Integration of safety analysis in model-driven software development

M.A. de Miguel J.F. Briones J.P. Silva A. Alonso

Technical University of Madrid, E.T.S.I. Telecomunicación, Ciudad Universitaria, Madrid 28040, Spain

Abstract: Safety critical software requires integrating verification techniques in software development methods. Software architectures must guarantee that developed systems will meet safety requirements and safety analyses are frequently used in the assessment. Safety engineers and software architects must reach a common understanding on an optimal architecture from both perspectives. Currently both groups of engineers apply different modelling techniques and languages: safety analysis models and software modelling languages. The solutions proposed seek to integrate both domains coupling the languages of each domain. It constitutes a sound example of the use of language engineering to improve efficiency in a software-related domain. A model-driven development approach and the use of a platform-independent language are used to bridge the gap between safety analyses (failure mode effects and criticality analysis and fault tree analysis) and software development languages (e.g. unified modelling language). Language abstract syntaxes (metamodels), profiles, language mappings (model transformations) and language refinements, support the direct application of safety analysis to software architectures for the verification of safety requirements. Model consistency and the possibility of automation are found among the benefits.

1 Introduction

In model-driven developments (MDD), (MDD is a paradigm for development of software applications) models are on the critical path of software development. MDD assumes a development sequence based on different types of models, especially platform-independent and platform-specific models. However, safety critical software involves some other types of models that support the verification of safety requirements. Integrating both modelling approaches reduces the number of problems because of inconsistencies and model development costs, but this requires adapting traditional MDD model paths and some tool integration support. We propose solutions to support integration with tool-independent implementations.

Not only can architectural models be used for the description of development properties, but they can also include safety properties that characterise software components, requirements and software systems in general. Using the architecture model as a common reference in both activities improves the consistency of safety analysis and software development. In this approach, the software development models are the main focus in safety critical software analysis, but safety analysis concepts complement the architectural modelling languages. Software architectures in safety critical systems are designed taking into account safety requirements defined in previous phases of the software development process, and safety engineers must ensure, as much as possible, that architectures proposed guarantee safety requirements before starting implementation or maintenance.

In this paper, we propose solutions to incorporate safety requirements in software architecture based on safety objectives, and for the evaluation of these software architectures based on safety analysis methods: fault tree analysis (FTA) and failure mode effects and criticality analysis (FMECA). The results of these analyses are used to detect inconsistencies of software architectures and safety requirements, and for the evaluation of architectures.

Fundamental to the model-driven architecture (MDA) (MDA integrates the application of several standards for the application of an MDD approach. http://www.omg.org/mda), unified modelling language (UML) is a general modelling language that does not include modelling elements for the description of safety concepts in particular. Some extensions to UML are required to integrate safety concepts with development architectures.

The safety concepts used in our work are described in the Eurocontrol recommendations . The practical experiences that we include in this document are the results of MODELWARE project. These results have been developed in close collaboration with Thales ATM, experts in safety critical systems (air traffic control). The primary goals of our research are to integrate safety analysis activities and software architecture development, to make both activities consistent, and also to provide some metrics for the evaluation of safety-aware software architectures.

The rest of this section completes the introduction describing the integration process, giving an illustrative example, and presenting the language engineering challenges involved in the research. Section 2 clarifies the modelling of safety attributes in software architectures and to achieve it some safety concepts need to be explained. Section 3 describes safety analysis languages. Section 4 give certain details of the safety modelling framework implemented in our research. Section 5 explains how safety analysis models can be created from the concepts presented in Section 2. Section 6 includes an example and some discussions. Section 7 includes related work and finally section 8 includes some discussions.

1.1 Overview of the integration process

Safety critical software components require complex development processes. The goal of performing safety analyses along with the definition of the architecture and performing early evaluations of the architecture, reflects the need of assuring the safety of the system from the beginning in order to reduce the development cost of safety for critical parts of the system.

Safety analysis models are used in the verification and safety evaluation of safety critical software. The assessment must reflect the safety guarantees of the software developed and, thus, safety models and development models must be consistent and well integrated. Software engineers propose initial software architectures that safety engineers annotate and improve with safety-specific concepts, such as safety requirements and software assurance levels (SWALs). Safety analysis models provide initial safety evaluations.

Safety analysis methods, in general, do not have an associated software architecture, and so we need to identify a modelling approach for the description of software architecture, and define rules for the application of analysis methods in these architectures. A general approach is the following: (i) In a conceptual model, the main abstractions of architectures and the safety concepts that complement them are identified. (ii) The representation of these architectures in UML requires the identification of UML modelling elements that support architectural concepts, and some UML extensions to represent specific safety notions. (iii) The same concepts must be represented in terms of safety analysis models, and we must define some rules for the representation of architecture and safety concepts in models. Conceptual models represent the intermediate internal models to support the mappings from annotated architectural models to safety analysis models.

Annotations have to be made in a formal manner to allow the later reuse of data in safety analysis. This requires the definition of UML extensions that support safety modelling concepts. Current generation of modelling tools include some facilities that support this MDD approach (e.g. interchange file formats, application programming interfaces (API) to access repositories, and transformation languages). However, the interoperability of modelling tools is tool dependent, and general standard facilities are not supported yet (e.g. facilities for the invocation of general modelling services, such as analysis of safety models).

The automatic generation of safety analyses from the architecture models is the key to the process. Without it, safety engineers have to redo the safety analyses each time they want to assess the safety of the architecture.

The results of the analysis are incorporated back into the architecture as formal annotations. Both software and safety engineers can communicate using the same document, the architectural models, to propose improvements if the architecture does not fit cost and safety requirements. The whole process is cyclical and ends when costs and safety requirements are fulfilled (Fig. 1).

1.2 Illustrative example

Fig. 2 depicts some of the visual diagrams and tables found during the process. The left of the figure represents a common structural diagram in a software modelling language. The safety goal represents a probabilistic

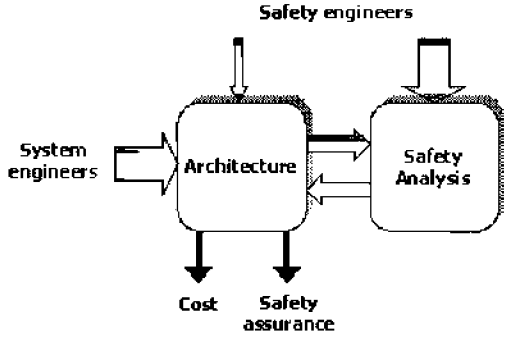


Figure 1 *Integration of development architectures and safety analysis*

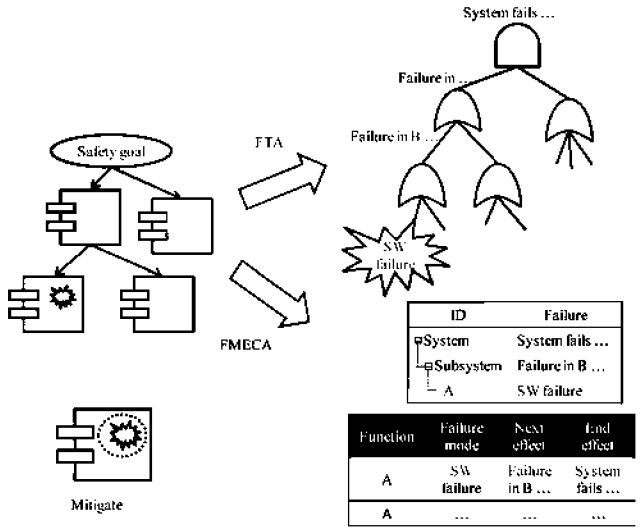


Figure 2 *Illustrative example*

objective concerning safety. If the failure of a component is easy to occur, the safety goal will not be able to be met. Safety analyses are used to determine how the process development for each part of the system should be completed. If the failure of a function impacts critical safety goals, a more disciplined process for this function is needed to limit the likelihood of development faults. Another alternative is the use of mitigation solutions. They are included in the system and reflected in the safety analyses and, thus, the conclusions will differ.

Reliability and fault analysis methods are basic tools for the evaluation of alternative architectures and for the quantitative evaluation of safety assurance levels of components. There are several methods for the safety evaluation of systems FTA (FTA is a systematic way of prospectively examining a design for possible ways in which failure can occur. The analysis considers the possible direct proximate causes that could lead to the event and seeks their origins) and FMECA (FMECA is an analysis language to identify potential design weaknesses in hardware, software and systems in general) are two examples of well known safety analysis methods with tool support FMECA is a bottom-up approach to analyse component reliability. It allows us to determine which components might need special attention and any measures necessary during design and implementation. For each component, the effects are identified and analysed in order to evaluate the severity of their consequences. FTA can be used for qualitative and quantitative analysis. It is useful for the qualitative identification of design weaknesses and for the quantitative measurement of the probability of hazard occurrences. We use FMECA and FTA for the safety evaluation of architectural models. They are well established and they are supported by several tool vendors.

In the software domain, if a faulty piece is found, it should be fixed.

1.3 Language engineering challenges

In the software domain, modelling and programming languages have been continuously used. The term 'safety analysis models' was previously used to refer diagrams and tables used by safety engineers to assess how safe a system is. We could consider each safety analysis method a language (the FTA/FMECA language). Certainly, these languages significantly differ from those of the software domain, especially from those found in an MDD environment. Safety analysis languages are contained in specialised safety tools, being vendor-specific languages, or in the specialists' head, frequently as simply office tools used to draw analyses. The solutions proposed in this paper pursue to integrate both domains constituting a sound example of the use of software language engineering to improve efficiency in software-related domains. Several language-engineering issues were addressed:

- Safety analysis languages were formalised following the MDA principles. Two metamodels for FMECA and FTA were designed and implemented.

- A safety language was proposed. This language is platform-independent, where the platform is the particular analysis method used. Such a language does not exist and would be very valuable to improve human communication, to allow tool interoperability and to boost tool development. We acknowledge that the proposed language is only a prototype and much effort should be placed to achieve these ambitions.

- Language refinement is used to convert instances from this safety language into instances of the safety analysis languages. We used rule-based programming support to implement this refinement.

- Safety analyses are bound to system elements, for example, software modelling elements, but current analyses languages do not allow reflecting software architecture changes into safety analyses. We proposed the use of extensions to the software modelling language as a modelling language for safety. Thus, we integrate the languages of both domains using the software modelling language as a common reference. This approach, along with automation, enables the binding between language instance elements.

- Consistency management between language instances is accomplished by using automation. Automation involves several issues to be resolved, mainly implementation and those humanrelated.

- A requirement of the work was that several software modelling languages could be used. Tool support was created to fulfil this requirement allowing other modelling languages to be integrated, currently different implementations of UML.

- To couple the different languages presented in our work, a consistent tool chain was implemented.

2 Modelling safety-aware software architectures

2.1 Safety process and concepts

Software safety can be thought of as a function that ensures safety in early development phases and continues to do so throughout the entire software development. Frola and Miller introduce the safety-related activities, hazard and risk analysis, as part of project management activities (Fig. 3).

Hazard analysis is fundamental in any safety-aware software system development and aims to identify hazards. A hazard is a state or set of conditions of a system or object that, together with other conditions in the environment of the system or object, will lead inevitably to an accident Hazards are usually classified according to severity levels. A severity level identifies the worst possible consequences that could result from the hazard. Hazard analysis is the central matter in functional hazard assessment (FHA) Eurocontrol phase (FHA is a top-down iterative process, initiated at the beginning of the development or modification of an air navigation system whose objective is to determine how safe does the system need to be), and the risk assessment subprofile of the quality of service standard gives support to FHA activities. The concept of a safety objective is used to define, qualitatively or quantitatively, the maximum frequency at which a hazard can be expected to occur. The results of the hazard analysis are used to define these objectives, which are the main safety input for preliminary system safety assessment (PSSA) Eurocontrol phase (PSSA is a top-down iterative process, initiated at the beginning of the design or modification of an air navigation system whose objective is to determine whether the proposed architecture is able to achieve a tolerable risk). The work that underlies this paper provides solutions to software architecture modelling and PSSA. deals in detail with the embedding in PSSA of the aspects this paper presents. The results of PSSA are a collection of safety requirements

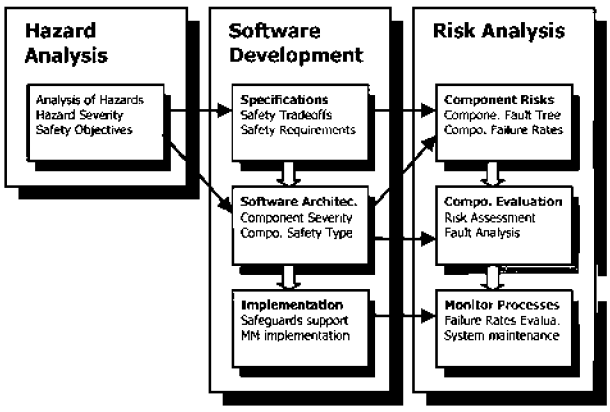


Figure 3 *General integration of safety analysis and software development*

allocated to system elements, among which the safety assurance level system elements need to be satisfied. We consider system elements to be software components and they will have to provide SWALs.

Architectural components have associated risks that are identified in hazard and risk assessment processes. These risks are a function of both, the likelihood of the hazard that leads to an accident and the consequences of the accident. Components can be supported by mitigation means to help in satisfying the assurance level. Mitigation means are solutions to reduce the likelihood or the severity of hazard consequences. Their risk reduction capacity depends on their platform-specific implementations and we might need a combination of mitigation means to achieve the desired reduction. The application of mitigation means has some associated costs. For example, to improve reliability two mitigations means are commonly used: functionality redundancy and fault recovery. They have significant cost consequences because of memory and CPU resource consumption. Mitigation means are reusable software elements and an organisation/project can define a catalogue of mitigation means and their implementation methods. Combinations of mitigation means can also be included in this catalogue. Examples introduced in Section 2.4 are part of a model library of mitigation means.

Although software components are typically identified in software architectural phases, safety analyses should be taken into account to improve architectures. These assessments are done based on probabilities of component failure occurrences and risk reduction of mitigation means. These numbers are estimations that must be monitored and confirmed in the implementations. Architectural decisions should set up: components used and their arrangement, their assurance levels, and the concrete configuration of the mitigation means employed.

2.2 Conceptual model of the safety-aware architecture language

Fig. 4 introduces the kernel of a conceptual model, showing the main concepts for the description of safety-aware software architectures. Based on this conceptual model, we have created a metamodel that represents safety concepts of component-based safety-aware architectures (SAA). We use this metamodel for the internal representation of architectures, for evaluation and for transformation purposes. The following are the basic elements of this conceptual model and metamodel:

- *Safety objective.* These represent the safety results of an FHA Eurocontrol phase and define the safety requirements in the system. Eurocontrol document FHA – Severity Classification Schema SAF.ET1.ST03.1000-MAN-01-01-03-D has been a fundamental document for the characterisation of safety objectives.

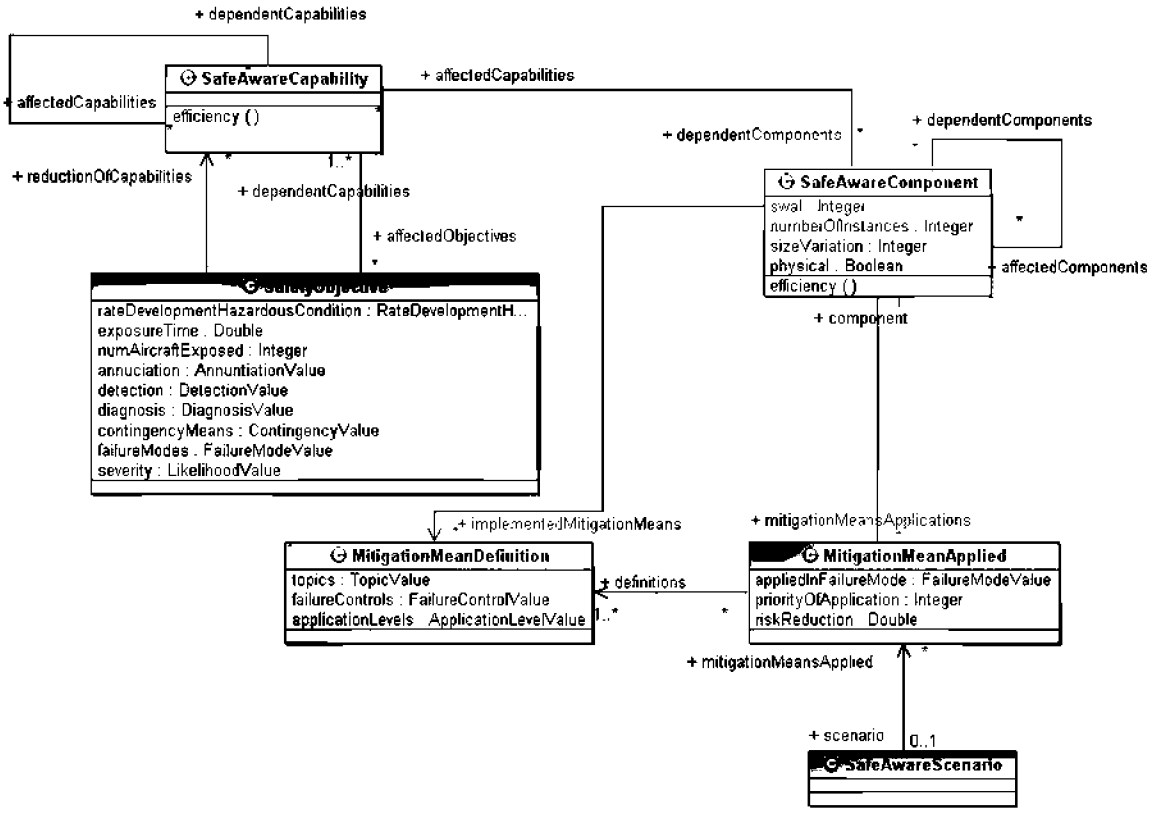


Figure 4 *Conceptual model for the description of safety-aware component-based architectures*

• *Safety-aware capability*. SafetyAwareCapability represents software capabilities that have associated safety objectives. The safety-aware capabilities are supported directly and indirectly by a set of safety-aware components.

• *Safety-aware component*. SafetyAwareComponent represents logical and physical components that support Safety AwareCapabilities. The SWAL of a software component depends on the severity of the safety objectives affecting that component. SWAL establishes levels of confidence in the software development process, by using a disciplined method to limit the likelihood of development errors that could impact safety A failure in a component without mitigation means and SWAL1 would directly cause a failure of system function resulting in a hazard whose end-effect will be catastrophic. A failure in a component with a SWAL4 would have no immediate effect on safety.

• *Mitigation mean definition*. Mitigation means are any software that allows avoidance, detection, propagation control or mitigation of effects of failures in order to meet the safety requirements resulting from the safety analysis. A mitigation mean definition characterises a software means independent of its application.

• *Mitigation mean application*. A SafetyAwareComponent applies a set of mitigation means to meet safety requirements. The combination of a specific set of mitigation means in a component produces a specific risk reduction.

• *Scenario*. Software architectures can have alternative associated solutions that produce different safety results. The scenario facilitates the representation of alternative solutions in the same model. The safety evaluation of a model is based on a specific scenario.

2.3 Integration of safety concepts in a software modelling language: UML

Based on this conceptual model, we define the abstract syntax of a language (metamodel) for the description of SAA. We use a profile for the concrete representation of concepts included in SAA, but we do the transformation and analysis based on abstract syntax. The profile has been designed: (i) to represent, based on UML modelling elements, SAA concepts and (ii) to integrate safety concepts with component-based software architectures.

Fig. 5 is a structural diagram in UML2 that includes the implementation of this profile in Objecteering 6 (the metamodel of this Objecteering version is an evolution from UML 1.4–UML 2.0). We have implemented this profile in Objecteering 5.3 (the metamodel of this version is based on UML 1.4) and UML 2 Eclipse plugin and rational software architect (RSA) (RSA is an IBM tool for software development which integrates some basic tools for the design of UML models and development of web services and Java software). The differences of these versions are the base metaclasses of UML metamodels, the implementation

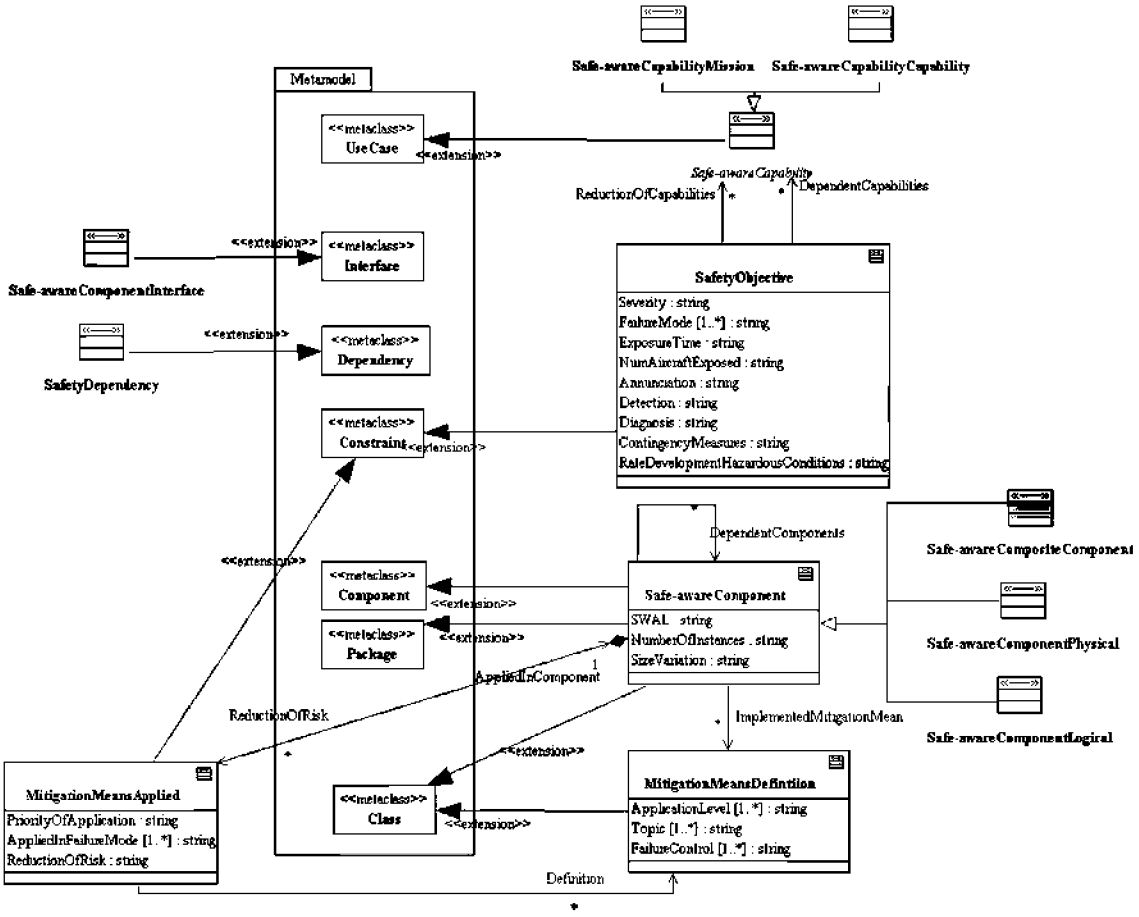


Figure 5 *UML safety architecture profile*

of associations between stereotypes and the use of enumeration types and primitive types in the definition of stereotype attributes (Objecteering does not allow the use of enumerations as a type of stereotype attributes).

This profile includes stereotypes for the description of most of the metaclasses included in SAA. Stereotype attributes represent properties included in metaclasses. We have defined a mapping from UML + safety architecture profile to SAA, and we can transform the UML models into the equivalent SAA abstract syntax.

2.4 Modelling examples

This section includes some simple examples of the application of the concepts introduced previously. Safety concepts annotate software architectures to improve their safety semantics. They include invented safety requirements, mitigation means and components similar to real elements, but simplified.

An example of a safety requirement, a result of FHA and input for PSSA, could be:

'The probability of Undetected Corruption Of Secondary Surveillance Trajectory Assignment for any aircraft shall be no greater than extremely remote after 2 min as long as the failure condition leads to undetected corruption trajectory code.'

This requirement leads to a safety objective that must formalise the requirement in terms of some safety objective parameters. The safety objectives include several parameters, in this case are especially important: (i) the amount of time the hazard exists (exposure time) is 2 min, (ii) the number of aircraft exposed is only one aircraft, (iii) the rate of development of the hazardous condition (e.g. sudden, moderate, slow) compared with the average time required for recovering from unsafe conditions is fast, (iv) the annunciation of the hazard requires an interpretation, because we need to recognise the wrong trajectory, and the diagnosis of the error can be sometimes incorrect, (v) in some cases, it may also be possible to consider the availability of alternative procedures, fall-back equipment and the ability to apply contingency measures; in this case we have some contingency measures to ensure that the second surveillance trajectory can be wrong and (vi) the requirement explicitly includes the likelihood as extremely remote.

Fig. 6 shows the way the safety objective can be edited. The safety objective is represented as a UML constraint (annotated with SafetyObjective stereotype and the stereotype attributes), but, because the safety objectives combine several parameters

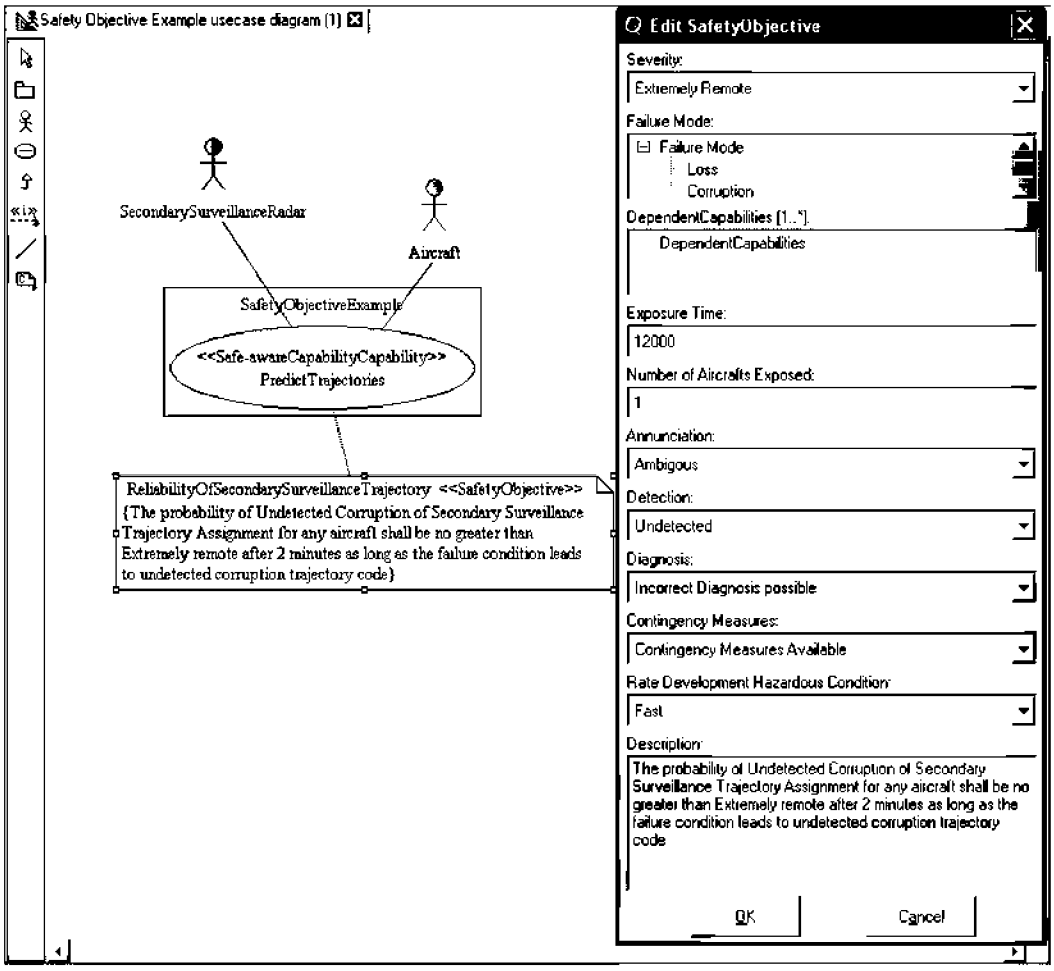


Figure 6 *UML safety objective example*

with specific values, we have developed specific wizards for their creation. The wizard updates the stereotype attributes and controls their consistency. In this figure, the safety objective is attached to a safety capability (represented with a UML use case) that represents the functionality in the system for the construction of surveillance trajectories. This safety requirement affects this capability and components that support this functionality. The model includes UML dependencies (annotated with SafetyDependency stereotype) that represent the safety-aware traceability from use cases to physical components.

We are going to introduce the design of a redundant and connections checked component. This is a redundant kind of component that verifies the consistency of its input and output messages at the arrival and output of new messages (component interfaces includes interceptors to evaluate it). The objective of this component is to support software components with high safety assurance levels. It reduces the hazard of loss based on redundancy and data-checking mitigation means: (i) Loss detectors, to identify the loss of the component and the location where it was lost. (ii) Replications, which distribute the invocation services depending on availability of component replicas. (iii) Isolation checks the initialisation of the node and the initialisation of any other component in the same node.

To avoid problems of corruption and functional errors, the component uses the following data-checking techniques: (i) Syntactic data checks, which ensure the syntactic redundancy of data types of messages. (ii) Semantic data checks, which ensure the consistency of data based on domain specific functionalities. (iii) Input data checks for the components that interface with external actors.

Replications and data checks are examples of mitigation means that require an integration process because the data checkers can raise exceptions that must be considered in the process of message delivery. Software architects would use this kind of component for the configuration/ characterisation of some safety critical application components.

Fig. 7 includes the component TrajectoryPrediction that supports the capability included in Fig. 6. We apply the mitigation means introduced to reduce the probability of error and ensure the severity that the safety objective imposes on this component (extremely remote). The SWAL of this component must ensure this severity, and only the mitigation

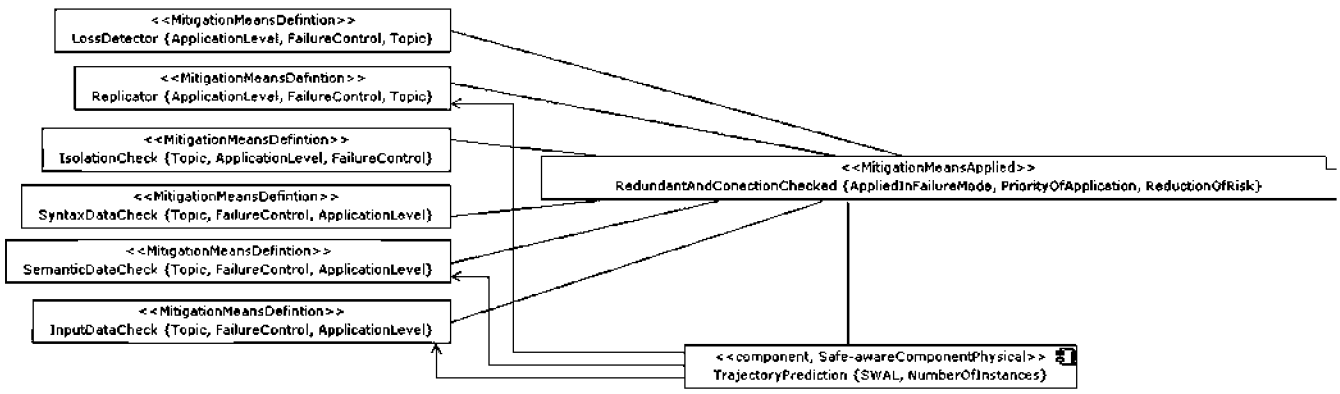


Figure 7 *UML mitigation means and applied examples of mitigation means*

means applied can reduce this SWAL. These mitigation means can be used to reduce the errors and losses in components that are affected by safety objectives such as the safety objective included in Fig. 6. The mitigation means are UML classes annotated with the MitigationMeansDefinition stereotype and its attributes: the level of application of mitigation means (component, process, system or hardware), the kind of failure control (avoidance, propagation, detection and mitigation) that the mitigation means provides and the topics of application of mitigation means (architecture, size and timing, initialisation and stop, input–output control, data management, internal communication, error recovery policy and fault tolerance). The combination of a certain set of mitigation means provides a specific reduction of risk. The MitigationMeansApplied determines this reduction, the priority of applicability (the same component can reduce the risk with different combinations) and the failure mode of application (loss, undetected corruption or error).

Mitigation mean definitions and applications can be used as reusable safety modelling elements: designed as model libraries and reused in different projects.

3 Safety analysis languages

3.1 FMECA metamodel

FMECA (failure modes, effects and criticality analysis) is an inductive approach to system design and reliability. It identifies each potential failure within a system or manufacturing process and uses severity classifications to show potential hazards associated with these failures

We identify three main elements in FMECA (Figs. 8–10 include the main diagrams of the FMECA metamodel):

- *FmecaSystem*. A FMECA safety analysis is constructed as a hierarchy of blocks. A FMECA system holds properties global to the analysis (such as name and reference identifiers of the system, or its life time) and the main block of the system. It supports the strategy of having some common phrases that could be reused for the description of failure modes and effects; this choice would be very useful if the metamodel is used to create FMECA editors. The SeverityCategory metaclass allows the characterisation of some categories for severities. FmecaSystem includes a classification of severity categories to be used for blocks and for the system itself. Fig. 8 includes FmecaSystem and its associations.

- *Block*. A block represents every component that makes up the system to be analysed. Blocks are specialised as SystemBlock, SubsystemBlock and FunctionBlock. This specialisation is done mainly because the same parameters in different blocks can have different semantics. That is, a failure rate in a block can be an input of the analysis or a result; in the conventional bottom-up execution of FMECA, failure rates are input in FunctionBlocks whereas

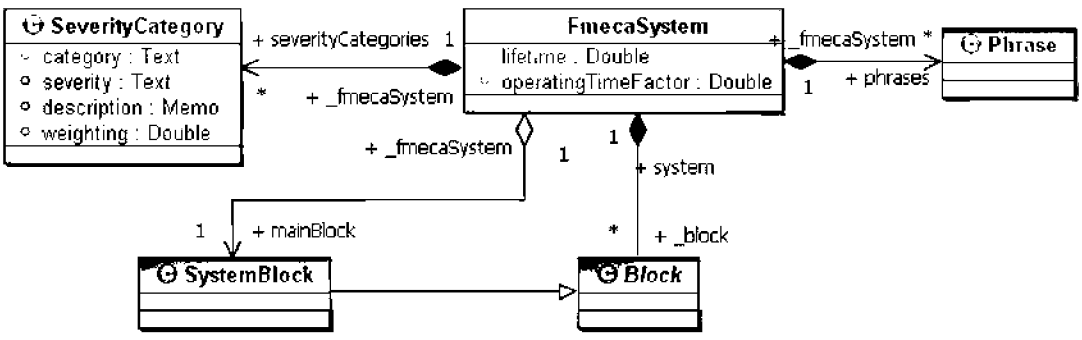


Figure 8 *FmecaSystem in FMECA metamodel*

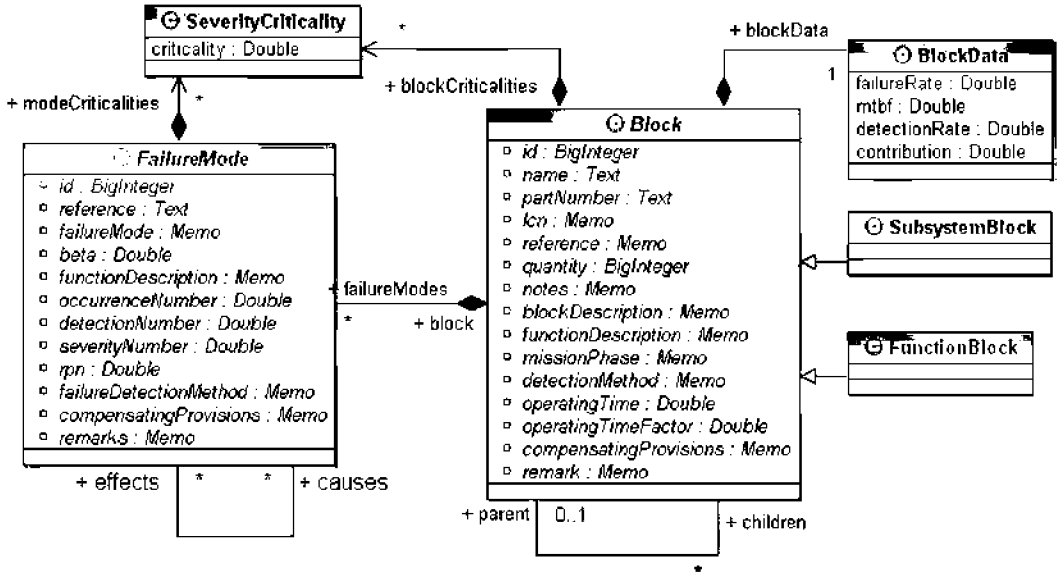


Figure 9 *Block in FMECA metamodel*

results in the SystemBlock. The BlockData metaclass contains those attributes whose semantics can change whereas Block contains all others. Fig. 9 includes Block and its associations.

• *FailureMode*. Each block has failure modes, modes in which the block could fail. These failure modes can be primary or derived from a cause-effect relationship if the failure occurrence depends on the failure of another block. A safety tool allows to mark these cause-effect dependencies at the same time as to provide a description for the failure mode among other parameters. Primary failure modes are represented by the class FunctionFailureMode, whereas derived ones are usually represented by SubsystemFailureMode. The final goal is to trace primary failure modes affecting end-effects, which are represented by the class SystemFailureMode. When the criticality analysis is performed in FMECA, both blocks and failure modes will end up with a criticality value for each severity level concerning it. Fig. 10 includes the description of FailureMode.

3.2 FTA metamodel

FTA [7, 6] is used during safety assessments to represent the logical interaction of component failures and other events in a system. It provides quantitative (probability theory is used), and qualitative (in the way of minimal cut sets that are combinations of events leading to a system-level failure when happening together) information. FTA's graphical representation consists of a tree made up of logical gates, such as AND & OR. Figs. 11–13 show diagrams that include the three main modelling elements of the metamodel:

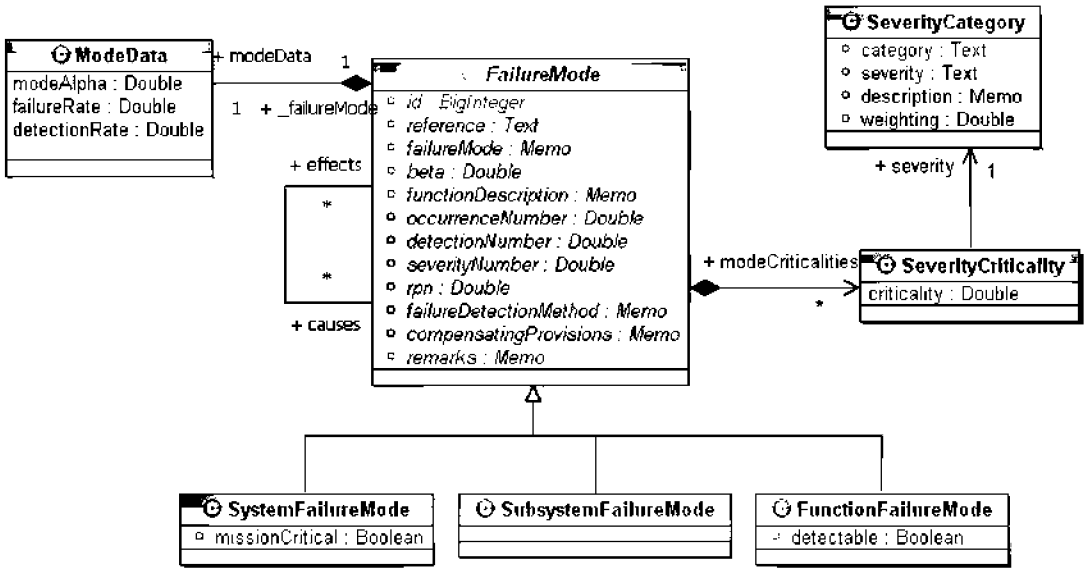


Figure 10 *FailureMode in FMECA metamodel*

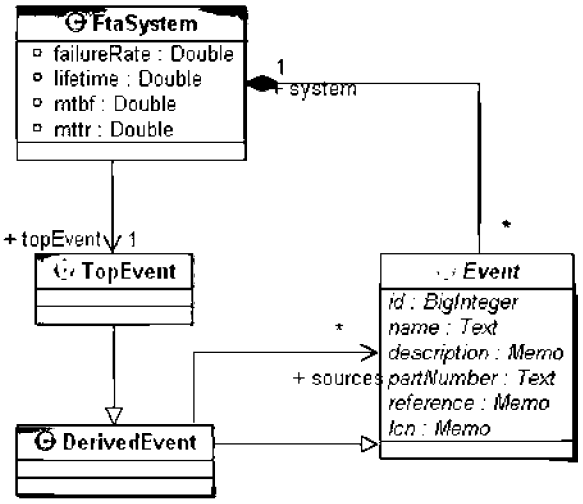


Figure 11 *FtaSystem in FTA metamodel*

reference to the TopEvent. Fig. 11 includes FtaSystem metaclass and its associations.

• *FtaSystem*. A FTA system is constructed as a set of interconnected gates. The metaclass FtaSystem holds all the properties global to the system (such as a name and reference IDs of the system, and life time). The first thing to be done when starting an FTA is to define the hazard to be analysed, which is represented with the metaclass TopEvent, a specialisation of DerivedEvent. FtaSystem references all the events within the system, with a special

• *Gate*. A gate is a logical function of some inputs. Typical logical functions are AND, OR and VOTE. The output of a gate is an event derived from its inputs, represented with DerivedEvent. The inputs are Events, either primary events (PrimaryEvent) or the output of other gates (DerivedEvent). Therefore we use to symbolise a gate by the combination of a DerivedEvent and a Gate, holding most of the properties of the former (name, identifiers and results) whereas by the latter mainly the kind of gate. The metaclass DerivedResult holds the data obtained from FTA calculations. The most important results will be attached to the TopEvent, but the calculations could output data for every DerivedEvent if it is considered as a subtree. Other output data are related to importance and minimal cut set analyses. Fig. 12 includes the metaclasses gate and DerivedEvent and their relations.

• *Event*. If DerivedEvents characterise the result of a gate, PrimaryEvents are the main events in an FTA. They represent a primary failure of a component, an external event or any other occurrence that could affect the global safety of the system. Primary events have an associated failure model, usually modelled with probability parameters. The most typical failure model is the RateFailureModel created from a constant failure and repair rate. BasicEvent is the metaclass to

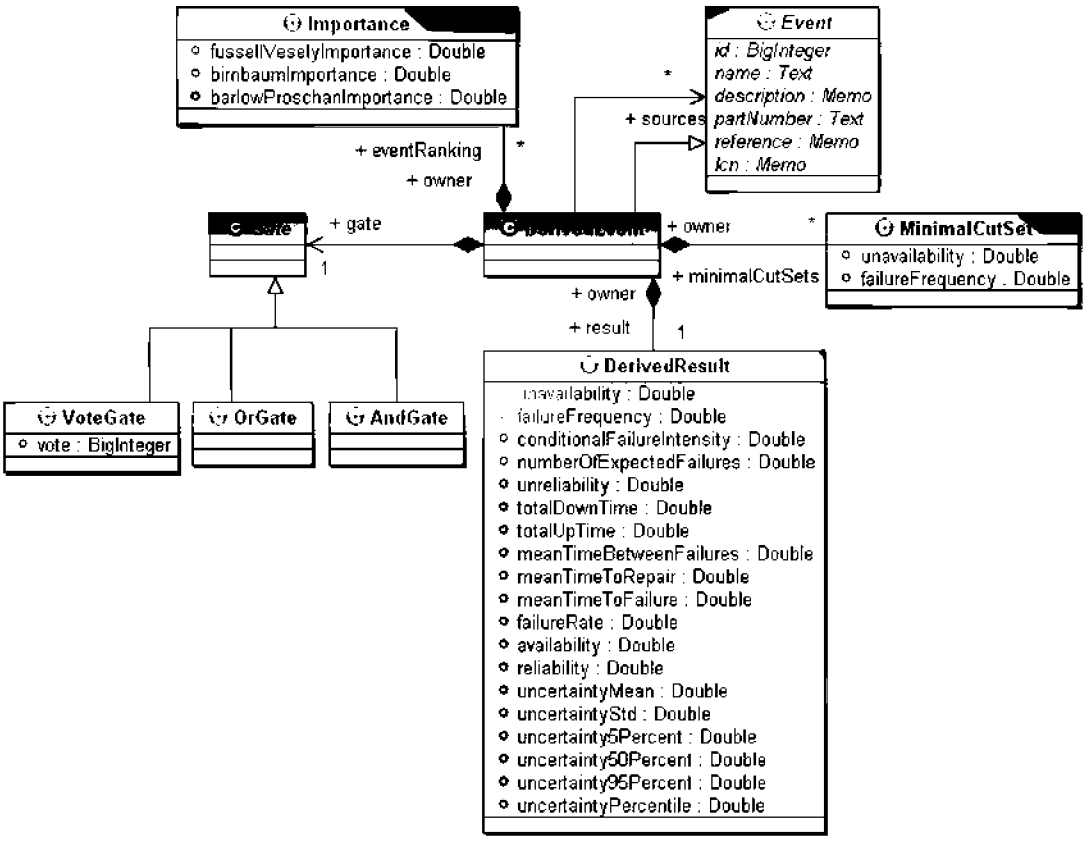


Figure 12 *Gate and DerivedEvent in FTA metamodel*

represent common primary events, such as the use of UndevelopedEvent limited to primary event whose main properties are not fully described. Fig. 13 includes Event and its subclass PrimaryEvent.

4 Safety modelling framework

The safety modelling framework (SMF) includes a set of tools (metamodels, profiles, model transformations and evaluators of metrics) for the analysis and development of safety-aware UML architectures. SMF has been designed (Fig. 14) with reference to meta-object facilities (MOF) (MOF includes a set of foundation standards, e.g. MOF core specification, XMI and JMI, that define a metadata management framework; http://www.omg.org/mof) and implemented using eclipse modelling framework (EMF)

EMF is a framework used to define and implement structured data models SMF includes a set of metamodels and a UML profile to support modelling languages and transformations based on these modelling languages.

Fig. 15 includes the SMF tool chain and coupling with external tools. Current implementation couples SMF with three modelling UML tools (TA: Objecteering 5.3 and 6.0 and RSA based on UML2 eclipse plugin) and one safety and reliability analysis tool (TB: ITEM). This tool chain includes four main phases: (i) coupling of SMF and UML modelling tools, (ii) transition from UML models to SAA, (iii) transition from SAA to safety analysis models (FTA and FMECA) and (iv) coupling with safety analysis tools. In this tool chain there are four modelling languages (UML, SAA, FTA and FMECA), and one UML profile (safety profile) adapted to two UML tools. There are two key model transformations: (i) T1: transformation from UML + Profile to SAA and (ii) T2: transformation from SAA to FTA and FMECA.

4.1 SMF interfaces

SMF interchanges (TA, TB) models with three external tools (Objecteering versions 5.3 and 6.0 UML2 Eclipse plugin and RSA and ITEM). SMF reproduces the metamodels of these tools inside EMF. Objecteering 5.3 and 6 use a metamodel-based approach to access to Objecteering repositories and models. Objecteering 5.3 includes a tool specific language (J language) that includes primitives to navigate over the metamodel. This tool and language do not allow defining new metaclasses. Objecteering 5.3 is a UML 1.4 metamodel (with some minimal differences) and Objecteering 6 metamodel is an evolution from 1.4 to UML 2.0 metamodel. We use three different interchange methods:

1. SMF interchanges metadata with Objecteering 5.3. XMI (http://www.omg.org/technology/documents/formal/xmi.htm) is an OMG standard for exchanging metadata information via XML. It can be used for any metadata whose meta-model can be expressed in MOF. The most common use of XMI is as an interchange format for UML models, although it can also be used for serialisation of models of other languages (metamodels). We have rebuilt Objecteering 5.3 and 6.0 metamodels in EMF, and EMF generators provide the Java API and XMI persistency. We have developed a module in language J in Objecteering 5.3 for importation/exportation of this type of XMI files. The interchange format is XMI files, and SMF and Objecteering 5.3 are executed in two independent processes. It is TA in Fig. 15 for Objecteering 5.3.

2. Objecteering 6.0 includes language J and a Java API. We use this API to access Objecteering repositories. The implementation is written in Java and uses EMF metamodel reflection to download Objecteering models in EMF repositories. This reflection combines EMF

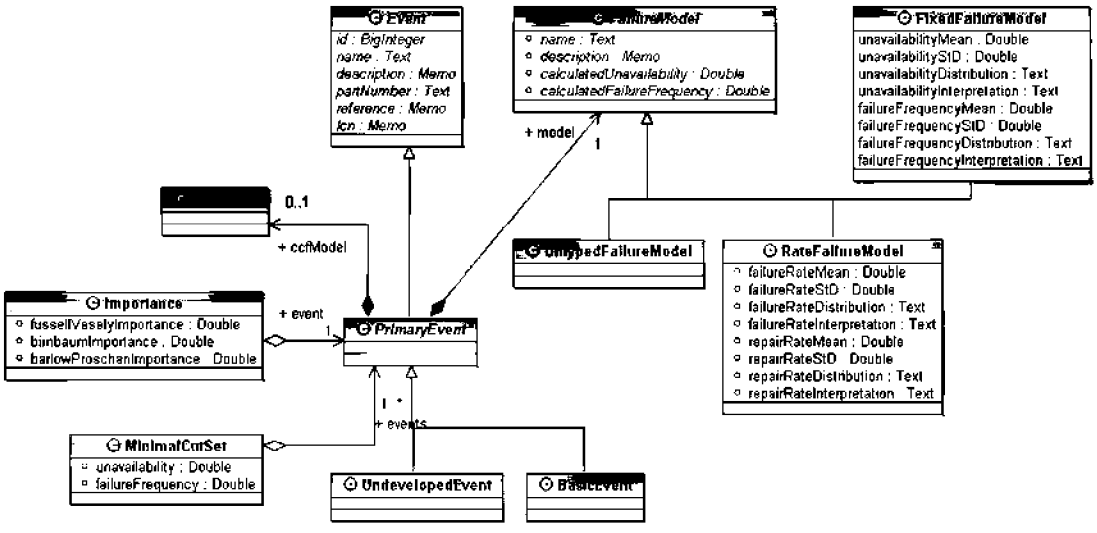


Figure 13 *Event and PrimaryEvent in FTA metamodel*

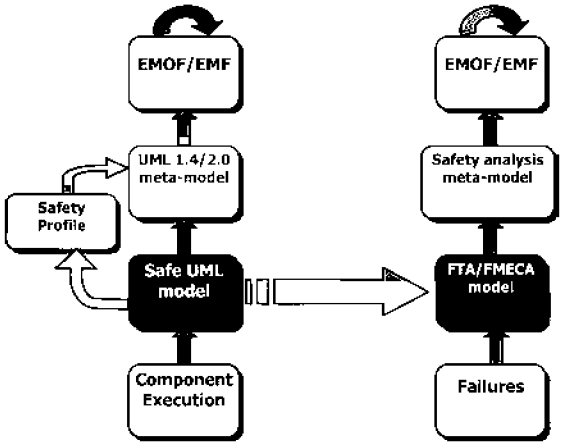


Figure 14 *Support of safety modelling languages and transformations*

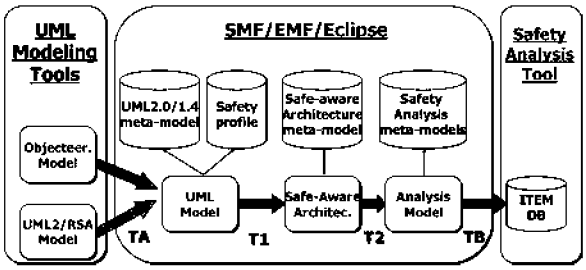


Figure 15 *Safety modelling framework tool chain*

reflection facilities (EMF provides dynamically the attributes and relations of metaclasses) and Java reflection for the Java API. SMF includes EMF notifiers for all Objecteering 6.0 EMF metaclasses, which automatically updates Objecteering repositories whenever any modification is detected in EMF repositories. These notifiers are implemented based on reflection facilities. It is TA in Fig. 15 for Objecteering 6. With UML2.0, we use the same approach, reusing UML2 plugin Java libraries. This is TA in Fig. 15 for UML2.

3. We access ITEM repositories in a Jet database via an Object/relational mapping framework. This framework provides facilities to access ITEM databases, and we reuse it as a Java library in SMF. This is TB in Fig. 15 for ITEM.

The design of model transformations depends on source and target modelling languages. We developed the transformations using a metamodel façade pattern. The transformation includes references to the metamodel accessors methods (set and get Java methods), and this makes the transformation dependent on the UML metamodel. Currently in SMF, we support three kinds of models as input (Objecteering 5.3, Objecteering 6.0 and UML2), with important differences. We have created a neutral UML metamodel, which presents any UML metamodel. This metamodel is based on UML2 but it only includes the modelling elements handled in SAA transformations and profiles (Neutral UML includes 35 metaclasses, and the merged version of UML2 metamodel includes 227 metaclasses). In this approach, transformation only includes references to Neutral UML accessors and Neutral UML handles the specific references to UML accessors. At run-time, a Neutral UML model references a concrete UML model (Objecteering 5.3, 6.0 or UML2). Any model element in Neutral UML includes a reference to a concrete element in the concrete model.

4.2 Safety modelling language repositories

Safety modelling languages are SAA, FMECA and FTA. For each modelling, language repository facilities were created to handle models that use those languages.

SAA is an abstract modelling language (there is no concrete syntax implementation of this language) used to describe SAAs based on methods included in Eurocontrol guidelines. The SAA metamodel defines a modelling language based on the abstractions and conceptual model introduced in Section 2.2. We could create a concrete syntax to represent these models, but currently we represent these concepts with UML modelling elements annotated with safety extensions. We use SAA models to compute metrics and to reduce the complexity of analysis model generation.

FMECA and FTA are languages to express such safety analyses. The languages are well understood, but to our knowledge there is no attempt to formalise them in a model-driven environment. These languages are currently used for implementing transformations, being the target and enabling to create safety analyses in a tool-independent way, neither did we create an associate concrete syntax. In its place, models are exported (TB) to the preferred safety tool, which indeed contains a concrete syntax for these languages.

4.3 Model transformations

Fig. 15 includes the transformation T1. This is a transformation from UML + profile to SAA. The safety profile was designed based on the SAA metamodel and the conceptual model and, thus, there is a direct correspondence between UML + profile elements and SAA modelling elements. This transformation is tedious but rather straightforward.

Transformation T2 represents the generation of safety analysis models taking SAA models as input. Although we have implemented this transformation in Java, we designed it using a set of rules to represent the transformation. Each transformation rule is designed with two collaborations: (i) query collaboration, which is the query on SAA models for the execution of the rule and (ii) construction collaboration, which represents the new instances created in FTA and

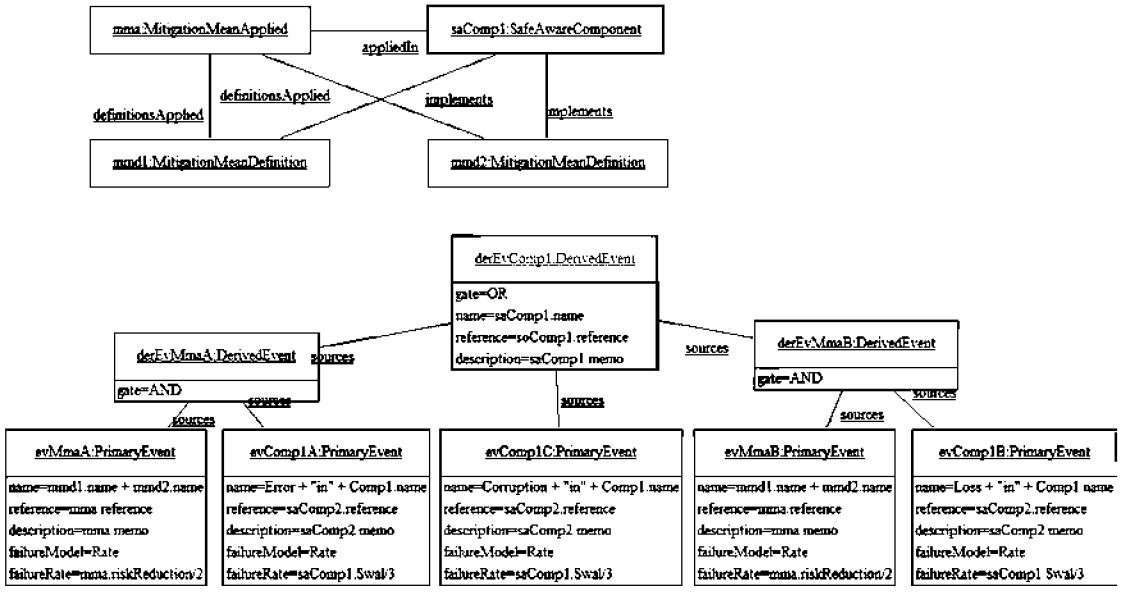


Figure 16 *FMECA: query collaboration on the top and construction collaboration at the bottom*

FMECA models during the rule execution. Further details of T2 will be given in the next section.

5 Safety analysis model generation

In this section, we introduce our general solutions for the construction of safety analysis models based on development architectures. Our approach identifies safety characteristics of the system and associates them with system elements that will be used later to generate the safety analyses. For this reason, it is essential that all safety characteristics and links to be used in the safety analysis are identified and formalised. The UML extensions for safety realise this condition for our safety analysis purposes. Safety analysts should learn how to include safety parameters into the architecture; that is, they should know how to use the UML extensions. In contrast, they do not need to know the details of the safety analysis languages. The creation of safety analyses models from safety annotations in UML is a transformation where we convert development models into safety analysis models. The following subsections will describe how this transformation converts UML entities (system elements and safety annotations) into safety analysis model entities (FMECA blocks, failure modes and FTA events and gates).

5.1 Safety analysis automation

As there are numerous safety analysis methods, there are also different ways to perform FMECA or FTA according to particular objectives. This variety found in safety analyses contrasts with analyses in other fields such as real-time systems, where the objective is unambiguous (makes the system schedulable) and the application of the analysis is implicit in the analysis (no different ways to perform the analysis). The aim that is, pursued by applying FMECA and FTA to software architectures, is framed in the general process of PSSA of Eurocontrol: to evaluate how an architecture can cause some hazards to occur, to estimate whether the architecture can fulfil the safety objectives of the system, to allocate SWALs to software components for making it possible to accomplish these objectives and to discern mitigation means to prevent hazards from occurring. It is important to note that some safety analyses are subject to automation: (i) Those that extensively use information stored in a way that allows its processing (for those an MDA approach facilitates this processing, obviously, by requiring that safety characteristics are recorded using a modelling language), (ii) Those from which we learn more from the results of the analysis (an organised table, a graph or some meaningful ciphers) than from the process.

In an evolutionary process, like the PSSA and Eurocontrol approaches, automation is of great importance, with safety analyses growing at the same time as software models, reflecting the progress of the design. Nevertheless, some potential issues may arise as a consequence of the way humans interact with automation. For instance, actions must be taken so that safety analysts understand what the automation is doing and why. The best approach is for them to collaborate in the automation process, which also helps to prevent them from disliking or not relying on the automation process. There is also the danger that analysts will abuse automation by losing awareness of the process and becoming less conscious of errors. Therefore even though automation may avoid tedious analyses, it should be applied with caution.

Safety engineers create safety analysis models instead of safety models not bound to a specific analysis. They do not model hazards, failure modes and safety relationships, but

rather FTA, FMECA, ETA or Markov; thus, their models must be reworked to perform any other type of safety analysis, however similar. On the other hand, safety models could enable to create different safety analyses. The creation of safety analysis models from safety annotations in UML is a model transformation in which we convert development models into safety analysis models. To formalise FTA and FMECA languages, we designed the corresponding metamodels. They intend to be agnostic to any specific implementation of these types of analyses in a safety tool.

5.2 FMECA model creation

FMECA model generation is derived from the model of software components. Each FMECA component corresponds to a software component. The following attributes are common to all components:

- *Failure mode*. Each component can have an associated set of failure modes.

- *Failure rate for each failure mode*. This is the failure rate for cases where the component does not use any mitigation means.

- *Hazards associated for each failure mode*. These are the final effects on the operational environment.

- *Severity level for each failure mode*. This is the worst possible consequence that could result from each component failure mode.

Safety-aware capabilities and safety-aware components from architecture models shape the structure of an FMECA model by constituting FMECA blocks. A function block can only represent a bottom-level safety-aware component (one which does not depend on any other). Conversely, a top-level safety-aware capability (one with at least one safety objective associated directly) is represented by a top-level subsystem block. The only way to include a mitigation mean in an FMECA model is in unison with the safety-aware component it affects.

A safety objective is an invariant that the system must fulfil, so that when it is not fulfilled we identify a failure mode of the system (the system block). Failure modes of safety-aware components are hard to identify. In a preliminary safety assessment, we use common keywords like corruption, loss and error; other schemes are valid though.

In FMECA, the dependency between blocks is like a parent-child relationship whereas the dependency between failure modes is a cause-effect relationship (limited from a child to its parent). In software models, safety-aware elements are interrelated by means of safety dependencies that we use to create the hierarchy of the analysis model. This proposal makes it necessary to replicate FMECA model elements in some cases; for instance, when a safety-aware component affects two safety-aware capabilities, we create two subsystem blocks for the safety-aware capability.

5.2.1 FMECA transformation rules: The following rules represent how the FMECA models are constructed.

1. The general rules for the generation of FMECA models are:

2. Only one FmecaSystem is created with its corresponding SystemBlock.

3. SafetyAwareCapability not affecting any other with SafetyDependency ⇒ A SubsystemBlock is created for the SafetyAwareCapability whose parent is the SystemBlock. A SystemFailureMode in the SystemBlock is created for each SafetyObjective that affects the SafetyAwareCapability.

4. SafetyAwareCapability affecting another SafetyAware Capability with a SafetyDependency ⇒ A Subsystem Block is created for the former, the latter being the parent.

5. SafetyAwareComponent affecting a SafetyAwareCapability with a SafetyDependency ⇒ A SubsystemBlock is created for the former, the latter being the parent.

6. SafetyAwareComponent affecting another SafetyAware Component with a SafetyDependency and affected by others ⇒ A SubsystemBlock is created for the former, the latter being the parent.

7. SafetyAwareComponent affecting another SafetyAware Component with a SafetyDependency and not affected by any other ⇒ A FunctionBlock is created for the former, the latter being the parent. A FunctionFailureMode is created for each foreseen failure mode (3 in the current scheme: Error, Loss and Corruption).

8. In the tree FmecaSystem, all the FailureModes of a child affect the FailureModes of the parent. If the parent has no FailureMode, one is created for it.

9. SWAL of the SafetyAwareComponent in combination with the RiskReduction of the MitigationMeanApplication ⇒ They are equally apportioned among failure modes.

10. Criticality of the SafetyObjective ⇒ It is assigned as the severity of the SystemFailureMode that represents the SafetyObjective.

Fig. 16 depicts an example including rules 3 and 7.

5.3 FTA model creation

FTA models should start by defining a hazard. As mentioned for FMECA, the non-fulfilment of a safety objective constitutes a hazard. Consequently, it seems reasonable that a different FTA model has to be created for each safety objective in the system and, thus, we will end up with several FTA models for only one system. In FTA, it can be reasonable to create an FTA model from a lower-level hazard although this is not supported by our FTA model generation.

Primary events will represent the failure of a bottom-level safety-aware component. The other important element in FTA, gates, will represent the failure of a higher-level safety-aware component or capability. Because of the lack of information about safety dependencies, we must suppose the worst case and use the OR gate; a gate will produce true data if one of the components supporting it fails.

Mitigation means impede failure propagation in a degree according to risk reduction, so that they are represented as AND gates. Only some failure modes of the component will be mitigated, those declared in the mitigation mean definition.

5.3.1 FTA transformation rules: The following rules represent how the FTA models are constructed.

1. SafetyObjective ⇒ An FtaSystem is created for each SafetyObjective. The TopEvent of the system is also created, holding an OrGate.

2. SafetyAwareCapability attached to a SafetyObjective ⇒ For the former, a DerivedEvent holding an OrGate is created. The output of this DerivedEvent is connected to the TopEvent.

3. SafetyAwareCapability affecting another SafetyAware Capability with a SafetyDependency ⇒ For the former, a DerivedEvent holding an OrGate is created. The output of the created DerivedEvent is connected as a source of the DerivedEvent of the latter.

4. SafetyAwareCapability affecting other SafetyAware Capability with a SafetyDependency ⇒ For the former, a DerivedEvent holding an OrGate is created. The output of the created DerivedEvent is connected as a source of the DerivedEvent of the latter.

5. SafetyAwareComponent affecting another SafetyAware Component with a SafetyDependency ⇒ For the former, a DerivedEvent holding an OrGate is created. The output of the created DerivedEvent is connected as a source of the DerivedEvent of the latter.

6. SafetyAwareComponent with no MitigationMean Applications ⇒ A PrimaryEvent is created for each foreseen failure mode (3 in the current scheme: Error, Loss and Corruption). The PrimaryEvents are connected as a source of the DerivedEvent representing the component.

7. SafetyAwareComponent with a MitigationMean Application but at least one of the MitigationMeanDefinitions of the MitigationMeanApplications is not implemented by the SafetyAwareComponent ⇒ as 6.

8. SafetyAwareComponent with MitigationMeanApplication and all the MitigationMeanDefinition's are implemented by the SafetyAwareComponent ⇒
a. For each foreseen failure mode not mitigated, a PrimaryEvent is created and connected to the DerivedEvent representing the component.
b. For each mitigated failure mode, a DerivedEvent holding an AndGate is created, the output of which is connected to the DerivedEvent representing the Component. Two PrimaryEvents are created and connected to it: one for the mitigated FailureMode and one representing the effects of the MitigationMeanApplication.

9. SWAL of the SafetyAwareComponent ⇒ The SWAL, previously converted to likelihood, is equally apportioned among the failure modes.

10. RiskReduction of the MitigationMeanApplication ⇒ It is equally apportioned among the mitigated failure modes.

Fig. 17 depicts an example including the rule 8 (also 9 and 10).

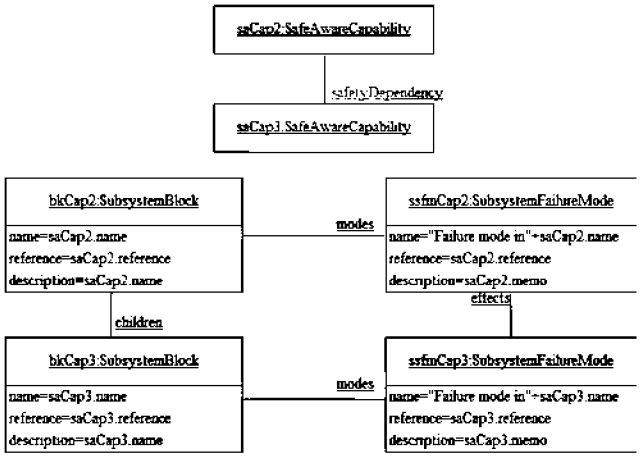


Figure 17 *FTA: query collaboration on the top and construction collaboration at the bottom*

5.4 Inputs and results

The automation of analysis model construction takes as input UML development architectures annotated with safety extensions and provides tables (FMECA), trees (FMECA, FTA) and diagrams (FTA) depicting the models. Safety analysts can now inspect them to see how the safety-critical elements of the architecture work together. FMECA

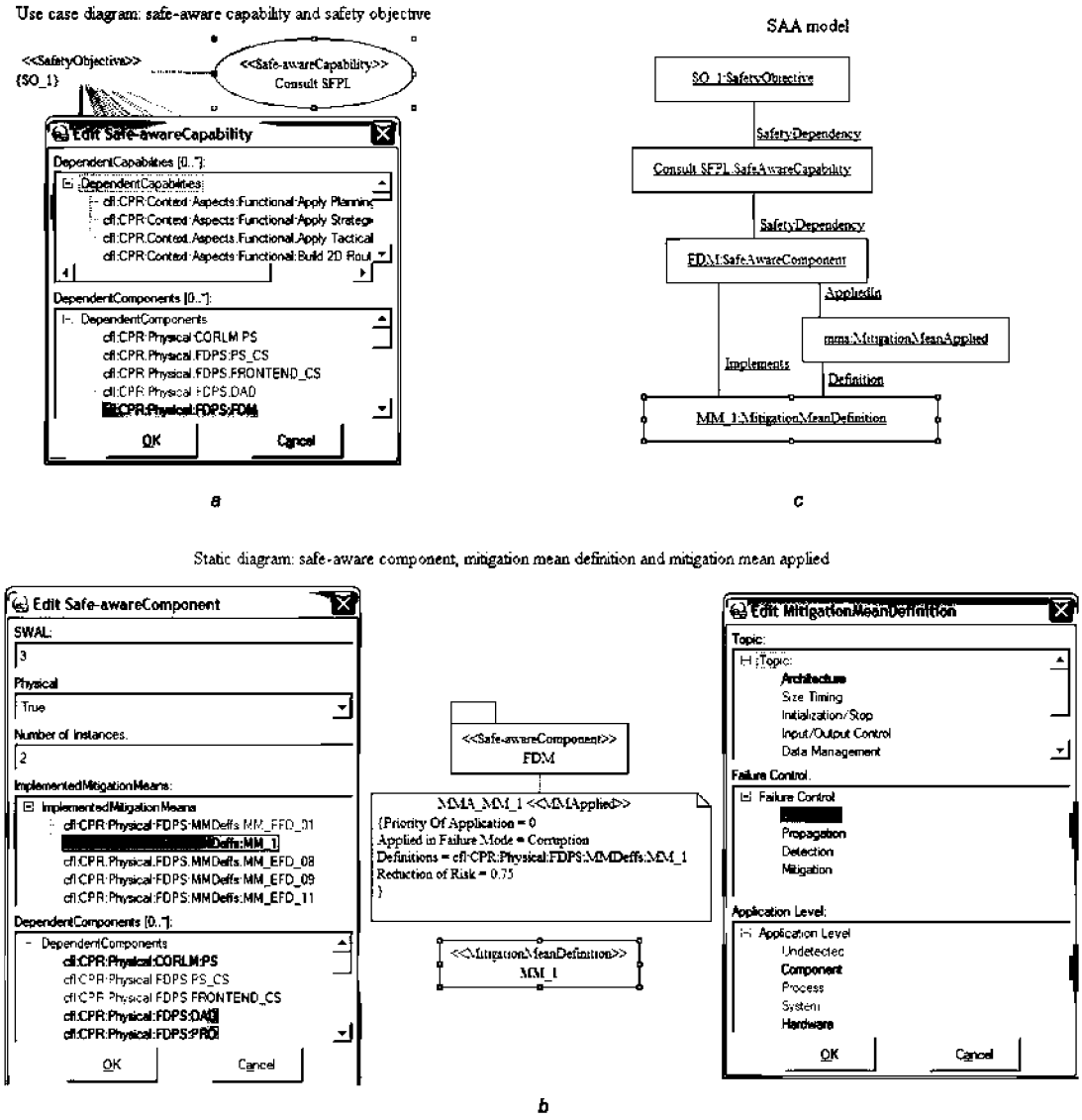


Figure 18 *UML models annotated with the safety profile and SAA model backing them*

a Use case diagram: safe-aware capability and safety objective
b Static diagram: safe-aware component, mitigation mean definition and mitigation mean applied
c SAA model

can indicate the severity of function failure modes by only setting the severity of system failure modes (this setting is already implemented in the automatic transformation). FTA can provide us the list of minimal cut sets, that is, the combinations of bottom-level failures leading to a hazard when happening together; we should pay attention to combinations with only one element.

One of our final aspirations with the safety analyses is to allocate SWAL to safety-aware components. Allocating a SWAL to a component does not imply calculating a failure rate for software, and hence SWALs cannot be used in the safety assessment process as can hardware failure rates. In the case of model-driven engineering with hardware, we could map component hardware assurance level to failure rates in safety analysis models, and the safety tool could provide quantity results.

The methodology provided by Eurocontrol strictly forbids allocating a failure rate to the software and we have to assume that the software fails. Allocation of SWALs is accomplished by looking at the overall system design in its operational environment. Therefore it is essential to keep the link between the safety-aware component to the end-effect and its maximum tolerable frequency of occurrence. Our work provides the link between a safety-aware component and the hazard but not to the end-effect, although it can be easily obtained from other documents. We have worked on safety analysis creation to provide quantity results by automating some hand-made processes ; these values can be used to derive SWALs.

6 Example and validation

6.1 Example

This section provides a comprehensive example including the whole process proposed in the paper. The process is rather complicated and the best way to become familiar with it is by means of a simple but comprehensive example model. The model does not contain a complex component-based architecture but a single software component. Another software component is included into the model to represent the mitigation mean application. This insertion would improve the safety of the system by reducing the occurrence probability of a component failure. In this example we have a capability, 'consult system flight plan (SFPL)', by which an air traffic controller (ATC) requests information about a flight plan. This capability needs to fulfil a safety objective, 'SO_1: The probability of detected and undetected corruption for greater than 5 min shall be no greater than 10−5 per hour'. The capability is hypothetically supported by a single component, 'flight data management (FDM)'. To achieve the safety goal, safety engineers have allocated a mitigation mean to this component 'MM_1: Processing servers are replicated on different nodes'.

Fig. 18*a* illustrates how, with the help of the safety profile, the constraint SO_1 is linked to the capability consult SFPL expressed as a use case. This relation is articulated within a use case diagram, and using the stereotypes SafetyObjective and SafetyAwareCapability. An *ad-hoc* editor enables us to make the component FDM dependent on the capability. Fig. 18*b* shows the safety-aware component, which has applied a mitigation mean. Within a static diagram we represent the component as a package annotated with a stereotype SafetyAwareComponent. A constraint with stereotype MMApplied indicates the application of MM_1, which the component properly implements (as shown in the *ad-hoc* editor on the left). Another editor (on the right) enables the definition of the mitigation mean. Finally 18*c* shows the safety-aware model that backs the previous models and editors' data.

So far, safety engineers need to collect information from several documents, meetings and know-how to build the analyses. SAA models collect the information likely to be used during the analyses construction, so that safety staff could build the analyses based on it. SAA models constitute the input of the transformation T2. Transformation T2 currently automates our proposal for the construction of FMECA and FTA. It basically consists in arranging the data included in SAA models to bring up safety analyses. The transformation creates in essence a graph of nodes from another one. The last step of the process would be the assessment of the analyses by safety staff, as soon as they import the analyses in their preferred safety (and graphical) tool.

Fig. 19*a* shows the output of the transformation for a FTA (note that not all model elements are included for simplicity, e.g. FtaSystem). This analysis will assess the safety objective SO_1, which is transformed into a TopEvent. 'Consult SFPL' and 'FDM' are mapped into DerivedEvent resulted from OR gates, whereas 'Error' and 'Loss' failure modes of FDM into BasicEvents. Since the failure mode 'Corruption' has been mitigated, an AND gate is used to

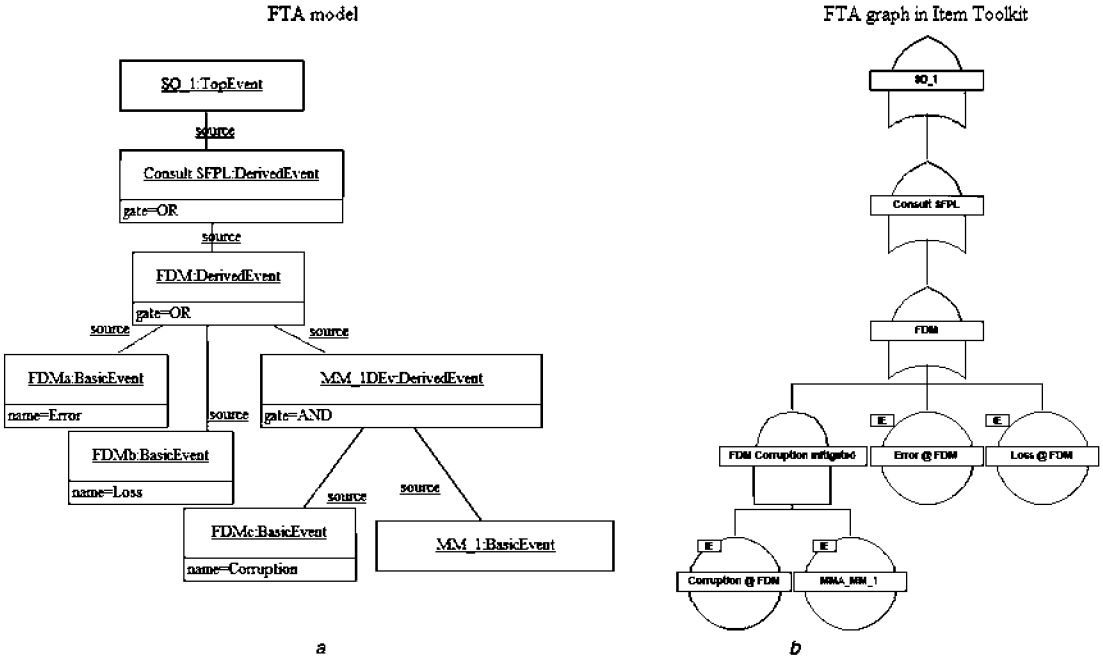


Figure 19 *Fault tree model and representation within item toolkit*

create a reduction of the failure mode's probability. This means that the corruption of FDM will only show up at the system level when both the failure mode occurs and the mitigation mean does not impede its propagation. When the output model is imported in the safety tool with the provided support, safety engineers can assess the FTA within the tool (Fig. 19*b*)).

Fig. 20 is the equivalent of Fig. 19 for FMECA. Side Fig. 20*a* shows the output of the transformation for a failure mode, effect and criticality analysis (as with FTA, note that not all model elements are included, e.g. FmecaSystem). This analysis assesses the system as a whole, as one can realise when the SystemBlock is 'atm'. 'Consult SFPL' is a SubsystemBlock and 'FDM' is translated into a FunctionBlock. The not-fulfilment of the only safety objective, 'SO_1', constitutes the only SystemFailureMode. Three FunctionFailureModes can be found: 'Loss', 'Corruption' and 'Error'. The failure mode at system-level is also called 'end-effect', and learning what are the failure modes at a low level which can cause its occurrence is one the main purposes of FMECA. This is shown in the table of Fig. 20*b*, which can be obtained with the safety tool (ITEM).

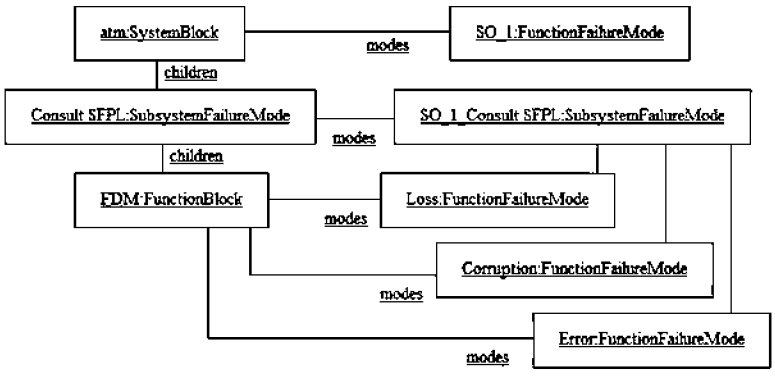
6.2 Validation and discussion

That was a simple example with a few elements and, thus, one could easily create manually the FTA within the safety tool. We applied SMF to the evaluation of a subsytem in a real project: EUROCAT. This is a flight plan management system that improves current implementations of flight plan systems. Thales-ATM is currently developing the system. The system baseline has more than three million lines of code and we cooperated on the early safety evaluations of one subsystem. This subsystem is modelled in Objecteering 5.3 and the size of the XMI model is 13 MB (the size of the UML2 merged metamodel is 1.3 MB, so the size of this subsystem model is around ten times the size of the UML2 metamodel). EUROCAT project represents a good scalability study for SMF. The performance bottle neck is in the generation and load of XMI files. Section 4.1 includes alternative solutions for Objecteering 6 that we have evaluated with this model, and the performance problems are solved using Java API. Thales-ATM uses version 5.3 because most tools, generators and other modules are not yet available in version 6.

The size of the EUROCAT subsystem allowed us to validate our proposal focusing on the analysis generation scalability. For this subsystem, we updated the architecture model with safety annotations based on FHA results of this project. According to these results we created safety objectives and we defined some mitigation means. The safety annotations were done with the UML profile and some graphical user interface wizards that made the application of UML extensions more user-friendly and secure. Profile and SAA metamodel provided good support for the integration of safety annotations in the architecture. In the end, we end up with a model with 21 safety-aware capabilities, 19 safety-aware components, five mitigation mean definitions and nine mitigation mean applications.

The model contained enough elements to validate the generation of safety analysis and those analyses fit the needs

FMECA model



a

FMECA result table

	End Effects					
	Mode-Ref. ID	End Effect	Severity ID	Severity Description	Detectable ID	Detectable
1	cfl.0	SO_1	II	Major severe - Large Reduction in Safety Margins	FDM.1	Loss
					FDM.2	Corruption
					FDM.3	Error

b

Figure 20 *FMECA model and result table in item toolkit*

they were designed for. The creation of the analyses without automation support would have required hard work. The stage of development implied changes in the architecture that had to be tracked by safety engineers and incorporated into safety analyses. What is more, the inherent evolutionary property of PSSA, to reach the combination of mitigation means that make it possible to meet safety requirements, added new changes to be tracked. In our approach, change is welcomed because reworking the analyses tracking changes is done automatically. Our approach scaled with the bottleneck previously mentioned. Generating the analyses took about 2 h caused by exporting the models using the Objecteering 5.3 programming support. Removing this step, the generation took less than 5 min.

Safety engineers could evaluate the generated analyses. However, we acknowledge that it is critical that analysis model generation needs to be adapted to safety engineers' particularities. They need to perform analyses to assess the architectures following a particular assessment methodology but also following their viewpoints (Section 5.1 coped with this concern). Work is needed in the SAA language to accept safety properties from different methodologies and analysis methods. A particular configuration in the analysis model generation may not fit the needs of every project and safety analysts. Safety engineers need to see the real benefits of the automated generation to cooperate in the election of the configuration. In the validation project, the easiness to create the analyses facilitated comparing alternative software architectures in search of optimality: safety feasibility, software requirement fulfilment, cost reduction and parameters.

Our work implied clear advantages: (i) engage safety engineers in the architecture design from the beginning benefiting the safety quality and cost of the project, (ii) safety information was stored in a way that can be processed not only for safety analysis generation, (iii) data consistency management where changes in the architecture design are translated into safety analyses, and safety properties are only annotated in architecture designs and (iv) analysis automation. The potential for the automatic generation of safety analyses becomes apparent when the system model grows or/and when there is a real benefit to maintain consistency between system models and safety analyses.

7 Related work

We will now introduce some related work about the support of safety in MDD. Some related work focuses on the automatic generation of code for safety critical systems like which use MDA for developing the software for the F-16 modular mission computer application software. In this experiment, they focused mainly on the automatic generation of software and on system portability across platforms. This reference is neither meant to represent safety concepts, nor possible safety assurance levels.

Software fault-tree analysis (SFTA) is an example of the adaptation of FTA to software systems. Most SFTA research efforts have been directed towards requirements or code. Components and software architectures require different levels of abstraction. Towhidnejad *et* propose some ideas for the application of FTA in software design. In this approach, FTA is part of the analysis of software designs.

Pai and Dugan propose some UML extension for the description of hardware and software redundancy, reliability dependencies and reconfigurations as well as, the transformation of UML models into FTA models. These notations are useful for the analysis of systems that mitigate risk with redundancy methods. However, they do not consider other methods such as checklists. The extensions are used to annotate deployment modelling elements, but they do not annotate architectural modelling elements such as interfaces or component behaviour description.

Other alternative analysis methods include the evaluation of contracts between components. The assembly and coupling of components developed by third parties create risks because of the incompatibility of the security and reliability characteristics of components. Some experimental methods are used for the specification of security attributes and for the evaluation of contracts

Avionics architecture description language (AADL [20]) is a good example of a modelling language that integrates reliability and safety analysis methods. The components can be equipped with reliability models, which are Markov chains that relate fault events and error states. The system description must describe how errors propagate among components. A reliability analysis tool combines the reliability models of individual components into a global Markov chain, and uses a separate tool (in this case the SURE/PAVE/PAWS tool from NASA Langley) or the Markov chain analysis. In safety analysis, each process has its own address space in an implementation. Safety levels and memory allocation properties can be declared for components. The MetaH tool partitioning analyser can partially verify that no error in a component with a lower safety level can propagate to a process with a higher safety level. Currently some efforts have been initiated to integrate AADL in modelling standards (UML 2.0 profile).

includes some UML extensions for the description of hazards and their relation to risks. The risk assessment subprofile of standard uses the new extensions and UML modelling elements in the description of models of hazards and risks. In this paper, we propose a solution that is, integrated with UML modelling elements, but we assume component-based software architectures, PSSA methods and FTA and FMECA analysis methods. The concepts described in this paper are specific to safety critical systems, and specially integrated with development architectures. They can be supported by quality of service (QoS) profile

or could be adapted as an improvement to a risk assessment subprofile.

They take the same choice of developing a UML profile, in this case for developing safety-critical systems compliant with RTCA DO-178B. This is indeed an extensive effort to create a profile, which could be used to automatically generate information that could even be submitted to certification authorities, for example, contributions to failure conditions and software requirement traceability. They use rules to extract this information from models. They do not seem to use the profile to generate analysis models, but certainly it could replace ours in our approach within an RTCA DO-178B environment.

We came across a recent research work on determining critical components and connectors using risk factors (considering complexity and severity levels) These risk factors are used to determine fault proneness among components, and hence they pay more attention to the components while coding and testing them.

An example of a problem identified by the Eurocontrol method is the execution of PSSA, and when the architecture is not completely defined it can lead to several problems; for instance, the architecture can be over-engineered to deal with uncertainty in the design [24]. The process we propose to perform PSSA can be considered as a lightweight PSSA as well. Nevertheless, our approach is based on the annotation of software architectures specified in UML and on the automatic generation of safety analyses, whereas their work focuses only on safety issues. Some other work in York enumerates several issues that arise when conducting PSSA, some of which we want to highlight as we think our proposal can contribute to this discussion. The first issue is to track changes in the design updating the assessment. This is how PSSA can motivate design decisions but tracking every change can cause the project to overrun budgets. A lightweight method of doing PSSA that becomes more rigorous as the design matures would be the best option, and an automatic lightweight method, as we propose, would be much better. The second issue deals with who owns the design and leads the design specification. Working on the same models must help safety and software engineers can cooperate as their communication is based on common foundations; when software engineers change the architecture, safety engineers can evaluate it immediately, and when safety engineers discover a way to create a safer version of the architecture, they can pass on the changes to software engineers. Our vision facilitates the creation of integrated project teams even though some difficulties involving cultural change might still need to be resolved. The third issue is to identify the failure modes of incomplete designs that have not yet been implemented. We are evaluating how to add support for the failure mode description in order to incorporate failure modes in architecture models, when identified; the automatic process would use these descriptions, adding them to the analysis results. The fourth issue concerns how to cope with the modification and evolution of the system, for which our approach offers some guidance.

8 Conclusions

MDD solutions provide support to improve the integration of software development and safety analysis. MDD infrastructures (e.g. meta object facilities and UML extensions) provide facilities to hold this integration. However, some improvements are needed; these improvements include the invocation of services in other tools and solutions to interchange modelling tool components that adapt model-driven facilities to domains and technologies. Safety analysis tools do not provide APIs for evaluating safety models from other modelling tools; modelling tools should provide API for the invocation of services from other tools, and the exportation of these modelling tool services should be defined with standard methods, currently not available. Current generation of UML modelling tools (e.g. RSA and Objecteering 6) need improvements to integrate basic MDD tools, such as metamodels, profiles, transformations, well-formedness rules, and JMI-based (Java Metadata Interface, based on MOF, defines a dynamic, platform-neutral infrastructure that enables the creation, storage, access, discovery, and exchange of metadata; http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html) code with a common asset. In SMF we have done important efforts to integrate these different types of tools.

Compared with hardware, software safety issues have received limited attention, since it is a fact that software has caused few safety problems. However, as software is increasingly becoming part of system functioning, gaining assurance for software is very important. The earlier this assurance is gained, the greater benefits we will be able to obtain. In this paper, we have presented a framework for conducting a preliminary assessment of the safety of a software system. Safety and software engineers can work together to arrive at a complete definition of software architecture with the tool support outlined in this paper. Part of this support is simply a specific implementation with room for different adaptations. For instance, safety engineers and certification authorities require their own type of analysis, by this we mean not only whether it is FMECA or FTA, but also how the analyses are implemented. Companies usually have a favoured safety tool to which our implementation could be tailored. Although the process we propose is largely tailored to Eurocontrol PSSA, we believe it can be applied fairly generally to the task of evaluating high-level models. Finally, work is also necessary because there is no common understanding of how to deal with software safety. After integrating our approach in a real development of air navigation systems, we corroborate some of the added value. The main benefits to safety and software engineers

working on the same models include the immediate availability of traceability between safety concepts and software elements, consistency in software architecture and safety information throughout process. Another achievement of our work is to automate the generation of safety analyses, which has already been discussed. In evolutionary processes like PSSA, this is a bonus in that it helps avoid the need to perform the analyses manually again and again. We consider that the separation of safety and safety analysis modelling is a must. Certain safety analyses can be considered as different arrangements of elements from a well known safety vocabulary such as hazard, fault, failure and failure propagation. Modelling and characterising safety vocabulary enable the derivation of a safety analysis from safety models. The UML profile proposed here could be extended in this way.

9 References

European organization for the safety of air navigation: 'Air navigation systems safety assessment methodology', Eurocontrol, April 2005

Modelware web page, available at: http://www.modelware-ist.org/

BRIONES J.F., DE MIGUEL M., SILVA J.P., ALONSO A.: 'Integration of safety analysis and software development methods'. Proc. 1st Int. Conf. System Safety, IEE, June 2006

BRIONES J.F., DE MIGUEL M., SILVA J.P., ALONSO A.: 'Application of safety analyses in model driven development'. Proc. 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems, LNCS, May 2007

REIBMAN A.L., VEERARAGHAVAN M.: 'Reliability modeling: an overview for system designers', *IEEE Comput.*, 1991, **24**, (4), pp. 49–57

LEVENSON N.: 'Safeware: system safety and computers' (Addison Wesley, 1995)

NUREG-0492 : 'Fault tree handbook' (Nuclear Regulatory Commision, US, 1981)

DUNN W.: 'Designing safety-critical computer systems', *IEEE Comput.*, 2003, **36**, (11), pp. 40–46

FROLA F., MILLER C.: 'System safety in aircraft acquisitions'. Technical report, Logistics, Management Institute, Washington, DC, 1984

Object Management Group: UML profile for modeling quality of service and fault tolerance characteristics and mechanisms final task force, OMG document number ptc/2005-05-02, available at: http://www.omg.org/cgi-bin/;doc?ptc/2005-05-02

Military standard: 'Procedures for performing a failure mode, effects and criticality analysis', 1980, MIL-STD-1629A

BUDINSKY F., STEINBERG D., MERKS E., ELLERSICK R., GROSE T.J.: 'Eclipse modeling framework' (Addison-Wesley Professional, 2003), ISBN 0131425420

Objecteering web page, available at: http://www.objecteering.com/

ITEM software web page, available at: http://www.itemuk.com/

Lockheed Martin 'MDA success story', available at: http://www.omg.org/mda/mda_files/LockheedMartin.pdf

LEVESON N., HARVEY P.: 'Analyzing software safety', *IEEE Trans. Softw. Eng.*, 1983, **SE-9**, (5), pp. 497–507

TOWHIDNEJAD M., WALLACE D., GALLO A.: 'Fault tree analysis for software design'. Proc. Software Engineering Workshop, 27th Annual NASA Goddard, IEEE Computer Society, December 2002

PAI G., DUGAN J.: 'Automatic synthesis of dynamic fault trees from UML system models'. Proc. Int. Symp. Software Reliable Engineering, IEEE Computer Society, November 2002

KHAN K., HAN J.: 'Composing security-aware software', *IEEE Software*, 2002, **19**, (1), pp. 34–41

AADL: 'SAE architecture analysis & design language', available at: http://www.aadl.info/

DE MIGUEL M., PAULY B., PERSON T., BRIONES J.F.: 'Model-based integration of safety analysis and reliable software development'. Proc. Workshop in Objectect Oriented Real-Time Dependable Systems, Words, January 2005

ZOUGHBI G., BRIAND L., LABICHE Y.: 'A UML profile for developing airworthiness-compliant (RTCA DO-178B), safety-critical software'. Carleton University Report TR SCE-06-19, December, 2006

GOSEVA-POPSTOJANOVA K., HASSAN A., GUEDEM A., ABDELMOEZ W., NASSAR D.E.M., AMMAR H., *ET AL.*: 'Architectural-level risk analysis using UML', *IEEE Trans. Softw. Eng.*, 2003, **29**, (6), pp. 946–960

DAWKINS S.K., KELLY T.P., MCDERMID J.A., MURDOCH J., PUMFREY D.J.: 'Issues in the conduct of PSSA'. Proc. Int. System Safety Conf., ISSC, August 1999

LISAGOR O., MCDERMID J.A., PUMFREY D.J.: 'Safety analysis of software architectures – lightweight PSSA'. Proc. Int. System Safety Conf., ISSC, August 2004