FROM SAFETY REQUIREMENTS TO SIMULATION-DRIVEN DESIGN OF SAFE SYSTEMS

Alfredo Garro[(a)], Andrea Tundis[(a)], Lena Buffoni-Rogovchenko[(b)], Peter Fritzson[(b)]

[(a)]Department of Computer Engineering, Modeling, Electronics, and Systems Science (DIMES), University of Calabria, via P. Bucci 41C, 87036, Rende (CS), Italy
[(b)]Department of Computer and Information Science (IDA), Linköping University, SE-581 83 Linköping, Sweden

[(a)]{garro, atundis}@dimes.unical.it,[(b)]{olena.rogovchenko, peter.fritzson}@liu.se

ABSTRACT
System safety is an important aspect of System Dependability which should be taken in consideration during the whole system lifecycle. However, often systems are built by considering mainly their functional aspects and safety requirements are verified and validated in the latest stages of the development process. For this reason and due to the deep integration of modern systems in the daily life of people, regulatory standards have been defined and have to be applied during the development of critical systems to guarantee a minimum and acceptable level of safety. In this context, the paper proposes a model-driven process, inspired by ISO-26262, which provides a methodological support for the verification and validation of safety requirements. In particular, the proposed framework combines model-driven engineering tools and techniques with OpenModelica, an equation based simulation environment based on the Modelica language. The proposal is experimented through a case study concerning the safety analysis of an Airbag System.

Keywords: Model-Based Systems Engineering, Safety Analysis, Requirements Engineering, Verification and Validation, Modelica, Automotive.

1. INTRODUCTION
The modeling of system requirements deals with formally expressing constraints and requirements that have an impact on the behavior of the system so as to enable their verification through real or simulated experiments and/or analytical techniques. The need of models for representing system requirements as well as for methods and techniques, especially centered on model-based approaches, able to support the modeling, evaluation, and validation of requirements and constraints along with their traceability is today even more prominent (Krause, Hintze, Magnus, and Diedrich 2012; Peraldi-Frati and Albinet 2010; Tundis, Rogovchenko-Buffoni, Fritzson, and Garro 2013; Yu, Xu, and Du 2009). In particular, while the modeling and verification of functional requirements are well supported by several tools and techniques, there is still a lack of models and methods specifically conceived to deal with non-functional requirements (such as reliability, availability, maintainability, safety, security); as a consequence, their verification is often postponed to the late stages of the development process with the risk of having to revise already implemented design choices, and, consequently, to miss project deadlines and exceed the budget (Garro, Tundis, and Chirillo 2011; Garro and Tundis 2012c).

Among non-functional requirements, Safety, which represents an important requirement to be satisfied for a wide range of systems (Laprie 1992), becomes even more crucial in several industrial domains such as nuclear plants, medical appliances, avionics, automotive and satellite (Guillerm, Demmou, and Sadou 2010; Garro, Tundis, Groß, and Riestenpatt Gen. Richter 2013; Lahtinen, Johansson, Ranta, Harju, and Nevalainen 2010; Rierson 2013). In particular, in the automotive domain, although Safety has always played a key role, the importance that is attributed to it has become far greater in recent times (Herpel and German 2009; Garro and Tundis 2012a; Navinkumar and Archana 2011). In modern automobile design, Safety Requirements can be generally categorized in three main classes: (i) *Passive safety*, which aims to minimize the severity of an accident; examples of passive safety elements are seatbelts, crumple zones, airbags; (ii) *Active safety*, which aims to avoid accidents and to minimize their effects if they occur; examples of active safety elements are: predictive emergency braking, seatbelt pre-tensioning, anti-lock braking systems and traction control; (iii) *Functional safety*, which aims to ensure that both the electrical and electronic systems (such as power supplies, sensors, communication networks, actuators, etc.), also including all active safety related systems, function correctly. In other words, Functional safety aims to guarantee the absence of unacceptable risk due to hazards caused by malfunctioning behavior of electrical and electronic systems.

The increasing importance that Safety is gaining as one of the main selling points with which to differentiate between car manufactures has led these competitors to join together to foster the definition of safety standards for automotive such as ISO-26262 (ISO-26262 2011). Its basis resides in the more generic IEC-61508 (IEC-61508 2010) which has a broad field of application (industrial process, control and automation, oil/gas, nuclear, etc.). However, ISO-26262 is totally dedicated to the automotive sector and allows car manufacturers to indemnify themselves from liability in case a malfunction remains undetected when following the standard (Lahtinen, Johansson, Ranta, Harju, and Nevalainen 2010). At the process level, this standard allows to follow a clear guidance on the development and validation of electrical and electronic systems, avoiding errors in the design and implementation, which could otherwise induce more expensive production activities and delay during the development (Täubig, Frese, Hertzberg, Lüth, Mohr, Vorobev, and Walter 2012). Moreover, a well-defined and standardized development process, which goes from the Requirements Analysis phases up to the System Testing phases, allows supporting the traceability of Safety Requirements during all the intermediate development stages.

In this paper a comprehensive approach, inspired by the ISO-26262 standard, for the definition of *Functional Safety Requirements* of systems is proposed along with a mechanism to enable their traceability and support their verification through simulation. The approach is based on an iterative process which is an extension for the Safety Analysis of Physical Systems of that proposed in (Garro and Tundis 2012b; Garro, Tundis, Groß, and Riestenpatt Gen. Richter 2013) and is constituted by the following main phases (see Figure 1): *Requirements Analysis*, *System Modeling* and *Virtual Testing*. Both the *Requirements Analysis* phase and the *System Modeling* phase are based on UML/SysML (System Modeling language) and supported by related modeling tools (IBM Rational Rhapsody); whereas, the *Virtual Testing* phase is enabled by the *OpenModelica* environment (OpenModelica), an Open Source simulation environment based on the Modelica language which is an equation-based object-oriented language for representing physical systems with *acausal* features (Fritzson 2004).

The rest of the paper is structured as follow: Section 2 introduces the safety analysis discipline along with a brief survey on the most common related techniques; then, in Section 3, the proposed simulation-driven design process for the safety analysis of systems is presented; in Section 4, this process is exemplified through a case study in the automotive; finally, conclusions are drawn and future work delineated.

2. SYSTEM SAFETY ANALYSIS AND RELATED TECHNIQUES

Safety Analysis is a discipline of Safety Engineering whose aim is to ensure that engineered systems provide acceptable levels of safety through the identification of safety related risks, eliminating or controlling them by design and/or procedures, based on acceptable system safety precedencies (FAA 2000; NASA).

System safety uses systems theory and systems engineering approaches to prevent foreseeable accidents and minimize the effects of unforeseen ones. It considers losses in general, not just human death or injury. Such losses may include destruction of property, loss of mission and environmental harm. Safety of systems needs to be planned in an integrated and comprehensive engineering framework that requires experience in the application of safety engineering principles by exploiting well-known analysis techniques to perform safety analysis for the identification and the management of *hazards*. The general definition of Safety is based on the main concept of *risk* which is the combination of the probability of a failure event and the severity resulting from the failure.

Several techniques for performing quantitative and qualitative safety analyses are currently available. *Quantitative analysis* techniques are based on the identification and modeling of physical and logical connections among system components and on their analysis through statistical methods and techniques, but very often probabilistic information is not so relevant or desired, for example, when one wants to study the reachability of a state of the system, as a consequence *Qualitative analysis* techniques are often preferred (Rouvroye and Van den Bliek 2002).

The *Fault Hazard Analysis* (FHA) is a deductive method of analysis that can be used exclusively as a qualitative analysis or, if desired, expanded to a quantitative one (Pomeranz and Reddy 2009). The Fault Hazard Analysis requires a detailed investigation of the subsystems to determine component hazard modes, causes of these hazards, and resultant effects to the subsystem and its operation. This type of analysis belongs to a family of reliability analysis techniques which comprehends FMEA/FMECA (Failure Mode and Effects Analysis/Failure mode effects and criticality analysis). The main difference between the FMEA/FMECA and the Fault Hazard Analysis is a matter of depth. Wherein the FMEA or FMECA looks at all failures and their effects, the Fault Hazard Analysis deals only with those effects that are safety related.

Fault Tree Analysis (FTA) is a popular and productive hazard identification tool (Clifton 1999). A FTA is a deductive or backward logic representation which involves specifying a top event to analyze (a system failure), followed by identifying all of the associated elements in the system that could cause that top event to occur. It provides a standardized discipline to evaluate and control hazards. The FTA process is used to solve a wide variety of problems ranging from safety to management issues. This tool is used by the professional safety and reliability community to both prevent and resolve hazards and failures. Both qualitative and quantitative methods are used to identify areas in a system that are most critical to safe operation. The output is a graphical presentation providing a map of “failure or hazard” paths.

Event Tree Analysis (ETA) is an analysis technique for identifying and evaluating the sequence of events in a potential accident scenario following the occurrence of an initiating event (Kenarangui 1991). ETA is an inductive or forward logic representation, which starts from an initiating event and includes all possible paths, whose branch points represent successes and failures. The objective of ETA is to determine whether the initiating event will develop into a serious mishap or if the event is sufficiently controlled by the safety systems and procedures implemented in the system design. An ETA can result in many different possible outcomes from a single initiating event and it provides the capability to obtain a probability for each outcome.

Common Cause Failure Analysis (CCFA) is an extension of FTA to identify "coupling factors" that can cause component failures to be potentially interdependent (Liudong and Wendai 2008). Primary events of minimal cut sets from the FTA are examined through the development of matrices to determine if failures are linked to some common cause relating to the environment, location, secondary causes, human error, or quality control. A cut set is a set of basic events (e.g. a set of component failures) whose occurrence causes the system to fail. A minimum cut set is one that has been reduced to eliminate all redundant "fault paths". CCFA provides a better understanding of the interdependent relationship between FTA events and their causes. It analyzes safety systems for "real" redundancy.

Sneak Circuit Analysis (SCA) is a method for the evaluation of electrical circuits (Price and Hughes 2002). SCA employs recognition of topological patterns that are characteristic of all circuits and systems. The purpose of this analysis technique is to uncover latent (sneak) circuits and conditions that inhibit desired functions or cause undesired functions to occur, without a component having failed. The process converts schematic diagrams to topographical drawings and searches for sneak circuits.

The *Energy Trace* is a hazard analysis approach addresses all sources of uncontrolled and controlled energy that have the potential to cause an accident (Booya, Arghami, Asilian, and Mortazavi 2007). Examples include utility electrical power and aircraft fuel. Sources of energy causing accidents can be associated with the product or process. The purpose of energy trace analysis is to ensure that all hazards and their immediate causes are identified. Once the hazards and their causes are identified, they can be used as top events in a fault tree or used to verify the completeness of a fault hazard analysis. Consequently, the energy trace analysis method complements but does not replace other analyses, such as fault trees, sneak circuit analyses, event trees, and FMEAs.

Even though the above mentioned techniques are fairly popular for the safety static analysis of systems, nowadays, with the increase of complexity and heterogeneity of modern systems, more dynamic and flexible analysis techniques, based on simulation methods as well as compliant with international safety standards for specific domains, such as ISO-26262 in the automotive one (Aljazzar, Fischer, Grunske, Kuntz, Leitner-fischer, and Leue 2009; SAE 2003; Stapelberg 2008; Struble 2005), are even more required. As an example, the Process Deployment Advisory Service defined on ISO-26262 in order to help identifying gaps in the development processes, including requirements traceability and requirements based-testing, is fully supported by popular tools such as MatLab/Simulink (Mathworks).

3. A SIMULATION-DRIVEN PROCESS FOR THE DESIGN OF SAFE SYSTEMS

In this section a methodological process for the development of safe systems, based on the validation of the design through simulation, is presented.

As shown in Figure 1, such process which is inspired by the ISO-26262 standard, is defined in terms of three main iterative phases: *Requirements Analysis*, *System Modeling*, and *Virtual Testing*, which aim to provide a methodological support according to the ISO-26262 standard.

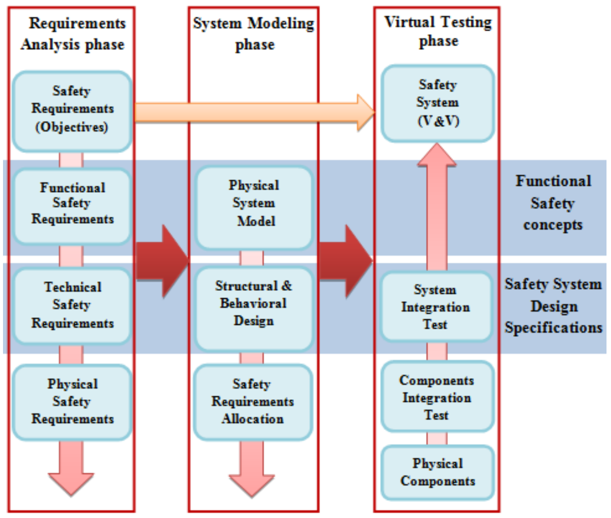
Requirements Analysis phase
System Modeling phase
Virtual Testing phase
Safety Requirements (Objectives)
Safety System (V&V)
Functional Safety Requirements
Physical System Model
Functional Safety concepts
Technical Safety Requirements
Structural & Behavioral Design
System Integration Test
Safety System Design Specifications
Physical Safety Requirements
Safety Requirements Allocation
Components Integration Test
Physical Components

Figure 1: Main phases of the proposed simulation-driven process for the design of safe systems

In the *Requirements Analysis* phase the system safety objectives are analyzed and Safety requirements, in terms of Functional, Technical and Physical requirements, are identified (Rubio, Ponce, and Madrid 2011; Sommerville and Sawyer 2003). They may consist of properties and safety performances to be considered in order to eliminate the risk or to reduce it to an acceptable level. Specifically, a process for their elicitation, definition, formalization and validation is defined according to a meta-model proposed in (Tundis, Rogovchenko-Buffoni, Fritzson, and Garro 2013).

In particular, the first step consists in the requirements elicitation that, according to the proposed meta-model, is obtained through *RequirementAssertions*. An iterative process between the user and the analyst is typically executed in order to state all the requirements, as much as possible, by associating to each of them a *Name* for their identification along with a possible *Description* in a text format by using the natural language in order to provide an explanation of specific or salient aspects, characteristics, or features (e.g. functional, technical or physical) of the system in a detailed way. At the end of this step the so called *User Requirements (URs)* are generated according to the meta-model.

The second step consists in the refinement of the *URs* in order to generate *System Requirements (SRs)*. This step is very crucial to make URs machine readable and executable in order to enable their verifiability during the simulation, as a consequence, it is really important what to represent and how to do it as well as when to use such requirements. First of all a *RequirementAssertion* could be involved in several verification tasks grouped in different *RequirementModel*, so the membership of each requirement to at least one of those *RequirementModels* must be identified. Then, the output values, associate to the evaluation of requirements, for describing if a requirement has been not violated, violated, and so on, have to be fixed. At the end, a *Metric* needs to be specified for each *RequirementAssertion*. In particular, it specifies the purpose of a *RequirementAssertion* in terms of verification mechanism. In Figure 2 the relationships among the User Requirements, System Requirements and Safety Requirements are represented.

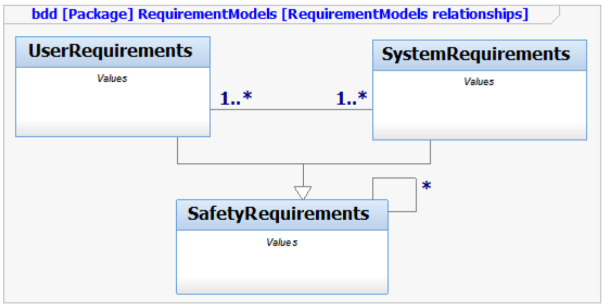

Figure 2: Relationships among User Requirements System Requirements and Safety Requirements

The representation of requirements is carried out by using Requirement diagrams available in SysML, a UML profile for modeling system, and exploiting tools such as IBM Rational Rhapsody (IBM) or Papyrus (Papyrus), in order to enable model-based system engineering.

It is worth to notice that not all the requirements can be formalized into something computable such as "a cable must be well connected", if the term "well connected" is not represented in a machine readable formalism.

In the *System Modeling* phase, a possible physical model of the real system in terms of its components is defined; in particular, the Structural and the Behavioral views are generated by breaking down the system in (sub)components.

Specifically the first step, according to the *Physical* side of the meta-model proposed in (Tundis, Rogovchenko-Buffoni, Fritzson, and Garro 2013), consists in building a possible *PhysicalSystemModel*, of the actual *PhysicalSystem* by specifying the *models* of its physical components (*PhysicalComponentModels*) and the related *Attributes* and, then, defining the *relationships* among them as well as their *behaviors*. In particular, the structural part of the system is described by using Block Definition Diagrams and Internal Block Diagrams in a top-down fashion. The behavior of the system, which is modeled by following a bottom-up approach, can be defined in terms of Activity, Sequence or Parametric diagrams in order to model the internal behavior of each system components as well as the flows of actions and interactions between components.

Then *SRs* belonging to the *RequirementModel* concerning Safety Requirements, can be further formalized in order to make them machine executable. In particular, a formal *Measure*, and its expected input and output values, can be associated to the defined *Metric*. Specifically, a *Measure* can be expressed by adopting an appropriate *ComputationalModel* which in turn could be represented through an *Algorithm*, a *Finite Automata*, a *Function*, a set of *Equation* or by their combination to enable the computational process.

Finally, the allocation between the *SafetyRequirements* and the *PhysicalSystemModel* is performed. Furthermore, inputs, required from the *Measure* of a *RequirementAssertion* for its evaluation, are explicitly included in the *PhysicalComponentModels*.

In the *Virtual Testing* phase, the Models of the system under consideration are transformed into executable models and represented in terms of the constructs offered by the OpenModelica platform (Open Source Modelica Consortium), an Open Source simulation environment based on the Modelica language, an equation-based object-oriented language for representing physical systems with *acausal* features, (Modelica and the Modelica Association). In particular, physical components are defined and integrated in order to build the physical system model and then the safety requirements to be verified are introduced into the overall model. Then, different simulation scenarios are set and simulations are executed; finally, simulation results can be analyzed on the basis of the system safety requirements identified in the first process phase. This analysis allows to evaluate the safety properties of the system, to compare different design choices for improving, possibly, the safety of the system under consideration.

As the process is iterative, if necessary, new partial or complete process iterations can be executed.

3.1. Relationships between the ISO-26262 standard and the proposed process

The above described process is inspired by the IEC-61508 standard and, in particular, by the ISO-26262 whose goal is to demonstrate the capability to develop certain products with acceptable risks.

ISO-26262 is organized in 10 parts as following:

- *Part 1 - Vocabulary*: which specifies the terms, definitions and abbreviated terms for application in all parts of ISO 26262;
- *Part 2 - Management of Functional Safety*: which specifies the requirements for functional safety management for automotive applications, including (i) project-independent requirements with regard to the organizations involved (overall safety management), and (ii) project-specific requirements with regard to the management activities in the safety lifecycle (i.e. management during the concept phase and product development, and after the release for production);
- *Part 3 - Concept phase*: which specifies the requirements for the concept phase for automotive applications (e.g. item definition, functional safety concept, etc.);
- *Part 4 - Product Development at system level*: which specifies the requirements for product development at the system level for automotive applications, such as the system design and system integration and testing;
- *Part 5 - Product Development at hardware level*: which specifies the requirements for product development at the hardware level for automotive applications (e.g. hardware design and hardware architectural metrics, hardware integration and validation);
- *Part 6 - Product Development at software level*: which specifies the requirements for product development at the software level for automotive applications such as software architectural design, software unit design and implementation, software integration and testing;
- *Part 7: Production and Operation*: which specifies the requirements for production, operation, service and decommissioning.
- *Part 8: Supporting Processes*: which specifies the requirements for supporting processes through qualified tools, system engineering approaches and best practices;
- *Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses*: concerning the measures required to avoid unreasonable risks.
- Part 10: *Guidelines on ISO-26262*.

In the Table 1 the matching between ISO-26262 parts and the phases of the proposed process are shown, by indicating in which phase of the process a specific part of such standard should be considered.

In particular *Vocabulary* and *Management of Functional Safety Concept phase* can be considered in the *Requirements Analysis phase* for the definition, the organization and categorization of requirements; then *Product Development at system level*, *Product development at the hardware level* and *Product development at the software level* can be taken into account in the *System Modeling phase,* when the design of the system is under definition, whereas the *Supporting Process* part can be considered during the *Virtual Testing phase* of the proposed process.

Table 1: Matching between ISO-26262 and the proposed process.

Parts of the Standard ISO-26262	Simulation-Driven Process for the Design of Safe Systems
Vocabulary Management of Functional Safety Concept phase	*Requirements Analysis phase*
Product Development at system level Product development at the hardware level Product development at the software level	*System Modeling phase*
Supporting Process	*Virtual Testing phase*

4. FROM SAFETY REQUIREMENTS TO A SAFE DESIGN IN THE AUTOMOTIVE DOMAIN: A CASE STUDY

In this Section, a case study in the automotive domain concerning the modeling of an airbag system, and the validation and evaluation of its design according to the safety requirements through simulation, is analyzed following the proposed process. In particular, after a brief introductive description of the system under consideration, its safety analysis is performed.

4.1. Airbag description

Airbags are one of the most important components of a motor vehicle system for the occupant protection. It is used along with and as a supplement to the seatbelt restraint system to provide passenger protection in case of collision In addition to the standard airbags for the driver and front passenger, an increasing number of specialized airbag variants (such as curtain airbags, kneebags, etc.) are used.

Each airbag should be specifically designed and optimized for its intended purpose. In addition to the deployment technology, which can in principle be based on the uniform pressure approach or the more recent corpuscular method, this includes the selection of the inflow method (Wang-Nefske or hybrid approach, etc.) as well as the verification and validation of the associated inflow data. Moreover, the deployment behavior is also determined by the correct adjustment of contact, discharge opening and porosity parameters. As a consequence a sensible and comprehensive simulation of airbag behavior as part of a simulation of the entire restraint system is indispensable.

An airbag is typically made of synthetic material and equipped with holes in the rear; it is usually composed by different subsystems such as:

- a *sensor* that detects the abrupt deceleration of the vehicle caused by an impact and the pressure;
- an *Airbag Control Unit* (ACU) that monitors the readiness of the entire airbag system.
- a *detonator* that triggers the substance contained in the explosive capsule through an electric current or a bump of a ferrule;
- a possible second capsule (*GasSoure*) that contains pre-compressed inert gas which inflates the airbag;
- a *warning light* which is illuminated if a fault is detected.

Specifically the ACU receives the signal of the sensor, processes it and sends the command to switch on a detonator; which in turn blows up the capsule of the detonator by developing a large amount of gas, to inflate the container.

4.2. Requirements Analysis phase

In this phase of the proposed process all the possible user requirements need to be identified and elicited.

As an example, in the following some *URs* are reported: (*Req1)* when the car decelerates very quickly, as in a head-on crash, the electrical circuit has to be turned on for initiating the process of inflating the airbag; (*Req2*) the process, from the initial impact of the crash to full inflation of the airbags, takes less than 40 milliseconds; (*Req3*) when a sensor detects a collision an immediate trigger should be sent to enable the deployment of the airbag; (*Req4*) in order for the airbag to cushion the head and torso with air for maximum protection, the airbag must begin to deflate (i.e., decrease its internal pressure) by the time the body hits it, otherwise, the high internal pressure of the airbag would create a hard surface instead of a protective cushion; (*Req5*) the airbag is ignited within a well-define threshold.

Starting from the collected *URs* the next step consist into their rewriting in *SRs* for making them more formal and by identifying their belonging *RequirementModel.* For example:

- *AbruptDeceleration(Req1)*: when the deceleration *d* is greater than a *threshold*, a *signal* to switch on the electronic circuit has to be sent;
- *InflationTime(Req2)*: The time to inflate the airbag has to take less than *40ms*, *inflationTime<=40;*
- *CollitionDetection(Req3)*: when the collision is detected by the sensor, a *collitionSignal* has to be generated;
- *DeflationTime(Req4)*: the airbag has to be able to deflate in a *deflationTime* lesser than a deflation *threshold*.
- *Activation(Req5)*: after a crash the airbag is deployed in *delayTime*=45ms.

Specifically, the relationships among the above mentioned safety requirements are represented in Figure 3. In particular the status of the *DeflationTime* is not violated if at least the status of the requirement *InflationTime* is not violated. In turn the status of the *InflationTime* is not violated if at least the status of the *Activation* requirement is fulfilled at least by both the *AbruptDeceleration* requirement and the *CollitionDetection* requirement. That is to say, the status of both *AbruptDeceleration* and *CollitionDetection* must be not violated.

Moreover different scenarios can be analyzed, such as:

- the airbag is not ignited or is inflated too late even though a critical crash occurred;
- the airbag is deployed unintentionally, which means that it is ignited even though no crash at all or only a non-critical crash has occurred;

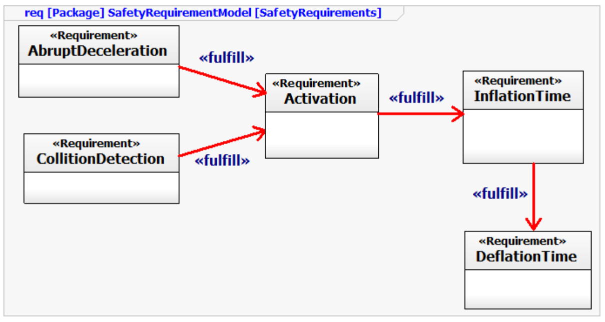
req [Package] SafetyRequirementModel [SafetyRequirements]
«Requirement» AbruptDeceleration
«Requirement» CollitionDetection
«Requirement» Activation
«Requirement» InflationTime
«Requirement» DeflationTime
«fulfill»
«fulfill»
«fulfill»
«fulfill»

Figure 3: Safety System Requirement relationships

4.3. System Modeling phase

In this phase both the physical structure of the system is built by composing components and then the behavior of each single component is specified.

As it is shown in Figure 4, a Block Definition Diagram (BDD) of an Airbag System is depicted, in terms of its subsystems and ports. Then, the interactions among these components are better specified by using the Internal Block Diagram (IBD), as it is shown in Figure 5.

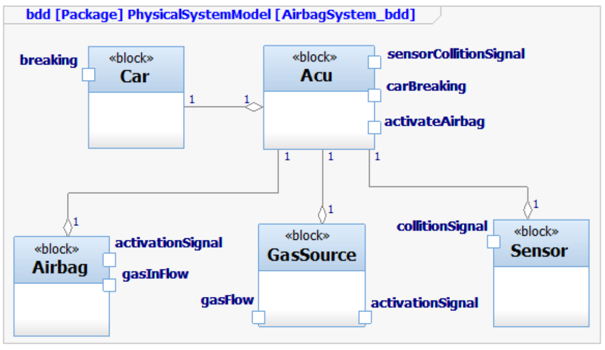

Figure 4: Physical System Model: Components of the Airbag System

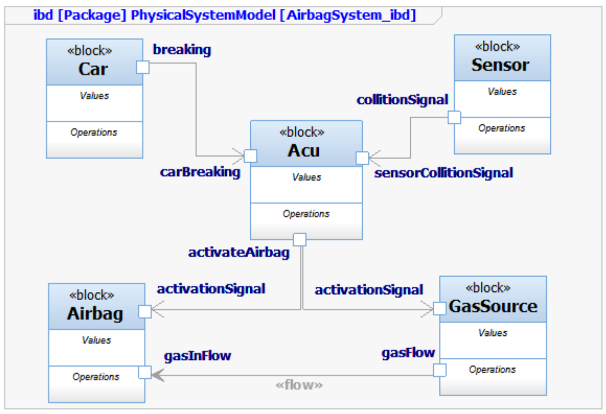

Figure 5: Physical System Model: Components interactions of the Airbag System

After the structure is built, Parametric diagrams are employed for representing the behavior of each subsystem as well as dynamic interactions among them, by exploiting a *Computational Model* based on *EquationsSet*. As an example, in Figure 6 the diagram concerning the behavior of the Airbag component is reported. In particular, in the first section of the diagram, the parameters taken in input from the model are defined, secondly a brief description about the use of such parameters is reported; then the behavior of the Airbag component, which exploits such input parameters, is represented in terms of equations.

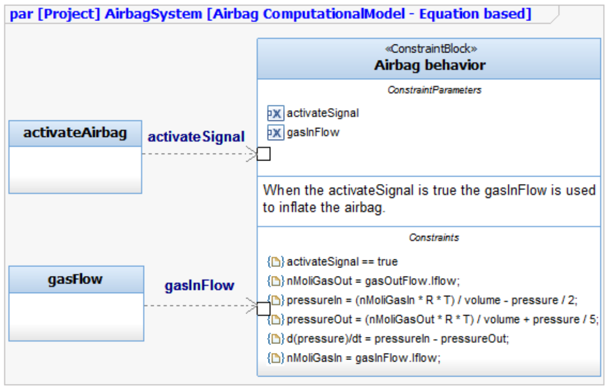
par [Project] AirbagSystem [Airbag ComputationalModel - Equation based]
activateAirbag
activateSignal
gasFlow
gasInFlow
«ConstraintBlock» Airbag behavior
ConstraintParameters
activateSignal
gasInFlow
When the activateSignal is true the gasInFlow is used to inflate the airbag.
Constraints
activateSignal == true
nMoliGasOut = gasOutFlow.lflow;
pressureIn = (nMoliGasIn * R * T) / volume - pressure / 2;
pressureOut = (nMoliGasOut * R * T) / volume + pressure / 5;
d(pressure)/dt = pressureIn - pressureOut;
nMoliGasIn = gasInFlow.lflow;

Figure 6: Computational Model of the Airbag component

Finally, requirements modeled in the previous phase, which need to be verified, are allocated to (i) a single physical component in order to check its behavior or (ii) a set of physical components in order to check if the interaction among them is or is not consistent as expected. In Figure 7 the allocation of some requirements to the airbag physical system model, is shown.

In particular, such a scenario wants to verify, the *InflationTime* of the airbag when a car-crash occurs. Specifically, the requirement is not violated when the status of the *Activation* requirement is not violated and both the *Acu* component and the *Airbag* component fulfill the internal rules specified by the *InflactionTime*.

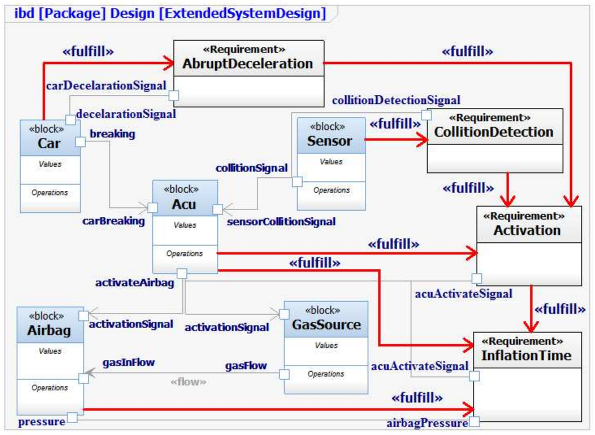

Figure 7: Allocation of Safety Requirements to the Airbag Physical System Model

4.4. Virtual Testing phase

In this phase the virtual testing is executed by exploiting the simulation in order to study the behavior of the system under consideration and analyze interesting aspects or, possibly, to discover some issues that are not immediately obvious when applying static analysis techniques. In order to enable the simulation, models generated in the previous phase need to be translated into the desired simulation platform in order to make them executable. In this case the OpenModelica environment has been chosen as simulation platform since: (i) it is equation based (by implementing the Modelica Language) and, as a consequence, compliant with the *Computational Model* which has been used to represent the behavior of the overall Airbag System; (ii) it is open source, thus allowing the possibility to extend both the language and the tool, to enable modeling of requirements and introduce allocation mechanisms.

In Figure 8, a fragment of source code in Modelica language, which represents the structure of the AirbagSystemDesign, is reported.

```
model AirbagSystemDesign
 import AirbagPhysicalComponentModel.*;
 Car car;
 Acu acu;
 Sensor sensor;
 Airbag airbag;
 GasSource gasSource(gasGain = 0.73);
 GasOut gasDestination(gasGain = 0.02);
equation
 connect(car.breaking,acu.carBreaking);
 connect(sensor.collitionSignal,acu.sensorCollitionSignal);
 connect(sensor.collitionSignal,acu.carBreaking);
 connect(acu.activateAirbag,airbag.activationSignal);
 connect(airbag.gasInFlow,gasSource.gasFlow);
 connect(airbag.gasOutFlow,gasDestination.gasFlow);
end AirbagSystemDesign;
```

Figure 8: Airbag System Design in Modelica

As we can see by looking at the source code, the transformation between SysML notation and Modelica constructs, is almost direct. In particular, each *SysML block* can be represented as a *Modelica Model*, whereas connections among *SysML blocks* can be enabled by the *connect* construct, which is already available in the Modelica language.

In Figure 9, a fragment of Modelica source code concerning the implementation of the behavior of the airbag component, is shown.

```
model Airbag
 Boolean activationSignal(start = false);
 GasFlow gasInFlow "Connector-flow";
 GasOut gasOutFlow "Connector-flow";
 Real nMoliGasIn(unit = "mol", start = 0);
 Real nMoliGasOut(unit = "mol", start = 0);
 constant Real volume(unit = "l") = 9.0;
 constant Real R(unit = "l*atm/mol*K") = 0.0821;
 constant Real T(unit = "K") = 295.0;
 Real pressureIn(start = 0.0, unit = "atm");
 Real pressureOut(start = 0.0, unit = "atm");
//Airbag Internal Pressure
 Real pressure(start = 0.0, unit = "atm") ;
equation
 if activationSignal == true then
   der(pressure) = pressureIn - pressureOut;
   nMoliGasIn = gasInFlow.lflow;
   nMoliGasOut = gasOutFlow.lflow;
   pressureIn = (nMoliGasIn * R * T) / volume - pressure / 2;
   pressureOut = (nMoliGasOut * R * T) / volume + pressure / 5;
 ...
  ...
 end if;
end Airbag;
```

Figure 9: Representation of the behavior of the Airbag component in Modelica

As we can see from the picture, the component behavior, which was described in the *System Modeling phase* through a SysML parametric diagram (see Figure 6), has been is translated in a set of equations by using the Modelica language.

Similarly, the requirements identified in the *Requirements Analysis phase* are formalized by exploiting some extensions of the Modelica language, proposed by the authors in (Rogovchenko-Buffoni, Fritzson, Garro, Tundis, and Nyberg 2013); specifically: (i) the *requirement* keyword is used for their representations, (ii) the *fulfill* relationship is used both for their allocation to the physical system and for their traceability, (iii) the *precondition equation* section is used to specify the conditions when the evaluation of the requirement has to be performed.

In particular, the source code of the formalized requirement *InflationTime* is reported in Figure 10, where the evaluation is based on the inflation time that the airbag takes to reach a specific safety level of pressure after the airbag is activated.

```
requirement InflationTime
  Real airbagPressure(unit="atm");
  Real safePressureLevel(unit="atm")=2.5;
  Boolean activateAirbag(start=false);
  constant Real maxInflationTime(unit="ms")=40;
  constant Real activationTime(unit="ms")=20;
precondition equation
 activateAirbag=true;
 (time < activationTime + maxInflationTime)=true;
equation
 (airbagPressure >= safePressureLevel)=true;
end InflationTime;
```

Figure 10: Formalization of the *InflationTime* requirement by using Modelica language extensions

The source code of the extend system design is reported in Figure 11, where the *fulfill* keyword is employed for creating the matching among the requirements as well as between requirements and physical components of the airbag system.

```
model ExtendedSystemDesign
      import SafetyRequirementModel.*;
 // System Design
 AirbagSystemDesign asd;
 // Safety Requirements
 AbruptDeceleration ad;
 CollitionDetection cd;
 Activation ac;
 InflationTime it;
equation
  // Fulfill equations
 {asd.car} fulfill ad;
 {asd.sensor} fulfill cd;
 {asd.acu, ad, cd} fulfill ac;
 {asd.airbag, asd.acu, ac} fulfill it;
  // Connect equations between Requirements
  // and PhysicalComponents
 connect(asd.airbag.pressure,it.airbagPressure);
 connect(asd.acu.activateAirbag,it.acuActivateSignal);
 connect(asd.car.decelarationSignal,ad.carDecelarationSignal);
 connect(asd.sensor.collitionSignal,cd.collitionDetectionSignal);
 connect(asd.acu.activateAirbag,ac.acuActivateSignal);
end ExtendedSystemDesign;
```

Figure 11: Formalization of the *ExtendedSystemDesign* by using Modelica language extensions

After obtaining the executable models, the tuning of the simulation parameters is performed in order to reach a safe working state of the system according to the specified requirements. Several simulations have been executed for testing virtually the System in different scenarios and evaluating its behavior. Moreover three possible values can be reached by a requirement.

In the considered experimentations (see Figure 12 and Figure 13) a pressure level (*safePressureLevel* in yellow color) of 2.5 atmospheres (atm) has been considered as minimum safe threshold, coupled with a maximum inflation time (*maxInflationTime*) of 40 milliseconds (ms) in order to reach such safe pressure level of the airbag, after that the activation airbag signal (*activateAirbag*) is arrived.

As it is shown in Figure 12, even though the *AbruptDeceleration* (in dark blue color) and *CollitionDetection* (in dark green color) requirements are satisfied as well as the *Activation* (in brown color) requirement, the *requirementStatus* of the *InflactionTime* (in red color) is negative, that is to say it is violated. Indeed, as we can see, the *airbagPressure* (in light blue color) is not able to reach the *safePressureLevel* (in yellow color) within the *maxInflationTime* (in 40 milliseconds) as expected.

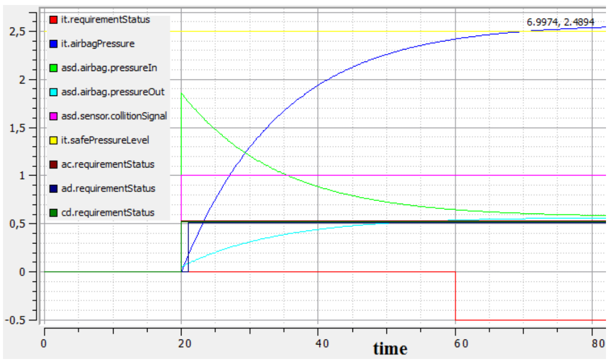

Figure 12: Violation of the *InflationTime* requirement

As it is shown in Figure 13, by setting opportunely some parameters of the airbag system, for example, by increasing the pressure to be provided in input to the airbag (*pressureIn* in light green color), it is possible to reach the necessary parameters tuning which fulfills all the requirements in the considered scenario.

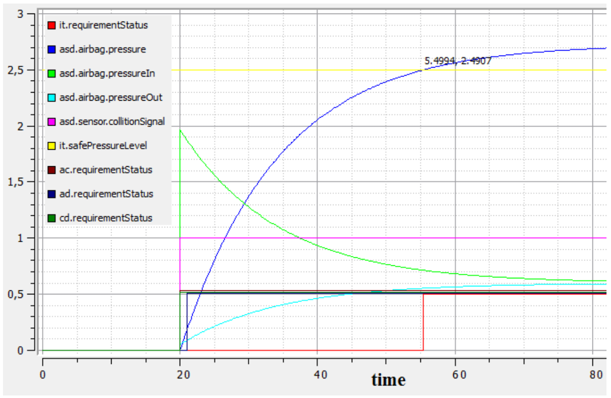

Figure 13: Fulfillment of the *InflationTime* requirement

5. CONCLUSIONS AND FUTURE WORKS

The paper has presented a model-driven process for supporting the safety analysis of systems which is inspired by ISO-26262 standard and exploits simulation techniques.

Two powerful languages for modeling systems have been combined in a comprehensive system engineering framework; specifically, SysML has been exploited for platform independent representation of the system; whereas, the Modelica language has been exploited for the executable representation of the systems according to an equation-based paradigm.

A prototype of the OpenModelica simulation platform, able to support both the modeling of requirements and their allocation, according to a well-defined reference meta-model, has been used for the simulation. Finally, a concrete experimentation has been conducted in the automotive domain which has allowed to point out both the flexibility and the effectiveness of the overall proposed process for safety analysis.

Future work includes both the improvements of the proposed model-driven process and its extension by introducing (i) some approaches and possible patterns for representing dysfunctional behavior and fault injection and, (ii) a probability model for enabling the representation of uncertainties in order to perform Fault Tree Analysis of a Modelica-based system model.

ACKNOWLEDGMENTS

Andrea Tundis has been supported by a grant funded in the framework of the “POR Calabria FSE 2007/2013”. This work is part of an ongoing research project, the MODRIO Project (ITEA 2), aiming at developing a model-based approach for system requirements verification and safety analysis through simulation.

REFERENCES

Aljazzar H., Fischer M., Grunske L., Kuntz M., Leitner-fischer F., Leue S., 2009. Safety Analysis of an Airbag System using Probabilistic FMEA and Probabilistic Counterexamples. *Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems (QEST).* September 17-18, Eger (Hungary).

Booya M., Arghami S., Asilian H., Mortazavi S., 2007. Safety analysis of a corn processing industry by energy trace and barrier analysis method: a case study. *Iran Occupational Health Journal*, 4 (3), 27-34.

FAA 2000. *System Safety Handbook*.

Clifton E., 1999. Fault tree analysis - a history. *Proceedings of the 17th Inernational Systems Safety Conference*, pp. 1-9. August 16-21, Orlando (Florida, USA).

Fritzson P., 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley IEEE Press.

Garro A., Groß J., Riestenpatt Gen. Richter M., Tundis A., 2013. Reliability Analysis of an Attitude Determination and Control System (ADCS) through the RAMSAS method. *To appear in Journal of Computational Science*, Elsevier.

Garro A., Tundis A., 2012a. Enhancing the RAMSAS method for System Reliability Analysis: an exploitation in the automotive domain. *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH).* July 28-31, Rome (Italy).

Garro A., Tundis A., 2012b. Modeling and Simulation for System Reliability Analysis: The RAMSAS Method. *Proceedings of the 7th IEEE International Conference on System of Systems Engineering (IEEE SoSE).* July 16-19, Genova (Italy).

Garro A., Tundis A., 2012c. A model-based method for system reliability analysis. *Proceedings of the Symposium on Theory of Modeling and Simulation (TMS).* March 26-29, Orlando (FL, USA).

Garro A., Tundis A., Chirillo N., 2011. System reliability analysis: a model-based approach and a case study in the avionics industry. *Proceedings of the 3rd Air and Space International Conference (CEAS)*. October 24-28, Venice (Italy).

Guillerm R., Demmou H., Sadou N., 2010. Engineering dependability requirements for complex systems - A new information model definition. *Proceedings of the 4th Annual IEEE Systems Conference*. April 5-8, San Diego (CA, USA).

Herpel T., German R., 2009. A simulation approach for the design of safety-relevant automotive multi-ECU systems. *Proceedings of the 4th IEEE International Conference on System of Systems Engineering (IEEE SoSE).* May 30 - June 03, Albuquerque (New Mexico, USA).

IBM Rational Rhapsody web site - http://www.ibm.com/.

IEC-61508, 2010. Functional safety of electrical/electronic/programmable electronic safety-related systems, Parts 1-7.

ISO-26262, 2011. Software Compliance: Achieving Functional Safety in the Automotive Industry.

ITEA 2 Projects: MODRIO web site - http://www.itea2.org/.

Kenarangui R., 1991. Event-tree analysis by fuzzy probability. *IEEE Transaction on Reliability*, 40 (1), 120-124.

Krause J., Hintze E., Magnus S., Diedrich C., 2012. Model based specification, verification and test generation for a safety fieldbus profile. *Proceedings of the 31st International Conference on Computer Safety, Reliability and Security (SafeComp).* September 25, Magdeburg (Germany).

Lahtinen J., Johansson M., Ranta J., Harju H., Nevalainen R., 2010. Comparison between IEC 60880 and IEC 61508 for certification purposes in the nuclear domain. *Proceedings of the 29th International Conference on Computer Safety, Reliability and Security (SafeComp).* September 14-17, Vienna (Austria).

Laprie J.C., 1992. *Dependability: Basic Concepts and Terminology*, Springer-Verlag.

Liudong X., Wendai W., 2008. Probabilistic common-cause failures analysis. *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 354-358. January 28-31, Las Vegas (NV, USA).

Mathworks MatLab/Simulink web site - http://www.mathworks.com/.

Modelica and the Modelica Association web site - https://www.modelica.org/.

NASA - http://askmagazine.nasa.gov/pdf/pdf31/NASA_APPEL_ASK_31i_introduction_to_system_safety.pdf

Navinkumar V.K., Archana R.K., 2011. Functional safety management aspects in testing of automotive safety concern systems (electronic braking system). *Proceedings of the 3rd International Conference on Electronics Computer Technology (ICECT).* April 8-10, India.

OpenModelica - Open Source Modelica Consortium (OSMC) web site - https://www.openmodelica.org/.

Papyrus - Eclipse project web site - http://www.eclipse.org/papyrus/.

Peraldi-Frati M., Albinet A., 2010. Requirement traceability in safety critical systems. *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness & Safety (CARS).* April 27, Valencia (Spain).

Pomeranz I., Reddy S.M., 2009. Hazard-Based Detection Conditions for Improved Transition Fault Coverage of Functional Test Sequences. *Proceedings of the 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 358 - 366. October 7-9, Chicago (IL, USA).

Price C.J., Hughes N., 2002. Effective automated sneak circuit analysis. *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 356 – 360. January 28-31, Seattle (WA, USA).

Rierson L., 2013. *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*, CRC Press.

Rogovchenko-Buffoni L., Fritzson P., Garro A., Tundis A., Nyberg M., 2013. Requirement Verification and Dependency Tracing During Simulation in Modelica. *To appear in proceedings of the 8th EUROSIM Congress on Modelling and Simulation*. September 10-13, Cardiff (Wales, UK).

Rouvroye J.L., Van den Bliek E.G., 2002. Comparing safety analysis techniques. *Journal of Reliability Engineering & System Safety*, Elseiver, 75 (3), 289–294.

Rubio D., Ponce S., Madrid F., 2011. ISO/IEC 17025 technical requirements in electrical safety laboratory for electromedical devices. *Proceedings of Pan American Health Care Exchanges (PAHCE 2011).* March 28 – April 1, Rio de Janeiro (Brazil).

SAE International, 2003. *Airbags and Safety Test Methodology, Society of Automotive Engineers*.

Stapelberg R.F., 2008. *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design*, 1st ed. Spinger-Verlag.

Sommerville I., Sawyer P., 2003. *Requirements Engineering: A good practice guide*. Wiley.

Struble D.E., 2005. *Advances in Side Airbag Systems, Society of Automotive Engineers Inc*.

Systems Modeling Language (SysML) web site - http://www.omgsysml.org/.

Tundis A., Rogovchenko-Buffoni L., Fritzson P., Garro A., 2013. Modeling System Requirements in Modelica: Definition and Comparison of Candidate Approaches. *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT).* April 19, Nottingham (UK).

Täubig H., Frese U., Hertzberg C., Lüth C., Mohr S., Vorobev E., Walter D., 2012. Guaranteeing functional safety: design for provability and computer-aided verification. *Journal Autonomous Robots*, 32, Springer, 303-331.

Yu G., Xu Z., Du J., 2009. An Approach for Automated Safety Testing of Safety-Critical Software System Based on Safety Requirements. *Proceedings in the International Forum on Information Technology and Applications (IFITA).* May 15-17, Chengdu (China).