Derivation of Failure Mode and Effects Analysis (FMEA) Table from UML Software Model by Epsilon Model Transformation

by

Paul David Deji

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

Discovering and documenting potential failures and irregular user behavior that can interrupt the normal system behaviour is very important during the development of critical systems. Failure Mode and Effects Analysis (FMEA) is a bottom-up inductive analysis method that helps to identify potential failure modes based on experience with similar products and processes. Model-Driven Development (MDD) is a software development paradigm that raises the level of abstraction of software development by changing the focus from code to models and automating the generation of code from models. MDD also eases the derivation of analysis models for different software non-functional properties (NFPs) in the early stage of software development.

The objective of this thesis is to develop a model transformation process that takes as input a UML software model with failure mode annotations and generates a FMEA model. The transformation is developed in Epsilon, a new family of languages specialized in model transformations, refinement and management.

Acknowledgements

All thanks to the Lord my God and King for his grace and mercies! I sincerely appreciate the support and patience of my supervisor Prof. Dorina Petriu, whose constant direction and advice helped me throughout the duration of the thesis up till the very end. She spent a large amount of time and effort to answer my questions, review my work, make corrections and give me valuable feedback throughout my thesis. She is a major reason this thesis was completed and I am very grateful to her.

I would also like to thank Professor Samuel A. Ajila, Professor Abdulmotaleb El Saddik, and Professor Chung-Horng Lung for taking out the time to review my thesis and attend my final presentation. A special "thank-you" goes to all my dear colleagues and friends from my research group and RADS lab, especially Mana and Karishma, for their continuous help and valuable support throughout this thesis. Thank you all!

Thanks to Ifeoluwa Oyelowo my "Beta" (who did not want to do my school work for me) you are indeed a precious jewel to me. In addition, I would like to express my appreciation for the support and encouragement of the Living Praise Choir and my friends, most especially, Oluwaseyi Daropale and Mr. Joseph. God bless you all!

Special thanks to Ifewale Aroyehun, my beloved sister ("my mum, sister and friend") and the entire Ifekunle's family. I can't thank you enough for your immeasurable support and care. Finally, I appreciate the support and prayers of my parents, Chief and Mrs. Paul and the entire Paul family. My love goes to you all!

Table of Contents

List of Tables

List of Figures

List of Code Fragments

List of Appendices

List of Acronyms

ADC	Analog to Digital Converter
Al	Alarm
ATL	Atlas Transformation Language
DSM	Domain-Specific Model
DSML	Domain-Specific Modeling Languages
EMF	Eclipse Modeling Framework
EOL	Epsilon Object Language
Epsilon	Extensible Platform of Integrated Languages for mOdel maNagement
ETL	Epsilon Transformation Language
FMEA	Failure Mode and Effect Analysis
FMECA	Failure Mode Effect and Critical Analysis
Ev1	Electrovalve 1
FTA	Fault Tree Analysis
HAZOP	Hazard and Operability
ISR	Interrupt Service Routine
LCS	Level Control System
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MOF	Meta-Object Facility
Mv1	Mechanical Valves1
NFP	Non-Functional Property

OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform-Independent Model
PSM	Platform-Specific Model
QVT	Query/View/Transformations
RPN	Risk Priority Number
RTES	Real-Time Embedded Systems
S1	Level Sensor1
TER	Tele-echography
UML	Unifying modeling language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

1 Chapter: Introduction

1.1 Motivation and Objectives

The application of sophisticated computer systems built for high risk tasks is on the rise in different domains such as military, automotive, aerospace, health care, financial, etc. The consequences of failures in such systems maybe very costly and severe, and can lead to loss, physical damages or casualties. Therefore, proper development methods of dependable and safe systems are extremely important. In [PET95] are discussed a number of software failures. In [CHA05] it is given a good summary of software failure and their cost from 1992 to 2005 (Sources: *Business Week*, *CEO Magazine*, *Computerworld*, *Info Week*, *Fortune*, *The New York Times*, *Time*, and *The Wall Street Journal*). For instance, it shows how in 1996 software specification and design errors in ArianeSpace (France) program caused the explosion of a $350 million Ariane 5 rocket. This could have been avoided if all necessary verification and validation of reliability, safety and other functional and non-functional requirements would have been properly done. This is one of the reasons why it is of high interest to conduct research in the area of automating some of some of the engineering safety tools for software development.

According to [MHE16]: "Safety analysis has the objective to assess system safety during design phase and ensure that the designed systems have satisfactory safety level". In accordance with NASA Standard on software safety, risk is defined in [HAS05] as a "function of the anticipated frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity". FMEA (Failure Mode and Effect Analysis) is defined in [YU11] as a

powerful and effective engineering tool for the analysis of all possible failure modes and the elimination of the potential failure.

The term Model-Driven Development (MDD) refers to a software development paradigm whose main focus is to raise the level of abstraction of software development by changing the focus from code to models, as well as to automate the code generation from models. This thesis work is based on MDD, using some of the techniques, languages and tools that support MDD. Another important merit of MDD, apart from automatic code generation, is to facilitate the derivation of formal analysis models for the verification and validation of non-functional properties (NFP), such as safety, performance, reliability, availability, scalability, security, etc. The NFP verification/validation adds more engineering aspects to the software development process, which is a step forward from the Model-Driven Development to the Model-Driven Engineering paradigm.

In order to analyze a certain NFP in the MDE context, the following steps can be followed: a) the addition of NPF-specific information to the software model with the help of specific profiles, b) the execution of a pre-defined model transformation to generate an analysis model for the given NFP, followed by c) the analysis of the generated model using an existing solver, and finally d) giving feedback to the model designer. For instance, these steps were followed to transform annotated Unified Modelling Language (UML) models to Fault trees in [ZHA14] and to Layered Queueing Networks in [ZAR16]. UML is a general-purpose object-oriented visual modelling language that offers numerous formalisms to allow developers to create models during all phases of the development life cycle [MUS09].

This thesis work is at the intersection of the following disciplines: software engineering, model-driven development and safety analysis. The thesis focuses on addressing system safety during the software development process with the use of a known safety analysis method called Failure Mode and Effect Analysis (FMEA). More specifically, the thesis develops an approach for transforming a UML software model consisting of composite structure diagram and sequence diagram annotated with safety information into an FMEA model, which in turn will be transformed into the usual FMEA table (textual) form defined in the literature.

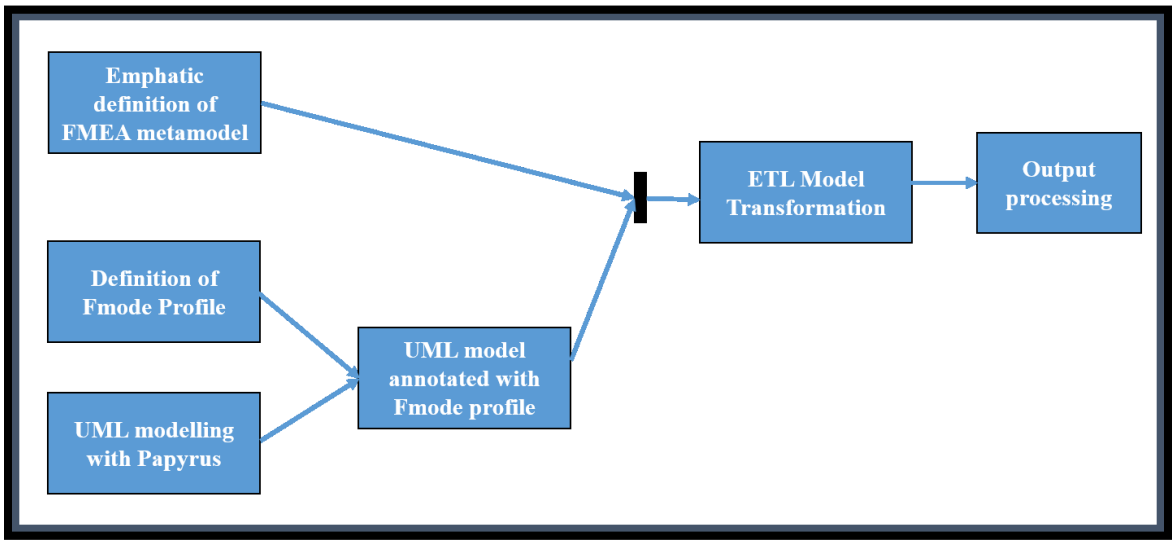

Figure 1-1 Activities for the proposed transformation

Figure 1-1, shows the activities necessary to transform an UML annotated with Fmode profile to an FMEA table. The UML modeling tool used for creating UML models is Papyrus [PAP15], which is an open-source tool based on Eclipse modeling framework (EMF). There are two preliminary steps performed in the thesis: the definition of the FMEA Metamodel using the language Emphatic [EMF15] and the definition of the Fmode profile. The input UML model conforms to the UML 2.5 Metamodel standardized by OMG. The transformation implemented in ETL generates the target FMEA model in

XML format. The XML document of the automatically generated FMEA model is then processed with XSLT to produce the corresponding FMEA table in text format, which will be presented to the developer.

1.2 Thesis Contributions

The main high-level goal of this thesis is to contribute to the integration of software safety analysis into the model-driven software development process. The first major objective of the thesis is to develop a model transformation process that accepts as input a UML software model with safety annotations, and generates a corresponding Failure Mode and Effect Analysis (FMEA) table as output. Other objective of the thesis is to use Epsilon, a new family of model transformation languages. To achieve the objectives mentioned above, the following contributions were made:

i. Development of an automatic multi-step FMEA model derivation process, which starts with a UML software model extended with the Fmode profile and generates the corresponding FMEA model in XML format, which is in then transformed into a textual format.

ii. The definition of the FMEA target metamodel expressed in the Emphatic language (presented in chapter 3).

iii. The definition of the Fmode profile for failure mode extensions to UML models (presented in chapter 3).

iv. The design and implementation of the model transformation: mapping of UML+Fmode model elements to corresponding FMEA model elements and designing and implementing a model transformation composed of ETL rules and

operations, which work together to generate the target FMEA model (presented in chapter 4).

v. Verification and validation of the ETL transformation. All the test cases and the running example used in the thesis for verification and the two case studies from literature used for validation are presented in chapter 5 and 3, respectively.

vi. Finally, since FMEA is usually presented in a tabular form (spreadsheet) a Text-to-Text transformation using XSLT to transform the generated FMEA model in XML format to its table format equivalent is presented in chapter 3.

1.3 Thesis Contents

This thesis consists of 6 chapters, including this introductory chapter, and an appendix. Chapter 2 introduces the background material required for this work, which includes a basic general description of software modeling languages, Model-Driven Development, Model-Driven Architecture (MDA), a brief description of UML and its major elements used in this thesis, an introduction to model transformation, the transformation languages we used in this work, description of some features of Software safety analysis and some engineering safety analysis tools with more attention to FMEA as a safety analysis tool. Also, some related works from literature on FMEA using some kind of model transformation are summarized.

Chapter 3 describes the design of the transformation process from annotated UML to FMEA model generation. It contains a detailed description of both metamodels of the source and target models with a running example.

Chapter 4 explains the main mappings from UML+FMode models to FMEA models. The mapping rules of the transformation are presented separately with the rules and operations.

Chapter 5 presents six test cases created by the author and two complete case studies from literature, in which all the main aspects of this model transformation were verified.

Chapter 6 is the conclusion that presents the main thesis accomplishments alongside with the limitations and some possible areas for future work.

2 Chapter: Background and State of the Art

2.1 Software Modeling Languages

In [GRO09] modeling is described as an integral part of complex software system development projects, whose main aim ranges from assisting developers and customers to communicate with each other, test case generation or automatic derivation of the developed system code. A modeling language can be defined as a language used to express the structure and behavior of systems, information or abstract knowledge, following a set of guiding rules. A complete definition of a modeling language consists of the description of its syntax, including well-formedness rules and its semantics [HAR04]. A well-known example of modeling language is the Unified Modeling Language (UML), whose goal is to represents software during the development process. According to its founders, UML is a graphical language for visualizing, specifying, constructing and documenting the artifacts of software-intensive system [BRJ01]. UML is standardized by the Object Management Group (OMG) [UML15] and is widely used in research and industry. UML is the modeling language used in this thesis for representing software systems.

Model and Metamodel

According to [ISO12], models describe the structural dimension of an object model and specify the collaborations among the objects. Structural models are used in conjunction with dynamic modeling techniques representing behavior, such as those based on finite state machines.

A metamodel is a special kind of model that defines the abstract syntax of a modeling language. The abstract syntax defines the set of modeling concepts expressed in the

language, their attributes and their relationships, as well as the rules for combining these concepts to construct partial or complete models. The UML metamodel used in the thesis is the standard specification adopted by OMG [UML15]. Another metamodel used in the thesis is the FMEA metamodel; its definition given in the next chapter was inspired from [TAGU11].

Model Driven Development (MDD)

MDD is a software development paradigm based on the idea of modelling. MDD is shifting the primary development focus from code (i.e., usual computer programming syntax and semantics) to models, thereby raising the abstraction level of the software development activities. MDD is characterized by automatic code generation from models. One of the advantages of MDD is the use of models that are very close to the problem domain, which has proven to make it easier to specify, understand, and maintain models [SEL03]. Also, the use of models provides easy means for verifying the model's non-functional properties, such as reliability, availability, safety, performance, scalability etc.

Model Driven Architecture (MDA)

Released in 2001, MDA is OMG's vision for model-driven development based on OMG standards; a more detailed presentation of the MDA definition was adopted by OMG in 2003 [MDA03]. One major aim of the MDA framework is the separation between the platform-independent specification of the software system and the platform-specific implementation of the system. This approach facilitates portability between different platforms and model reuse.

Model Driven Engineering (MDE)

Model Driven Engineering (MDE) includes Model-Driven Development (MDD), but also

extends the use of models to a wider range of uses, as presented below. MDE makes it easy to consider models as the key artifact of the software engineering process SEL03]. Complex systems can now be easily modeled and presented, but in recent time modeling has gone beyond mere presentation and documentation and can be used for the following engineering purposes:

i) Model Testing: Detection of errors at the model level is made easier with the provision of testing facilities at a very high level of abstraction. Unlike testing at the code level, which is very detailed, so error detection and elimination is not as easy.

ii) Documentation: With the consistent increase in the complexity level of software systems with modern technologies, comprehensive detailed documentation has become an important aspect of engineering work. Proper modeling of the system has proven to be a capable option for the system`s detailed specifications and comprehensive documentation.

iii) Code generation: Automatic or semi-automatic code generation of programming code is a vital and very important aspect of the model-driven approach (Model2Text transformation). Code can be automatically generated from the models, making the modeling more useful and not limited to just communication purposes.

iv) Verification of non-functional properties: Software models can be transformed into analysis models that can also, be expressed in different formats. Like in the case of this thesis, we developed a transformation from UML to FMEA model for safety analysis. Other modeling techniques for safety analysis are HAZOP, Fault

Tree, Petri nets, etc.

v) Model transformation: Models may have different purposes and application domains, different levels of abstractions and can be defined in different languages. The basic aim of a model transformation is to automatically transform a certain kind of model into another kind of model. Recently, specialized languages for model transformations have been defined and tool support has been developed, as described in section 2.2.

2.1.1 Unified Modeling Language (UML)

The Unified Modeling language (UML) standardized by Object Management Group (OMG) [UML15] is a general-purpose software modeling language used during the analysis, design, implementation, code generation, testing and maintenance phases of software systems by human users. The UML metamodel is defined in turn by using the Metal Object Facility (MOF) language [MOF11]. UML supports a set of graphical notations representing its concrete syntax, defined informally in the OMG standard specification [UML15].

UML provides fourteen different diagram types, which represent different perspectives of the system under development. Usually UML modeling tools are used to create UML models composed of multiple diagrams. Papyrus is the UML tool (editor) used in this thesis [Papy15]. Basically the UML diagrams are classified into two broad categories: structure diagrams and behavior diagrams. A structural diagram is used to describe the system's structure, as shown later in this thesis with the use of Composite structure diagram for the running example and the case study systems. The UML structure diagram category includes: Class diagram, Object diagram, Package diagram, Component

diagram, Deployment diagram, Composite structure diagram and Profile diagram. On the other hand, the behavior diagrams describe the interactions between structural entities and/or the behavior of a system, subsystem or component. The UML behavior diagram category includes: Use Case diagram, State Chart, Activity diagram and Interaction diagram, which in turn can be a Sequence diagram, Timing diagram, Communication diagram, and Interaction overview diagram. In this thesis, the Composite structure diagram is used to represent the system structure and the sequence diagram to represent the system behavior. (Both the Composite structure and Sequence diagrams are described in more detail below, as they are used in the transformation rules defined in Chapter 4).

UML metamodel

As described above, a metamodel is a model describing the abstract syntax and well-formedness rules of a modeling language. In other words, the UML metamodel describes the UML model elements, their attributes and relationships [UML15]. The metamodel is specified in the MOF language [MOF11] and it is organized in packages. The MOF language is aligned with UML, in the sense that MOF is composed of a subset of UML concepts used for modeling classes and packages. A UML model is an instance of the UML metamodel. In the rest of this subsection are presented the metamodel fragments for UML Composite Structure, Interaction and Profile, which are used for writing the transformation from UML to FMEA presented in Chapter 4.

UML Composite Structure diagram

A StructuredClassifier is a structural concept that may have an internal structure of Connectable Elements (each playing a given role) and an external structure of one or more Ports. The connectable elements can themselves be instances of a

StructuredClassifier, allowing for nested structures). The Ports act as local agents of remote collaborators which interact with the Structured Classifier owning the ports, allowing us to model the overall behavior (i.e., the interactions of this classifier with other parts of a system [ZHA14]). In this thesis, the composite structure diagram is used to represent the system structure, which may contain software components and hardware components. Figure 2-1 describes the abstract syntax of a UML StructuredClassifier, which owns any number of attributes representing the parts and any number of connectors that link the parts with each other. An attribute is modeled by a Property metaclass, which in turn is a Connectable Element (an abstract class representing the internal roles in the StructuredClassifier [UML15].

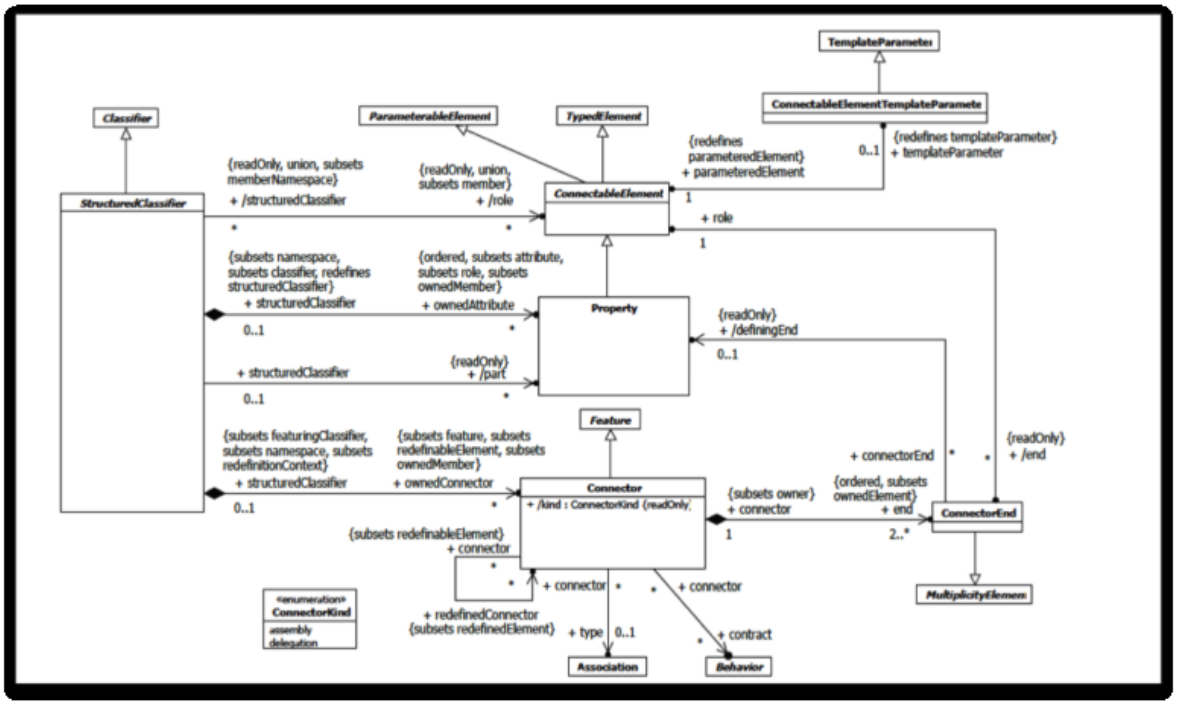

Figure 2-1 The UML2.5 Structural Classifier Metamodel [UML15]

UML Sequence Diagram

The UML Sequence diagram is the most common kind of UML Interaction Diagram that focuses on the interchange of Messages between the model's Lifelines. More specifically,

a sequence diagram describes an interaction by showing the sequence of Messages that are exchanged between the interaction participants (represented by Lifelines), along with their corresponding OccurrenceSpecifications on the Lifelines. Figure-2-2 shows the Lifeline metamodel fragment from [UML15]. In this thesis, Sequence diagrams are used to capture the message exchange or signal passing between components, emphasizing the elements that may fail during the component interaction.

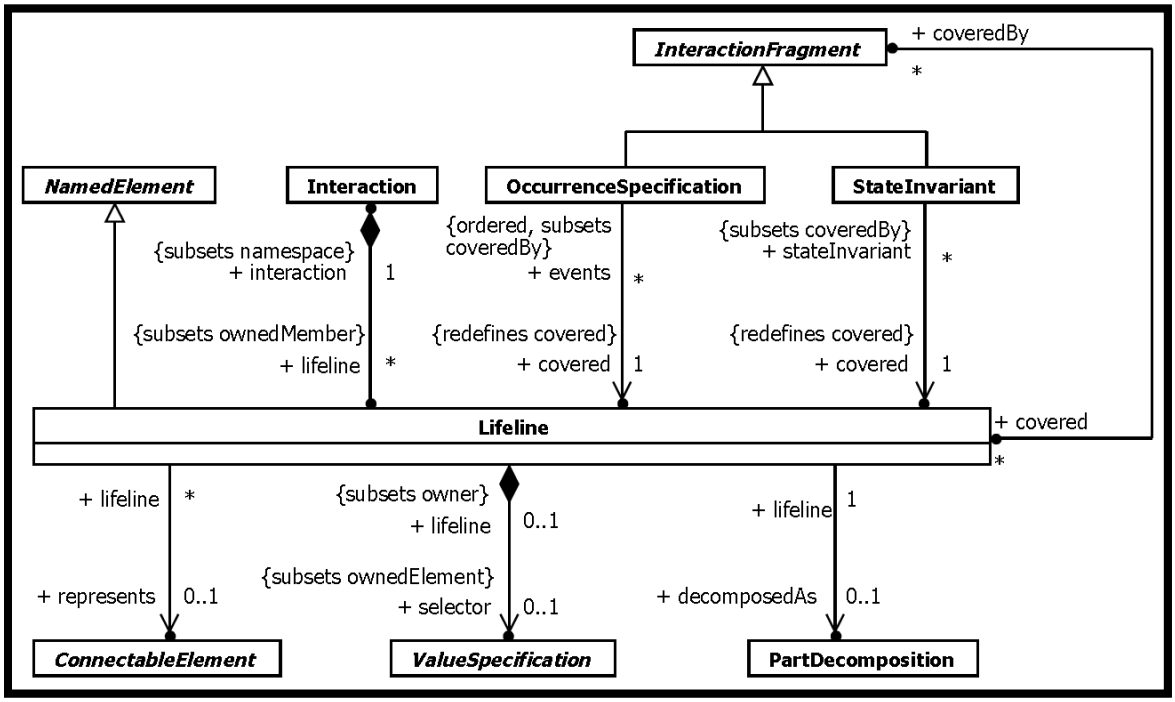

Figure 2-2 The Lifeline metamodel [UML15]

UML Profile

An important flexibility embedded in UML is the provision of a set of extension mechanisms (profiles) to accommodate the modeling of some domains for which the UML notation should be specialized. UML profile definition allows the customization and extension of its own syntax and semantics in order to adapt it to certain application

domains [FUE04]. As defined by UML 2.5 Standard, a Profile "is a restricted form of metamodel that can be used to extend UML".

The primary UML extension construct is the Stereotype, which may have a number of Properties (representing attributes) and Constraints. Stereotypes are used to extend one or more UML metaclasses with properties and constraints that are characteristic to a certain domain. Figure3-2 shows the metamodel of a UML profile. A UML Profile is defined as a UML package with the keyword «profile» that groups together related stereotype definitions, their attributes and constraints. The application of a profile to a model should not change that model in any way but merely define a view of the extended model [UML15].

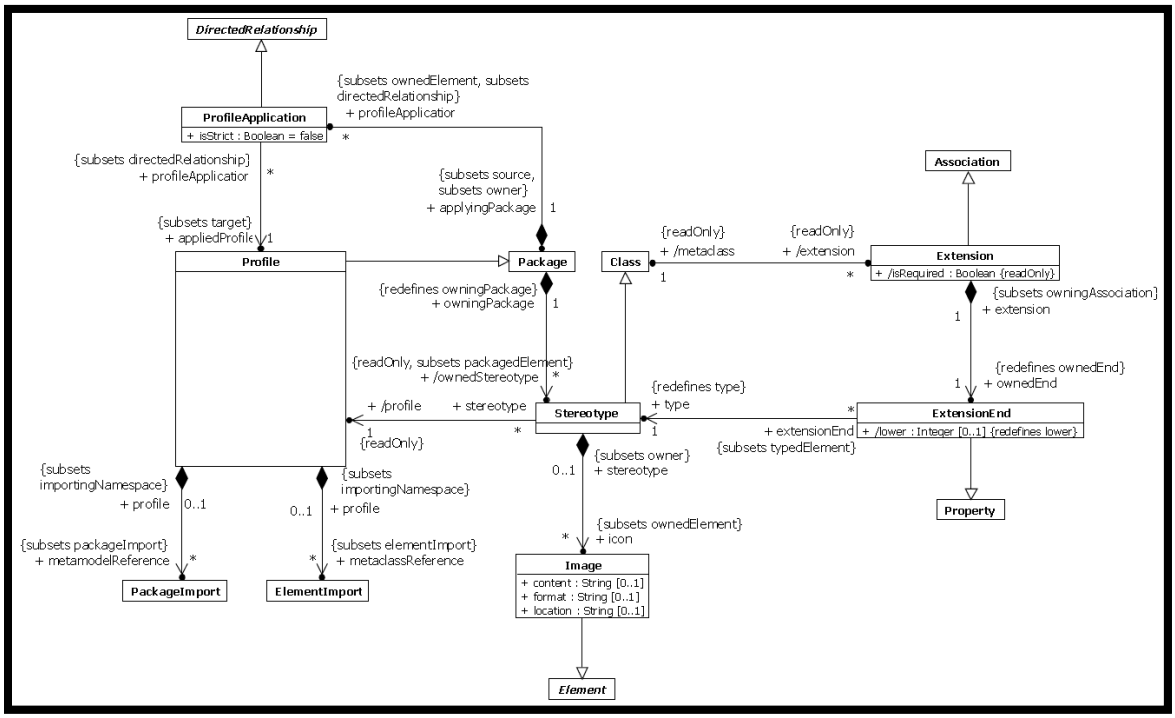

Figure 2-3 UML Profile metamodel [UML15]

A stereotype is a class that defines how an existing metaclass (or other stereotype) may be extended, and enables the use of platform or domain specific terminology in addition

to the standard UML notation. Certain stereotypes are predefined in the UML, others may be user defined.

UML Profiles can be used to define Domain Specific Modeling Languages based on UML extensions. According to the definition provided in [AMY06] "Domain-Specific Modeling Languages (DSMLs) are high-level languages specific to a particular application or set of tasks. They are closer to the problem domain and concepts than general-purpose programming languages. Improvements in productivity and comprehensibility are often cited as benefits."

Using UML profiles provides the additional advantage that the extended models can be processed with standard UML editors, without any need to change the tools, as profiles are standard mechanisms for extending UML models [PETR13].

In this thesis we define a profile that allows us to add failure mode annotations to the UML source model. These annotations are processed by the transformation developed in the thesis.

2.2 Model transformations

A model transformation is a special application that takes one or more input models (each defined by a source metamodel), and generates one or more output models (each defined by a target metamodel). Figure 2-1 illustrates the relationships between the models and metamodels involved in a model transformation. Model transformations may either be of the type of model-to-model (M2M) or model-to-text (M2T) transformation. The choice of what type of transformation to use is dependent on the main purpose of the transformation. In this thesis we use both M2M and M2T transformations.

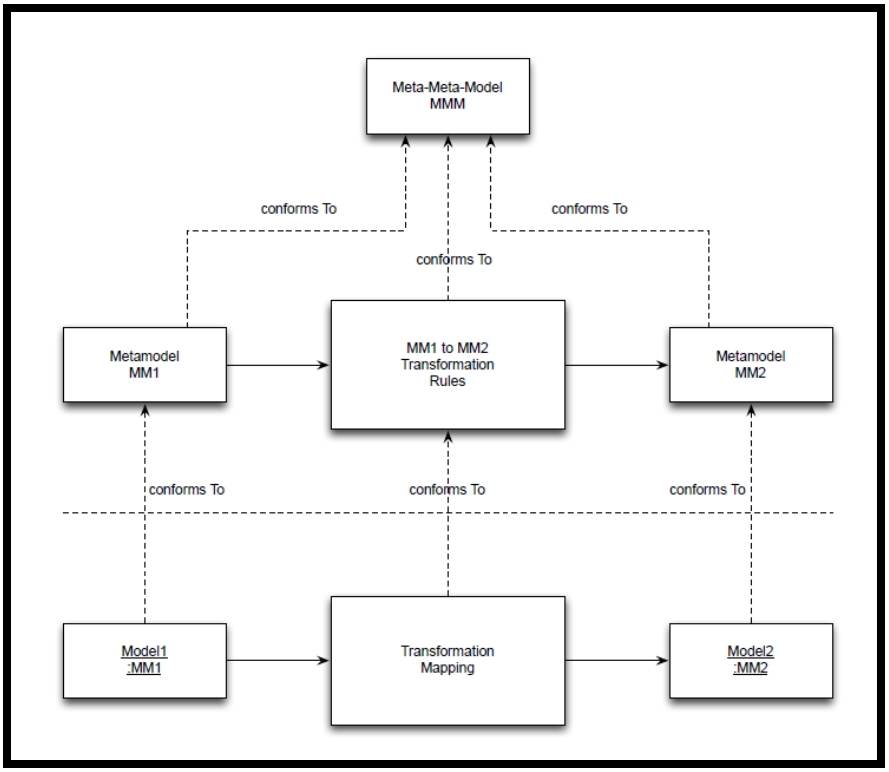

Figure 2-4 Pattern of model transformation [ZHA14]

Model-to-Model (M2M)

This is the type of transformation used in this thesis to transform an annotated UML model to the corresponding FMEA model, where the input and the generated output are both models. M2M is a very convenient way of transforming Platform-Independent-models (PIM) to Platform-Specific-Models (PSM), or like in the case of this thesis transforming an annotated software model to an analysis model for a given non-functional property (safety in this case).

Model-to-Text (M2T)

M2T is the form of model transformation, which generates textual artefacts from models. In this type of transformation, the input is a model while the generated output is a text artefact or an application code.

2.2.1 Specialized model transformation languages

Not long ago, model transformations had usually been carried out by using general-purpose programming languages [ZHA14], especially Java which is used in the open-source Eclipse Modeling Framework. However, more recently the development of specialized model transformation languages has become a major aid in model-to-model transformation. Special model transformation languages advantages include it syntax that provide easy access to the model elements and allows expressing constraints in the Object Constraint Language (OCL) style.

Model transformation languages are classified into three different categories: declarative, imperative and hybrid.

Declarative transformation languages provide a very good and solid transformation aid where both the source and target metamodels are of similar structure, by providing a model to model mapping of the two metamodels, though declarative transformation seems to provide a very good mapping ground for model transformation, it is limited in the cases where complex mapping is required.

Imperative transformation languages on the other hand, are transformation language, which is capable of supporting more complex transformation cases but operate at a much lower level of abstraction.

Hybrid transformation languages were developed to take care of the limitations of both purely declarative and purely imperative languages.

Examples of transformation languages include; ETL, QVT, ATL, MOF, JTL, GReA, etc.

2.2.2 Query/View/Transformation (QVT)

QVT is a modeling language standardized by OMG standard with more concentration on model transformation, view and query as the name implies. OMG issued a Request for Proposal (RFP) in 2002 on MOF Query/View/Transformations to develop a standard transformation language with OMG standards compatibility, QVT standard language evolved out of this proposal [QTV11]. The QVT aids two levels of declarative language; the Relations and the Core, it also supports an imperative language; the Operational mapping. The Relations language is basically employed in the matching of object patterns, creating object templates and tracing of model elements, while Core language on the other hand, seems to be much simpler and easier compere to the Relations language, it is used for matching patterns with a variable sets evaluated against a set of models. Operational mappings language this language is usually applied when invoking the transformation imperative representations from the declarative language.

2.2.3 Atlas Transformation Language (ATL)

ATL is another model transformation languages proposed during the OMG standardization process. It was released earlier compared to QVT, so it had quite a number of user of its toolset [ATL12]. ATL is hybrid, having both declarative and imperative characteristics. An ATL program is composed of different rules describing the matching/mapping and navigation of source model elements to generate the target model elements.

2.2.4 Epsilon family of languages

Epsilon (Extensible Platform of Integrated Languages for mOdel maNagement) is a family of consistent and interoperable task-specific languages, designed to handle model

management tasks such as model transformation, code generation, model comparison, merging, refactoring and validation [KOL15]. The Epsilon family framework contains Epsilon Object Language (EOL), Epsilon Transformation Language (ETL), Epsilon Wizard Language (EWL), Epsilon Merging Language (EML), Epsilon Validation Language (EVL), Epsilon Generation Language (EGL), and Epsilon Comparison Language (ECL).

In this thesis we used the Epsilon Transformation Language (ETL) for the M2M transformation from UML to FMEA. ETL is a hybrid transformation language used for model-to-model transformation in the Epsilon family. ETL also possess the EOL imperative features for proper handling of complex model transformation. ETL is capable of expressing the transformation at a higher level of abstraction than a general-purpose language such as Java. ETL accepts an arbitrary number of input models and generates an arbitrary number of output models [KOL15].

2.3 Software Safety Analysis

Software system safety engineering application focus more on an effective system level hazard analysis process.

2.3.1 Dependability Analysis

Dependability is a nonfunctional property (NFP) of a system, defined in [AVIZ04] as the ability to avoid failures that are more frequent and severe than acceptable. Dependability covers a set of attributes: availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences for the users and the environment), integrity (absence of improper system alterations) and maintainability

(ability to undergo modification and repairs). The evaluation of the dependability attributes can be done with quantitative and/or qualitative methods [LYU96].

Quantitative tools can be used to measure both system Availability and Reliability attributes, while the other safety attributes are more subjective. For instance, in the case of system safety it may not be easily determined using a particular measurement what is the system safety confidence level, unlike the case of system reliability, which can be determined using quantitative measures as failures over time.

Three concepts are important for dependability analysis: fault, error and failure. *Fault* can be defined as a system defect which can either result into a system error or not; system *error* is propagation from system fault, while system *failure* represents an instance in time when a system`s behavior is contrary to its defined specifications. When a fault is activated, can result to an error (system invalid state) and the system`s invalid state propagates to become an error which may in turn lead to another system failure (a vivid and observable deviation of the system from its specifications) or another system error

According to the NASA standard [NAS04] failure is "non-performance or incorrect performance of an intended function of a product. A failure is often the manifestation of one or more faults".

2.3.2 Safety Analysis Methods

System safety analysis methods include the following:

- Fault Tree Analysis (FTA) is a deductive, top-down method safety analysis method with the focus on analyzing effects of the initiating faults and events on a system.

- Failure Mode and Effects Analysis (FMEA) is a bottom-up, inductive analysis method with the main focus on analyzing the effects of a single component failure on subsystems or system [GSF96].
- Failure Mode, Effect and Critical Analysis (FMECA) is an extension of FMEA which is also, a bottom-up, inductive system safety analytical method. It extends FMEA with its analysis concentration on the system's components with high risk criticality, which is then used as a base to draw the probability of failure modes against the severity of their consequences. The components with relatively high failure modes occurrence and high severity level are given more attention with remedial effort. Mil-Std 1629 (ships), "Procedures For Performing a Failure Mode, Effects and Criticality Analysis" [USM74] was published in 1974. FMECA seems to be of greater importance in safety critical fields such as Military, Medicine (health), Aeronautics, Automobile, space and North Atlantic Treaty Organization (NATO) military applications etc.
- Dependence diagram (DD) known as reliability block diagram (RBD) and Markov analysis. It is equivalent to a success tree analysis (STA) with the logical inverse of an FTA, and it depicts the system using paths rather than gates. DD and STA produce probability of success instead of probability of failure.
- Hazard and Operability study (HAZOP) is a qualitative technique based on breaking down a complex design process into a number of smaller and simple sections (nodes), which are then reviewed individually, to identify and evaluate any associated risk to personnel or equipment.

The two most commonly used safety analysis techniques are FMEA and FTA. As indicated before, this thesis focuses on FMEA.

2.3.3 Failure Mode and Effects Analysis (FMEA)

FMEA history can be traced back to its initial development in the late 1940s by the US military to study failure classification. Examples of very important projects in which FMEA was used in the 1960s were the Apollo space missions. Another major use of FMEA as a tool was by Ford Motors in the 1980s with the focus on risks reduction in their automobile production.

FMEA is a technique for assessing the dependability of system's components and the system safety. FMEA has been used with success in the process industry in support of safety and reliability. Some of the advantages of FMEA is that it is easy to understand, can decrease cost, permit reduction of the development time and increase customer satisfaction.According to the definition given in [DEL04] FMEA is an engineering analytical technique that has proven to be domain independent (it has been used in diverse field of engineering) for identification and reduction of hazards in designing, manufacturing and maintenance of systems. The analyses enhance the complete/exhaustive analysis of all the system's components or its sub-systems to determine all the potential failure modes and its effects on the entire system stability [SPA03].

There are four common categories of FMEA: a) system FMEA class, b) design FMEA class, c) process FMEA class and e) machinery FMEA class. These categories of FMEA focus on different levels of analysis, according to the classes. System FMEA class focuses more on potential failure in the system level interactions, the design FMEA class

has its main focus on potential failure of product design, the process FMEA class analyzes potential failure in the process that produces/makes the product, and the machinery FMEA class is used to analyze the potential failure in the machinery that performs the process [MOR11]. FMEA is generically created within spreadsheets manually with a set of participants, which could include but are not limited to the following: designers, customers, owners, suppliers, etc.

FMEA usually involves the following steps:

- Determine the participants
- Brain storming using Murphy's law ("Anything that can go wrong will go wrong").Identify all components to be analyzed, the system and subsystems, all processes, functions (all conditions that could fail to meet required level of quality and reliability). The team must have the ability to describe the failure cause and its effects both local and propagated effect on the system.
- Criteria for FMEA analysis are based on the following failure characteristics: Severity, Occurrence frequency and Detection method.

The following ranking (defined as an enumeration type) were used in this thesis similar to [MRAZ05]:

Table 2-1 Severity Ranking and suggested criteria

Effect	Criteria	Severity	Rank
None		No Effect	1
Very Minor	Minor disruption to production line	A portion of the product may have to be reworked. Defect not noticed by average customers; cosmetic defects.	2
Minor	Minor disruption to production line.	A portion of the product may have to be reworked. Defect noticed by average customers; cosmetic defects.	3
Very low	Minor disruption to production line.	The product may have to be sorted and reworked. Defect noticed by average customers; cosmetic defects.	4
Low	Some disruption to production line.	100% of product may have to be reworked. Customer has some dissatisfaction. Item is fit for purpose but may have reduced levels of performance.	5
Moderate	Some disruption to production line.	A portion of the product may have to be scrapped. Customer has some dissatisfaction. Item is fit for purpose but may have reduced levels of performance.	6
High	Some disruption to production line.	Product may have to be sorted and a portion scrapped. Customer dissatisfied. Item is useable but at reduced levels of performance.	7
Very High	Major disruption to production line.	100% of product may have to be scrapped. Loss of primary function. Item unusable. Customer very dissatisfied.	8
Hazard with warning	May endanger machine or operator.	Failure occurs without warning. The failure mode affects safe operation and involves noncompliance with regulations	9
Hazard without warning	May endanger machine or operator	Failure occurs without warning. The failure mode affects safe operation and involves noncompliance with regulations	10

Table 2-2 Occurrence Ranking and suggested criteria

Notional Probability of failure	Evaluated Failure Rate	Rank
Remote: Failure is unlikely. No Failures ever associated with almost identical processes	1in 1,500,000	1
Very Low: Only Isolated Failures associated with almost identical processes	1in 150,000	2
Low: Isolated Failures associated with similar processes	1in 15,000.	3
Moderate: Generally associated with processes similar to previous processes Failures, but not in 'major' proportions	1in 2,000.	4
Beyond Moderate	1in 400	5
Moderately	1in 80	6
High: Generally associated with processes similar to previous processes that have often failed	1in 20	7
Very High: Failure is almost inevitable	1in 8.	8
Extremely High	1in 3	9
Almost Certain	1in 2	10

Table 2-3 Detection Ranking and suggested criteria

Detection	The likelihood the Controls will detect a Defect	Rank
Almost Certain	Current controls are almost certain to detect the Failure Mode. Reliable detection controls are known with similar processes.	1
Very High	Very High likelihood the current controls will detect the Failure Mode.	2
High	High likelihood that the current controls will detect the Failure Mode.	3
Moderately High	Moderately high likelihood that the current controls will detect the Failure Mode.	4
Moderate	Moderate likelihood that the current controls will detect the Failure Mode.	5
Low	Low likelihood that the current controls will detect the Failure Mode.	6
Very low	Very Low likelihood that the current controls will detect the Failure Mode	7
Remote	Remote likelihood that the current controls will detect the Failure Mode	8
Very Remote	Very Remote likelihood that the current controls will detect the Failure Mode	9
Almost Impossible	No known controls available to detect the Failure Mode.	10

The priority setting is usually based on the Failure mode Risk Priority Number (RPN):

$$RPN = Severity * Occurrence * Detection$$

After each analysis, the suggestion of corrective actions for system dependability and safety is another merit of performing FMEA.

Analysis assumption: It is assumed that only the component undergoing analysis is in its failure mode at the moment of analysis, while others are in perfect state.

The table below shows the structure of FMEA table headings.Full and complete tables for the running example and case studies are given later in chapters 3 and 5, respectively. This table heading is found in literature. Its elements are used in the FMEA metamodel, as explained in chapter 3 of this thesis.

Table 2-4 Structure of FMEA table

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause

2.4 Related work on FMEA using software models and/or model transformation

The survey [BERN12] of existing works on software dependability analysis based on the transformation of UML software model (source model) to different types of dependability analysis models (target model) shows that there are very few works that attempt to transform UML software models to FMEA models.

This section explores some related works on FMEA making use of software models, with special attention to the semi or automatic derivation of the FMEA table by model transformation technique. For example, in [DAV09] the use of both UML/SysML models is explored, by building FMEA models from the functional analysis of the software models. The authors analyze a set of use cases, taking into account the sequence diagram of every use case, from which all system functions can be identified. But this approach had a significant limitation, namely the production of "too many lines" in the generated table, which can lead to difficulty in risk identification. Therefore, the authors proposed to create a "dysfunctional database" that describes the significant failure modes for each component. The enhanced algorithm identifies the component type that participates in a use case and fetches the corresponding failure modes from the database. Thereafter, the generated FMEA table was more precise, due to the fact that only the significant failure modes were identified in the database.

FMEA was used in [GUI03] for a Tele-echography (TER) system. TER was developed to allow an expert physician (the operator) to move or adjust by hand a scanner virtual probe in an unconstrained way and then the system safely reproduces the movement on the remote patient. The paper presents the risk management general concepts and the holistic functional analysis of the system, and applies FMECA for risk assessment. The

project used some UML elements like Use Case, Sequence diagram, Object diagram, Deployment and Class diagram for the system analysis, but there is no automated model transformation.

In [HAS05] it is proposed a severity assessment methodology by combining three different analysis techniques: Functional Failure Analysis (FFA), Failure Mode and Effect Analysis (FMEA), and Fault Tree Analysis (FTA), but the analysis was not really automated.

In [BOW01] it is shown how FMEA can be applied to a microprocessor based control system. The details of this work are used as a running example in this thesis.

In [KUM13] introduce the so-called functional modeling of the system functions. The identified system functions are used in a parameter diagram to identify the FMEA potential failure modes and failure causes. Furthermore, the parameter diagram information is inputted into a modified FMEA format that links the higher level functions, failure modes and failure causes to the lower level building blocks and associated failure modes.

In [GUI04a] the UML is used as a description language and FMEA is used as a risk analysis technique.

In [OZA04] it is analyzed how FMEA can be better performed at a higher level of software development rather than the very detailed level.

Table 2-5 Summary of FMEA papers using software models and/or model transformation

Paper	Source Model	Target Model	Proposed approach
[DAV09] "Improving reliability studies with SysML."	SysML UML Sequence Diagram and Dysfunctional Database	FMEA	The introduction of Dysfunctional Database
[GUI03] "Integration of UML in human factors analysis for safety of a medical robot for tele-echography,"	UML Diagrams	FMECA	Manual generation of FMECA
[HAS05] "UML Based Severity Analysis Methodology"	UML Diagrams (Use case and Sequence diagram)	FMEA, FFA and FTA	Not specific and target model generation not automated.
[BOW01] "Software Failure Modes and Effects Analysis for a Small Embedded Control System"	Software Flow Chart	FMEA	Functional blocks breakdown
[KUM13] "FMEAs using a functional modeling based approach,"	Functional Model (Functional Diagram)	FMEA	Use of Parameter Diagram (P-diagram)
[GUI04a] UML based risk analysis – Application to a medical robot	UML Use case, Sequence	PHA, FMEA or FTA	Manual generation of the target model.
[OZA04] "Failure Modes and Effects Analysis during Design of Computer Software"	UML Diagrams	SWFMEA	Manual generation of FMEA

3 Chapter: High Level View of the Proposed Approach

This chapter presents the high-level view of the process used for deriving the FMEA model from UML + FMEA Profile model. It describes also the source and target models.

3.1 Overview of the Thesis approach

In order to generate a FMEA model automatically from a UML software model, we have developed a multi-step process shown in Figure 3.1, which includes models transformation and model editing steps.

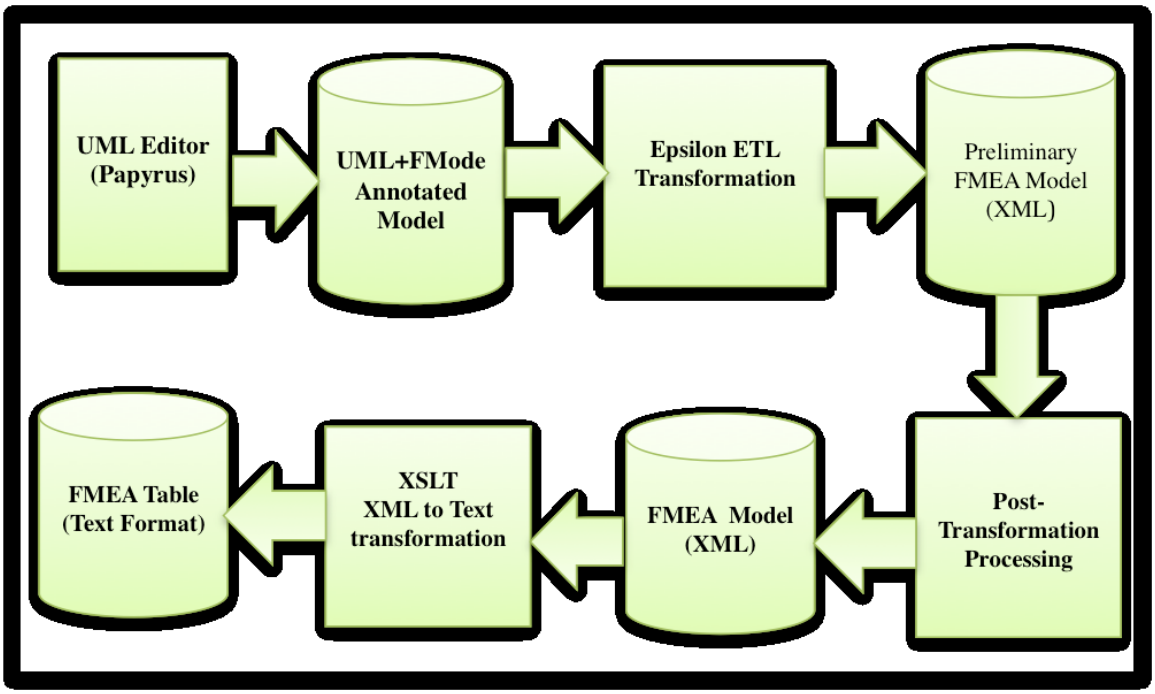

Figure 3-1 The multi-step process for deriving the FMEA model

This process is composed of the following steps:

Step 1. Building the source model: With the help of a standard UML editor, construct the source UML software model and annotate it with failure mode information provided by the safety analyst as described in section 3-3. . In this work we used the open-source UML editor Papyrus [PAP15] whose development is supported by PolarSys [Pola16], an

Eclipse Industry Working Group created by large industry players and tool providers to work together on the creation and support of Open Source tools for the model-driven development of embedded systems.

Step 2. Epsilon ETL transformation: this is the major step of the process, in which the -target FMEA model is generated from the source UML model. The generated model is exported by the transformation tool to XML format. This step bridges the big semantic gap between UML and FMEA models.

Step 3. Post-transformation processing: At this stage some very minor editing of the generated XML file (preliminary FMEA model) is carried out in order to prepare it for the next step, as described in section 4.5.

Step 4. Transformation to text: This step transforms the generated FMEA model from XML to the FMEA table in text format using XSLT, as described in section 4.6.

3.1.1 Fmode Profile Definition

In order to transform a UML model to FMEA, we need to add failure mode information to the UML model elements, such as components, classes and messages. This requires the definition of the Fmode profile shown in Figure 3-3. The most important element of the profile is the Fmode stereotype, which extends several UML metaclasses as shown in the diagram below. The Stereotype Fmode has the following attributes: Name (of type String), Severity (of type SeverityLevel), Occurrence (of type OccurrenceLevel) Detection (of Type DetectionLevel), FailureLocalEffect (of type String) and FailureSystemLevelEffect (of type String). Three enumeration types, SeverityLevel, OccurrenceLevel and DetectionLevel are also defined, corresponding to Tables 2-1, 2-2 and 2-3.

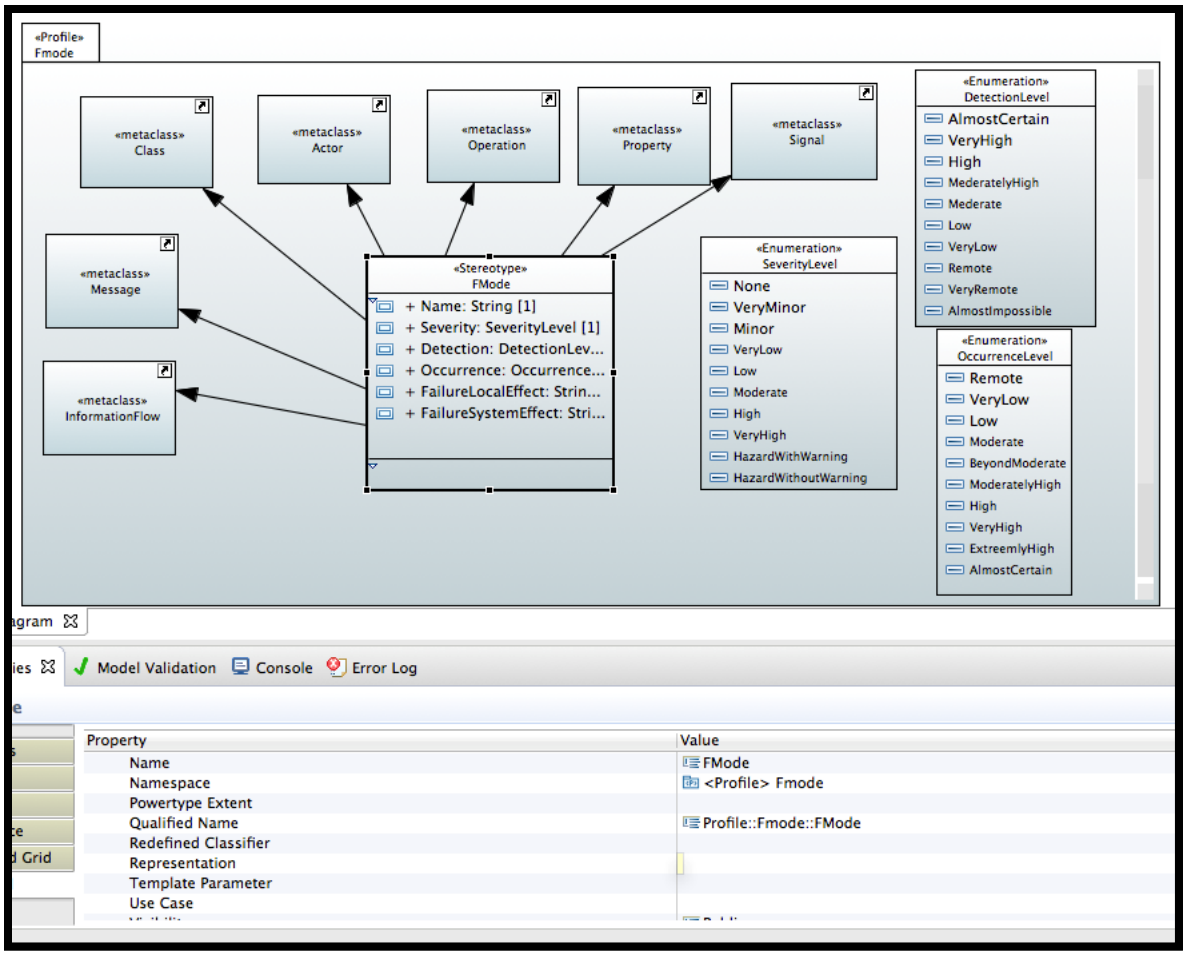

Figure 3-2 Fmode Profile Diagram

3.2 FMEA Domain Model

As already mentioned, FMEA is a classical system safety analysis technique which is widely used in the safety critical industries like aerospace, automotive etc. [PAP04]. The FMEA domain model as shown in Figure3-4 is similar with the domain model proposed in [TAGU11]. This model consists of the following main elements: System, Component, Failure mode, Failure Cause, Failure Effects (Local Failure Effect and System Level Failure Effect), Detection Method, Occurrence and Severity. The mapping of the

elements of the FMEA domain model to UML model elements is as follows: System corresponds to the UML Model package element (the root element), Component corresponds to Part in the Composite Structure and UML Lifeline in the Sequence diagram; Failure Cause corresponds to UML Message in Sequence diagram and Information Flow in Composite Structure diagram. The remaining domain model elements, such as Failure Mode, Failure Effect and Detection Method are specialized to the FMEA domain and cannot be mapped directly to any UML model elements.

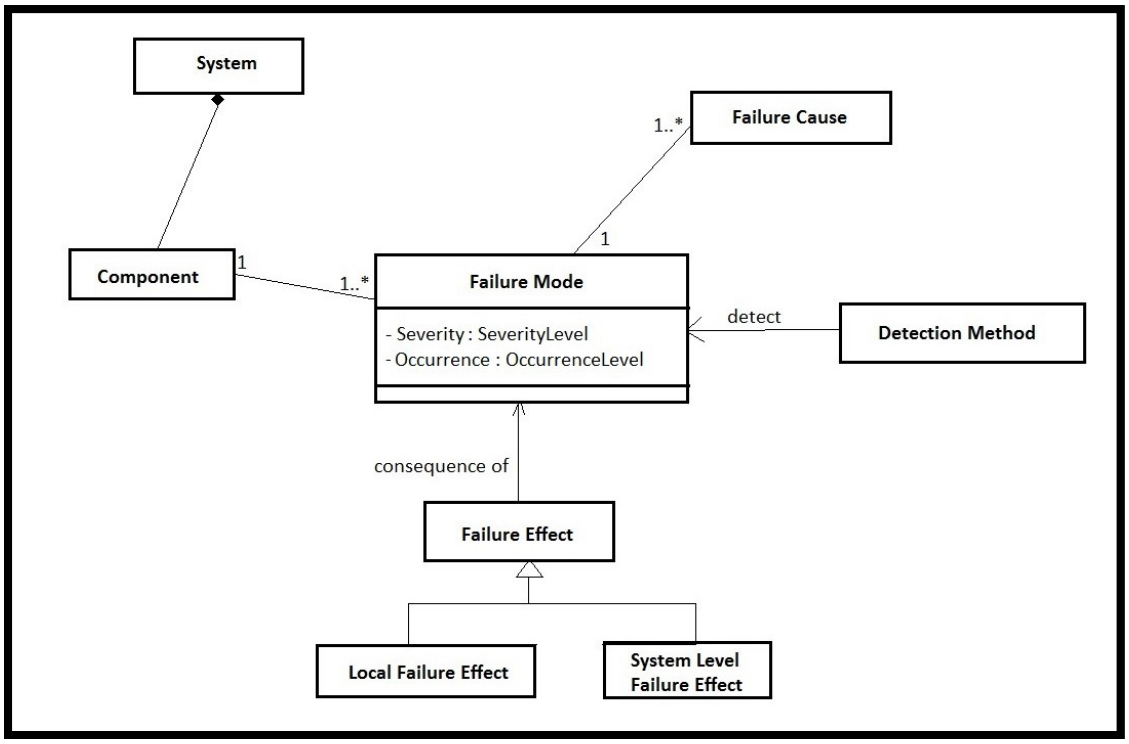

Figure 3-3 FMEA Domain Model Diagram similar to [TAGU11]

Therefore, we defined the *Fmode* profile (see section 3.1.1), which contains the *Fmode* stereotype and its attribute that correspond to these specialized elements. When we need to add failure mode information to a UML model, we apply the *Fmode* profile, as described in section 3.3.

3.3 The Source Model

The source model represents a software system expressed in UML2.5 [UML14] extended with the *Fmode* profile to add failure mode information. The UML diagrams composing the model and the transformed model elements in the transformation rules are described with stereotypes applied to them. This section explains in details how the UML model elements and *Fmode* stereotype are used in the source model, explaining how the source model and its elements are required to be constructed in order to be compatible with the transformation.

The basic UML diagrams contained in a source model for the transformation are a *Composite Structure* diagram representing the system structure and one or more *Sequence* diagrams representing the system behavior, all annotated with the Fmode profile. Figure 3-4 and Figure 3-5 show examples of Composite Structure diagram and Sequence diagrams respectively, both diagrams together form a source model for the transformation developed in the thesis. A UML *Composite Structure* diagram represents the system structural composition based on system's the classes' decomposition and encapsulation that shows the internal collaboration structure of the system's parts (i.e., instance roles linked together by connectors) [UML15]. A UML *Sequence* Diagram is a kind of system behavior diagram that focuses on the Message exchange between a number of Lifelines, it shows the interactions that model the ordered sequences of event occurrences in the system [UML15]. In this thesis, UML Sequence diagram is used to model the flow of interaction and event occurrence that compose a scenario; a model may contain one or more scenarios, each represented by a sequence diagram.

3.3.1 Running Example of a Small Embedded Control System

As a running example to be used for explaining the transformation, we have selected an FMEA case study from literature, in order to reduce the potential "author bias" that arises when using only self-constructed systems. This example is the control system of a ball-in-a-tube system, which has the basic objective of regulating the inflow of air into a tube to constantly suspend a small ball at a predetermined height called the set-point [BOW01]. The height or suspension level of the ball corresponds to the rate of the air inflow; the ball goes up when the flow is stronger, and comes down when is weaker. The system consists of a 3-foot long clear plastic tube, a lightweight ball (ping pong ball), a small electric fan, an infrared sensor circuit for detecting the balls position in the tube, a drive circuit, and an MC68000 micro controller. The block diagram is shown below.

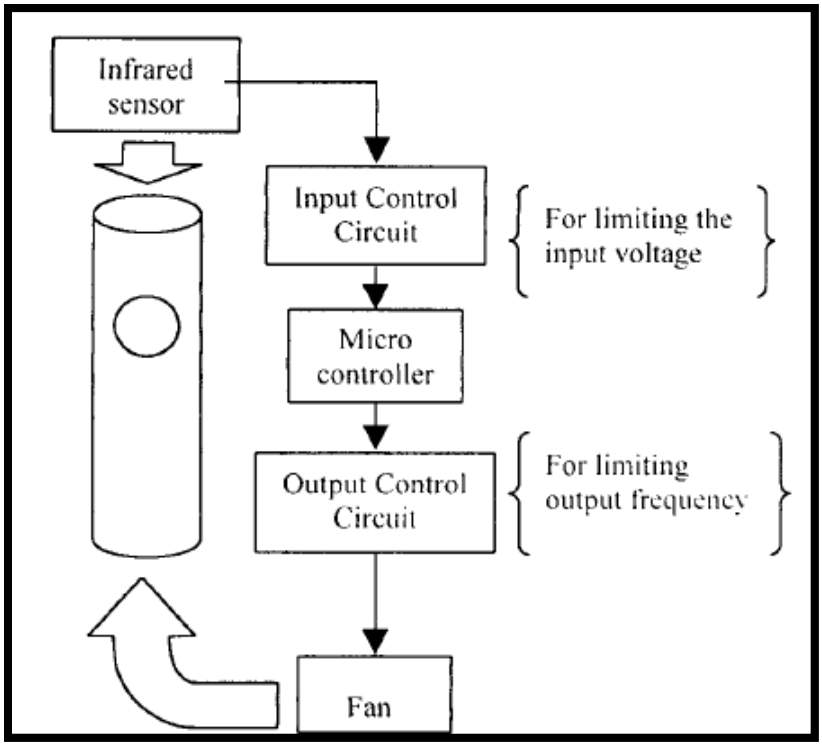

Figure 3-4 The Block Diagram of the close loop control system [BOW01]

As shown above, the height of the ball is measured by the system's infrared sensor. The white color of the ball enhances the infrared light from an emitter to properly reflect from its surface into the system's sensor. The output voltage of the sensor is sampled by the Analog to Digital Converter (ADC) built into the microprocessor. The input voltage received is further converted into a one-byte value by the ADC, which in turn is what determines ball height. Below are the Composite Structure diagram and the Sequence diagram of the system and the generated FMEA by the transformation. Figure 3-8 below represents the ball-in-a-tube control system composite structure diagram showing the internal structure of the system and the application of the *Fmode* Stereotype.

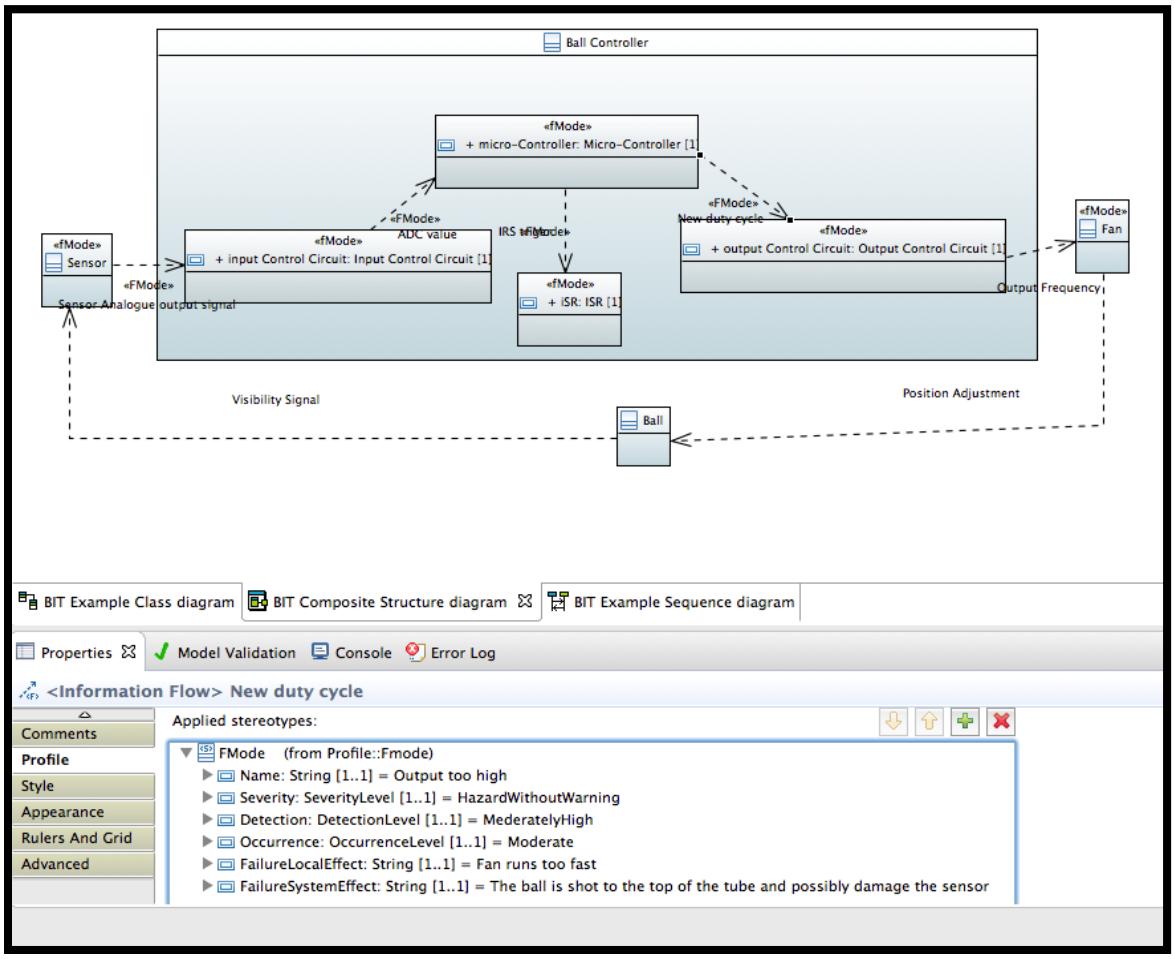

Figure 3-5 The source model composite structure diagram for the running example

Also, Figure 3-9 represents the Sequence diagram of the control system, showing the interaction messages and the Fmode profile application.

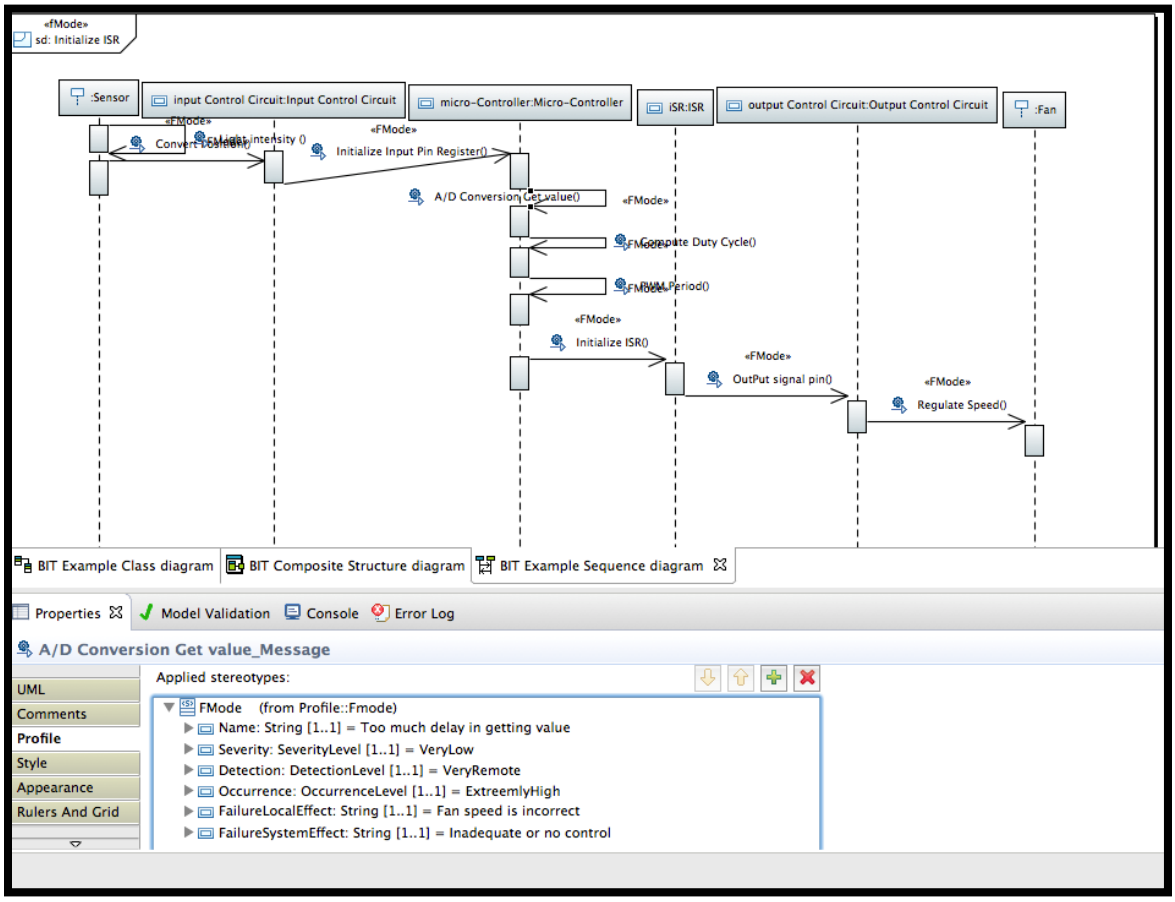

Figure 3-6 The source Model interaction (Sequence) diagram for the running example

The structure model from Figure 3-5 contains a composite structure representing the subsystem *Ball Controller*, which in turn contains the following parts: *Input Control Circuit, Micro-Controller, ISR* and *Output Control Circuit*. The *Ball Controller* interacts with other model elements represented as class instances: *Sensor, Ball* and *Fan*. The failure mode annotations are attached to structural elements via the *Fmode* stereotype. For example, the sub-window at the bottom of Figure 3-5 shows the attribute values of the stereotype *Fmode* applied to the selected *InformationFlow* element called *New Duty*

Cycle. All the stereotype's attributes (*Name, Severity, Detection, Occurrence, FailureLocalEffect, FailureSystemEffect*) are shown in the sub-window at the bottom of the screen shot from Figure 3-6, with the values assigned by the system safety analyst. Each of the stereotype's attributes is transformed to the corresponding target element located on the seventh line in the FMEA table shown in Table 3-1. The *Failure Cause* corresponds to the *InformationItem* conveyed by the *InformationFlow,* while the *Property* (*InformationTarget*) corresponds to the target model *Component*. Attaching failure mode annotations as stereotype to different model elements in the structure and behavior views of the system aids the system safety analyst to be able to focus on the specific context where failures may occur.

Figure 3-6 shows a sequence diagram of the running example, which represents the exchange of messages between the *Lifelines*. Adding failure mode annotation to the diagram helps the safety analyst to focus on the system interaction development. For instance, in Figure 3-6 the attributes of the *Fmode* stereotype applied to the *Message* "A/D Conversion Get Value_Message" are shown within the blue box at the bottom of the figure. Each of the stereotype's attribute is transformed into a column of the corresponding target failure mode element, which is located on the sixth line of the generated FMEA table shown in Table 3-1. Its *Failure Cause* corresponds to the received *MessageOccurenceSpecification* while the *Lifeline* receiving the message corresponds to the target model *Component*.

3.4 The target model

The FMEA Metamodel used for the transformation is shown in the Figure3-7 below. It is a simplified representation of the FMEA domain model from Figure 3-3, where the class *DetectionMethod* has become the *Detection* attribute of *FailureMode*, and the two subclasses of *FailureEffect* have become the *FailureLocalEffect* and *SystemFailureLevel-Effect* attributes of *FailureMode*. Only the class *FailureCause*, which has a one-to-many association with *FailureMode* is represented as a separate class.

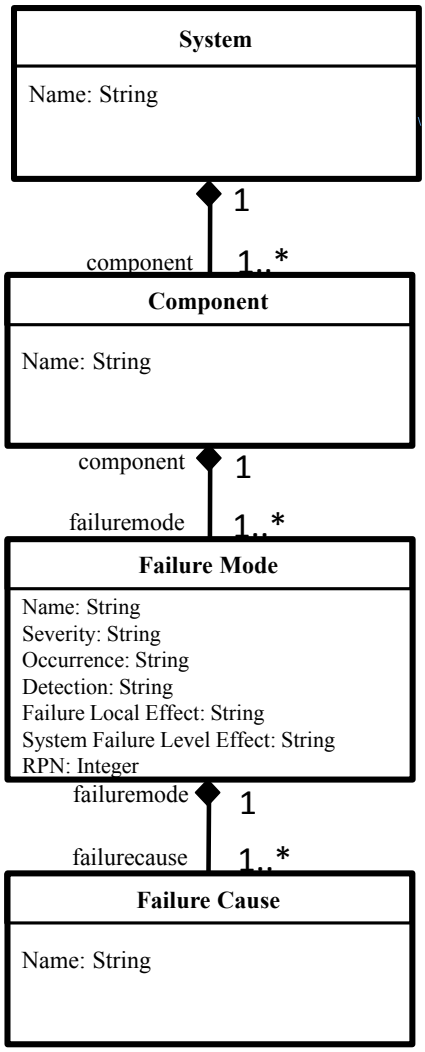

Figure 3-7 Modified FMEA Metamodel Corresponding to the Emphatic description

```
@namespace(uri="FMEA", prefix="")
package System;

class System {
  attr String name;
  val Component[*] component;
}
class Component {
  attr String name;
  val FailureMode[*]#component failuremode;
}
class FailureMode {
  attr String name;
  attr String FailureLocalEffect;
  attr String SystemFailureLevelEffect;
  attr String severity;
  attr String occurrence;
  attr String detection;
  attr int RPN;
  ref Component[0..1]#failuremode component;
  val FailureCause[*]#failuremode failurecause;
}
class FailureCause {
  attr String name;
 ref FailureMode[1]#failurecause failuremode;
}
```

Figure 3-8 The Emphatic description of the FMEA metamodel

Figure3-8 is the description in the language Emphatic of the FMEA metamodel in class diagram form, as required by the Epsilon transformation. It shows the metamodel main package, the classes, their attributes and types and also the relationships and association role names. Figure 3-9 below shows the mapping of the FMEA metamodel represented as a class diagram to its corresponding Emphatic description.

Figure 3-9 Mapping between the FMEA Metamodel represented as class diagram and the corresponding Emphatic description

The outcome of the Model-to-Model transformation from UML to FMEA is shown in the screen shot in Figure3-13 below. The UML model can be exported to XML format, an intermediate format [LI14] that shows the FMEA objects, the relationships and the attributes (as shown in the Appendix).

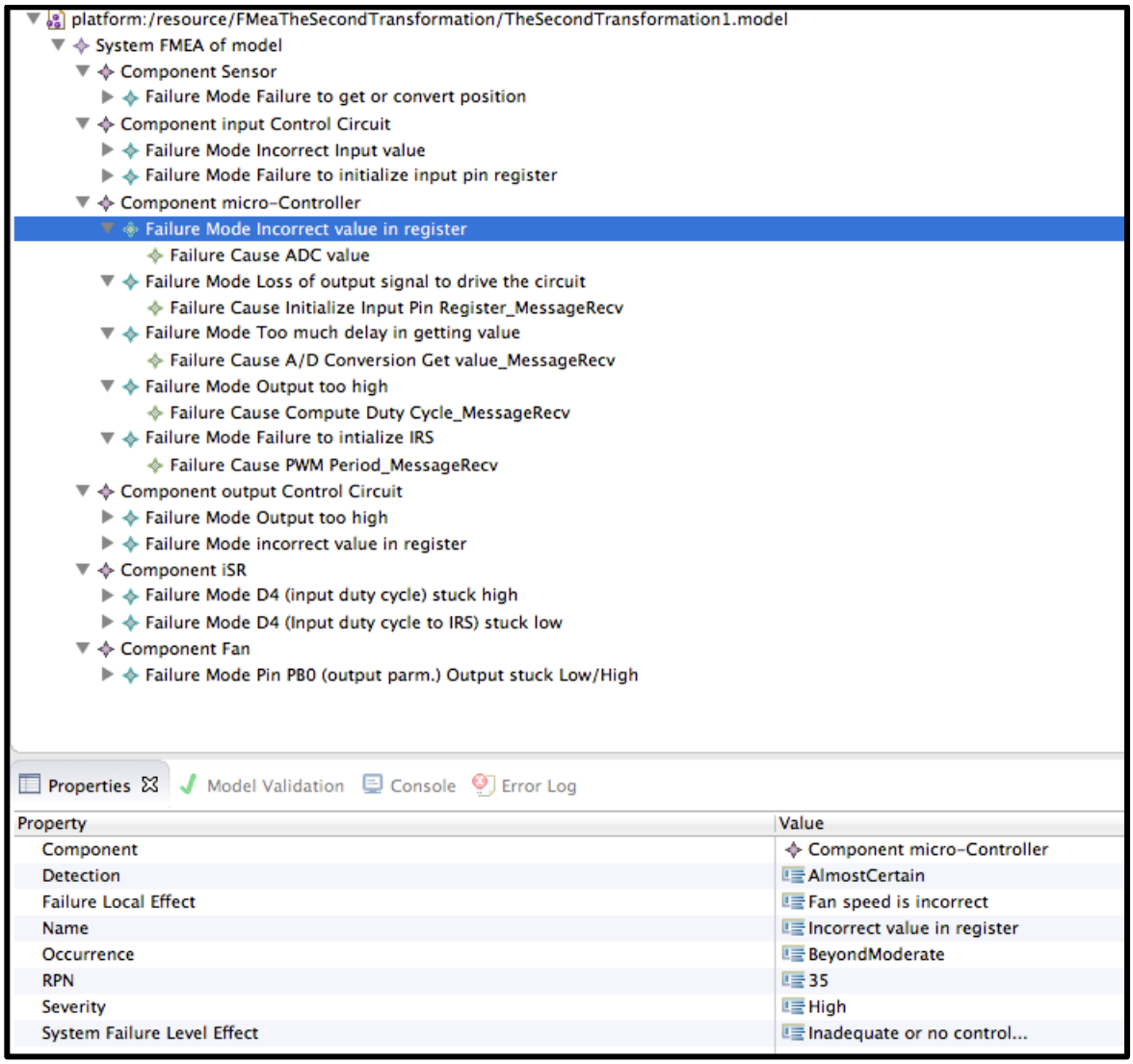

Figure 3-10 The FMEA Target model produced by the transformation

The Table below shows the output FMEA table generated from the auto-generated FMEA XML file (shown in Appendix A1) generated from the UML model. The component column shows that a component can have more than one Failure mode, each one with its own set of elements: *Local Effect, System Level Effect, Severity, Occurrence, Detection, RPN,* and *Failure Cause*.

Table 3-1The FMEA table of Ball in a Tube control system

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Sensor	Failure to get or convert position	Fan speed does not change	Inadequate or no control	Hazard without Warning	Moderate	Almost Certain	40	Light intensity _MessageRecv
Input Control Circuit	Incorrect Input value	Can not read Input	Ball falls to the bottom of the tube	Very High	Very Low	Low	96	Sensor Analogue output signal
	Failure to initialize input pin register	Fan does not run	Ball falls to the bottom of the tube	Hazard without Warning	Beyond moderate	Remote	400	Convert Position_MessageRecv
Micro-Controller	Incorrect value in register	Fan speed is incorrect	Inadequate or no control	High	Beyond moderate	Almost Certain	35	ADC value
	Loss of output signal to drive the circuit	Fan does not run or system does not respond	Ball falls to the bottom of the tube	High	Very High	Moderately High	224	Initialize Input Pin Register_MessageRecv
	Too much delay in getting value	Fan speed is incorrect	Inadequate or no control	Very Low	Extremely High	Very Remote	324	A/D Conversion Get value_MessageRecv
	Output too high	Fan runs too fast	Ball is shot to the top of the tube and possibly damages sensor	Hazard With Warning	Moderately High	Very Low	378	Compute Duty Cycle_MessageRecv
	Failure to initialize IRS	Can not run IRS	Ball falls to the bottom of the tube	High	Low	Very Low	147	PWM Period_Message Recv
Output Control Circuit	Output too high	Fan runs too fast	The ball is shot to the top of the tube and possibly damage the sensor	Hazard without Warning	Moderate	Moderately High	160	New duty cycle
	Incorrect value in register	Incorrect output	Inadequate or no control	Low	Moderate	Very Remote	180	OutPut signal pin_MessageRecv
ISR	D4 (input duty cycle) stuck high	Fan runs too fast	Ball is shot to the top of the tube and possibly damages sensor	High	Low	Moderate	21	IRS triger
	D4 (Input duty cycle to IRS) stuck low	Fan dose not run or runs too slow	Ball falls to bottom of tube	H azard without Warning	Moderately High	Remote	480	Initialize ISR_MessageRecv
Fan	Pin PB0 (output parm.) Output stuck Low/High	Fan run incorrectly (fast if stuck high and too slow or dose not run at all if stuck low)	Ball shot to the top of the tube or ball falls to the bottom of the tube (if stuck low)	Low	Moderate	Remote	160	Regulate Speed_MessageRecv

4 Chapter: Design and Implementation of the Transformation

This chapter presents the design and implementation of the transformation. The design starts by deciding the mapping between the source and target models. The implementation is the description of the ETL code and the effect of each rule and operation. Figure 4-1 below shows the mapping of the source model elements with the target model.

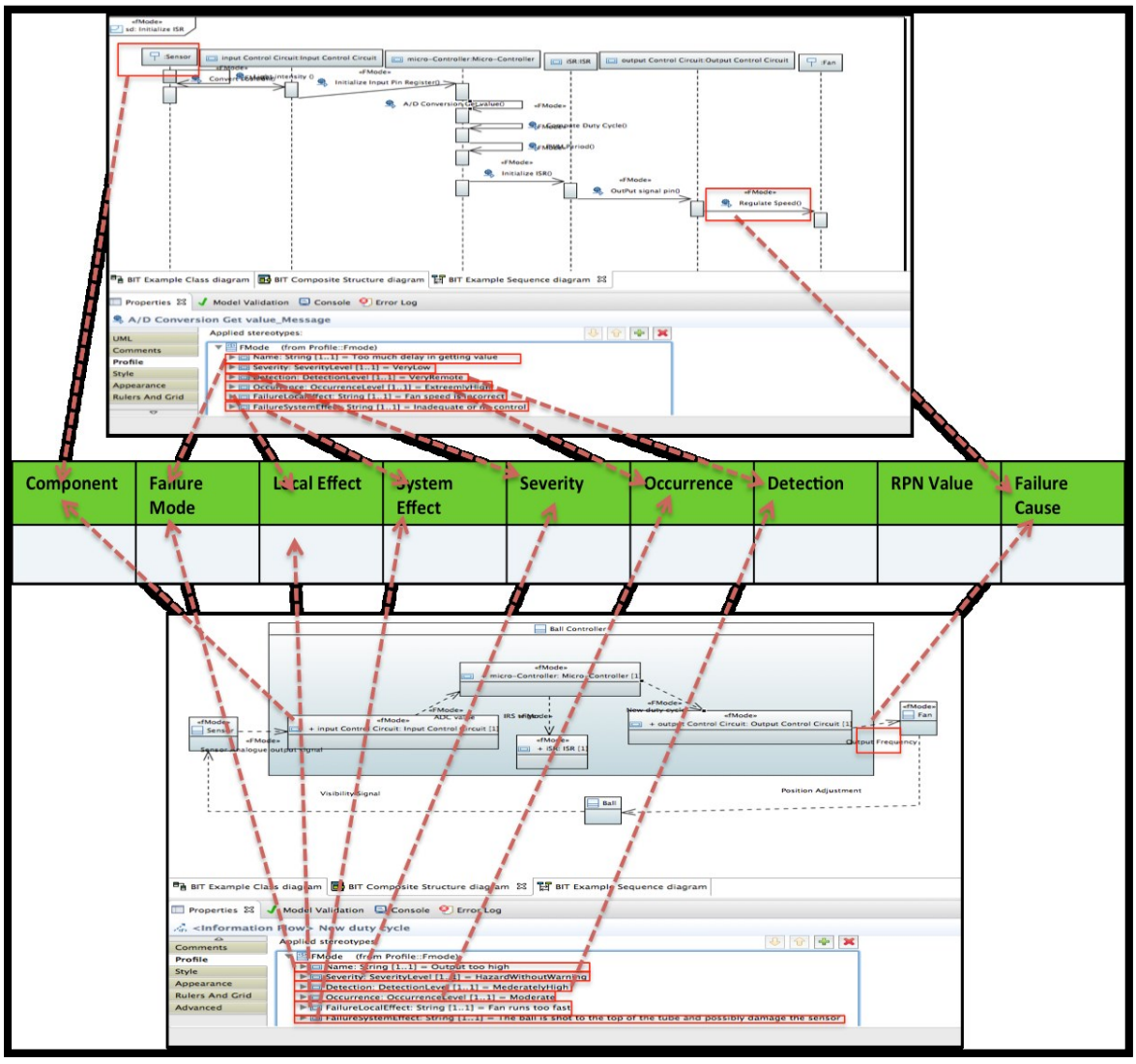

Figure 4-1Mapping between the source and target model

4.1 Mapping between the Source and Target Elements

At the beginning of the design of the transformation, we need to decide what elements of the target model are obtained from what elements of the source model. Table 4-1 shows this mapping.

Table 4-1 Mapping Table

UML Model Element	FMEA Stereotype, attribute	FMEA Element
Model	None	FMEA Model
Property (Name)	Fmode	Component
InformationFlow	Fmode(Name)	Failure Mode
InformationFlow	Fmode (FailureSystemEffect)	SystemFailureLevelEffect
InformationFlow	Fmode(FailureLocalEffect)	FailureLocalEffect
InformationFlow	Fmode (Severity)	Severity
InformationFlow	Fmode (Occurrence)	Occurrence
InformationFlow	Fmode (Detection)	Detection
InformationItem	None	FailureCause
Lifeline	None	Component
Message	Fmode(Name)	FailureMode
Message	Fmode (FailureSystemEffect)	SystemFailureLevelEffect
Message	Fmode(FailureLocalEffect)	FailureLocalLevelEffect
Message	Fmode (Severity)	Severity
Message	Fmode (Occurrence)	Occurrence
Message	Fmode (Detection)	Detection
MessageOccurrence Specification	None	FailureCause

4.2 Rules and Operations

This thesis transformation has been designed in an ETL module, which consists of all the transformation rules and operations used to carry out all source-to-target model mappings given in Table 4-1. The rule table (Table 4-2) below shows the FMEA elements that are created for the source elements discovered in the input model. Each operation described in the Table 4-3 below represents a function called by the rules or by other operations during the transformation. An operation verifies a set of conditions and returns the corresponding results.

The Epsilon Transformation Language (ETL) is one of the Epsilon task specific language [KOL15] that can be used in the transformation of an arbitrary number of input models into an arbitrary number of output models of different modeling languages and technologies at a high level of abstraction. ETL is a hybrid kind of transformation language, as it provides both a declarative rule-based execution scheme, as well as imperative features for handling complex transformation scenarios [KOL15]. ETL rule may be either "Matched" or "Lazy" rules, where a Matched rule corresponds to the declarative style, while a Lazy rules corresponds to the imperative style. From the code fragments that show the rules syntax the logic for connecting each generated target element with another is defined within each rule. The containment association relationships are used to link the target elements together. The first rule *Model2System* is a Matched rule, which transform the root element (*UML Model*) of the source model to the target's (FMEA Model) root element (*System*). *System* in turns has a containment relationship with the target's element *Component*. The target element *Component* has a containment relationship with the target element *Failure Mode*, which in turn has a

containment relationship with the target element *Failure Cause*. The generated FMEA *Failure Cause* has to be added as a *child* to the target element *Failure Mode*. The matched rule *Informationflow2FailureMode* and *Message2FailureMode* invoke the lazy rules *InformationItem2FailureCause* and *MessageOccurrenceSpecification2Failure Cause* to generate a target *Failure Cause* for each target's *Failure Mode* element found in the input model, and then adds the newly created target elements to the corresponding container *FMEA model element*.

Table 4-2 Table of Rules in the UML to FMEA Transformation

Rule Name	Description
Model2System	Transform UML Model Element (source root) to FMEA System Element (target root)
Lifeline2Component	Transform UML Lifeline to FMEA Component Element
Property2Component	Transform UML Composite Structure Property (a part of the composite structure) to FMEA Component Element
Message2FailureMode	Transform UML Sequence Diagram message to FMEA Failure Mode Element
InformationFlow2FailureMode	Transform UML Composite Structure Diagram Information Flow to FMEA Failure Mode Element
MessageOccurrenceSpecification2FailureCause	Transform UML Sequence Diagram Message Occurrence Specification to FMEA Failure Cause Element
InformationItem2FailureCause	Transform UML Composite Structure Diagram Information Item to FMEA Failure Cause Element

Table 4-3Table of Operations in the UML to FMEA transformation

Operation	Description
MessageEndisReceive ()	For the given UML Message defined in Sequence diagram, returns true and the message if the message occurrence specification event is a receive event
MessageEndisSend ()	For the give UML Message defined in Sequence diagram, returns true and the message if the message occurrence specification event is a send event
MessagehasStereotype ()	For the given UML Message defined in Sequence diagram, returns the String value of all the properties of its FMode stereotype
MessageOccurrenceSpecificationisThisReceiveMessage ()	For the given UML Message Occurrence Specification defined in Sequence diagram, returns true if the Message Occurrence specification event is a ReceivedMessage.
MessageOccurrenceSpecificationisThisSendMessage ()	For the given UML Message Occurrence Specification defined in Sequence diagram, returns true if the Message Occurrence specification event is a SendMessage.
InformationFlowhasStereotype()	For the given UML Information Flow defined in Composite Structure diagram, returns the String value of all the properties of its FMode stereotype
LifelinehasStereotype ()	For the given UML Lifeline defined in Sequence diagram, returns true if the Lifeline possessed the FMode stereotype
PropertyhasStereotype ()	For the given UML Class property defined in Composite Structure diagram, returns true if the Property possessed the FMode stereotype
operation ReturnDetectionValue ()	Returns the equivalent integer value of the target Detection string value
operation ReturnSeverityValue()	Returns the equivalent integer value of the target Severity string value
operation ReturnOccurrenceValue( )	Returns the equivalent integer value of the target Occurrence string value

4.3 Detailed description of Rules

4.3.1 Rule Model2System

The *System* element is the top element of the target FMEA model. The *System* element serves as the root container of the target model. *System* corresponds to the source root container, the UML *Model* element. Just as shown in Code Fragment 4-1 below, the UML *Model* element, which is the root container of the source model, indicated in the rule line “transform”, is transformed to the top FMEA model element *System*.

The rule contains a target element preceded with the keyword “to”, which is labeled with the variable *Fmea*. This is a variable that represents the generated target node of type *System*. The assignment statement involving the target node *Fmea*:

Fmea.name = "FMEA of "+m.name

shows that the *Fmea.name* attribute is initialized with a string composed of “FMEA of” to which the name of the source *Model* appended. The operation *isDefined ()* is invoked on the variable *i* that represents the UML *Lifeline* element of the source model, which returns a Boolean result. For each *true* returned, the call to *i.equivalent()* returns the FMEA model element *Component* transformed from *i*, which is then added to the collection *Fmea.component*. The operation *equivalent()* is a built-in ETL operation that resolves the target elements that have been transformed from its corresponding source elements by other rules in the transformation [KOL15]. Figure 4-2 illustrates the transformation of the source model element name to the corresponding target model element name.

```
rule model2system
        transform m: UML!Model
        to Fmea: FMEA!System {
        Fmea.name = "FMEA of "+m.name;
        var Lifeline:Any = UML!Lifeline.all;
        for (i in Lifeline) {
              if (i.isDefined()){
              Fmea.component.add( i.equivalent());
            }
        }
}
```

Code Fragment 4-1 Rule Model2System

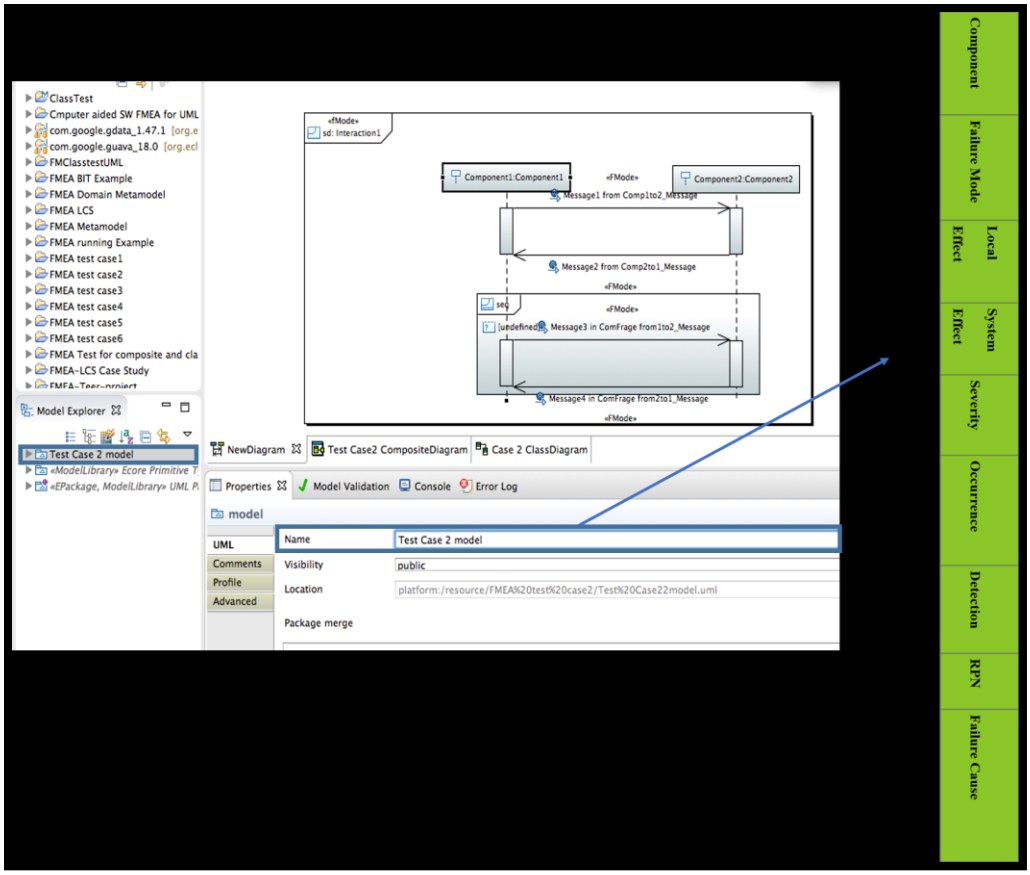

Figure 4-2 Transformation rule example: Rule Model2System

4.3.2 Rule Lifeline2Component

Each *Lifeline* element from the UML Sequence diagram of the source model is mapped to a *Component* element of the target FMEA model as depicted Code Fragment 5-4 and shown in Figure 4-3 and Figure 4-4.

Figure 4-3 Transformation rule example: Rule Lifeline2Component

In this transformation, the name of a target *Component* will be initialized to the name of its corresponding source UML *Lifeline* or UML *Property* (i.e. *Lifeline.represents*). The ETL built-in operation *isDefined()* is invoked on the variable *l.represents* (see Figure 4-3)

```
rule lifeline2component
      transform l: UML!Lifeline
      to C: FMEA!Component
      {
      C.name = l.name;
      var o:Collection;
              if (l.represents.isDefined()){
                  var Inf:Any = InformationFlow;
                        var In : String;
                        for (i in Inf) {
                         if
                         (i.target.selectOne(a|a.name=l.represents.name)
                         .isDefined()){
                              C.failuremode.add( i.equivalent());
                              }
                        }
                  }

                  if (l.name.isDefined()){

                  var Inf:Any = Message;

                  var In : String;

                    for (i in Inf) {

                        if  ( i.receiveEvent.covered.selectOne

                        (a|a.name=l.name).isDefined()){

                              C.failuremode.add( i.equivalent());

                        }

                    }

                  }

}
```

Code Fragment 4-2 Rule Lifeline2Component

and *l.name* which refer to UML *Property* and UML *Lifeline.name,* respectively, to return a Boolean result. For each *true* returned, the call *.equivalent()* returns the FMEA model element *Failure Mode* transformed from both UML *InformationFlow* and UML *Message*, which is then added to the collection *FC.failuremode* .

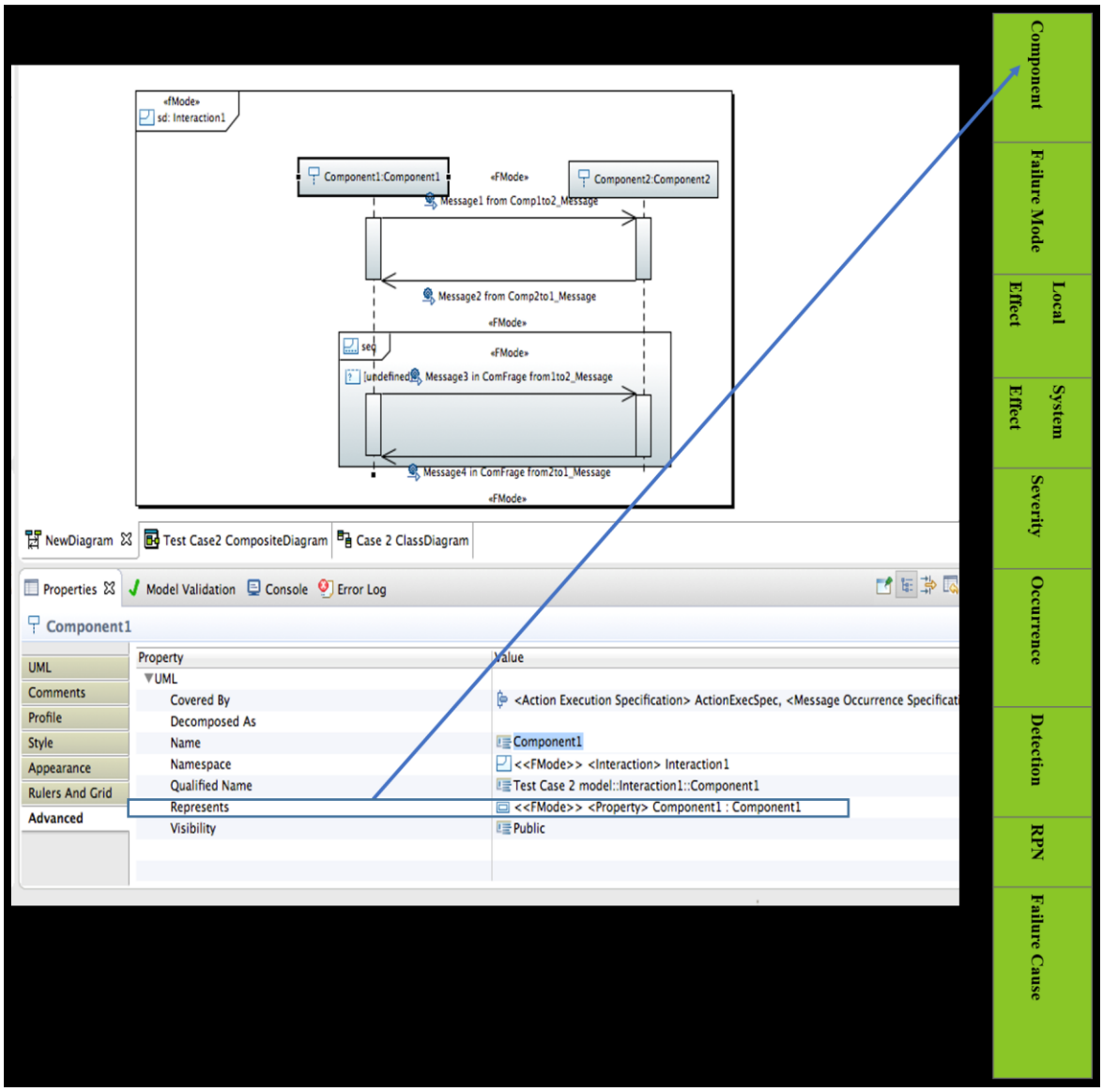

Figure 4-4 Transformation rule example: Lifeline2Component with reference to the represented Composite structure Property

4.3.3 Rule InformationFlow2FailureMode

This rule transforms the source model element UML *InformationFlow* from the Composite Structure diagram to the target component *Failuremode*, as it can be seen in Code Fragment 4-3 and Figure 4-5. The "guard" section of the rule checks for the UML *InformationFlow* stereotyped with *Fmode*. If there are UML *InformationFlow* in the source model that are not stereotyped with the *Fmode* stereotype, those elements will not be considered in the transformation rule. Every transformation rules that include a "guard" section will be applicable only to the elements that satisfy the stated guard's condition. Furthermore, operation *getAllAttributes()* retrieves all the attributes of the *Fmode* stereotypes applied to *InformationFlow*, which includes *name, Occurrence, Severity, Detection, FailureLocalEffect* and *FailureSystemEffect* and transforms them to the corresponding target model element *FailureMod*e attributes: *name, Occurrence, Severity, Detection, FailureLocalEffect* and *FailureSystemLevelEffect* respectively, as shown in the Code Fragment 4-3 and Figure 4-5, which illustrates the source elements to target elements transformation. The *RPN* column corresponds to the product of the integer value of *Severity, Occurrence and Detection* of each *FailureMode* element. The conversion of these three attributes from string value to integer value is performed with the operations *ReturnOccurrencevalue(), ReturnServerityValue()* and *ReturnDetectionValue()*, respectively.

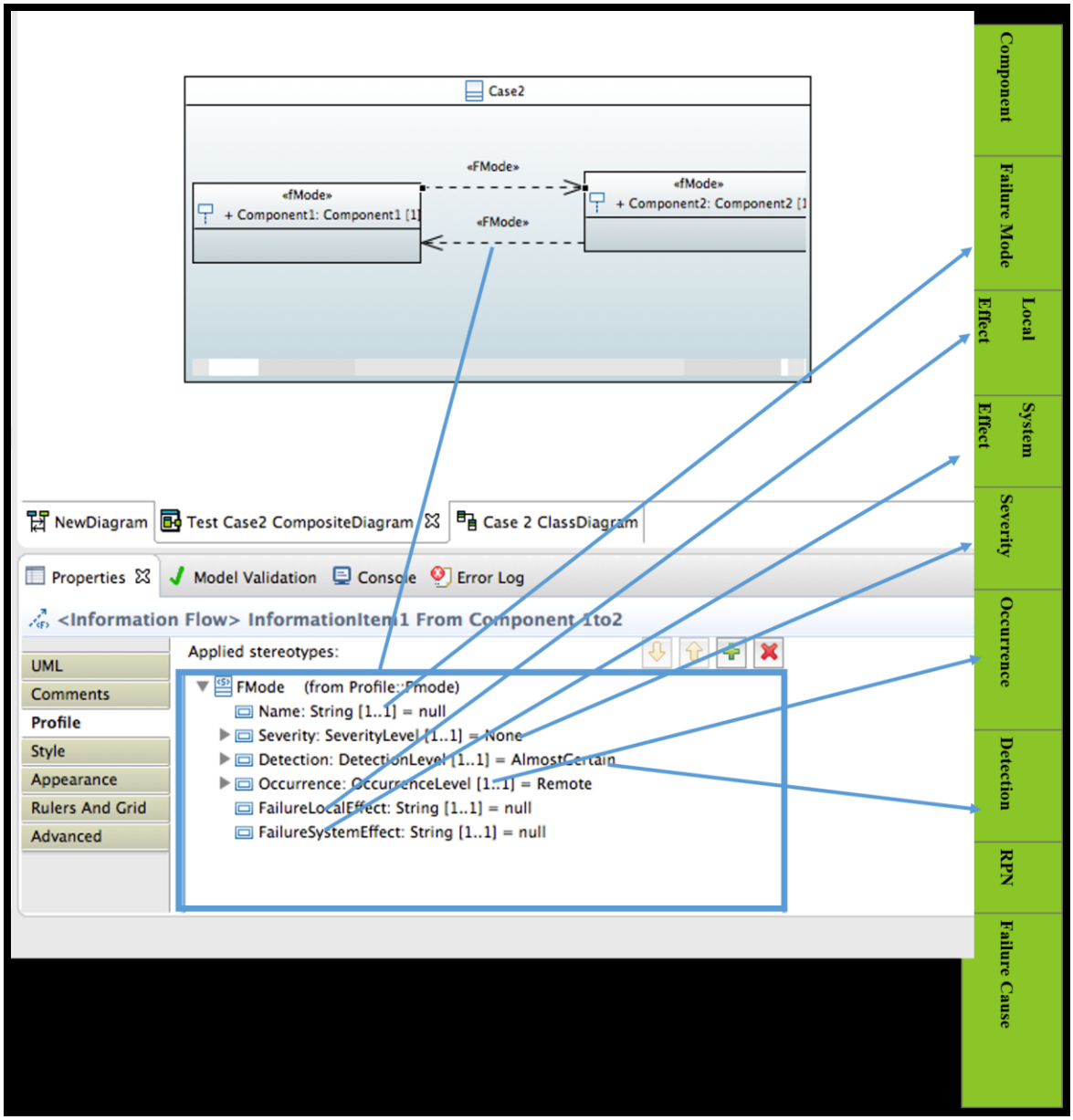

Figure 4-5 Transformation rule example: InformationFlow2FailureMode (Name, Effects, Severity, Occurrence and Detection)

```
rule informationflow2FailureMode
transform I :UML!InformationFlow
to FM : FMEA!FailureMode {
        guard:I.hasStereotype("FMode")
        var hasSte:Any = I.hasStereotype("FMode");
var stereotype:Any=I.getAppliedStereotypes();
if(stereotype.isDefined()){
    for(s in stereotype){
        if(s.name="FMode"){
            if(s.getAllAttributeS().selectOne(a|a.name="FailureLocalEffect").isDefined()){
            FM.FailureLocalEffect=I.getValue(s,"FailureLocalEffect").asString();
            if(s.getAllAttributeS().selectOne(a|a.name="FailureSystemEffect").isDefined())
            {FM.SystemFailureLevelEffect=I.getValue(s,"FailureSystemEffect").asString()
            ;
            if(s.getAllAttributeS().selectOne(a|a.name="Name").isDefined()){
            FM.name=I.getValue(s,"Name").asString(); }
            if(s.getAllAttributeS().selectOne(a|a.name="Occurrence").isDefined()){

            FM.occurrence =I.getValue(s,"Occurrence").name.asString();}
            if(s.getAllAttributeS().selectOne(a|a.name="Severity").isDefined()){
            FM.severity=I.getValue(s,"Severity").name.asString(); }
            if(s.getAllAttributeS().selectOne(a|a.name="Detection").isDefined()){
            FM.detection=I.getValue(s,"Detection").name.asString();}
            var Det:Integer =1;
            var Sev:Integer=1;
            var Occ:Integer;
            Sev=ReturnSeverityValue(FM.severity);
            Det=ReturnDetectionValue(FM.detection);
            Occ=ReturnOccurrenceValue(FM.occurrence);

        FM.RPN= Det*Sev*Occ;}
        }
    }
    var Item :Any = I.conveyed;
      for (i in Item){

        FM.failurecause.add(i.equivalent());
          }
}
```

Code Fragment 4-3 Rule InformationFlow2FailureMode

4.3.4 Rule Message2FailureMode

This rule is structured in a similar manner to the above rule *InformationFlow2Failure Mode,* as shown in the code fragment 4.4 and illustrated in Figure 4-6. It transforms the source model element UML *Message* from a sequence diagram to the target element *FailureMode*. Also, the "guard" section of the rule checks whether the source element *Message* is stereotyped with *Fmode*. If there are messages in the source model that are not stereotyped with *Fmode*, such messages will be ignored by the transformation rule.

If the Stereotype "*Fmode*" is applied to the UML *Message* and its attributes are defined, the operation *getAllAttributes()* returns the value of the following attributes: *name, Occurrence, Severity, Detection, FailureLocalEffect and FailureSystemEffect*. For every source model element *Message*, these attributes are transformed to the corresponding target model element *FailureMod*e attributes: *name, Occurrence, Severity, Detection, FailureLocalEffect* and *FailureSystemLevelEffect*, respectively. Here as well, the *RPN* corresponds to the product of the integer value of *Severity, Occurrence* and *Detection* of the corresponding target element *FailureMode*. The integer values of *Severity, Occurrence,* and *Detection* are obtained by calling the operations *ReturnServerityValue(), ReturnOccurrencevalue()* and *ReturnDetectionValue()*. These operations when invoked return the integer value corresponding to the string value of the respective element.

```
rule Message2FailureMode
        transform op : UML!Message
        to FM: FMEA!FailureMode{
        guard:op.hasStereotype("FMode")
         var hasSte:Any = op.hasStereotype("FMode");
         var stereotype:Any=op.getAppliedStereotypes();
         if(stereotype.isDefined()){
              for(s in stereotype){
                 if(s.name="FMode"){
                          if(s.getAllAttributeS().selectOne(a|a.name="Occurrence").isDefined())
                          { FM.occurrence =op.getValue(s,"Occurrence").name.asString();}
                          if(s.getAllAttributeS().selectOne(a|a.name="FailureLocalEffect").isDefined())
                          {FM.FailureLocalEffect=op.getValue(s,"FailureLocalEffect").asString();}
                          if(s.getAllAttributeS().selectOne(a|a.name="FailureSystemEffect").isDefined(
                          ){FM.SystemFailureLevelEffect=op.getValue(s,"FailureSystemEffect").asStri
                          ng();}
                           if(s.getAllAttributeS().selectOne(a|a.name="Name").isDefined()){
                           FM.name=op.getValue(s,"Name").asString();}
                          if(s.getAllAttributeS().selectOne(a|a.name="Severity").isDefined()){
                          FM.severity=op.getValue(s,"Severity").name.asString();}
                           if(s.getAllAttributeS().selectOne(a|a.name="Detection").isDefined()){
                          FM.detection=op.getValue(s,"Detection").name.asString(); }
                    var Det:Integer ;
                    var Sev:Integer;
                    var Occ:Integer;
                    Sev=ReturnSeverityValue(FM.severity);
                    Det=ReturnDetectionValue(FM.detection);
                    Occ=ReturnOccurrenceValue(FM.occurrence);
               FM.RPN= Det*Sev*Occ;
                          }
                 }
        }

         var Item :Any = op.receiveEvent;
        for (i in Item){
        FM.failurecause.add(i.equivalent());
        }
}
```

Code Fragment 4-4 Rule Message2FailureMode

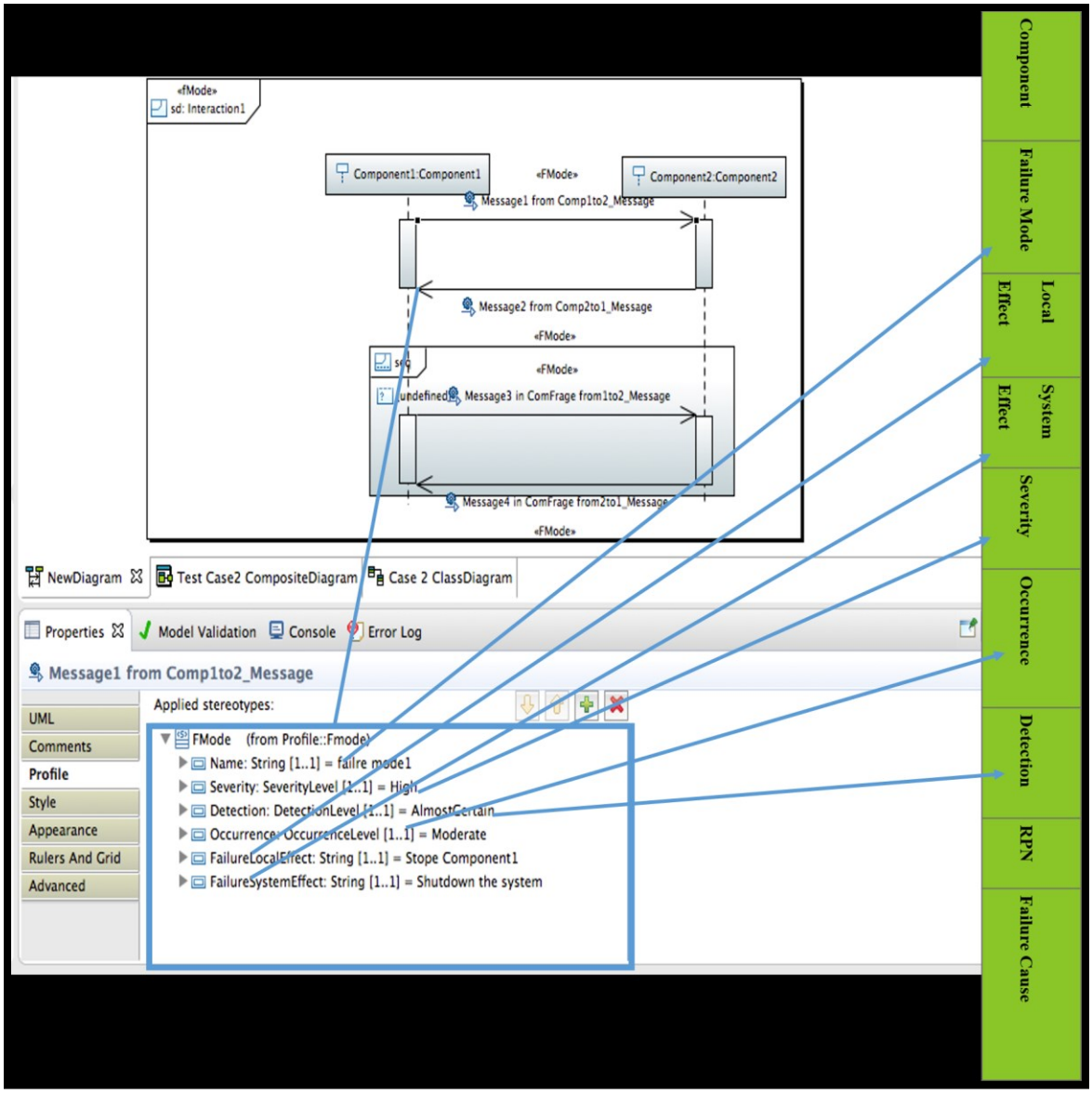

Figure 4-6 Transformation rule example: Message2FailureMode (Name, Effects, Severity, Occurrence and Detection)

4.3.5 Rule InformationItem2FailureCause

This rule transforms the source model element *InformationItem* to the corresponding target model element *FailureCause*. The assignment statement *FC.name = In.name* shows how the "*FC.name*" attribute is initialized with the *name* of the source *InformationItem* element, as shown in Code Fragment 4.5 and Figure 4-7.

```
@lazy
rule informationItem2Failurecause
      transform In :UML!InformationItem
      to FC : FMEA!FailureCause {
            FC.name=In.name;
}
```

Code Fragment 4-5 Rule InformationItem2Failurecause

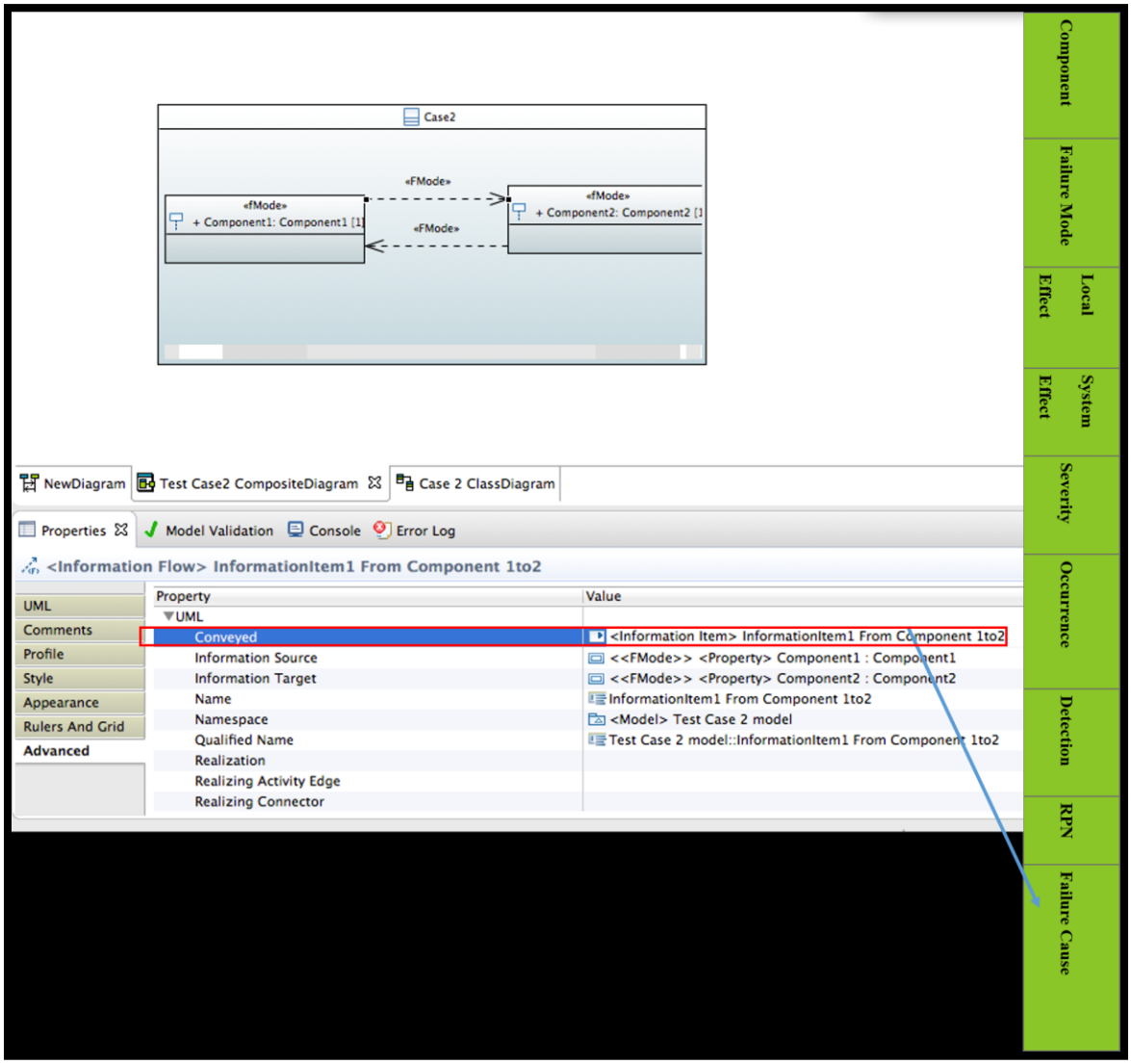

Figure 4-7 Transformation rule example: InformationItem2Failurecause

4.3.6 Rule MessageOccurrenceSpecification2FailureCause

This rule transforms the source model element *MessageOccurrenceSpecification* to the corresponding target model element *Failurecause*. In this rule the "guard" statement section of the rule checks the source element *MessageOccurrenceSpecification* invoking the operation *M.isThisReceiveMessage()*. If there are any *MessageOccurrence Specification* elements for which the guard condition is false, they will be ignored by the transformation rule.

The assignment statement *FC.name = M.name* shows that the *FC.name* attribute is initialized with the name of the source element MessageOccurrenceSpecification, as shown in Code Fragment 4-6 and Figure 4-8.

```
@lazy
rule MessageOccurrenceSpecification2FailureCause
          transform M:
UML!MessageOccurrenceSpecification
          to FC: FMEA!FailureCause {
            guard: M.isThisReceiveMessage()
            FC.name=M.name;
}
```

Code Fragment 4-6 Rule MessageOccurrenceSpecification 2Failurecause

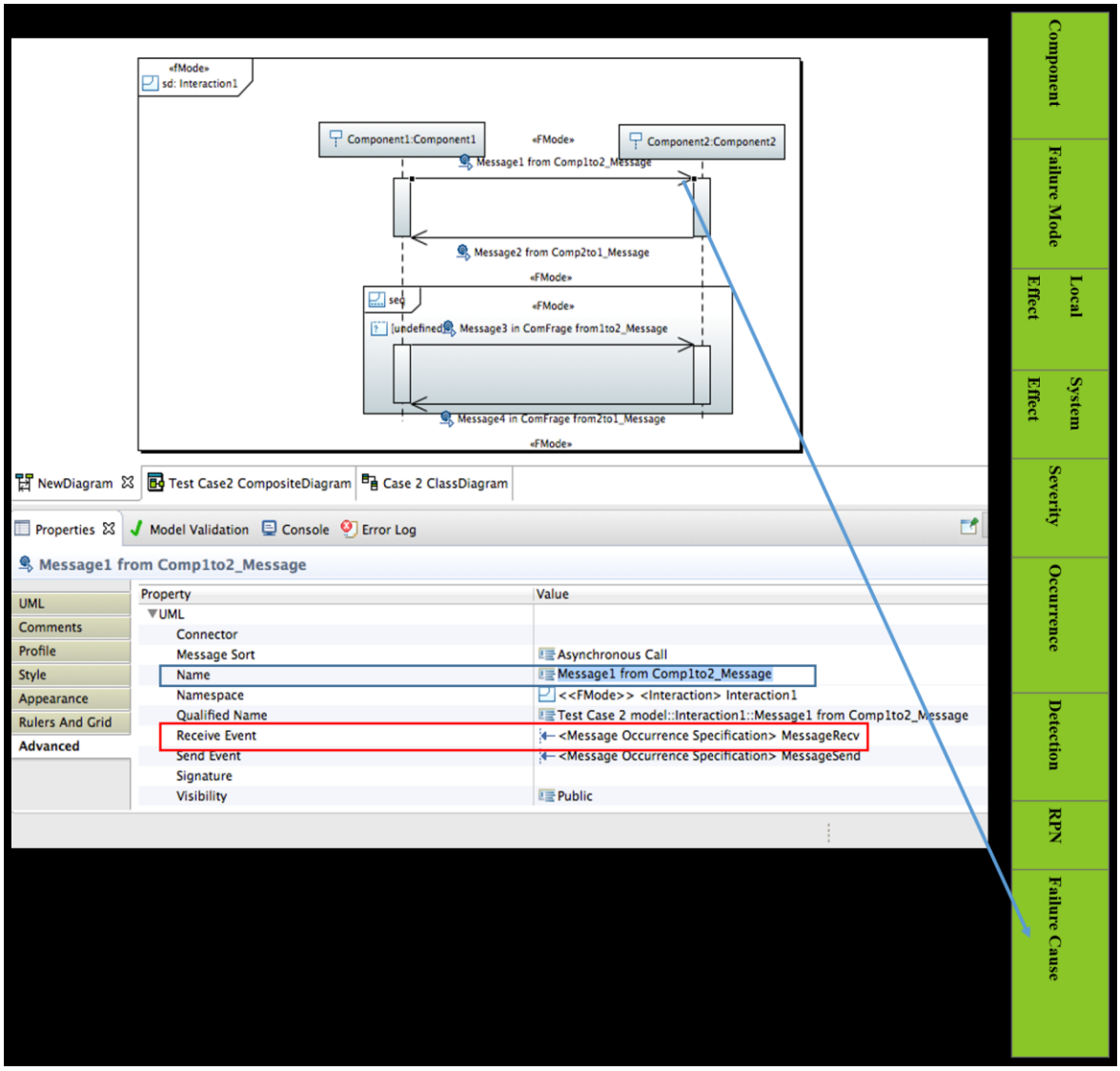

Figure 4-8 Transformation rule example: MessageOccurrenceSpecification2Failurecause

4.4 Detailed descriptions of Operations

4.4.1 Operation hasStereotype()

This operation is called in the context of a *Message* of the source model and returns a Boolean type value. As we can see in the Code Fragment 4.7, the ETL built-in operation *getAppliedStereotypes()* is invoked on the variable *self* that refers to the context of this operation. *Operation getAppliedStereotypes()* returns a collection containing all the

stereotypes that have been applied to the respective Message. For each Stereotype instance in this collection, if the stereotype's attribute name matches the String parameter passed as an argument to this operation, the operation will return *true*.

```
operation UML!Message hasStereotype(name:String):Boolean{
          var c: Collection;
          c=self.getAppliedStereotypes();
                for (s:Stereotype in c){
                    s.println();
               if(s.name=name){
               return true;         }
      }
          return false;
}
```

Code Fragment 4-7 Operation MessagehasStereotype()

4.4.2 Operation MessageOccurrenceSpecificationisThisReceiveMessage()

This operation is called in the context of the source element *MessageOccurrenceSpecification* as mentioned above to check first whether the context element is a message. If it is a received message, then the operation returns true, otherwise it returns false.

```
operation MessageOccurrenceSpecificationisThisReceiveMessage():
Boolean{
    if(self.message.isDefined())
            if(self.isSend())
               return false;
            else if(self.isReceive())
               return true;
}
```

Code Fragment 4-8 Operation MessageOccurrenceSpecificationisThisReceiveMessage()

4.4.3 Operation MessageEndisSend()

This operation is invoked to check whether a *MessageEnd* corresponds to a send event, in which case the operation returns *true*, otherwise it returns *false*.

```
operation MessageEndisSend() : Boolean {
 return self.message.sendEvent.asSet().includes(self);
}
```

Code Fragment 4-9 Operation MessageEndisSend()

4.4.4 Operation MessageEndisReceive()

This operation is called in the context of the source element *Message* to check if *MessageEnd* is a received message, in which case the operation returns true, otherwise it returns false.

```
operation MessageEnd isReceive() : Boolean {
return self.message.receiveEvent.asSet().includes(self);
}
```

Code Fragment 4-10 Operation MessageEndisReceive()

4.4.5 Operation InformationFlowhasStereotype()

Here is another kind of *hasStereotype* operation named *InformationFlowhasStereotype*, which is called in the context of an *InformationFlow* element from the source model. The *getAppliedStereotypes()* returns a collection containing all the stereotypes that have been applied to the respective *InformationFlow* element. For every Stereotype instance in this

collection it is checked whether the attribute name matches the *String* parameter passed as an argument. If the condition is true for one stereotype, the operation will return *true*.

```
operation UML!InformationFlowhasStereotype(name:String):Boolean{
            var c: Collection;
            c=self.getAppliedStereotypes();
            for (s:Stereotype in c){
                  if(s.name=name){
                  return true;
                        }
                  }
            return false;
}
```

Code Fragment 4-11 Operation InformationFlow HasStereotype()

4.4.6 Operation PropertyhasStereotype()

This is similar to operation *MessagehasStereotype()* described above, only that it is called in the context of a UML *Property* of the source model. The *getAppliedStereotypes()* returns a collection containing all the stereotypes that have been applied to the respective *Property*. For at least one Stereotype instance in this collection, if the stereotype's attribute name matches the String parameter passed as an argument to this operation, the operation will return *true*.

```
operation PropertyhasStereotype(name:String):Boolean{
            var c: Collection;
            c=self.getAppliedStereotypes();
                  for (s:Stereotype in c){
                  if(s.name=name){
                  return true;
                        }
                              }
            return false;}
```

Code Fragment 4-12 Operation UML!Propert hasStereotype()

4.4.7 Operation ReturnOccurrenceValue()

This operation is called in the context of a target element *Occurrence* to check if the *Occurrence* String value is defined. If true, it returns the corresponding integer value of that string value, as shown in Code Fragment 4-13 conditional statements.

```
operation ReturnOccurrenceValue(FMoccurrence:String):Integer{
        if (FMoccurrence ="Remote"){return 1;
        }else if(FMoccurrence="VeryLow"){
                return 2;
        }else if(FMoccurrence="Low"){
                return 3;
        }else if(FMoccurrence="Moderate"){
                return 4;
        }else if(FMoccurrence="BeyondModerate"){
                return 5;
        }else if(FMoccurrence="ModeratelyHigh"){
                return 6;
        }else if(FMoccurrence="High"){
                return 7;
        }else if(FMoccurrence="VeryHigh"){
                return 8;
        }else if(FMoccurrence="ExtreemlyHigh"){
                return 9;
        }else if(FMoccurrence="AlmostCertain"){return 10;
        }
}
```

Code Fragment 4-13 operation ReturnOccurrenceValue()

4.4.8 Operation ReturnSeverityValue()

This is another operation similar to the aboved operation. This operation is called in the context of a target element *Severity* to check if its string value is defined. If true, then it returns the corresponding integer value of that string as in the Code Fragment 4-14 conditional statements.

```
operation ReturnSeverityValue(FMseverity:String):Integer {
        if (FMseverity ="None") {return 1;
        }else if(FMseverity="VeryMinor"){
                return 2;
        }else if(FMseverity="Minor"){
                return 3;
        }else if(FMseverity="VeryLow"){
                return 4;
        }else if(FMseverity="Low"){
                return 5;
        }else if(FMseverity="Moderate"){
                return  6;
        }else if(FMseverity="High"){
                return 7;
        }else if(FMseverity="VeryHigh"){
                return 8;
        }else if(FMseverity="HazardWithWarning"){
                return 9;
        }else if(FMseverity="HazardWithoutWarning"){return 10;
        }
}
```

Code Fragment 4-14 operation ReturnSeverityValue()

```
operation ReturnDetectionValue(FMdetection:String):Integer{
        if (FMdetection ="AlmostCertain") { return 1;
        }else if(FMdetection="VeryHigh"){
                return 2;
        }else if(FMdetection="High"){
                return 3;
        }else if(FMdetection="MederatelyHigh"){
                return 4;
        }else if(FMdetection="Moderate"){
                return 5;
        }else if(FMdetection="Low"){
                return 6;
        }else if(FMdetection="VeryLow"){
                return 7;
        }else if(FMdetection="Remote"){
                return 8;
        }else if(FMdetection="VeryRemote"){
                return 9;
        }else if(FMdetection="AlmostImpossible"){return 10;
        } }
```

Code Fragment 4-15 operation ReturnDetectionValue()

4.4.9 Operation ReturnDetectionValue()

This operation is called in the context of the target element *Detection* to check if the *Detection* string value is defined. If true, it returns the corresponding integer value of that String value, as shown in the Code Fragment 4-15.

4.5 Post-transformation processing

The result of the ETL transformation is a preliminary FMEA model in XML format, meant to be an input to an XSLT transformation that will transform it to the FMEA table. But there is a need to manually edit the auto-generated XML file in order to delete the version and path "*xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA*"" generated with the FMEA XML file, so that the XML file can be processed by XSLT.

4.6 Generating the FMEA Table (XSLT transformation)

Producing the FMEA table is the final phase of the FMEA transformation. We decided to use the Extensible Stylesheet Language Transformations (XSLT), which is a transformation language for translating XML documents/files to another document format or another form of XML format.

XSLT can be referred to as a "turing-complete" language, that is, XSLT can specify any computation that can be performed by a computer [KEP04]. XSLT was used in this thesis to transform the XML file of the generated FMEA model to text format (FMEA table), where the input document is the FMEA XML document. In [BOS11] the author showed how XSLT transformations could be mapped to any XML Schemas to generated ontologies automatically, without subsequent manual adaptations. The Code Fragment 4-

16 below shows the XSLT transformation for the generated FMEA XML document to FMEA table, specifying the columns and the headings of the table.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
  <h1 style="text-align:center">The <xsl:value-of select="System/ @name"/></h1>
  <table border="1">
   <tr bgcolor="#9acd32">
    <th>Component</th>  <th> Failuremode  </th> <th>Local Effect</th> <th>System Effect</th>
    <th>Severity</th> <th>Occurrence</th> <th>Detection</th> <th>RPN Value</th>
    <th>Failure Cause</th> </tr>
   <xsl:for-each select="System/component">
   <tr>
    <td><xsl:value-of select=" @name"/></td>
    <td><xsl:for-each select="failuremode">
      <xsl:value-of select=" @name"/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@FailureLocalEffect"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@SystemFailureLevelEffect"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@severity"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@occurrence"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@detection"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode">
      <xsl:value-of select="@RPN"/><br/><br/><br/><br/><br/>
     </xsl:for-each></td>
     <td><xsl:for-each select="failuremode/failurecause">
      <xsl:value-of select="@name"/><br/><br/><br/><br/>
     </xsl:for-each></td>
    </td>
    </td><xsl:for-each>
     </tr>
   </tr>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Code Fragment 4-16 FMEA Text-to-Text Transformation code (XSLT)

5 Chapter: Verification and Validation

This section of the thesis presents different test cases used for the verification and validation of the transformation process and developed rules. Also, two case studies from [DAV10] and [GUI03] were used in order to mitigate the possible bias that could arise when using only self-built products for verification and validation.

5.1 Test Cases

In accordance with the IEEE standard IEEE-STD-610, software verification can be defined as the process of evaluating software to determine if the products of a given development phase satisfy the conditions imposed at the start of that phase, while software validation is referred to as the process of evaluating software during or at the end of the development process to determine whether it satisfies the specified requirements. The transformation processes and rules developed in this thesis have being verified by executing different level of tests using different test cases. A number of important transformation features were identified and test cases were used to verify the effectiveness (satisfaction the imposed conditions) of transformation, to check whether it produced the expected output.

Table 5-1 Test cases

Test Cases	Tested Features
Test case 1	Simple message and Execution Occurrence
Test case 2	Combined Fragment
Test case 3	Nested Combined Fragment
Test case 4	More Than One Interaction Diagram
Test case 5	Composite Structure and an Interaction diagram
Test case 6	Composite Structure and 2 Interaction Diagram
Case study 1	The Level Control System (LCS) [DAV10]
Case study 2	Robotic Tele-Echography (TER) Project [GUIO03]

5.1.1 Test case 1

For this test case, the motive is to start the testing with the simplest level: simple messages and execution occurrence transformation to verify if all basic elements are correctly transformed. Figure 5-1 shows the sequence diagram of test case1 model, which contains two messages and two execution occurrences, and two lifelines. The FMEA model generated from the above UML model is shown in Figure 5-2.

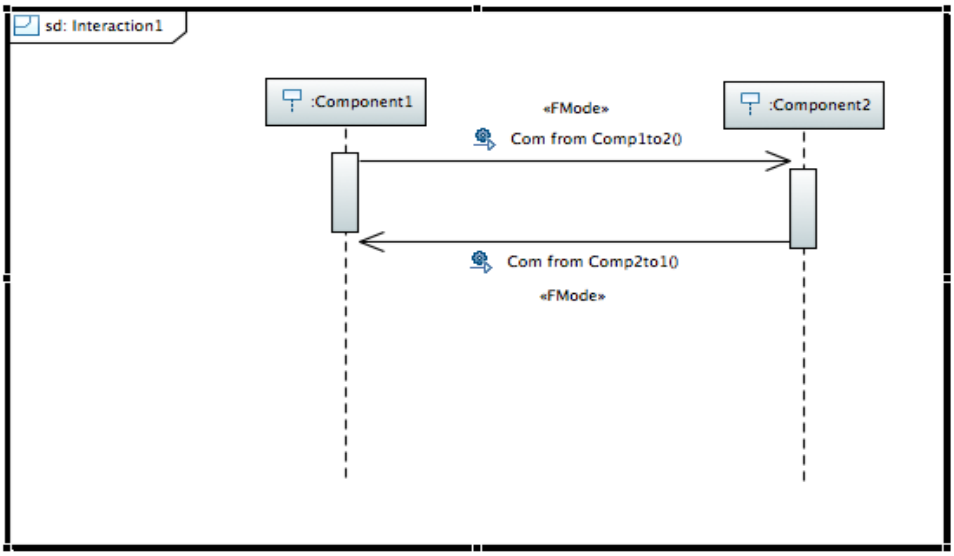

Figure 5-1 Test Case 1 Sequence Diagram

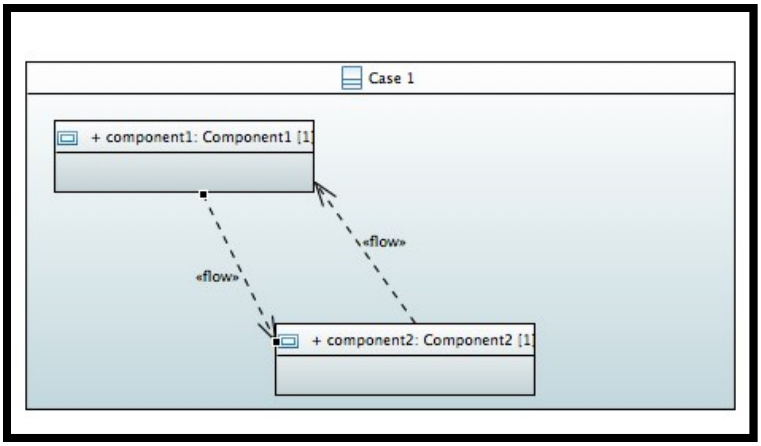

Figure 5-2 Test Case 1 Composite Structure Diagram

Table 5-2 The FMEA table for Test Case 1

Component	Failure Mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN	Failure Cause
Component1	Component1 Failure mode	Affect component 2	shutdown the system	Low	Beyond Moderate	High	75	Com from Comp2to1_MessageRecv
Component2	Component2 Failure mode	Affects Component 1	No effect on the system	Very Minor	Very Low	Almost Certain	4	Com from Comp1to2_MessageRecv

5.1.2 Test Case 2

Test Case2 is another simple example, but bigger than Test Case 1. It consists of just a sequence diagram with a combined fragment to test the capability of the transformation process to handle such a sequence diagram. Figure 5-3 below shows the sequence diagram and Figure 5-4 the structure diagram for test case 2.

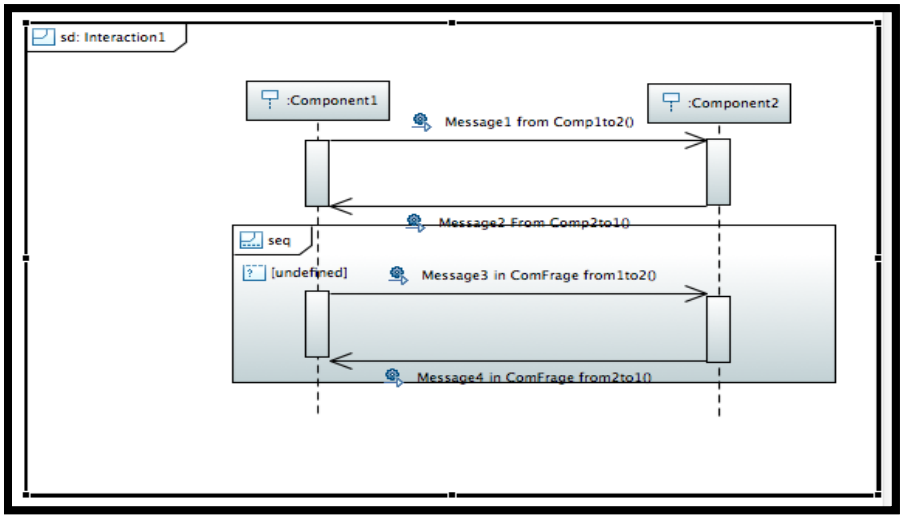

Figure 5-3 Test Case 2 Sequence diagram

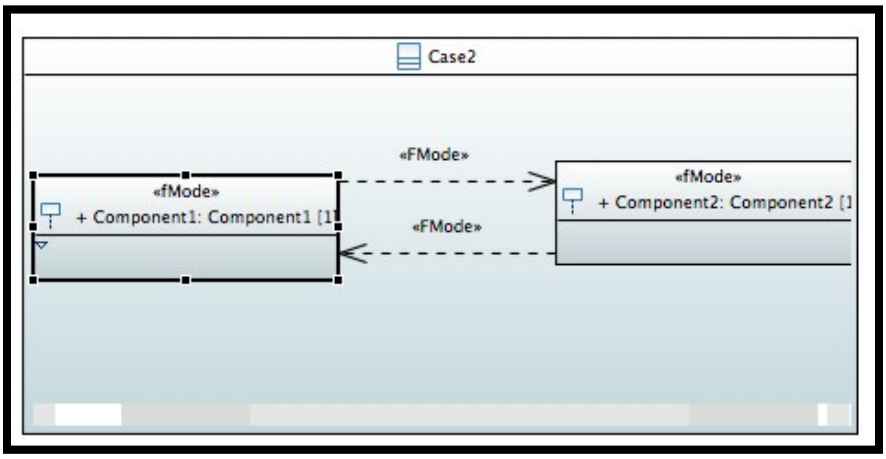

Figure 5-4 Test Case 2 Composite Structure Diagram

Table 5-3 The FMEA table for Test Case2

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Component 1	Failure mode2	No effect	None	None	Remote	Almost Certain	1	Message0Recv
	Failure mode4	Low Effect on Component2	Low Effect	Very Minor	Very Low	Almost Certain	4	Message2Recv
Component 2	Failure mode1	Stop Component1	Shutdown the system	High	Moderate	Almost Certain	28	MessageRecv
	Failure mode3	Affects Componenet 1	Affects the system	Moderate	Remote	Very High	12	Message1Recv

5.1.3 Test Case3

This is a more complex test case verifying the transformation process ability to transform a nested combined fragment. Figure 5-5 shows the sequence diagram of a nested combined fragment (Test Case 3).

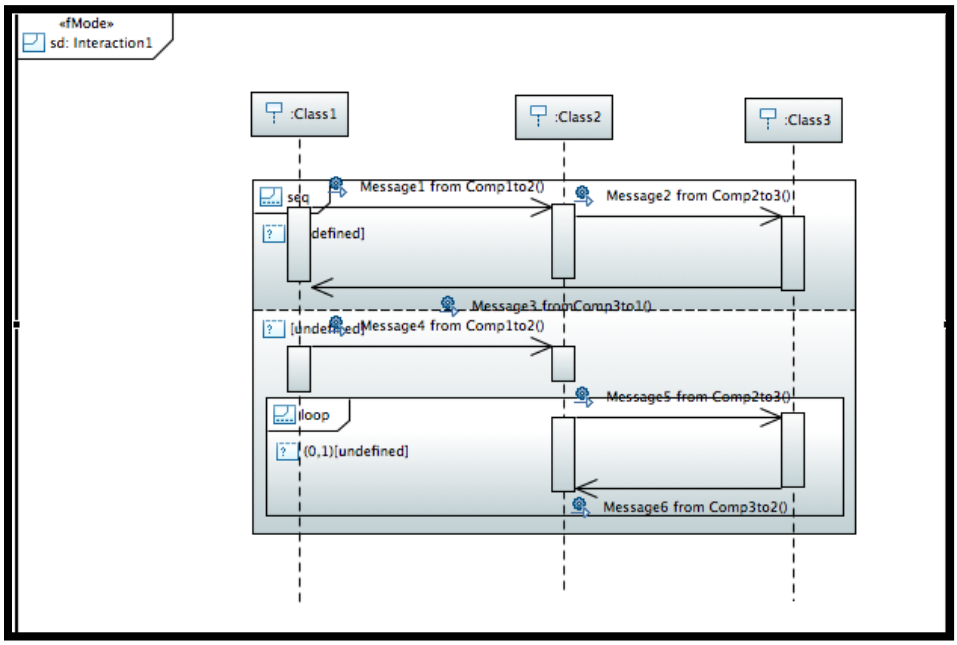

Figure 5-5 Test Case 3 Sequence diagram

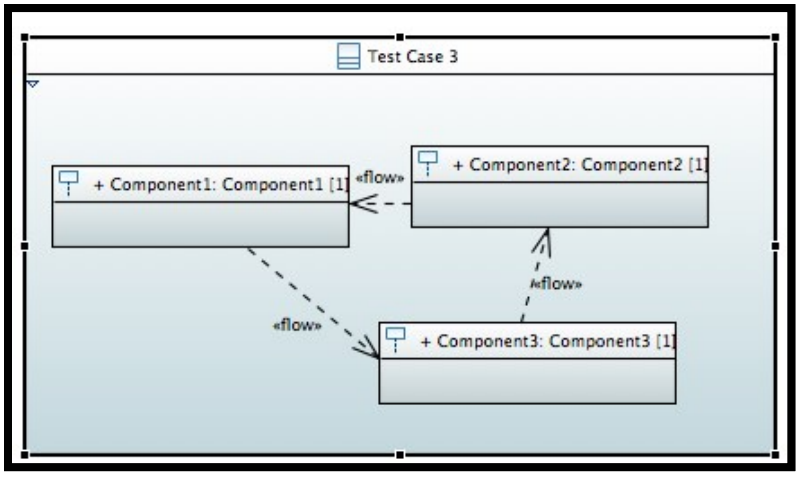

Figure 5-6 Test Case 3 Composite Structure Diagram

Table 5-4 The FMEA of Test case3

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Component 1	Failure Mode 3	Affect component 2	Low effect	None	Low	Moderately High	12	Message3 fromComp3to1 _MessageRecv
Component 2	Failure Mode 1	Affect other component	Affect the system	None	High	Moderate	35	Message1 from Comp1to2_MessageRecv
	Failure Mode 4	Moderate effect on other component	Affect the system	Moderate	Beyond Moderate	Almost Certain	30	Message4 from Comp1to2_MessageRecv
	Failure Mode 6	Affect other component	Moderate effect on the system	Very Low	Remote	Low	24	Message6 from Comp3to2_MessageRecv
Component 3	Failure Mode 2	Affect component 1	Low	None	Moderately High	Almost Certain	6	Message2 from Comp2to3_MessageRecv
	Failure Mode 5	Affect other component	Shut down the system	Hazard With Warning	Moderate	Almost Certain	36	Message5 from Comp2to3_MessageRecv

5.1.4 Test Case4

Here we carry out the testing of the transformation process to determine how well it transforms a source model with more than one interaction (sequence diagram).

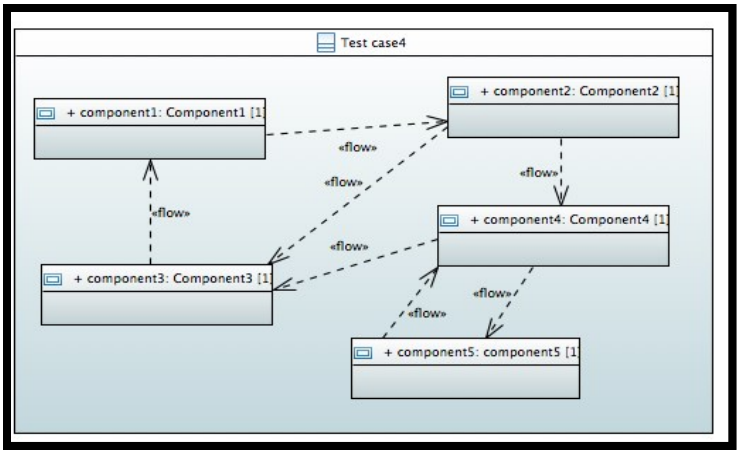

Figure 5-7 Test Case 3 Composite Structure Diagram

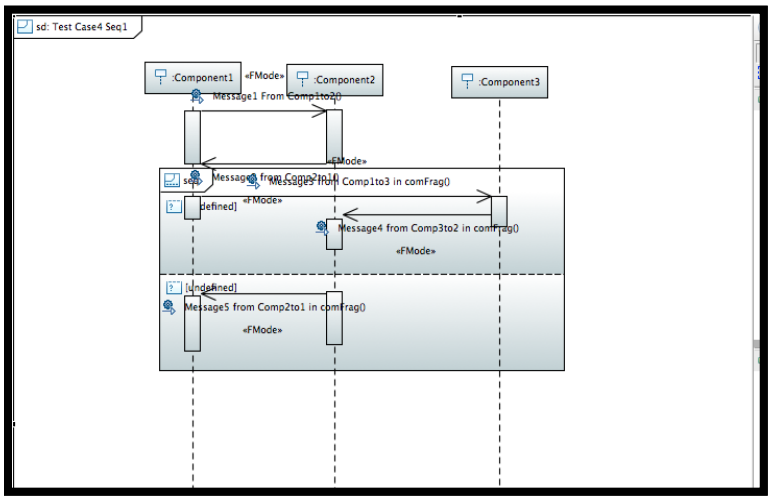

Figure 5-8 Test Case4 Sequence diagram1

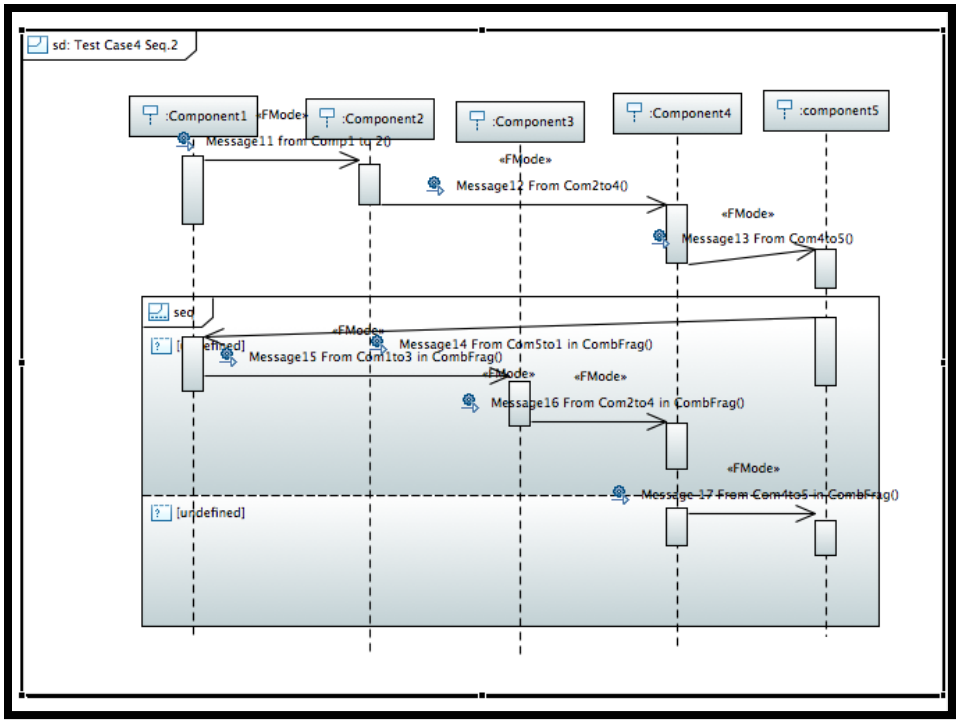

Figure 5-9 Test Case4 Sequence diagram2

Table 5-5 FMEA table for Test Case4

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Component5	Failure Mode 3	None	None	None	Remote	Almost Certain	1	Message13 From Com4to5_MessageRecv
	Failure Mode 6	No effect	None	None	Remote	Almost Certain	1	Message17 From Com4to5 in CombFrag_MessageRecv
Component4	Failure Mode 2	Affects Component 4	None	None	Low	Almost Certain	3	Message12 From Com2to4_MessageRecv
	Failure Mode 6	None	None	None	Remote	Almost Certain	1	Message16 From Com2to4 in CombFrag_MessageRecv
Component1	Failure Mode 8	Low effect on other component	Low	High	Remote	Very High	14	Message2 From Comp2to1_MessageRecv
	Failure Mode	Low effect on others	No visible	None	Remote	Almost Certain	1	Message5 From Comp2to1 in

	11		effect					comFrag_Message Recv
	Failure Mode 4	Affects Component 5	No effect	None	Remote	Almost Certain	1	Message14 From Com5to1 in CombFrag_MessageRecv
Component2	Failure Mode 7	Low effect	Low effect	None	Low	Low	18	Message1 From Comp1to2_MessageRecv
	Failure Mode 10	High	High effect	High	Beyond Moderate	Almost Certain	35	Message4 From Comp3to2 in comFrag_Message Recv
	Failure Mode 1	Affect Component1	None	None	Remote	Almost Certain	1	Message11 From Comp1 to 2_MessageRecv
Component3	Failure Mode 9	Low effect on other component	Very Low effect	None	Moderately High	Almost Certain	6	Message3 From Comp1to3 in comFrag_Message Recv
	Failure Mode 5	Affect other component	Affect the system	None	Moderate	Low	24	Message15 From Com1to3 in CombFrag_MessageRecv

5.1.5 Test Case 5

At this stage the aim is to test the transformation process ability to transform more than one source element (both structural and interaction model elements). So the source model here consists of a simple sequence diagram and a composite structural diagram as shown in the diagram below.

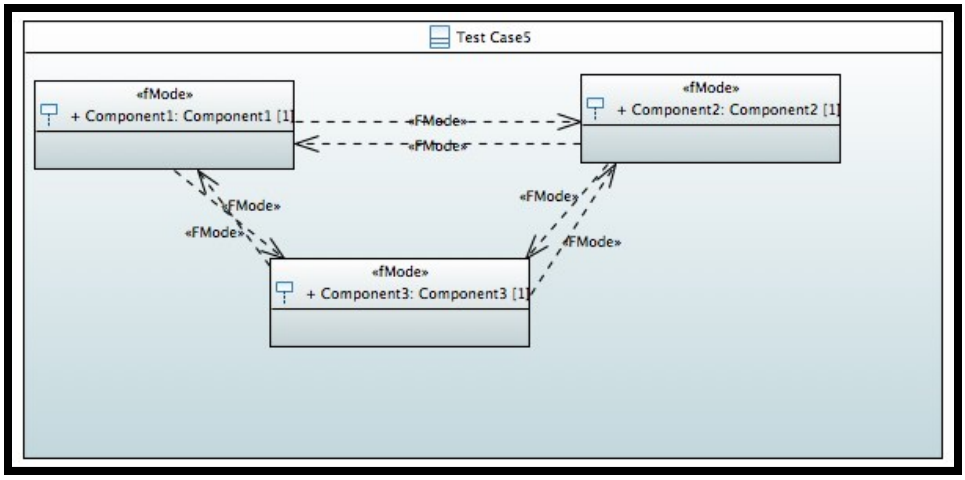

Figure 5-10 Test Case 5 Composite Structure Diagram

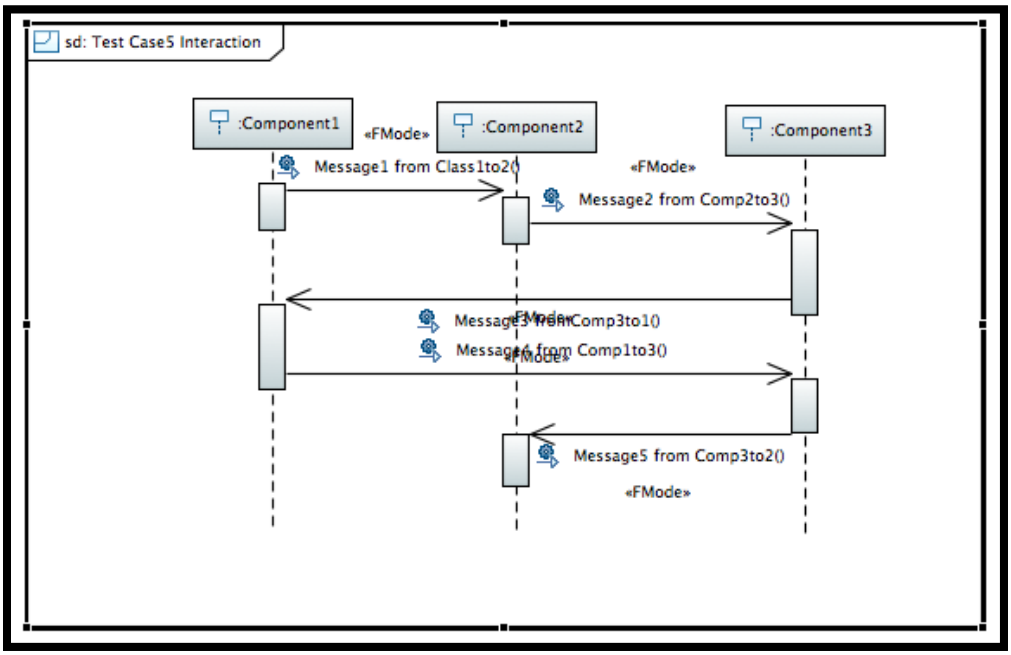

Figure 5-11 Test Case 5 Sequence Diagram

Table 5-6 FMEA table for Test Case5

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Component1	CS Failure Mode 2	Affects Component 2	None	None	High	Almost Certain	7	InformationItem3 from Comp2to1
	CS Failure Mode 5	Affect component 3	None	None	Remote	Almost Certain	1	InformationItem5 from Comp3to1
	Failure Mode 3	Affect component 3	Low effect on the system	None	Very High	Almost Certain	8	Message3 fromComp3to1_MessageRecv
Component2	CS Failure Mode 1	Low effect on others	Moderate effect on the system	None	Low	Very Low	21	InformationItem1 from Comp1to2
	CS Failure Mode 4	Affect other components	Low effect on the system	Low	Remote	High	15	InformationItem6 from Comp3to2
	Failure Mode 1	Low effect on other component	No effect	None	Remote	Almost Certain	1	Message1 from Class1to2 _MessageRecv
	Failure Mode 5	Minor effect	Moderate effect	Very Low	Remote	Low	24	Message5 from Comp3to2_MessageRecv
Component3	CS Failure Mode 5	Affect other component	Low Effect	Very High	Remote	Almost Certain	8	InformationItem2 from Comp1to3
	CS Failure Mode 3	Affects Others	High effects on the system	Low	High	Almost Certain	35	InformationItem4 from Comp2to3
	Failure Mode 2	Low effect	None	None	Remote	Moderate	5	Message2 from Comp2to3_MessageRecv
	Failure Mode 4	None	No effect	None	Remote	Almost Certain	1	Message4 from Comp1to3_MessageRecv

5.1.6 Test Case 6

This include the combination of all the previous test cases in one source model (all the possible source model features) that is, more than one sequence diagram with combined fragments and a composite structure diagram to test the completeness of the

transformation process and its capability to transform every possible source model elements in one source model (input features)

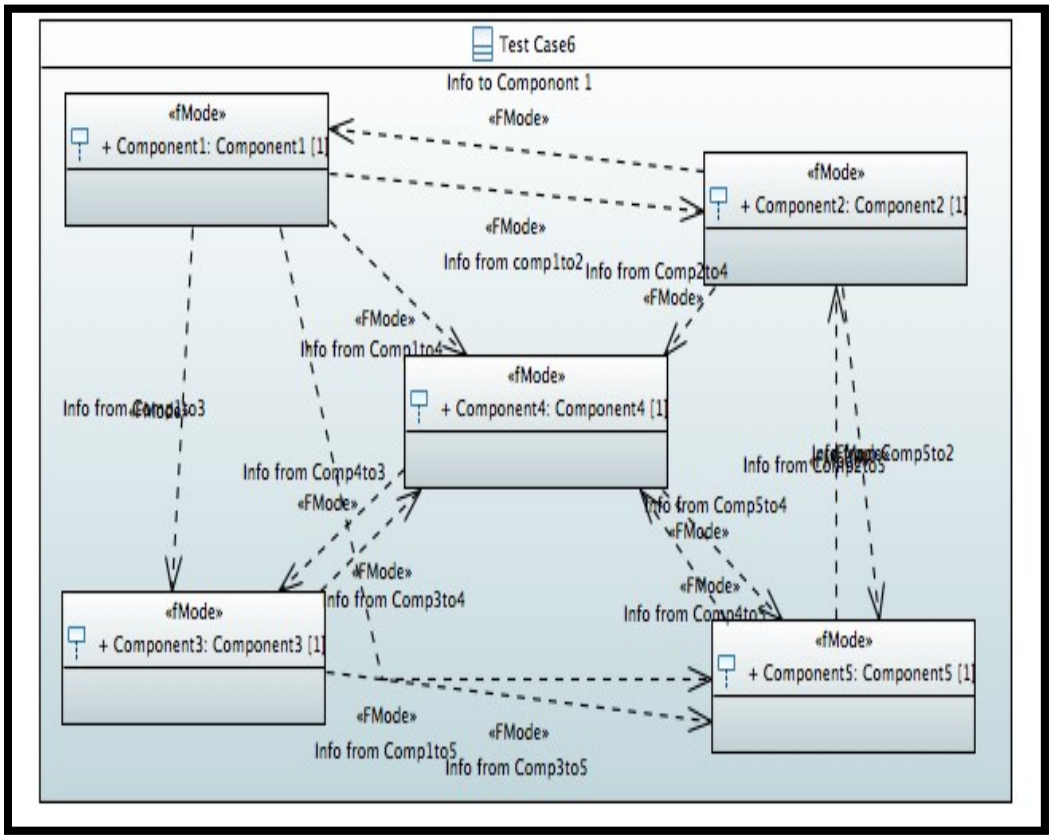

Figure 5-12 Test Case6 Composite Diagram

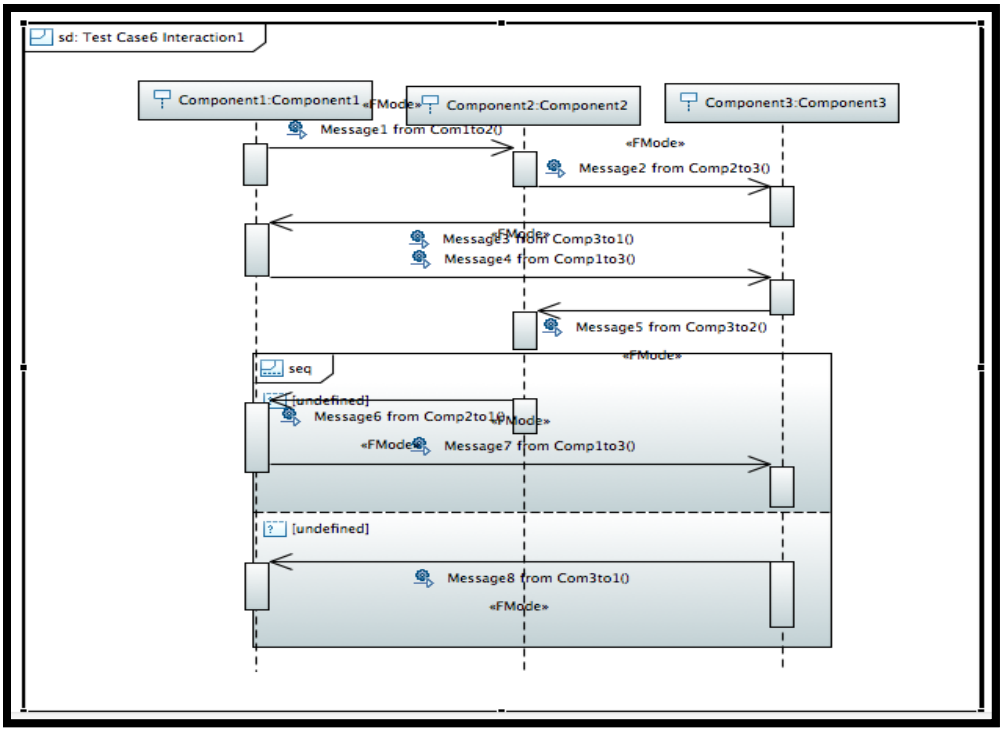

Figure 5-13 Test Case6 Sequence diagram1

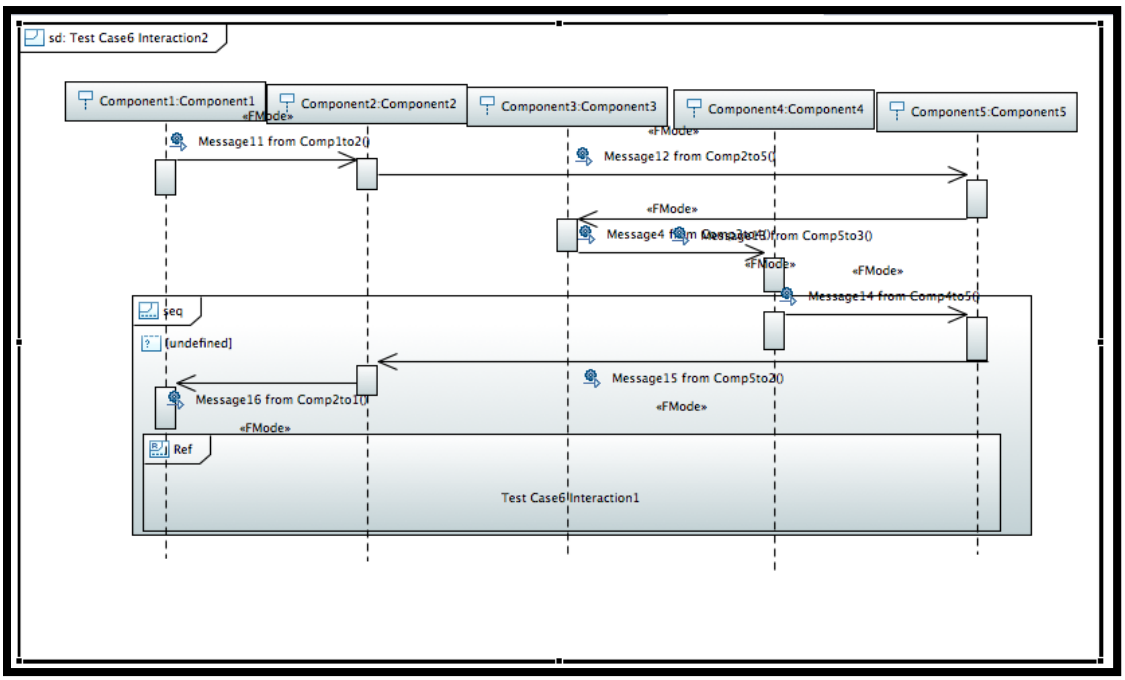

Figure 5-14 Test Case6 Sequence diagram2

Table 5-7 FMEA table for Test Case6

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Component1	Failure Mode1 of Component 1	Affects Component 2	No effect	Low	Remote	Almost Certain	5	Info to Component 1
	Failure Mode2 of Component	None	No effect	None	Remote	Almost Certain	1	Message3 from Comp3to1_MessageRecv
	Failure Mode3 of Component 1	Affect the next component	No effect	None	Remote	Remote	8	Message6 from Comp2to1_MessageRecv
	Failure Mode4 of Component 1	Affect component 3	Low effect on the system	Low	Remote	Remote	40	Message8 from Com3to1_MessageRecv
	Failure Mode5 of Component 1	Affects component 2	Low effect	None	Beyond Moderate	Almost Certain	5	Message16 from Comp2to1_MessageRecv
Component2	Failure Mode1 of Component 2	Affect Other component	High	Very Low	Extremely High	Low	216	Info from comp1to2
	Failure Mode2 of Component 2	Affect one component	None	None	Remote	Almost Certain	1	Info from Comp5to2
	Failure Mode3 of Component 2	No effect	None	None	Remote	Moderate	5	Message1 from Com1to2_MessageRecv
	Failure Mode4 of Component 2	No effect	None	None	Remote	Almost Certain	1	Message5 from Comp3to2_MessageRecv
	Failure Mode5 of Component 2	Affect other component	Affects the system	None	Extremely High	Low	54	Message11 from Comp1to2_MessageRecv
	Failure Mode6 of Component 2	Low effect	No effect	None	Remote	Almost Certain	1	Message15 from Comp5to2_MessageRecv
Component3	Failure Mode1 of Component 3	No effect	Low Effect	Low	Beyond Moderate	Almost Certain	25	Info from Comp1to3
	Failure Mode2 of Component 3	Affects others	No Effect	None	Very High	Moderate	40	Info from Comp4to3
	Failure Mode3 of Component 3	Affect Component 2	None	None	Remote	Almost Certain	1	Message2 from Comp2to3_MessageRecv

	Failure Mode4 of Component 3	Affect other compon ents	No effect on the system	High	Remote	Almost Certain	7	Message4 from Comp1to3_Messa geRecv
	Failure Mode5 of Component 3	Affect compon ent 2	No effect on the system	None	Remote	Almost Certain	1	Message7 from Comp1to3_Messa geRecv
	Failure Mode6 of Component 3	Affect compon ent 5	None	None	Remote	Almost Certain	1	Message13 from Comp5to3_Messa geRecv
Component4	Failure Mode2 of Component 4	None	No effect	None	Remote	Almost Certain	1	Info from Comp1to4
	Failure Mode1 of Component 4	High effect	None	None	Remote	Almost Certain	1	Info from Comp2to4
	Failure Mode4 of Component 4	Low effect	No effect on the system	High	Remote	High	21	Info from Comp5to4
	Failure Mode3 of Component 4	No effect	None	None	Remote	Almost Certain	1	Info from Comp3to4
	Failure Mode5 of Component 4	None	No effect	None	High	Almost Certain	7	Message4 from Comp3to4_Messa geRecv
Component5	Failure Mode4 of Component 5	Affects others	Affect the system	Very High	High	Almost Certain	56	Info from Comp1to5
	Failure Mode1 of Component 5	None	Moderat e effect	None	High	Very Low	49	Info from Comp4to5
	Failure Mode2 of Component 5	None	No effect	None	Remote	Almost Certain	1	Info from Comp2to5
	Failure Mode3 of Component 5	None	No effect	None	Remote	Almost Certain	1	Info from Comp3to5
	Failure Mode5 of Component 5	Affects Compon ent 2	Affects the System	Minor	Remote	Very Low	21	Message12 from Comp2to5_Messa geRecv
	Failure Mode6 of Component 5	No effect	None	None	Remote	Almost Certain	1	Message14 from Comp4to5_Messa geRecv

5.2 Case Study1 (LCS)

This case study is taken from [DAV10], where the authors present a UML/SysML model of Level Control System (LCS) system, and derived its FMEA model. The decision to use a case study built by someone else is in order to reduce the "author bias" in method/project development, in which self-built test cases by a software developer may be too selective (willingly or unwillingly). The case-study contains both sequence and composite structure diagram as the input (Source model) unlike the author's approach with the use of SysML for the design and description of the system architecture and behavior, as well as their algorithms for the FMEA generation.

As described in [DAV10] the LCS is mounted on a tank filled with an unspecified fluid and fluid level. The basic responsibility of the LCS is to insure that the level of the fluid in the tank will never exceed the capacity of the tank. The components of LCS include; two electrovalves (Ev1, Ev2) connected upstream of the tank, two mechanical valves (Mv1, Mv2), two level sensors (S1, S2) command the electrovalves and one Alarm (Al). The LCS working principle can be described as follows: "If S1 detects a too high level of fluid in the tank, it closes Ev1. If S2 detects a too high level, it closes Ev2 and activates Al. When Al rings, an operator closes Mv1. If the fluid level is still too high after 3 min, the operator opens Mv2 that drains the tank". Figure 5-1 and 5-2 presents the annotated UML Composite structure and Sequence diagram of the system, respectively, and Table 5-8 shows the FMEA table derived by our transformation.

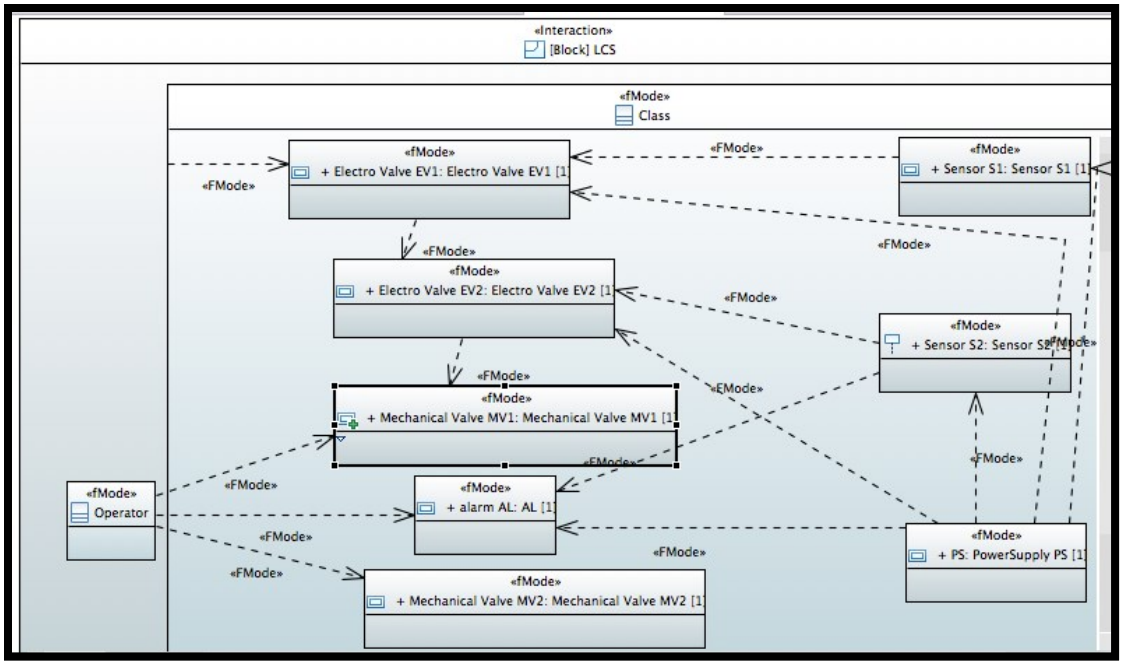

Figure 5-15 the case study 1 (LCS) Composite Structure Diagram

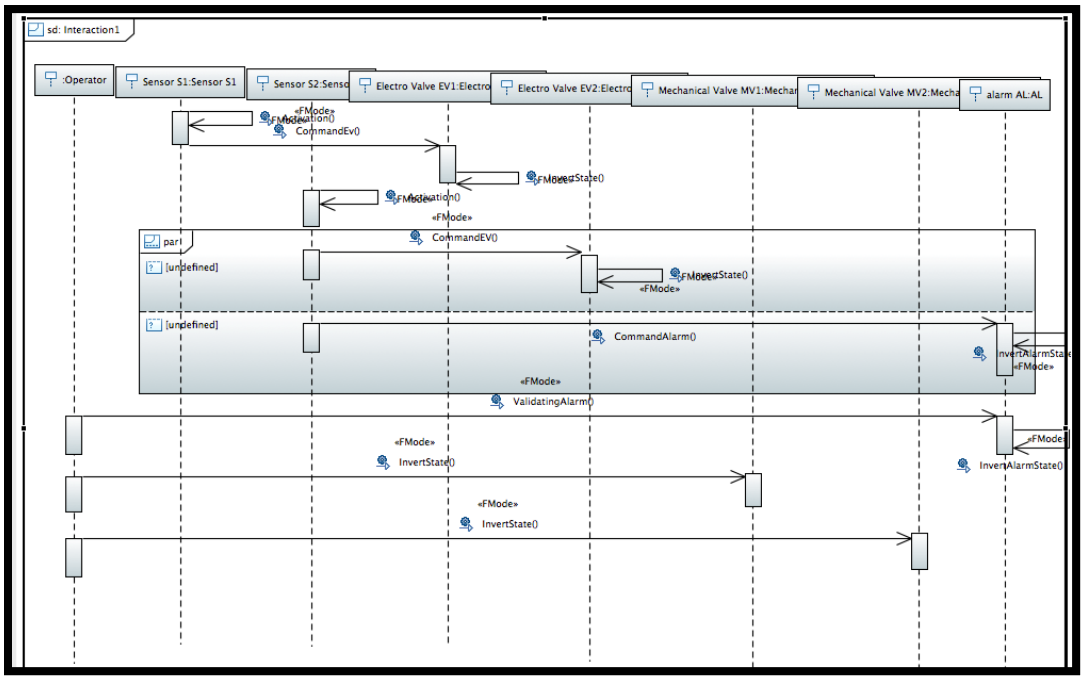

Figure 5-16 Case study1 (LCS) Sequence diagram

Table 5-8 FMEA Table for Case Study 1

Component	Failure mode	Local Effect	System Effect	Severity	Occurrence	Detection	RPN Value	Failure Cause
Operator	Wrong movement	Incorrect validation	System error	Low	Moderate	Almost Certain	20	Stable movement_MessageRecv
Sensor S1	No detection	Effect On Ev1 through CiS1-CiEv1 [CommandInterface], On Ev1 by CommandEv On S1 by Activation	Internal effect and affects the system smooth runnings	Very High	Moderate	Almost Certain	32	PiS1:PowerInput
	False detection	Effect On Ev1 through CiS1-CiEv1 [CommandInterface], On Ev1 by CommandEv On S1 by Activation	Internal effect and affects the system smooth runnings	Very High	Beyond Moderate	Moderate	40	Activation_MessageRecv
Sensor S2	PiS2 Power failure	No invert state signal sent	The system sensor not responding	None	Remote	Almost Certain	1	PiS2:PowerInput
	No activation signal	Electro valves remain closed	System activation error	Hazard With Warning	Remote	Moderate	9	Activation_Message0Recv
Electro Valve EV1	No PiEV1 power detection	No signal to EV2	System shutdown /hangs	None	Remote	Almost Certain	1	PiEV1:PowerInput
	Wrong commandCiEV1	Wrong signal to EV2	System error	High	Remote	Low	42	CiEV1:CommandInterface
	No Fluid detected	EV1 remain closed	System not responding	None	Remote	Almost Certain	1	InEV1:Fluid
	Inaccurate Command EV	EV1 invert state error	System output error	Very High	Remote	Low	48	CommandEv_MessageRecv
	Stuck opened	On LCS through InEv1-In1LCS [Fluid] On Ev2 through OutEv1-InEv2 [Fluid] On Ev1 by InvertState	Affects the system's running	Minor	Extremely High	Almost Certain	27	InvertState_MessageRecv

Electro Valve EV2	PiEV2 Power failure	No output signal sent	System output failure	None	Remote	Almost Certain	1	PiEV2:PowerInput
	Failed CiEV2 detection	No reply signal to EV1	System output error	None	Low	Low	18	CiEV2:CommandInterface
	No Fluid detected	EV1 remain open	System error	None	Remote	Almost Certain	1	InEV2:Fluid
	Ev command error	Affects the other	System output error	Very High	Moderate	Low	192	CommandEV_MessageRecv
	Inaccurate state inversion	Affects the other components	System output error	Minor	High	Almost Certain	21	InvertState_Message0Recv
Mechanical Valve MV1	Stuck	Affects immediate components	Affects the system	Low	Remote	Low	30	InMV1:Fluid
	Inaccurate actuator signal	Affects the system alarm	System unstable	Very Low	Moderately High	Very Low	168	ActuatorMV1
	No invert signal received	Affect other components	System continue in the same state	Very High	Very Low	Almost Certain	16	InvertState_Message1Recv
Mechanical Valve MV2	Inaccurate MV2 actuator	Wrong alarm signal	System unstable	Hazard With Warning	Remote	Low	54	ActuatorMV2
	Wrong invert signal	Affects others	The system receives wrong inversion	None	High	High	21	InvertState_Message2Recv
alarm AL	PiAL2 Power failure	Affects the system alarm sub-system's	Bad system alarm	Very High	Remote	Low	48	PiAL:PowerInput
	Wrong command	Affects the system	Affects other components	None	Remote	Almost Certain	1	CiAL:CommandInterface
	Wrong alarm signal	Affects the system speaker	Bad system alarm	None	Low	Moderate	3	Speaker:Speaker
	Wrong command alarm signal	Affects the operator and the alarm sub-system	Wrong system alarm	High	Very Low	Almost Certain	14	CommandAlarm_MessageRecv
	Wrong invert signal	Shuts the alarm	The system alarm shutdown	Low	Moderate	Very Low	140	InvertAlarmState_MessageRecv
	No validation signal detected	Affects the Alarm	Lack of proper alarm system	High	Low	High	63	ValidatingAlarm_MessageRecv
	No invert alarm	Affects other alarm sub-system	Wrong system alarm	High	High	Almost Certain	49	InvertAlarmState_MessageRecv

5.3 Case study-2 (TER)

This example from [GUI03] is the second case study from literature used to validate this thesis work. The Tele-echography (TER) system allows an expert physician (operator) to move or adjust by hand a scanner virtual probe in an unconstrained way, and in turn safely reproduces this adjustment/movement on the distant robotic remote site where the patient is located. This gives the operator the liberty to move the virtual probe to control the remote one on the patient. The risk analysis was presented with using Failure Modes Effects and Critically Analysis (FMECA).

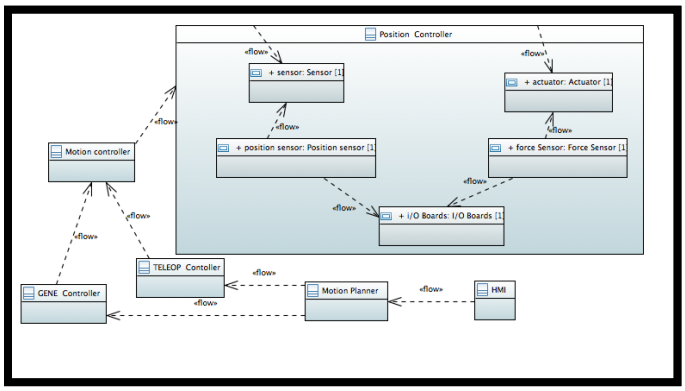

Figure 5-17 Case Study2 Composite diagram

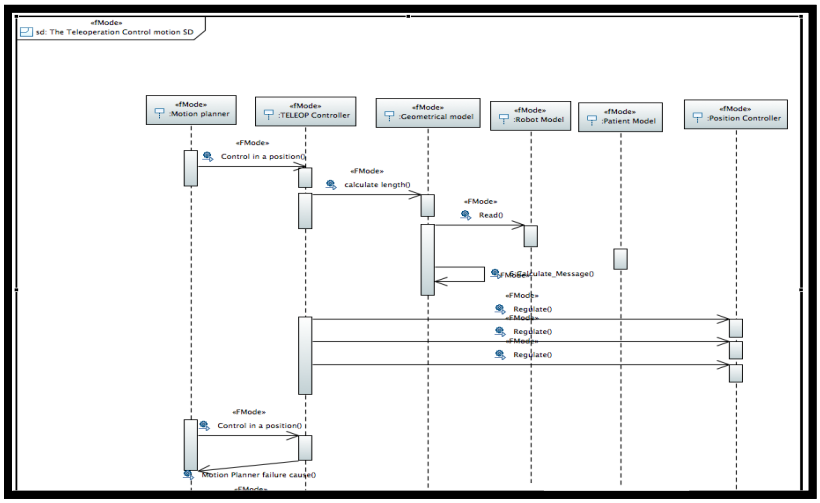

Figure 5-18 Case study2 (TER) Sequence Diagram1

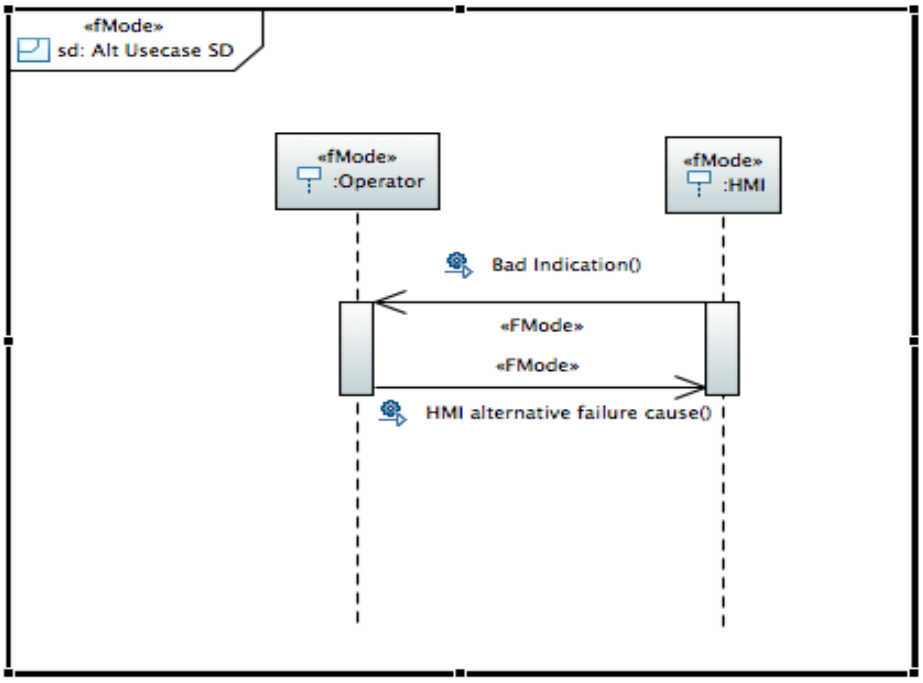

Figure 5-19 Case study2 (TER) Sequence Diagram2

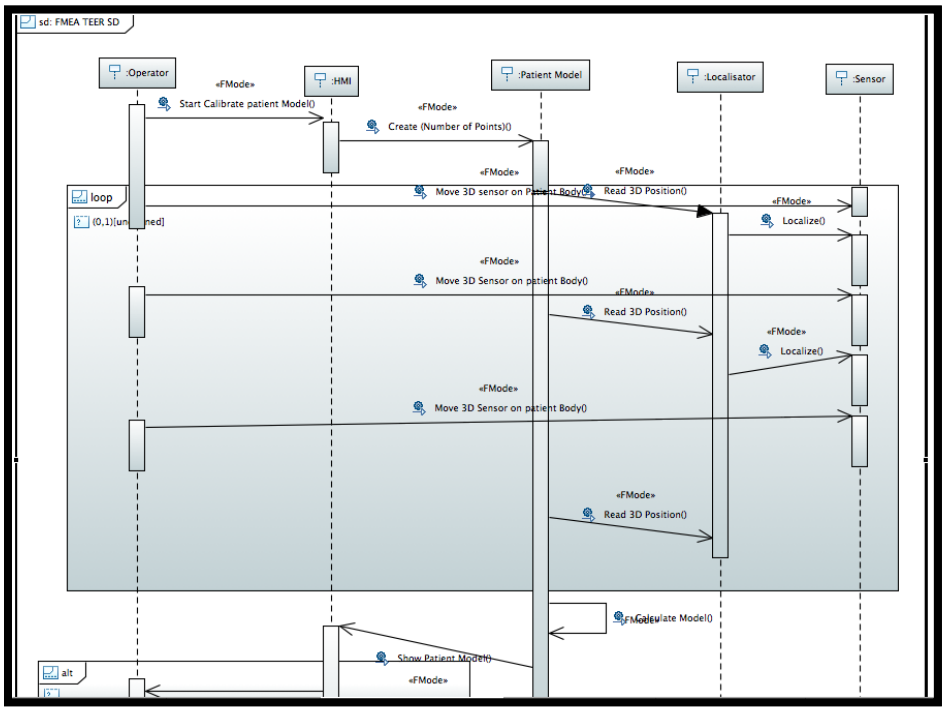

Figure 5-20 Case study2 (TER) Sequence Diagram3

Table 5-9 The FMEA Table For Case Study2

Component	Failure mode	Local Effect	System Effect	Severity	Detection	Detection	RPN Value	Failure Cause
Localizer	Mis-interpretation of 3D position	Wrong signal	System fails	None	Extremely High	High	27	Read 3D Position_MessageRecv
	Error reading 3D position	No signal	No response	Very High	Remote	Low	48	Read 3D Position_Message0 Recv
	Mis-interpretation of 3D position	No signal	No response	None	Remote	Almost Certain	1	Read 3D Position_Message1 Recv
3D Sensor	Wrong Movement of 3D	Error in calculation	Incorrect response	None	Remote	Almost Certain	1	Move 3D sensor on Patient Body_MessageRecv
	Error during localization	Affect other components	Localization effect on the system	None	Remote	Almost Certain	1	Localize_MessageRecv
	Wrong 3D movement	Wrong calculation	System crash	Low	High	Almost Certain	35	Move 3D Sensor on patient Body_MessageRecv
	No Position signal	No effect	None	None	Remote	Very Low	7	Localize_Message0 Recv
	Wrong 3D Movement	Incorrect response	No effect	None	Remote	Low	6	Move 3D Sensor on patient Body_Message0Recv
Operator	Wrong representation of the patient	Wrong interpretation by the operator	Wrong output wit incorrect moves	None	Remote	Almost Certain	1	3D representation Of Patient_MessageRecv
	Bad 3D sensor manipulation	Bad Patient Model	Pressure on the Patient too high	None	Remote	Almost Certain	1	Bad Indication_Message Recv
HMI	Wrong Signal	Incorrect calibration response	Wrong output	None	Remote	Almost Certain	1	Start Calibrate patient Model_MessageRecv
	Invisible Patient Model	Loss of signals	No response	Very High	Remote	Moderate	8	Show Patient Model_MessageRecv
	Wrong input	Invalid update	Wrong output	None	Remote	Almost Certain	1	Validate_MessageRecv
	HMI failure	Affects the operator input signal	Wrong output	None	Remote	Almost Certain	1	HMI alternative failure cause_MessageRecv
Motion Planner	Failed Motion Planner	No response	System hangs	None	Remote	Almost Certain	1	Motion Planner failure cause _MessageRecv

Table 5-8 The FMEA Table For Case Study2 (Continued)

Component	Failure mode	Local Effect	System Effect	Severity	Detection	Detection	RPN Value	Failure Cause
TELEOP Controller	Wrong Control signal	Wrong signal	Wrong output	Low	Low	Almost Certain	15	Control in a position_MessageRecv
	Loss of control	High	Low	Moderate	Remote	Almost Certain	18	Control in a position_Message0Recv
Geometrical model	Wrong Calculation	High	Low	Very High	Remote	Very Remote	72	Calculate_Message_MessageRecv
	Failed length Calculation	High	None	None	Remote	Almost Certain	1	Calculate length_MessageRecv
Position Controller	Loss of signal regulation	Severe effect	High	None	Remote	Almost Certain	1	Regulate_Message Recv
	Loss of signal regulation	Very high	High	Low	Moderate	Moderate	20	Regulate_Message0 Recv
	Loss of signal regulation	Extremely high	High	None	Remote	Almost Certain	1	Regulate_Message1 Recv
Robot Model	Failed signal to read	Low	None	Low	Remote	Almost Certain	5	Read_MessageRecv
Patient Model	Wrong Number Points	Affects HMI	Affects system smooth running	Moderate	Remote	Almost Certain	6	Create (Number of Points)_MessageRecv
	Wrong Calculation	Wrong signal to the other components	Wrong output	Hazard With Warning	Almost Certain	Almost Certain	90	Calculate Model _MessageRecv
	Read wrong signal	Wrong input	Wrong response	None	Remote	Almost Certain	1	Read_Message0Recv

6 Chapter: Conclusions

6.1.1 Achievements

This thesis has successfully proposed, implemented, and verified/validated a multi-step process for model to model transformations using the Epsilon transformation language family (ETL+EOL) and the text-to-text transformation language XSLT, in which the transformation input, a UML software model with failure mode annotations is transformed into a target FMEA tabular model. The source model is composed of composite structure diagrams and sequence diagrams with applied *Fmode* stereotypes.

Papyrus [PAP15] is the UML modeling tool used in this work. We selected Papyrus because it is an open source tool based on the Eclipse environment, supported by PolarSys [Pola16], an Eclipse Industry Working Group created by large industry players and tool providers working together on the creation and support of Open Source tools for the model-driven development of embedded systems. Another reason we selected Papyrus was because the Epsilon engine is able to directly accept the Papyrus annotated UML models as source models. The transformation accepts the standard UML metamodel, but also require a target metamodel for the FMEA domain, which is not standardized; we had to define it with a special metamodeling language named Emphatic, provided by the Eclipse platform. With the use of the FMEA metamodel, ETL was able to generate an XMI file of the target model, which could then be transformed to an FMEA table in text format by an XSLT text-to-text transformation.

6.1.2 Limitations

The transformation implementation has some limitations, as follows. The transformation accepts only UML source models built as described in Chapter 3. If the source model

does not follow the modeling style presented in Chapter 3, the transformation stops running.

Another limitation is that, the UML editor used in this work, Papyrus, does not yet implement all the metaclasses defined in the UML metamodel at the time of this thesis implementation. E.g., in deployment diagrams it is not easy to model communication networks because the links between deployment nodes are not implemented. During our modeling it was also discovered that Papyrus does not support the use of connector between ports of the classes in the composite structure diagram. The sequence diagram's lifelines could only represent properties in the composite structure diagram.

The number of existing works that use a model-driven approach for building FMEA tables is very limited. We searched very hard the literature to find examples of FMEA analysis applied in an MDD context, in order to extend the set of our test cases with models created by other authors. The goal was to reduce the potential "author bias" in the project or implementation testing when creating test cases.

This work is based on just the traditional FMEA, which can as well be extended to the Failure Mode, Effect and Critical Analysis (FMECA) version.

FMEA is an induction safety analysis (bottom-up) approach that can be combined with Fault-Tree Analysis (FTA), which is a deductive analysis (top-down) approach. We have not done this in the thesis, proposing it as a direction for future work.

Due to all the mentioned limitations our implementation experienced some difficulties. However, we were able to design solutions to deal with some of the limitations and left some others as future work.

6.1.3 Directions for Future Work

The transformation process of UML+Fmode models to FMEA models developed in this thesis can be further developed in the following directions:

- As shown in the FMEA multi-step process approach for this work, the generation of FMEA table requires another level of transformation (Text2Text) using XSLT to generate the FMEA table. It would be of a great benefit to the user if all these steps could be integrated in to one single script which does not require manual intervention of the user to go from one step to another.
- The transformation could be extended from the generated FMEA to other alternatives, such as FMECA.
- The UML editor Papyrus used in this implementation is evolving quickly. The transformation should be updated to include the new features supported by the tool as they become available.
- The development of more pre-transformation verifications processes of the input model to verify the correctness and completeness of the model before the transformation in order to avoid problems or errors would make the transformation more user friendly.
- The FMEA analysis could be combined with Fault-Tree Analysis (FTA), both in the context of model–driven development.

Appendices

Appendix A

A.1 Running Example: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Sensor"/>
  <component name="Input Control Circuit">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Sensor Analogue output signal "/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Convert Position_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Micro-Controller">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Initialize Input Pin
Register_MessageRecv"/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Get value_MessageRecv"/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Compute Duty Cycle_MessageRecv"/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="PWM Period_MessageRecv"/>
    </failuremode>
  </component>
  <component name="ISR">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="ADC value"/>
    </failuremode>
```

```
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Initialize ISR_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Fan">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="regulate Speed_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Ball">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Position_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Output Control Circuit">
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="New duty cycle"/>
    </failuremode>
    <failuremode>
      <failurecause name="Initialize Output pin_MessageRecv"/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="OutPut signal pin_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

A.2 Case Study 1: XML Target Model

This is Appendix A, Sub-Appendix 2

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Operator">
    <failuremode name="Wrong movement" FailureLocalEffect="Incorrect
validation" SystemFailureLevelEffect="system error" severity="Low"
occurrence="Moderate" detection="AlmostCertain" RPN="20">
      <failurecause name="Stable movement_MessageRecv"/>
```

```
    </failuremode>
  </component>
  <component name="Sensor S1">
    <failuremode name="No detection" FailureLocalEffect="Effect On
Ev1 through CiS1-CiEv1&#xD;[CommandInterface], On Ev1 by
CommandEv&#xD;On S1 by Activation"
SystemFailureLevelEffect="Internal effect and affects the system
smooth runnings" severity="VeryHigh" occurrence="Moderate"
detection="AlmostCertain" RPN="32">
      <failurecause name="PiS1:PowerInput"/>
    </failuremode>
    <failuremode name="False detection" FailureLocalEffect="Effect
On Ev1 through CiS1-CiEv1&#xD;[CommandInterface], On Ev1 by
CommandEv&#xD;On S1 by Activation"
SystemFailureLevelEffect="Internal effect and affects the system
smooth runnings" severity="VeryHigh" occurrence="BeyondModerate"
detection="Mederate" RPN="40">
      <failurecause name="Activation_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Sensor S2">
    <failuremode name="PiS2 Power failure" FailureLocalEffect="No
invert state signal sent" SystemFailureLevelEffect="The system
sensor not responding" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="PiS2:PowerInput"/>
    </failuremode>
    <failuremode name="No activation signal"
FailureLocalEffect="Electro valves remain closed"
SystemFailureLevelEffect="System activation error"
severity="HazardWithWarning" occurrence="Remote"
detection="Mederate" RPN="9">
      <failurecause name="Activation_Message0Recv"/>
    </failuremode>
  </component>
  <component name="Electro Valve EV1">
    <failuremode name="No PiEV1 power detection"
FailureLocalEffect="No signal to EV2"
SystemFailureLevelEffect="System shutdown/hangs&#xA;"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="PiEV1:PowerInput"/>
    </failuremode>
    <failuremode name="Wrong commandCiEV1" FailureLocalEffect="Wrong
signal to EV2" SystemFailureLevelEffect="System error"
severity="High" occurrence="Remote" detection="Low" RPN="42">
      <failurecause name="CiEV1:CommandInterface"/>
    </failuremode>
    <failuremode name="No Fluid detected" FailureLocalEffect="EV1
remain closed" SystemFailureLevelEffect="System not responding"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="InEV1:Fluid"/>
    </failuremode>
```

```
    <failuremode name="Inaccurate CommandEV" FailureLocalEffect="EV1
invert state error" SystemFailureLevelEffect="System output error"
severity="VeryHigh" occurrence="Remote" detection="Low" RPN="48">
      <failurecause name="CommandEv_MessageRecv"/>
    </failuremode>
    <failuremode name="Stuck opened" FailureLocalEffect="On LCS
through InEv1-In1LCS&#xD;[Fluid] On Ev2 through OutEv1-
InEv2&#xD;[Fluid] On Ev1 by InvertState"
SystemFailureLevelEffect="Affects the system's runnings&#xA;"
severity="Minor" occurrence="ExtreemlyHigh"
detection="AlmostCertain" RPN="27">
      <failurecause name="InvertState_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Electro Valve EV2">
    <failuremode name="PiEV2 Power failure" FailureLocalEffect="No
output signal sent" SystemFailureLevelEffect="System output failure"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="PiEV2:PowerInput"/>
    </failuremode>
    <failuremode name="Failed CiEV2  detection"
FailureLocalEffect="No reply signal to EV1"
SystemFailureLevelEffect="Sytem output error" severity="None"
occurrence="Low" detection="Low" RPN="18">
      <failurecause name="CiEV2:CommandInterface"/>
    </failuremode>
    <failuremode name="No Fluid detected" FailureLocalEffect="EV1
remain open" SystemFailureLevelEffect="system error" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="InEV2:Fluid"/>
    </failuremode>
    <failuremode name="Ev command error" FailureLocalEffect="Affects
the other" SystemFailureLevelEffect="Syetem output error"
severity="VeryHigh" occurrence="Moderate" detection="Low" RPN="192">
      <failurecause name="CommandEV_MessageRecv"/>
    </failuremode>
    <failuremode name="Inaccurate state inversion"
FailureLocalEffect="affects the other components"
SystemFailureLevelEffect="System output error" severity="Minor"
occurrence="High" detection="AlmostCertain" RPN="21">
      <failurecause name="InvertState_Message0Recv"/>
    </failuremode>
  </component>
  <component name="Mechanical Valve MV1">
    <failuremode name="Stucked" FailureLocalEffect="Affects
immediate components" SystemFailureLevelEffect="Affects the system"
severity="Low" occurrence="Remote" detection="Low" RPN="30">
      <failurecause name="InMV1:Fluid"/>
    </failuremode>
    <failuremode name="Inaccurate actuaor signal"
FailureLocalEffect="Affects the system alarm "
SystemFailureLevelEffect="System unstable " severity="VeryLow"
occurrence="ModeratelyHigh" detection="VeryLow" RPN="168">
```

```
      <failurecause name="ActuatorMV1"/>
    </failuremode>
    <failuremode name="No invert signal received"
FailureLocalEffect="Affect other components"
SystemFailureLevelEffect="system continue in the same state"
severity="VeryHigh" occurrence="VeryLow" detection="AlmostCertain"
RPN="16">
      <failurecause name="InvertState_Message1Recv"/>
    </failuremode>
  </component>
  <component name="Mechanical Valve MV2">
    <failuremode name="Inaccurate MV2 actuator"
FailureLocalEffect="wrong alarm signal "
SystemFailureLevelEffect="System unstable "
severity="HazardWithWarning" occurrence="Remote" detection="Low"
RPN="54">
      <failurecause name="ActuatorMV2"/>
    </failuremode>
    <failuremode name="wrong invert signal "
FailureLocalEffect="affects others " SystemFailureLevelEffect="The
system receives wrong inversion" severity="None" occurrence="High"
detection="High" RPN="21">
      <failurecause name="InvertState_Message2Recv"/>
    </failuremode>
  </component>
  <component name="alarm AL">
    <failuremode name="PiAL2 Power failure"
FailureLocalEffect="Affects the system alarm sub-system's "
SystemFailureLevelEffect="Bad system alarm" severity="VeryHigh"
occurrence="Remote" detection="Low" RPN="48">
      <failurecause name="PiAL:PowerInput"/>
    </failuremode>
    <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="CiAL:CommandInterface"/>
    </failuremode>
    <failuremode name="Wrong alarm signal"
FailureLocalEffect="Affects the system speaker "
SystemFailureLevelEffect="bad system alarm " severity="None"
occurrence="Low" detection="Mederate" RPN="3">
      <failurecause name="Speaker:Speaker"/>
    </failuremode>
    <failuremode name="Wrong command alarm signal"
FailureLocalEffect="Affects the operator and the alarm sub-system"
SystemFailureLevelEffect="wrong system alarm" severity="High"
occurrence="VeryLow" detection="AlmostCertain" RPN="14">
      <failurecause name="CommandAlarm_MessageRecv"/>
    </failuremode>
    <failuremode name="Wrong invert signal"
FailureLocalEffect="shuts the alram " SystemFailureLevelEffect="The
system alarm shutdown" severity="Low" occurrence="Moderate"
detection="VeryLow" RPN="140">
      <failurecause name="InvertAlarmState_MessageRecv"/>
```

```
    </failuremode>
    <failuremode name="No validation signal detected"
FailureLocalEffect="Affects the Alarm "
SystemFailureLevelEffect="Lack of proper alarm system "
severity="High" occurrence="Low" detection="High" RPN="63">
      <failurecause name="ValidatingAlarm_MessageRecv"/>
    </failuremode>
    <failuremode name="No invertalram received"
FailureLocalEffect="Affects other alarm sub-system "
SystemFailureLevelEffect="Wrong system alarm " severity="High"
occurrence="High" detection="AlmostCertain" RPN="49">
      <failurecause name="InvertAlarmState_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

A.3 Case Study 2: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA">
  <System name="FMEA of TEERModel"/>
    <component name="Localizer">
      <failuremode name="mis-interpretation of 3D position"
FailureLocalEffect="Wrong signal" SystemFailureLevelEffect="System
fails" severity="None" occurrence="ExtreemlyHigh" detection="High"
RPN="27">
        <failurecause name="Read 3D Position_MessageRecv"/>
      </failuremode>
      <failuremode name="Error reading 3D position"
FailureLocalEffect="No signal" SystemFailureLevelEffect="No
response" severity="VeryHigh" occurrence="Remote" detection="Low"
RPN="48">
        <failurecause name="Read 3D Position_Message0Recv"/>
      </failuremode>
      <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
        <failurecause name="Read 3D Position_Message1Recv"/>
      </failuremode>
    </component>
    <component name="3D Sensor">
      <failuremode name="wrong Movement of 3D"
FailureLocalEffect="error in calculation "
SystemFailureLevelEffect="incorrect response" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
        <failurecause name="Move 3D sensor on Patient
Body_MessageRecv"/>
      </failuremode>
```

```
      <failuremode name="" FailureLocalEffect=""
SystemFailureLevelEffect="" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
        <failurecause name="Localize_MessageRecv"/>
      </failuremode>
      <failuremode name="Wrong 3D movement"
FailureLocalEffect="Wrong calculation"
SystemFailureLevelEffect="System crash" severity="Low"
occurrence="High" detection="AlmostCertain" RPN="35">
        <failurecause name="Move 3D Sensor on patient
Body_MessageRecv"/>
      </failuremode>
      <failuremode name="No Position signal " FailureLocalEffect="No
effect" SystemFailureLevelEffect="None" severity="None"
occurrence="Remote" detection="VeryLow" RPN="7">
        <failurecause name="Localize_Message0Recv"/>
      </failuremode>
      <failuremode name="Wrong 3D Movenment"
FailureLocalEffect="incorrect response" SystemFailureLevelEffect="No
effect" severity="None" occurrence="Remote" detection="Low" RPN="6">
        <failurecause name="Move 3D Sensor on patient
Body_Message0Recv"/>
      </failuremode>
    </component>
    <component name="HMI"/>
    <component name="Operator">
      <failuremode name="Wrong representation of the patient"
FailureLocalEffect="Wrong interpretation by the operator"
SystemFailureLevelEffect="Wrong output wit incorrect moves"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="3D representation Of
Patient_MessageRecv"/>
      </failuremode>
      <failuremode name="Bad 3D sensor manipulation"
FailureLocalEffect="Bad Patient Model"
SystemFailureLevelEffect="Presure on the Patient too high"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="Bad Indication_MessageRecv"/>
      </failuremode>
    </component>
    <component name="HMI">
      <failuremode name="Wrong Signal"
FailureLocalEffect="Incorect colibration response"
SystemFailureLevelEffect="Wron output" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
        <failurecause name="Start Calibrate patient
Model_MessageRecv"/>
      </failuremode>
      <failuremode name="Invissible Patient Model "
FailureLocalEffect="Lost of signals" SystemFailureLevelEffect="No
response" severity="VeryHigh" occurrence="Remote"
detection="Mederate" RPN="8">
```

```
        <failurecause name="Show Patient Model_MessageRecv"/>
      </failuremode>
      <failuremode name="Wrong input" FailureLocalEffect="Invalid
update" SystemFailureLevelEffect="Wrong output" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
        <failurecause name="Validate_MessageRecv"/>
      </failuremode>
      <failuremode name="HMI failure " FailureLocalEffect="Affects
the operator input signal" SystemFailureLevelEffect="wrong output"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="HMI alternative failure
cause_MessageRecv"/>
      </failuremode>
    </component>
    <component name="Motion Planner">
      <failuremode name="Failed Motion Planner "
FailureLocalEffect="No response" SystemFailureLevelEffect="System
hangs" severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="Motion Planner failure
cause_MessageRecv"/>
      </failuremode>
    </component>
    <component name="TELEOP Controller">
      <failuremode name="Wrong Control signal"
FailureLocalEffect="Wrong signal " SystemFailureLevelEffect="Wrong
output" severity="Low" occurrence="Low" detection="AlmostCertain"
RPN="15">
        <failurecause name="Control in a position_MessageRecv"/>
      </failuremode>
      <failuremode name="Lost of control" FailureLocalEffect="High"
SystemFailureLevelEffect="Low" severity="Moderate"
occurrence="Remote" detection="High" RPN="18">
        <failurecause name="Control in a position_Message0Recv"/>
      </failuremode>
    </component>
    <component name="Geometrical model">
      <failuremode name="Wrong Calculation"
FailureLocalEffect="High" SystemFailureLevelEffect="Low"
severity="VeryHigh" occurrence="Remote" detection="VeryRemote"
RPN="72">
        <failurecause name="6:Calculate_Message_MessageRecv"/>
      </failuremode>
      <failuremode name="Failed lenght Calculation"
FailureLocalEffect="High" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="calculate length_MessageRecv"/>
      </failuremode>
    </component>
    <component name="Position Controller">
      <failuremode name="Lost of signal regulation "
FailureLocalEffect="Severe effect" SystemFailureLevelEffect="High"
```

```
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="Regulate_MessageRecv"/>
      </failuremode>
      <failuremode name="Lost of signal regulation "
FailureLocalEffect="Very high " SystemFailureLevelEffect="High"
severity="Low" occurrence="Moderate" detection="Mederate" RPN="20">
        <failurecause name="Regulate_Message0Recv"/>
      </failuremode>
      <failuremode name="Lost of signal regulation "
FailureLocalEffect="Extremely high" SystemFailureLevelEffect="High"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
        <failurecause name="Regulate_Message1Recv"/>
      </failuremode>
    </component>
    <component name="Robot Model">
      <failuremode name="failed signal to read"
FailureLocalEffect="Low" SystemFailureLevelEffect="None"
severity="Low" occurrence="Remote" detection="AlmostCertain"
RPN="5">
        <failurecause name="Read_MessageRecv"/>
      </failuremode>
    </component>
    <component name="Patient Model">
      <failuremode name="Wron Number Points"
FailureLocalEffect="Affects HMI" SystemFailureLevelEffect="Affects
system smooth running" severity="Moderate" occurrence="Remote"
detection="AlmostCertain" RPN="6">
        <failurecause name="Create (Number of Points)_MessageRecv"/>
      </failuremode>
      <failuremode name="wrong calculation "
FailureLocalEffect="wrong signal to the other components"
SystemFailureLevelEffect="wrong output" severity="HazardWithWarning"
occurrence="AlmostCertain" detection="AlmostCertain" RPN="90">
        <failurecause name="Calculate Model_MessageRecv"/>
      </failuremode>
      <failuremode name="REad wrong signal"
FailureLocalEffect="Wron input" SystemFailureLevelEffect="Wrong
response" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
        <failurecause name="Read_Message0Recv"/>
      </failuremode>
    </component>
  </System>
</xmi:XMI>
```

Appendix B

B.1 Test Case1: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Component1">
    <failuremode name="Component1 Failure mode"
FailureLocalEffect="Affect component 2"
SystemFailureLevelEffect="shutdown the system" severity="Low"
occurrence="BeyondModerate" detection="High" RPN="75">
      <failurecause name="Com from Comp2to1_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name="Component2 Failure mode"
FailureLocalEffect="Affects Component 1"
SystemFailureLevelEffect="No effect on the system " severity="Very
Minor" occurrence="Very Low" detection="AlmostCertain" RPN="1">
      <failurecause name="Com from Comp1to2_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

B.2 Test Case2: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Component1">
    <failuremode name="failure mode2" FailureLocalEffect="No effect"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message0Recv"/>
    </failuremode>
    <failuremode name="failure mode4&#xA;" FailureLocalEffect="Low
Effect on Component2&#xA;" SystemFailureLevelEffect="Low Effect"
severity="Very Minor" occurrence="Very Low"
detection="AlmostCertain" RPN="4">
      <failurecause name="Message2Recv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name="failre mode1"
FailureLocalEffect="Stope Component1"
SystemFailureLevelEffect="Affects the system" severity="High"
occurrence="Moderate" detection="AlmostCertain" RPN="28">
      <failurecause name="MessageRecv"/>
```

```
    </failuremode>
    <failuremode name="failure mode3" FailureLocalEffect="Affects
Componenet1" SystemFailureLevelEffect="Affects the system"
severity="Moderate" occurrence="Remote" detection="Very High"
RPN="12">
      <failurecause name="Message1Recv"/>
    </failuremode>
  </component>
</System>
```

B.3 Test Case3: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of Test
case3 model">
  <component name="Component1">
    <failuremode name=" Failure Mode 3" FailureLocalEffect="Affects
component 2" SystemFailureLevelEffect=" Low effect" severity="None"
occurrence="Low" detection=" Moderately High" RPN="12">
      <failurecause name="Message3 fromComp3to1_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name=" Failure Mode 1" FailureLocalEffect="Low
effect on other component" SystemFailureLevelEffect="Affect the
system" severity="None" occurrence="High" detection="Moderate"
RPN="35">
      <failurecause name="Message1 from Comp1to2_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 4" FailureLocalEffect="Moderate
effect on other component" SystemFailureLevelEffect="Affect the
system" severity="Moderate" occurrence="Beyond Moderate"
detection="AlmostCertain" RPN="30">
      <failurecause name="Message4 from Comp1to2_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 6" FailureLocalEffect=" Affect
other component " SystemFailureLevelEffect="Moderate effect on the
system" severity="VeryLow" occurrence="Remote" detection="Low"
RPN="24">
      <failurecause name="Message6 from Comp3to2_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component3">
    <failuremode name=" Failure Mode 2" FailureLocalEffect="Affect
component 1" SystemFailureLevelEffect="Low" severity="None"
occurrence="ModeratelyHigh" detection="AlmostCertain" RPN="6">
      <failurecause name="Message2 from Comp2to3_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 5" FailureLocalEffect="Affect
other component" SystemFailureLevelEffect="Shut down the system"
severity="HazardWithWarning" occurrence="Moderate"
detection="AlmostCertain" RPN="36">
```

```
      <failurecause name="Message5 from Comp2to3_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

B.4 Test Case4: XML Target Model

```
?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Component5">
    <failuremode name=" Failure Mode 3" FailureLocalEffect="None"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message13 From Com4to5_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 6" FailureLocalEffect="No
effect" SystemFailureLevelEffect="None" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="Message17 From Com4to5 in
CombFrag_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component4">
    <failuremode name=" Failure Mode 2" FailureLocalEffect="Affects
Component 4" SystemFailureLevelEffect=" None" severity="None"
occurrence="Low" detection="AlmostCertain" RPN="3">
      <failurecause name="Message12 From Com2to4_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 6" FailureLocalEffect="None"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message16 From Com2to4 in
CombFrag_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component1">
    <failuremode name=" Failure Mode 8" FailureLocalEffect="Low
effect on other component" SystemFailureLevelEffect="Low"
severity="High" occurrence="Remote" detection="Very High" RPN="14">
      <failurecause name="Message2 from Comp2to1_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 11" FailureLocalEffect="Low
effect on other component"
SystemFailureLevelEffect="No visible effect" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="Message5 from Comp2to1 in
comFrag_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 4" FailureLocalEffect="Affects
Component 5" SystemFailureLevelEffect="No effect" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
```

```
      <failurecause name="Message14 From Com5to1 in
CombFrag_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name=" Failure Mode 7"
FailureLocalEffect="Low effect" SystemFailureLevelEffect="Low
effect" severity="None" occurrence="Low" detection="Low" RPN="18">
      <failurecause name="Message1 From Comp1to2_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 10" FailureLocalEffect="High"
SystemFailureLevelEffect="High effect" severity="High"
occurrence="BeyondModerate" detection="AlmostCertain" RPN="35">
      <failurecause name="Message4 from Comp3to2 in
comFrag_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 1" FailureLocalEffect="Affect
Component1" SystemFailureLevelEffect="None" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="Message11 from Comp1 to 2_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component3">
    <failuremode name=" Failure Mode 9" FailureLocalEffect="Low
effect on other component" SystemFailureLevelEffect="Very low
effect" severity="None" occurrence="ModeratelyHigh"
detection="AlmostCertain" RPN="6">
      <failurecause name="Message3 from Comp1to3 in
comFrag_MessageRecv"/>
    </failuremode>
    <failuremode name=" Failure Mode 5" FailureLocalEffect="Affect
othercomponent" SystemFailureLevelEffect="Affect the system"
severity="None" occurrence="Moderate" detection="Low" RPN="24">
      <failurecause name="Message15 From Com1to3 in
CombFrag_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

B.5 Test Case5: XML Target Model

```
 <?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of test
case5">
  <component name="Component1">
    <failuremode name="CS Failure Mode 2"
FailureLocalEffect="Affects Componet 2"
SystemFailureLevelEffect="None" severity="None" occurrence="High"
detection="AlmostCertain" RPN="7">
      <failurecause name="InformationItem3 from Comp2to1"/>
    </failuremode>
```

```
    <failuremode name="CS Failure Mode 5" FailureLocalEffect="Affect
component 3 " SystemFailureLevelEffect="None" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="InformationItem5 from Comp3to1"/>
    </failuremode>
    <failuremode name="Failure Mode 3" FailureLocalEffect="Affect
component3" SystemFailureLevelEffect=" Low effect on the system"
severity="None" occurrence="VeryHigh" detection="ALmost Certain"
RPN="8">
      <failurecause name="Message3 fromComp3to1_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name="CS Failure Mode 1" FailureLocalEffect="Low
effect on others" SystemFailureLevelEffect="Moderate effect on the
system" severity="None" occurrence="Low" detection="VeryLow"
RPN="21">
      <failurecause name="InformationItem1 from Comp1to2"/>
    </failuremode>
    <failuremode name="CS Failure Mode 4" FailureLocalEffect="Affect
other component" SystemFailureLevelEffect="Low effect on the system"
severity="Low" occurrence="Remote" detection="High" RPN="15">
      <failurecause name="InformationItem6 from Comp3to2"/>
    </failuremode>
    <failuremode name="Failure Mode 1" FailureLocalEffect="Low
effect on others component" SystemFailureLevelEffect=" No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Message1 from Class1to2_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode 5" FailureLocalEffect="Minor
effect" SystemFailureLevelEffect="Moderate effect"
severity="VeryLow" occurrence="Remote" detection="Low" RPN="24">
      <failurecause name="Message5 from Comp3to2_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component3">
    <failuremode name="CS Failure Mode 5" FailureLocalEffect="Affect
other component" SystemFailureLevelEffect=" Low Effect"
severity="VeryHigh" occurrence="Remote" detection="AlmostCertain"
RPN="8">
      <failurecause name="InformationItem2 from Comp1to3"/>
    </failuremode>
    <failuremode name="CS Failure Mode 3"
FailureLocalEffect="Affects Others" SystemFailureLevelEffect="High
effect on the system" severity="Low" occurrence="High"
detection="AlmostCertain" RPN="35">
      <failurecause name="InformationItem4 from Comp2to3"/>
    </failuremode>
    <failuremode name="Failure Mode 2" FailureLocalEffect="Low
effect" SystemFailureLevelEffect="None" severity="None"
occurrence="Remote" detection="Mederate" RPN="5">
      <failurecause name="Message2 from Comp2to3_MessageRecv"/>
    </failuremode>
```

```
    <failuremode name="Failure Mode 4" FailureLocalEffect="None"
SystemFailureLevelEffect="No effect" severity="None"
occurrence="Remote" detection="AlmostCertain" RPN="1">
      <failurecause name="Message4 from Comp1to3_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

B.6 Test Case6: XML Target Model

```
<?xml version="1.0" encoding="ASCII"?>
<System xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns="FMEA" name="FMEA of
model">
  <component name="Component1">
    <failuremode name="Failure Mode1 of Component 1"
FailureLocalEffect="Affects Component 2"
SystemFailureLevelEffect="No effect" severity="Low"
occurrence="Remote" detection="AlmostCertain" RPN="5">
      <failurecause name="Info to Componont 1"/>
    </failuremode>
    <failuremode name="Failure Mode2 of Component 1"
FailureLocalEffect="None" SystemFailureLevelEffect="No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Message3 from Comp3to1_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode3 of Component 1"
FailureLocalEffect="Affect the next component"
SystemFailureLevelEffect=" Noeffect" severity="None"
occurrence="Remote" detection="Remote" RPN="8">
      <failurecause name="Message6 from Comp2to1_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode4 of Component 1"
FailureLocalEffect="Affect component 3"
SystemFailureLevelEffect="Low effect on the system" severity="Low"
occurrence="Remote" detection="Remote" RPN="40">
      <failurecause name="Message8 from Com3to1_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode5 of Component 1"
FailureLocalEffect="Affects Component 2"
SystemFailureLevelEffect="Low effect" severity="None"
occurrence="Beyond Moderate" detection="AlmostCertain" RPN="5">
      <failurecause name="Message16 from Comp2to1_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component2">
    <failuremode name="Failure Mode1 of Component 2"
FailureLocalEffect="Affect other component"
```

```
SystemFailureLevelEffect="High" severity="VeryLow"
occurrence="ExtreemlyHigh" detection="Low" RPN="216">
      <failurecause name="Info from comp1to2"/>
    </failuremode>
    <failuremode name="Failure Mode2 of Component 2"
FailureLocalEffect="Affect one component"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Info from Comp5to2"/>
    </failuremode>
    <failuremode name="Failure Mode3 of Component 2&#xA;"
FailureLocalEffect="No effect" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="Mederate" RPN="5">
      <failurecause name="Message1 from Com1to2_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode4 of Component 2"
FailureLocalEffect="No effect" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Message5 from Comp3to2_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode5 of Component 2"
FailureLocalEffect="Affect other component"
SystemFailureLevelEffect="Affect the system" severity="None"
occurrence="ExtremelyHigh" detection="Low" RPN="54">
      <failurecause name="Message11 from Comp1to2_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode6 of Component 2"
FailureLocalEffect="Low effect" SystemFailureLevelEffect="No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Message15 from Comp5to2_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component3">
    <failuremode name="Failure Mode1 of Component 3"
FailureLocalEffect="No effect" SystemFailureLevelEffect="Low Effect"
severity="Low" occurrence="Beyond Moderate" detection="Almost
Certain" RPN="25">
      <failurecause name="Info from Comp1to3"/>
    </failuremode>
    <failuremode name="Failure Mode2  of Component 3&#xA;"
FailureLocalEffect="Affects others" SystemFailureLevelEffect="No
Effect" severity="None" occurrence="VeryHigh" detection="Mederate"
RPN="40">
      <failurecause name="Info from Comp4to3"/>
    </failuremode>
    <failuremode name="Failure Mode3 of Component 3"
FailureLocalEffect="Affect Component 2"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message2 from Comp2to3_MessageRecv"/>
    </failuremode>
```

```
    <failuremode name="Failure Mode4 of Component 3"
FailureLocalEffect="Affect other components"
SystemFailureLevelEffect="No effect on the system" severity="High"
occurrence="Remote" detection="AlmostCertain" RPN="7">
      <failurecause name="Message4 from Comp1to3_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode5 of Component 3"
FailureLocalEffect="Affect component 2" SystemFailureLevelEffect="No
effect on the system" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message7 from Comp1to3_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode6 of Component 3"
FailureLocalEffect="Affect component5 effect"
SystemFailureLevelEffect="None" severity="None" occurrence="Remote"
detection="AlmostCertain" RPN="1">
      <failurecause name="Message13 from Comp5to3_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component4">
    <failuremode name="Failure Mode2 of Component 4"
FailureLocalEffect="None" SystemFailureLevelEffect="No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Info from Comp1to4"/>
    </failuremode>
    <failuremode name="Failure Mode1 of Component 4&#xA;"
FailureLocalEffect="High effect" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Info from Comp2to4"/>
    </failuremode>
    <failuremode name="Failure Mode4 of Component 4"
FailureLocalEffect="Low effect" SystemFailureLevelEffect="No effect
on the system" severity="High" occurrence="Remote" detection="High"
RPN="21">
      <failurecause name="Info from Comp5to4"/>
    </failuremode>
    <failuremode name="Failure Mode3 of Component 4"
FailureLocalEffect="No effcet" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Info from Comp3to4"/>
    </failuremode>
    <failuremode name="Failure Mode5  of Component 4"
FailureLocalEffect="None" SystemFailureLevelEffect="No effect"
severity="None" occurrence="High" detection="AlmostCertain" RPN="7">
      <failurecause name="Message4 from Comp3to4_MessageRecv"/>
    </failuremode>
  </component>
  <component name="Component5">
    <failuremode name="Failure Mode4 of Component 5"
FailureLocalEffect="Affects others&#xA;"
SystemFailureLevelEffect="Affect the system&#xA;"
```

```
severity="VeryHigh" occurrence="High" detection="AlmostCertain"
RPN="56">
      <failurecause name="Info from Comp1to5"/>
    </failuremode>
    <failuremode name="Failure Mode1 of Component 5"
FailureLocalEffect="None" SystemFailureLevelEffect="Moderate effect"
severity="None" occurrence="High" detection="Very Low" RPN="49">
      <failurecause name="Info from Comp4to5"/>
    </failuremode>
    <failuremode name="Failure Mode2 of Component 5"
FailureLocalEffect="None" SystemFailureLevelEffect="No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Info from Comp2to5"/>
    </failuremode>
    <failuremode name="Failure Mode3 of Component 5"
FailureLocalEffect="None" SystemFailureLevelEffect="No effect"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Info from Comp3to5"/>
    </failuremode>
    <failuremode name="Failure Mode5  of Component 5"
FailureLocalEffect="Affects Component 2"
SystemFailureLevelEffect="Affect the System" severity="Minor"
occurrence="Remote" detection="VeryLow" RPN="21">
      <failurecause name="Message12 from Comp2to5_MessageRecv"/>
    </failuremode>
    <failuremode name="Failure Mode6 of Component 5"
FailureLocalEffect="No effect" SystemFailureLevelEffect="None"
severity="None" occurrence="Remote" detection="AlmostCertain"
RPN="1">
      <failurecause name="Message14 from Comp4to5_MessageRecv"/>
    </failuremode>
  </component>
</System>
```

References

[AMY06] D. Amyot, H. Farah, and J.F. Roy, "Evaluation of Development Tools for Domain-Specific Modeling Languages," in *System Analysis and Modeling: Language Profiles.* Vol. 4320, R. Gotzhein and R. Reed, Eds., ed: Springer Berlin Heidelberg, pp. 183-197, 2006.

[ATL12] "ATL User Guide", wiki.eclipse.org/ATL/User_Guide, 2012, last accessed July 28, 2016.

[AVIZ04] Avizienis A., Laprie J.C., Randell B., Landwehr C. "Basic concepts and taxonomy of dependable and secure computing". *IEEE Transactions on Dependable and Secure Computing* 01(1):11–33. (2004)

[BERN12] Bernardi, Simona, Merseguer, José, Petriu, Dorina C, "Dependability modeling and analysis of UML-based software systems", *ACM Computing Surveys*, Vol. 45, Issue 1, November 2012,

[BOS11] Thomas Bosch, Brigitte Mathiak "XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas"6th International Conference on Internet Technology and Secured Transactions, 11-14, Abu Dhabi, United Arab Emirates, December 2011

[BOW01] Bowles, J. B and Chi Wan, "Software Failure Modes and Effects Analysis for a Small Embedded Control System", Proc. of the 2001 Reliability and Maintainability Symposium, Philadelphia PA, pp. 1 – 6, January 2001.

[BRJ01] Booch, G., Rumbaugh J., Jacobson, I., "The Unified Modeling Language user guide", (Addison-Wesley Object Technology Series) April 2001

[DAV09] David, P., Idasiak, V., and Kratz, F. Improving reliability studies with SysML. In RAMS09: Proceedings of the Reliability, 2009

[DAV10] David, P., Idasiak, V., and Kratz, F., 2010. "Reliability study of complex physical systems using SysML", Reliability Engineering and System 95 431 – 450. (2010)

[DEL04] Zacanas De Lemos "FMEA Software Program for Managing Preventive Maintenance of Medical Equipment" 2004.

[EMF15] Eclipse Foundation, "Eclipse Modeling Framework (EMF)", www.eclipse.org/modeling/emf/, last visited July26, 2016

[FUE04] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. "An Introduction to UML Profiles" UPGRADE .The European Journal for the Informatics Professional. Vol. V, No. 2, April 2004.

[GRO09] Hans Gronniger, Jan Oliver Ringert, and Bernhard Rumpe "System Model-Based Definition of Modeling Language Semantics" IFIP International Federation for Information Processing 2009

[GSF96] Goddard Space Flight Center (GSFC) (1996-08-10). Performing a Failure Mode and Effects Analysis, Goddard Space Flight Center, http://www.everyspec.com/NASA/NASA-GSFC/GSFCCode Series/GSFC_431_REF_000370_2297/, 1996.

[GUI03] Guiochet. J., G. Motet, C. Baron, and G. Boy, "Integration of UML in human factors analysis for safety of a medical robot for tele-echography," in Proc. IEEE Int. Conf. Intell. Robots Syst., Las Vegas, NV, USA, Oct., pp. 3212–3217. 2003.

[GUI04a] Guiochet, J., Baron, C.: "UML based risk analysis – Application to a medical robot". Proc. of the Quality Reliability and Maintenance 5th International Conference, Oxford, UK, pp. 213–216, Professional Engineering Publishing, I Mech E. April, 2004 (2004)

[HAR04] Harel, D., Rumpe, B.: "Meaningful Modeling: What's the Semantics of "Semantics"? Computer" 37(10), 64–72 (2004)

[HAS05] Hassan, A., Goseva-Popstojaneva, K., Ammar, H. 2005. "UML Based Severity Analysis Methodology". In Proc. of Annual Reliability and Maintainability Symposium (RAMS 2005). IEEE, Pages: 158 - 164, DOI: 10.1109/RAMS.1408355. 2005

[ISO12] ISO/IEC/IEEE "International Standard-Information technology-Modeling Languages Part2: Syntax and Semantics for IDEF1X97 (IDEF Object)" ISO/IEC/IEEE 31320-2:2012 First Edition 2012/09/15

[KEP04] Stephan Kepser, "A Simple Proof for the Turing-Completeness of XSLT and XQuery" Proceedings of the Extreme Markup Languages® 2004 Conference, Montréal, Quebec, Canada 2-6 August 2004.

[KOL15] D. Kolovos, L. Rose, A. Garcia-Dominguez, R.Paige, The Epsilon Book www.eclipse.org/epsilon/doc/bool/, 2015, last accessed January 2016.

[KUM13] Kumar. C, Maass. E. "FMEAs using a functional modeling based approach," IEEE, 59th Annual Reliability and Maintainability Symposium, pp. 1-6, 2013.

[LI14] Chao Li, Liang Dou, Zongyuan Yang “A Metamodeling level transformation from UML sequence diagrams to Coq” , ICTCS 2014, pages 147-157, 2014.

[LYU96] LYU, M.R., (Editor). Handbook of Software Reliability Engineering. IEEE Computer Society Press. 1996

[MART11] Object Management Group, “UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.1, OMG doc. formal/2011-06-02, 2011.

[MDA03] Object Management Group, "MDA Guide Version 1.0.1", document omg/03-06-01, 2003.

[MHE16] Faïda Mhenni, Nga Nguyen, and Jean-Yves Choley “SafeSysE: A Safety Analysis Integration in Systems Engineering Approach” IEEE Systems Journal, Volume: PP, Issue: 99 Pages: 1 – 12, Year: 2016.

[MOF11] Object Management Group, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1”, OMG Document: formal/2011-01-01, January 2011.

[MOR11] Mark A. Morris “Failure Mode and Effects Analysis based on FMEA 4th Edition” ASQ Automotive Division Webinar Nov. 30, 2011

[MRAZ05] Mraz, Miha, and Bernhard Huber. FMEA - FMECA. Rep. Ljubljana: University of Ljubljana, 2005

[MUS09] Sadaf Mustafiz “Dependability-Oriented Model-Driven Requirements Engineering for Reactive Systems”, PhD thesis, School of Computer Science McGill University, Montreal, Quebec, October 2009.

[NAS04] NASA-STD-8719.13B, “Software Safety NASA Technical Standard”, July 8, 2004

[OZA04] Nathaniel Ozarin, “Failure Modes and Effects Analysis during Design of Computer Software” Proceedings of Reliability and Maintainability, USA, pp.201-206, 2004. [PAP15] Eclipse Foundation, “Papyrus Modeling Environment”, eclipse.org/papyrus, last accessed January 2016

[PET95] Ivars. Peterson. Fatal Defect: Chasing Killer Computer Bugs. Random House, Inc., New York, 1995.

[PETR13] Dorina C. Petriu, Mohammad Alhaj, Rasha Tawhid. “Software Performance Modeling”. 2013.

[QVT11] Object Management Group, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Version 1.1”, OMG Document: formal/2011-01-01, January 2011.

[SEL03] Selic, Bran (2003) “The pragmatics of model-driven development”. *IEEE Software*, Vol. 20, Issue 5, pp.19–25, May 2003.

[SEL14] Selic, Bran, and Gerard, Sebastien, *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, 2014 Elsevier Inc.

[SPA03] Spath Patrim L “Using failure mode and effects analysis to improve patient safety. (Home Study Program)” AORN Journal, Vol.78 Item 1, p15 -23, July 2003..

[TAGU11] Kenji Taguchi. “Meta Modeling Approach to Safety Standard for Consumer Devices”. National Institute of Advanced Industrial Science and Technology. 2011 June 22.

[UML15] Object Management Group, “OMG Unified Modeling Language” Version 2.5, OMG document formal-15-03-01, 2015

[USM74] US Mil-Std-1629 (ships) "Procedures for Performing a Failure Mode, Effects and Criticality Analysis", Nov. 1.1974

[YU11] Suiran Yu, Qingyan Yang, Jiwen Liu, Minxian Pan “A Comparison of FMEA, AFMEA and FTA”. Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference Pages:954 – 960 Year: 2011

[ZAR16] Mana Hassanzadeh Zargari “Automatic Derivation of LQN Performance Models from UML software models using Epsilon” Master Thesis, Electrical and Computer Engineering Carleton University, Ottawa, Ontario. 2016

[ZHA14] Zhao Zhao “UML Model to Fault Tree Model Transformation for Dependability Analysis” Master thesis, Department of System and Computer Engineering, Carleton University Ottawa, ON, Canada August 2014.