perform an accurate error propagation analysis. Specifics of the mechatronic domain brings additional complexity. The fact is that mechatronic systems incorporate the assembly of heterogeneous components (mechanical, electrical, computer, and information technology) with various mutual interactions. The goal of mechatronic system design is to ensure a proper and coordinated operation of these elements within a feedback structure under all possible operational conditions. According to a book "Mechatronic Systems Design: Methods, Models, Concepts" by K. Janschek [Jan11], one of the big challenges of mechatronics is the use of appropriate models, which describe this mutual interaction on a common abstract layer. The error propagation analysis, as an essential part of the mechatronic system design, also requires a specific model. This model must be able to operate with abstract entities to represent various properties of the heterogenous mechatronic components. Development of a sufficient error propagation model is the main challenge of this thesis.

2. State of The Art

This chapter represents the state of the art in studies that concern error propagation issues. It distinguishes between error propagation analysis for software and hardware and focuses on several high-level error propagation models that can be adapted to the mechatronic domain.

Error propagation is closely connected to the system reliability research domain. Reliability evaluation is one of system analysis tasks where the knowledge about error behavior is the most valuable. Moreover, this research area is the origin of the error propagation analysis. Existing reliability models often underlay error propagation models and vice versa; an error propagation analysis is used for more accurate reliability assessment. Therefore, core principles of reliability evaluation have been borrowed by the error propagation analysis. Due to this fact, existing approaches to system reliability evaluation are also discussed in this chapter.

2.1. System Reliability

System reliability is considered to be the ability of a system, or an element of the system, to perform required functions for a specified period of time. It is often reported in terms of probability. The most common reliability measures are listed in the following passage.

Failure rate is the probability of system failure per unit of time.

Mean time to failure (MTTF) is an estimate of the average time until the first failure of a system.

Mean time between failures (MTBF) is an average time between two successive system failures.

Reliability is a key property of safety-critical mechatronic systems, most of which consist of a mix of software and hardware elements. Reliability of hardware parts can be well estimated with the help of numerous methods and techniques of classical reliability. A lot of metrics for reliability analysis of single components exist, as well as models for reliability assessment of entire systems. In most cases, failure rates for existing hardware components are known and defined in specifications. Reliability of an entire hardware system can be estimated using this information and system level reliability models. A good survey of the hardware reliability models is given in [EFS+08].

The situation with the software reliability evaluation is more complicated. The history of reliability evaluation goes from hardware to software. By this reason, the first concepts of software reliability engineering were adapted from the older techniques of the hardware

reliability. However, the application of hardware methods to software has to be done with care, since there are fundamental differences in the nature of hardware and software faults. Weak reliability of hardware usually occurs at the beginning of utilization - the reason being burn-in, and than after a period of time, hardware wear-out. The software reliability usually depends on a number of unfixed bugs and design drawbacks. It increases during the testing/debugging phase and after bug fixes at the operational phase of the software development life cycle. Because of this distinction, well-established hardware dependability concepts might perform very differently (usually not well) for software. It was proposed in [EFR+08] that "hardware-motivated measures such as MTTF, MTBF should not be used for software without justification".

For the last 20 years, the software reliability engineering has been a separate domain. H. Pham gives the following classification of existing software reliability models [Pha10]: error seeding models, failure rate models, curve fitting models, reliability growth models, time-series models, and non-homogeneous Poisson process models. These models are based on the software metrics like the number of lines of codes, the number of operators and operands, cyclomatic complexity, a group of object-oriented metrics and many others (see [XSZC00, RR96] for further information about the software metrics). The majority of them are black box models that consider the software as an indivisible entity. There are several open repositories with software failure data that can be used for calibration of these models. The most well-known and available are PROMISE Data Repository [GTT07] and NASA IV&V Facility Metrics Data Program repository [Fac04].

A separate domain of the reliability models consists of *architecture-based software reliability models* [GPT01, GWHT04] that consider software as a system of components with given failure rates or fault activation probabilities (those can be evaluated using the black box models). Reliability of an entire system is evaluated by processing information about system architecture, failure behavior, and internal properties of system components. Most of these models are based on probabilistic mathematical frameworks like Markov chains, stochastic Petri nets, stochastic process algebra, and/or probabilistic queuing networks. In addition to reliability evaluation, these architecture-based models help to identify unreliable parts of the system. These models are abstract enough to cope with the heterogeneity of components of mechatronic systems. Therefore, several ideas of architecture-based reliability models are used in this thesis.

2.2. Classical Error Propagation Analysis

This section provides a survey of classical approaches to error propagation analysis. The majority of them have grown from former reliability models. Therefore, similar to the reliability domain, the classical approaches to error propagation analysis for hardware and software systems have fundamental differences.

2.2.1. Hardware Error Propagation Analysis

Typically, error propagation analysis of hardware is based on one of the classical reliability evaluation techniques: *failure modes and effect analyses* (FMEA), *hazard and operability studies* (HAZOP), *fault trees analysis* (FTA), *event trees* (ET) etc.

From the system engineering perspective, the most well-known approach to analysis of error propagation is the safety engineering technique - FMEA. It is a manual process of identifying failure modes of a system, starting with an analysis of single component failures. Generally, this process of failure analysis consists of several activities: identifying failures of individual components, modeling the failure logic of the entire system, analyzing the effect of a failure on other components, and determining and engineering the migration of potential hazards.

As a rule in the safety domain, developers model and analyze potential failure behavior of a system as a whole. With the emergence of component-based development approaches, investigations began exploring component oriented safety analysis techniques, mainly focusing on creating encapsulated error propagation models. These failure propagation models describe how failure modes of incoming messages, together with internal component faults, propagate to failure mode of outgoing messages. According to the [GPM09], *failure propagation transformation notation* (FPTN) [FMD93] was the first approach to promote the use of failure propagation models. Other relevant techniques are *hierarchically performed hazard origin and propagation studies* (HiP-HOPS) [PMSH01] and *component fault trees* (CFT) [KLM03]. A limitation of these safety analysis techniques is their inability to handle cycles in the control flow architecture of the system; cycles of course appear in most realistic systems. Another approach, *fault propagation and transformation calculus* (FPTC) [Wal05], is one of the first techniques that could automatically carry out failure analysis on systems with cycles.

The FMEA and the FPTN provide means for manual or non-compositional analysis that is expensive, especially in a typical component-based development process, because the failure analysis has to be carried out again in the case of changes in the components. The FPTC does not provide facilities for quantitative analysis, particularly in terms of determining the probability of specific failure behavior. These disadvantages exclude the possibility of using the listed models for the error propagation analysis of mechatronic systems.

2.2.2. Software Error Propagation Analysis

In the software engineering domain, the majority of classical error propagation approaches are based on *fault injection* or *error injection* techniques, conjugated with further statistical evaluation. One of the classical papers about software error propagation was published by J.M. Voas [Voa92]. It presents a dynamic technique for statistical estimation of three characteristics that affect computational behavior of a program: (i) the probability that a particular section of a program is executed, (ii) the probability that the particular section affects the data state, and (iii) the probability that a data state produced by the section

has an effect on program output. The author claims that these characteristics can be used to predict whether faults are likely to be uncovered by software testing. Another well-known approach to error propagation analysis is based on *propagation, injection, and execution* (PIE) technique [JL90]. It is an extension of the previous work of the same author. One more paper of J.M. Voas [Voa97] introduces an *interface propagation analysis* (IPA). The IPA is also a fault injection technique based on so-called 'garbage' injection into interfaces between system components and an observation how this 'garbage' propagates through the system.

An empirical study about propagation of data-state errors was presented in [MJ96]. Results of this study have been also obtained by specific fault injection. Candea et al. present a technique for automatically capturing dynamic fault propagation information in [CDCF03]. The authors use instrumented middleware to discover potential failure points in the application. Their technique builds a failure propagation graph among components of the system, using controlled fault injection and observation of the fault propagation. Khoshgoftaar et al. in [KAT+99] describe identification of software modules, which do not propagate errors, induced by a suite of test cases. The attention of this paper is focused on propagation of data state errors from a location in the source code to the outputs or observable data state during random testing with inputs drawn from an operational distribution. The authors present empirical evidence that static software product metrics can be useful for identifying software modules, where the effects of a fault are not observable.

A number of papers depict the influence of software error propagation phenomena on system reliability. Sanyal et al. in [SSB97] describes Bayesian reliability prediction of error propagation probability in component based systems. The authors use event control flow graphs to compute event failure probabilities among system components. They study the impact of component failure rate on error propagation in these systems. The underlying idea is to consider event probabilities, event dependencies, and fault propagation in order to compute the probabilities of occurrence of every event in the system. Finally, Zhang et al. in [ZFJ09] introduce an extension of a classical reliability model, presented by R. Cheung in [Che80], by considering error propagation phenomena. However, this paper contains only doubtful theoretical discussion without any numerical evaluation.

2.3. Error Propagation Models

This section discusses four candidate error propagation models, presented in the last ten years. These models were originally developed for the software engineering domain. However, all of them have a strong theoretical foundation and operate with abstractive entities. This makes them the best candidates for error propagation analysis of the heterogeneous components of mechatronic systems.

2.3.1. Abdelmoez's Model

Three articles [ANAS02, ANS+04, NRS+02] by a group of researchers from the *West Virginia University* (Abdelmoez et al.) talk about the different aspects of a single software error propagation model. This model is based on system architecture analysis using UML [OMG10b] diagrams. Application area, as defined in the papers, is *commercial of the shelf* (COTS) software.

The main advantage of this model is its applicability even in the design phase of the system development. The structure and semantics of the source code are not available at this phase, but the information about the flow of control and data within system components and between the components is presented in the corresponding UML diagrams. The authors use *state* and *sequence* UML diagrams in reference case studies. Abdelmoez's model is a probabilistic model like the majority of the existing error propagation models. The main function of this model is computation of the probabilities of error propagation from one software component to another.

The authors distinguish between conditional and unconditional error propagation probabilities. The *conditional error propagation probability* from a component A to a component B is defined as "the probability that an error in A is propagated by B because the outcome of executing B will be affected by the error in A" [NRS+02]. It is implied that the error propagates from A to B under the condition that the component A will actually transmit a message to the component B:

$$EP(A, B) = Pr([B](x) \neq [B](x')|x \neq x')$$

where $[B]$ is a function of the component B, which captures all the outcomes of the executing of B (a state of B and outputs of B), A variable x denotes a message instance, used in the communication between the components A and B, x represents a corrupted message, and $EP(A, B)$ represents the probability that a fault and an associated error state in A will be propagated to B. In the case of the architecture of N components, EP is an N x N matrix, where an element $EP(A, B)$ is the error propagation probability from the component A to the component B. The value of $EP(A, A)$ always equals to 1, meaning that an error in the given component will always change its expected outcome.

Also the authors consider the *unconditional error propagation probability*. It is denoted by $E(A, B)$ and defined as the probability that an error propagates from A to B, without being conditioned by an actual occurrence of a message from A to B. $E(A, B)$ is calculated using the *transmission probability matrix*, denoted by $T(A, B)$. Each element of $T(A, B)$ indicates the probability of a connector from A to B being activated during a canonical execution. The purpose of the matrix T is "to reflect the variance in frequency of activations of different connectors during a typical execution" [NRS+02]. The unconditional error propagation is computed as follows:

$$E(A, B) = EP(A, B) \cdot T(A, B)$$

The element $T(A, B)$ shows the number of messages between components A and B in

the given UML model divided by the number of all observed messages in the system. In other words, $T(A, B)$ represents an estimate of the probability that a message is sent from A to B. The authors have found analytically that the error propagation probability can be expressed in terms of the probabilities of the individual A-to-B messages and states, via the following formula:

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

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

A combination of Abdelmoez's model and a UML-based model for early reliability assessment for COTS systems, introduced by Singh et al. in [SCC+01, CSC02], has been presented by Popic et al. in [PDAC05, Pop05]. The goal of Popic's research is to extend the Singh's model by considering error propagation.

Singh's model can be applicable early on in the software development life-cycle because of seamless integration with UML diagrams (use-case, sequence, and deployment UML diagrams are considered in the paper). The model supports reliability prediction in the system design phase. The authors assume that information about failure rates for components and connectors is available. The failures among different components are considered to be independent events. Component failures follow the principle of regularity, i.e., a component is expected to exhibit the same failure rate whenever it is invoked. The failure probability of a component C_i in a scenario j is represented in Singh's model as the following:

$$\Theta_{ij} = Pr(failure\_of\_C_{ij}) = 1 - (1 - \Theta_i)^{bp_{ij}}$$

Where Θ_i is the probability of failure of the component, and bp_{ij} represents the number of busy periods that the component exhibits in the sequence diagram.

Popic considers the possibility of error propagation between the components and changes the previous expression to the next one:

$$\Theta_{ij} = Pr(failure\_of\_C_{ij}) = 1 - (1 - \Theta_i)^{bp_{ij}} \cdot \prod_{k=1}^{N} (1 - E(k, i)\Theta_{kj})$$

Where E represents the *unconditional* error propagation matrix, which was described in Abdelmoez's model. The last expression can be transformed into a system of equation. The solution of this system gives the probabilities of failure for each of the system components.

2.3.2. Hiller's Model

Another approach to error propagation analysis was presented by Hiller et al., the group of researches from the *Chalmers University of Technology in Gteburg*, Sweden, in [JHS01, HJS01, HJS05, HJS02]. This approach also concerns the analysis of data errors propagation in software. The primary application area is *modular software for embedded systems*. The authors define the concept of *error permeability* through software modules, as black-boxes with multiple inputs and outputs. The error permeability through a module is the probability of an error in an input permeating to one of the outputs. In contrast to Abdelmoez's model, Hiller et al. pay more attention to the process of error propagation through the modules, but not between the modules. In [HJS01] the permeability and a set of related measures are applied to find weak parts that are most likely exposed to propagating errors. Based on the performed error permeability analysis, the authors describe how to select suitable locations for error detection mechanisms (EDM) and error recovery mechanisms (ERM).

The permeability of a module is defined in the following manner. For a particular module M with m inputs and n outputs, error permeability is the conditional probability of error occurrence in the output, given that there is an error in the input. Thus, for an input i and an output k of a module M, error permeability$P^M_{i,k}$ is defined as follows:

$$0 \leq P^M_{i,k} = Pr(error\_in\_output\_k \mid error\_in\_input\_i) \leq 1$$

This measure indicates how *permeable* is the pair *input i/output k* of the software module M. The error permeability is the basic measure upon which the authors define a set of related measures. The *relative permeability*, denoted by P^M:

$$0 \leq P^M = (\frac{1}{m}\frac{1}{n}) \sum_i \sum_k P^M_{i,k} \leq 1$$

This expression does not automatically reflect the overall probability that an error permeates from the input of the module to the output. Rather, it is an abstract measure that can be used to obtain a relative value across the modules. In order to distinguish modules with a large number of input and output signals from those with a small number of input and output signals, the authors removed the weighting factor in the previous equation, and defined the *non-weighted relative permeability* $\hat{P^M}$as follows:

$$0 \leq \hat{P^M} = \sum_i \sum_k P^M_{i,k} \leq m \cdot n$$

Using the permeability for each *input i/output k* pair, the authors construct a permeability graph. Each node in this graph represents a particular module and has a number of incoming and outgoing arcs. Each arc has a weight associated with it, which represents the corresponding error permeability value. This graph enables two types of error analysis: (i) determining the paths in the system along which errors will most likely propagate

to certain output signals (*output error tracing*), and (ii) determining which output signals are most likely affected by errors occurring in the input signals (*input error tracing*).

In order to find the modules that are most likely to be exposed to propagated errors along the obtained paths with the most probable propagation, the authors define an *error exposure measure* of a particular module as a normalized sum of weights of all incoming paths. The weight for each path is the product of the error permeability values along the path. The error exposure is the mean of the weights of all incoming arcs of a node. Analogous to the non-weight relative permeability, the authors define a *non-weighted error exposure measure*.

Finally, using these two sets of measures, the authors define two rules for the placing of EDMs and ERDs. *The first rule*: The higher the error exposure values of a module, the higher the probability that it will be subjected to errors propagating through the system if errors are present. The authors concluded that it may be more effective to place EDMs in the modules with higher error exposure than in those with lower error exposure. *The second rule*: The higher the error permeability values of a module, the higher the probability of consequent modules being subjected to propagating errors if errors should pass through the module. Therefore, Hiller at al. suggest that it is cost effective to place ERDs in the modules with higher error permeability values than in those with lower error permeability values.

In the other paper [JHS01], the authors present a more specific approach to error propagation analysis. This approach is aimed at the systematic development of software in a such way, that inter-modular error propagation is reduced by design. The main theoretical contribution of this research is the definition of several metrics, which quantitatively characterize inter-modular error propagation.

The authors define the three phase of error propagation process through the software with *error containment modules* (ECM): (i) an error occurring in a source module ECM_S, (ii) an error propagating out of the source module, (iii) and the resulting error in a target module ECM_T. The number of potential propagation media for errors between $ECM_S$and ECM_T is defined and denoted by m. M_j represents j^{th} propagation medium of m.

The *probability of error propagation* out of M_j, is defined as $P_{M_j}^{I_k}$, where I_k is the k^{th}input of ECM_S. After that the authors define *the error transmission probability* as the probability of an error occurrence at the output of ECM_S to propagate, through M_j to the input set of ECM_T. This metric is denoted by P_j^1:

$$P_j^1 = \frac{Pr(I)}{N} \sum_{k=1}^{N} Pr(M_j|I_k)$$

Where N is a number of inputs of source ECM, and $Pr(I)$ is the probability of an error occurring in the input set I of the ECM. Once the error has propagated via M_j to input of ECM_T, the probability of an error occurring in the state $ECM_T$is known as *error transparency*, denoted by P_j^2. It shows how vulnerable ECM_T is to errors propagating

from ECM_S. Using these two metrics Jhumka et. al define influence of ECM_S to ECM_T:

$$I_{S,T} = 1 - \prod_{j=1}^{m}(1 - I_{S,T}^{M_j})$$

Where $I_{S,T}^{M_j} = P_j^1 \cdot P_j^2$. In addition to the influence metric, *a total separation metric* was defined as:

$$ECM_S \vdash ECM_T = (1 - I_{S,T}) \prod_{k}(1 - I_{S,k} I_{k,T}) \prod_{l,m}(1 - I_{S,l} I_{l,m} I_{m,T})$$

Where $k, l, m, \ldots$ represent intermediate ECMs. The separation value gives an estimate of the level of interaction between ECMs, as all other ECMs are considered. 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.

In conclusion, Hiller et al. pay a lot of attention to numerical evaluation of the introduced concepts. All related computations have been performed using error injection techniques with the help of the PROPANE software tool, introduced in [HJS05, HJS02].

2.3.3. Mohamed's Model

The third error propagation model is introduced by Mohamed et al. in [MZ08]. It describes an approach to error propagation analysis based on identification of so-called *architectural service routes* (ASR). A defined application area of this approach is the reliability analysis of COTS software. An ASR is considered to be a sequence of components connected using *provided* or *required* interfaces. The authors use UML component diagrams to obtain ASRs of the system. A distinctive feature of this approach is that the authors focus on different error types: not only data corruption, but e.g. silent and performance errors.

An error of the type $F \in T$, which occurs in component x is defined as $f_x \in F$. The component y is another system component that exists in one or more ASRs between x and y. The authors define the probability of the masking of f_x as follows:

$$P(f_x \Rightarrow m_y) = \prod_{k=1}^{|\Psi^{x,y}|} \sum_{i=1}^{L_k^{x,y}} \prod_{f_{j+1} \in T} \left(\prod_{j=1}^{i-1} P_{j+1}^{e_j f_{i+1}} \right) P_i^{e_{j+1} m_i}$$

The probability of error propagation to y is defined as follows:

$$P(f_x \Rightarrow f'_y) = \sum_{i=1}^{L_k^{x,y}} \prod_{f_{j+1} \in T} \left(\prod_{j=1}^{L_k^{x,y}-1} P_{j+1}^{e_j f_{i+1}} \right) P_i^{e_{j+1} f'_y}$$

Where $\Psi^{x,y}$ is a set of possible ASRs between the components x and y. $L_k^{x,y}$ - length of k^{th}ASR from x to y. $P_x^{ef} = P(e_x \Rightarrow f_x)$ - probability that an input error e will cause

the failure f_x in the component x. $P_x^{em} = P(e_x \Rightarrow m_x)$ - probability that an input error e will be masked in the component x.

Using the defined probabilities, the authors analyze an error propagation aspect with respect to its scattering effect, based on failure types and its ability to localize faults. Also, they determine upper and lower bounds of failure propagation among system components and present the relation between system reliability and architectural attributes. The attention in Mohamed's model is focused upon: fault localization ability, error masking upper bound, error propagation lower bound, error masking and shortest ASR relationship, error masking and a number of ASRs relationships, error propagation and the shortest ASR relationship, error propagation and a number of ASRs relationship.

Two formulas for system reliability evaluation have been given using this error propagation model. The first one defines reliability of the system via the error masking probability. The second formula defines the reliability using the error propagation probability of system outputs. The idea of reliability assessment is continued and modified to consider error type-awareness in [MZ10b]. In [MZ10a], the previous work was extended to propose a selection framework for incorporating reliability in software architectures.

2.3.4. Cortellessa's Model

The last model, considered in this chapter, is presented in [CG06, CG07]. Two Italian researchers introduce a very comprehensive approach to reliability analysis of component-based systems that takes into account error propagation phenomena. This approach provides useful support for several engineering tasks: placing of error detection and recovery mechanisms, focusing the design effort on critical components of the system, and devising cost effective testing strategies.

Cortellessa et al. use the classical definition of faults, errors, and failures [LAK92]. The authors consider reliability of a component-based system as the probability of failure-free operation that strongly depends on the following factors:

- An internal failure probability of each component - the probability that the component will generate an error, caused by some internal fault.
- An error propagation probability of each component - the probability that the component will propagate an erroneous input it has received to its output interface.
- A propagation path probability of the component assembly - the probability of error propagation through each possible error propagation path from a component up to the system output.

The authors assume that data errors always propagate through the control flow, and a system operational profile is known and satisfies the Markov property (see Appendix A

for a detailed description of Markov models). The operational profile of the system under consideration is defined as the matrix P:

$$P = [p(i,j)], (0 \leq i,j \leq C+1)$$

An entry $p(i,j)$ of this matrix represents the probability that a component i, during its execution, transfers the control to a component j. C is a number of components. The first and the last rows of the matrix P correspond to two "fictitious" components that represent the entry point and the exit point of the system respectively. The authors define two attitudinal measures: $intf(i)$ and $ep(i)$.

- $intf(i)$ - "is the probability that, given a correct input, a failure will occur during the execution of i, causing the production of an erroneous output" [CG07]. In other words, it shows the probability of fault activation during the execution of the component i, which leads to error propagation to the output of this component.

- $ep(i)$ represents an error propagation probability through the component i. It is the probability that an erroneous input of this component becomes the cause of an error on the output of this component.

Using the definitions listed above and the ChapmanKolmogorov equation for discrete time Markov chains (DTMC), the authors come to the following formulas:

$$err(i) = \sum_{k=0}^{\infty} \sum_{h=0}^{C} err^{(k)}(i,h)p(h,C+1)$$

$$Rel = 1 - err(0)$$

Where Rel is the system reliability. A variable $err(i)$ is the probability that the system will complete its execution producing an erroneous output, given that the execution started at a component i. A variable $err^{(k)}(i,j)$ shows the probability that the execution will reach a component j after exactly k control transfers and j produces an erroneous output, given that the execution started at the component i. The next recursive equation shows how to compute $err(i)$ using the defined $intf(i)$ and $ep(i)$ measures:

$$\begin{aligned}
err^{(k)}(i,j) &= p^{(k)} \cdot intf(j) + \\
&\quad + ep(j) \cdot (1 - intf(j)) \sum_{h=0}^{C} err^{(k-1)}(i,h)p(h,j) \\
err^{(0)}(i,j) &= 0, \quad \forall i \neq j \\
err^{(0)}(i,j) &= intf(j), \quad \forall i = j
\end{aligned}$$

The first part of the equation, $p^{(k)} \cdot intf(j)$, represents the probability of fault activation and error propagation on k^{th} step of system execution. The second part shows the probability of error propagation through the component, under the condition that an error occurs in the previous step in a component h. The authors also represent the same equation in a matrix form and use them to perform a sensitive reliability analysis.

To obtain the $intf$ parameters, the authors refer to the surveys of the architecture-based models [GPT01] and [GWHT04], discussed in Section 2.1. For computation of the ep parameters the authors refer to Hillers model (see Section 2.3.2).

Another paper [CP07] discusses a path-based approach to error propagation analysis in the composition of software services. It introduces a model that generates possible execution paths within a service-oriented architecture (SOA) using a set of scenarios. These scenarios are obtained from *UML collaboration diagrams*, *message sequence charts*, or *UML sequence diagrams*. Cortellessa et al. focus on *no-crash failures*, which means that such a failure does not provoke the immediate termination of the whole SOA system. Instead of this, it can propagate to the next service or can be masked.

The introduced model considers a SOA composed by a number elementary services. The authors define an input domain for each service as a number of disjoint equivalence classes. Through the composition of the elementary services, the SOA offers external services (i.e. system functionalities) to the user. After that, a *service dependency graph* is defined to describe system behavior. Each node of this graph represents an elementary service. Each directed edge from a node i to a node j represents the invocation of the service j from the service i. The defined graph model is used for the probabilistic analysis of error propagation through possible execution paths.

2.4. Summary

Four error propagation models that can be considered as candidates for the analysis of mechatronic systems are listed in Section 2.3. The key properties of these models are also shown and compared in Table 2.1.

Abdelmoez's model is a design-level model for error propagation analysis of COTS systems that was also extended for reliability evaluation in [PDAC05]. This model uses information about system states and messages in order to compute the probability of error propagation between system components. The advantage of this model is the possibility of its application in the early phases of system development. However, it requires a very detailed and specific UML description that should also be very accurate for obtaining trustworthy results.

Based on this concept, Hiller et al. introduce the concept of error permeability through software modules and an error propagation model. This model is defined for modular software of embedded systems and can be used for dependable system design. Hiller's model seams more suitable for real-world application than Abdelmoez's model because it operates at the source-code level. The detailed case studies and the software tool PROPANE have proven this fact. However, the discussed concept can only be applied

to the software part of a mechatronic system because the theoretical background of this model is not comprehensive enough.

Mohamed et al. present another approach to error propagation analysis and its application for system reliability assessment, based on the definition of the architecture service routes. In spite of several deviations, Mohammed's model can be considered an offshoot of Cortellessa's model. Cortellessa's model is based on the Markov representation of system control flow. It was originally developed for COTS systems and later extended for SOA systems. This model has the strongest mathematical background in comparison to the other error propagation models that have been discussed in this chapter. The authors demonstrate its applicability for smart placing of error detection and error recovery mechanisms, planning of cost-effective testing strategies, and system reliability evaluation. Therefore, after the literature overview, the general idea of Cortellessa's model has been selected as the starting point of this thesis.

Table 2.1.: The comparison table of the suitable models for the error propagation analysis of mechatronic systems.

Author and years	Abdelmoez et al., 2002/2004/2005
Application areas	Commercial of the shelf.
Required data	State and sequence UML diagrams.
Main idea	An early estimate of the error propagation probabilities between system components in terms of states and messages.
Purpose	General use and reliability assessment.
Deficiencies	Not abstract enough. Requires very specific and detailed system models.

Author and years	Hiller et al., 2001/2005/2007
Application areas	Modular software for embedded systems.
Required data	Source code and reliability measurements.
Main idea	The concept of error permeability through a system module.
Purpose	Placement of EDM and ERM. Reduction of error propagation by design.
Deficiencies	More oriented to module level rather than system level analysis. Applicable only for a software part of the system.

Author and years	Mohamed et al., 2008/2010
Application areas	Commercial of the shelf.
Required data	Component UML diagrams, estimated fault activation and error propagation probabilities.
Main idea	Error propagation through an architectural service route.
Purpose	Reliability assessment.
Deficiencies	Not comprehensive enough. Can be considered an offshoot of Cortellessas model.

Author and years	Cortellessa et al., 2006/2007
Application areas	Commercial of the shelf and service-oriented architecture.
Required data	Fault activation, error propagation, and control flow transition probabilities.
Main idea	Probabilistic error propagation analysis using Markovian representation of control flow.
Purpose	Placement of EDM and ERM. Identification of critical components. Development of cost-effective testing strategies.
Deficiencies	Does not distinguish between control and data flows.