Design-Time Safety Assessment of Robotic Systems Using Fault Injection Simulation in a Model-Driven Approach

Garazi Juez Uriagereka, Estibaliz Amparan, Cristina Martinez Martinez, Jabier Martinez, Aurelien Ibanez, Matteo Morelli, Ansgar Radermacher, Huascar Espinoza

Design-Time Safety Assessment of Robotic Systems Using Fault Injection Simulation in a Model-Driven Approach

Garazi Juez Uriagereka,
Estibaliz Amparan,
Cristina Martinez Martinez,
Jabier Martinez
Digital Trust Technologies Area (TRUSTECH)
TECNALIA - ICT Division
Derio, Spain
name.firstSurname@tecnalia.com

Aurelien Ibanez
AKEO PLUS
Chateau-Gaillard, France
a.ibanez@akeoplus.com

Matteo Morelli,
Ansgar Radermacher,
Huascar Espinoza
CEA, LIST
Laboratory of embedded and autonomous systems
Gif-sur-Yvette, France
name.surname@cea.fr

***Abstract*—The rapid advancement of autonomy in robotic systems together with the increasing interaction with humans in shared workspaces (e.g. collaborative robots), raises pressing concerns about system safety. In recent years, the need of model-driven approaches for safety analysis during the design stage has gained a lot of attention. In this context, simulation-based fault injection combined with a virtual robot is a promising practice to complement traditional safety analysis. Fault injection is used to identify the potential safety hazard scenarios and to evaluate the controller's robustness to certain faults. Besides, it enables a quantitative assessment w.r.t. other techniques that only give qualitative hints, such as FMEA. Thus, it facilitates the refinement of safety requirements and the conception of concrete mitigation actions. This paper presents a tool-supported approach that leverages models and simulation-assisted fault injection to assess safety and reliability of robotic systems in the early phases of design. The feasibility of this method is demonstrated by applying it to the design of a real-time cartesian impedance control system in torque mode as a use case scenario.**

***Index Terms*—Safety, robotic systems, fault injection, RobMoSys**

I. Introduction

Through the advance of more complex systems, such as autonomous robots or automated road vehicles, novel challenges on dependability assessment arise. This also increases the amount of potentially hazardous situations, for which traditional safety and security analysis techniques are not sufficient anymore. In order to manage this new complexity, the extension of traditional analysis techniques with simulation-based approaches poses as a promising solution to virtually validate the safety of the system under analysis. In other words, an early safety validation of the system can be achieved by combining model-driven development with simulation-based technology, which includes fault injection techniques and a virtual environment (e.g. a virtual robot).

Nowadays, safety is becoming a crucial property of robotic systems. ISO 12100 [1], ISO [2] 13849 and IEC 62061 [3] are some of the most accepted safety standards in robotics, covering aspects such as functional safety. Functional safety is the aspect of safety that aims to avoid unacceptable risks due to electrical/electronic failures, but also considering that the system should be designed to properly handle likely human errors, and operational/environmental stress. The safety analysis and validation steps are fundamental aspects to perform the safety assessment. Some of the commonly used risk assessment methods are Preliminary Hazard Analysis, Hazard Operability Analysis, FMEA (Failure Mode and Effects Analysis) [4] and FTA (Fault Tree Analysis) [5]. Furthermore, simulation-based fault injection completes these analyses by finding unexpected hazards (fault forecasting) and verifying the implemented safety mechanisms. Those techniques require knowing the effects of certain failures in advance. However, it might be the case that sometimes is not possible to know the effect of a certain failure on different levels. To overcome these failure chain issues, it is necessary to combine the aforementioned traditional techniques with simulation-based fault injection approaches.

Model-driven solutions and software component models can increase software development productivity through automated generation of code, qualification/certification evidence and documentation. For instance, iterative investigation and verification/validation can lower costs by identifying issues before the real implementation is carried out. Furthermore, these approaches facilitate the reuse of pre-qualified software components. Traditionally, safety analysis techniques, which allow deriving safety requirements and performing safety verification and validation, rely solely on the skill and expertise of the safety engineer. Consequently, the task of accomplishing either a FMEA or FTA remains a manual and time-consuming activity.

Two remarkable challenges exist when developing reusable robotic systems and software. Firstly, a modular design needs to be provided. Second, solutions for an early safety assessment are needed, in which model-based safety analysis

and robustness simulations are combined by means of fault injection. One of the most attractive benefits of applying the method presented here is the development of composable, replaceable and reusable safe components.

The contribution of this work is an approach called eITUS (Experimental Infrastructure Towards Ubiquitously Safe Robotic Systems) [6], that is one of the six Integrated Technical Projects (ITPs) that has been selected from the RobMoSys first open call. RobMoSys [7] envisions an integrated approach built on top of the current code-centric robotic platforms, by applying model-driven methods and tools. RobMoSys focuses on building an open and sustainable, agile and multi-domain European robotics software ecosystem.

In this context, one of the main challenges of eITUS is to assess safety properties of robotic systems employing simulation-based fault injection techniques. To address this concern, a simulation-based fault injection framework is coupled with the robot and environment simulator called Gazebo [8]. Gazebo consists of a physics engine that utilises a specific model annotated with robot dynamics and an environment model. The added value of including robot and environment models is that it allows quantitatively estimating the effects of a failure on robot system level.

This paper is structured as follows, Section 2 presents background information. Then, Section 3 details the approach and Section 4 explains how the approach was implemented in the case study. Later on, Section 5 introduces the related work. Finally, Section 6 presents the conclusions and outlines future work.

II. Background

A. Model-based Design for Safe-Aware Compositional Robotic Systems

Model-Driven Development (MDD) is an approach that allows robotic system developers to shift their focus from implementation to the robotics knowledge space and to promote efficiency, flexibility and separation of concerns for different development stakeholders. This reinforces the model-centric vision stated by the Multi-Annual Roadmap for Robotics 2020 [9], where models are not only used by persons but also by robots at run-time, e.g. to monitor and be aware of what they are doing. One key goal of MDD approaches is to be integrated with available development infrastructures from the robotics community, such as middleware (e.g. ROS [10]), real-time control [11], algorithm reuse, and simulation (Gazebo [8]). However, there is a high price to pay since there is not a strong integration between MDD and these latter technologies and even less guidance with respect to ensuring system safety. This is the focus of the RobMoSys project, which envisions an open platform to share models, design patterns, tool assets and methodological knowledge for robotics technologies, and now in addition eITUS, as an industrial experiment to integrate safety methods and mechanisms.

SmartSoft [12] combines a service-oriented component-based approach with Model Driven Software Development. Its component model is represented by a metamodel called SmartMARS, which is implemented as a UML profile. The Domain Specific Language RobotML [13] also proposes an UML implementation and a toolchain based on Papyrus, with the added possibility to describe component behaviour at the same abstraction level as component specifications. RobotML provides means to define the system architecture, the communication mechanisms between components and the behaviour of robotic components that form the system architecture. The BRICS (Best Practice in Robotics) FP7 EU project [13] also promotes the Model-Driven Engineering approach in order to solve robotic software engineering issues. They developed the BCM metamodel to describe a minimal component model suitable for code generation for multiple target middlewares (ROS, OROCOS-RTT). The P-RC2 (Platform for Robot Controller Construction) [14] project also relies on model-based engineering and component-based design approaches. P-RC2 proposes an architecture, a methodology and a set of tools to model, implement, validate and deploy robotic controllers. The feasibility of this design time concept has moreover been proven in a model-driven controller design tool based on Eclipse/Papyrus, which in turn is supported by an evolution of the RobotML metamodel. Components are implemented by contributors (consortium and community) with domain code which is middleware-agnostic, and can be deployed using automatic code generation to widespread middlewares such as OROCOS for industrial applications.

B. Fault Injection-based Safety Assessment for Robotic Systems

Out of a mission and safety perspective, it is of interest to ensure the availability of robotics functions, therefore requiring a fail-operational design. Traditionally, robotic systems have been known for being closed and working in controlled environments. However, autonomous systems in contrast will have to perform a variety of automated tasks in uncertain environments. Due to the unpredictability of certain environmental conditions, robotics systems need to comply with strong dependability requirements to deliver a service that can be trusted. A set of basic definitions of dependability are introduced by Jean-Claude Laprie. Dependability stands for "the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers" [15].

The safety analysis and validation steps are fundamental aspects to perform the safety assessment. Some of the commonly used risk assessment methods are Preliminary Hazard Analysis, Hazard Operability Analysis, FMEA and FTA. Furthermore, fault injection simulations complete these analyses by finding unexpected hazards (fault forecasting) and verifying the implemented safety mechanisms. Fault injection is a common technique for validating the effectiveness of fault-tolerance mechanisms by studying the behaviour of the system in presence of faults. The effects of software or hardware faults are emulated by randomly changing code or data at different software locations. However, with the increasing size of software in todays complex systems, it is a challenging task to define specific fault types and locations that can

effectively emulate realistic fault scenarios. Consequently, to reach dependability at a high safety criticality level, different methods like fault injection are necessary. Arlat et al. [16] defined fault injection as a dependability validation technique that is based on conducting controlled experiments. In detail, the observation of the system behaviour in the presence of faults is explicitly induced by the deliberate introduction of faults into the system. Arlat et al. also proposed an effective way to characterise a fault injection environment known as the FARM (Faults, Activation, Readouts, Measurements) environment model [17]. The FARM model (cf. Fig. 1) is composed of: (1) the set of *Faults* to be injected, (2) the set of *Activations* exercised during the experiment, (3) the *Readouts* to define observers of system behaviour, and (4) the *Measures* obtained to evaluate safety properties.

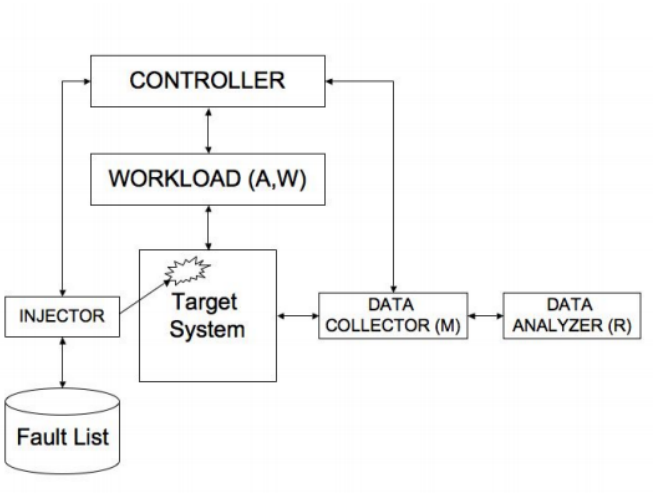

Fig. 1. FARM Model

Regarding fault injection techniques, researchers and engineers have created many novel methods to inject faults [18], which can be implemented at different design levels and are commonly used to pursue the following objectives:

- Understanding a system's behaviour in the presence of real faults.
- Verification of the fault tolerance mechanisms included in the target system for removing design faults.
- Forecasting of faulty behaviour of a target system, by obtaining measurements of the coverage or efficiency and the latency of fault tolerance mechanisms.
- Exploring the effects of different workloads on the effectiveness of fault tolerance mechanisms.
- Identifying weak links or single points of failures (a single fault leading to severe consequences) in the design.

Fault injection is traditionally used for emulating hardware faults and consists of various techniques. Depending on whether faults are injected into a system model or into an actual physical system prototype, techniques are divided into hardware-based fault injection, software-based fault injection or simulation-based fault injection [18]. Hardware-based fault injection directly injects faults into the real target system with external equipment, e.g., placing the target system under heavy-ion radiation or by means of pin-level hardware fault injection. In software-based fault injection, the modified real software is running on the real target system. At last, simulation-based fault injection injects faults into a model of the target system. Those fault models will depend on the level of abstraction of the modelled system. Each technique has different advantages and drawbacks as listed in [18]. Usually, these techniques are combined, for example, by mixing simulation and hardware fault injection. According to Piscitelli et al. [19], in simulation-based fault injection, the target system as well as the possible hardware faults are modelled and simulated by a software program, usually called fault simulator. In order to carry out simulation-based fault injection experiments, the hardware model or the software state of the target system is modified, which allows simulating the effect of a hardware fault in the system. In this way, the system behaviour in the presence of faults can be analysed during early design phases. Three of the main advantages of this technique are (i) the prevention of damage to the target system, (ii) reduced cost compared to other techniques such as hardware-implemented fault injection, (iii) higher observability and controllability of the experiments. The main drawbacks are in terms of fault representativeness and simulation time. The user should select an appropriate fault model and represent it accordingly so that the results of the fault injection simulations are meaningful. Thus, the fidelity of the results strongly depends on the accuracy of the models used.

III. The Fault Injection Framework

Among the aforementioned different fault injection techniques, the work presented here focuses on applying the simulation-based fault injection solution in combination with a model-driven approach. The eITUS fault injection framework sets up, configures, executes and analyses the simulation results. Model-based design combined with a simulation-based fault injection technique and a virtual robot poses as a promising solution for an early safety assessment of robotics systems. The added value of coupling robotics, environment and controller models is that the maximum time before the robot gets into a hazardous event can be calculated by means of simulation. In other words, it allows quantitatively estimating the relationship of an individual failure to the degree of misbehaviour on the robot level.

The fault injection framework has been developed to set up, configure, execute and analyse the obtained simulation traces. This configuration process includes setting up a virtual environment (e.g., robot environment) and the needed information to build up the faulty *System Model Under Test* (SMUT). All the safety analysis techniques (e.g., FMEA view) developed for RobMoSys have been integrated in the Papyrus4Robotics toolchain [20]. Papyrus is an industrial-grade open source model-based engineering tool. It is based on standards and supports model-based design in UML, SysML

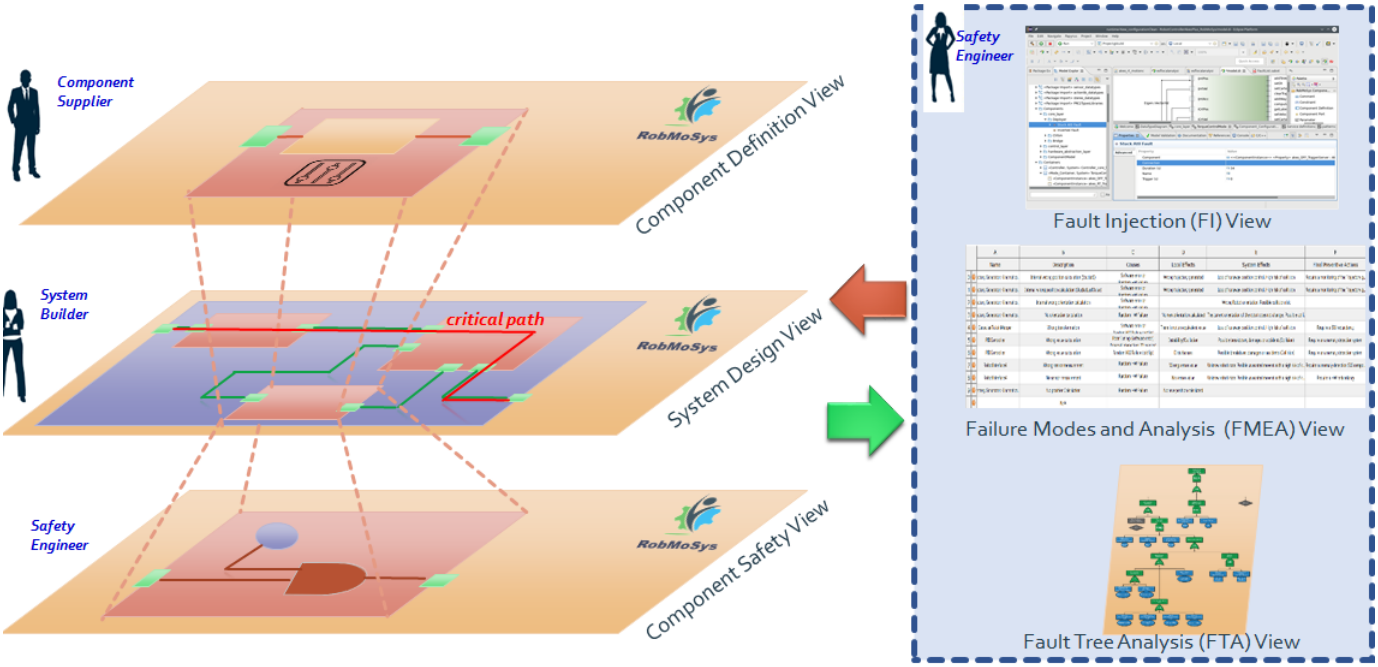

Fig. 2. Safety Views

and MARTE. The platform uses the UML profile mechanism to enable the implementation of Domain Specific Modelling Languages (DSMLs) that assist RobMoSys' ecosystem of users in designing robotic systems. Papyrus4Robotics features a modelling front-end which conforms to the RobMoSys' foundational principles of separation of roles and concerns. Fig. 2 illustrates how safety analysis is related to RobMoSys views and introduces a new role called safety engineer. In detail, Fig. 2 depicts the relationship between safety engineering activities and system engineering processes. Moreover, the three major safety analysis techniques are integrated into the model-driven development process.

Fig. 3 shows the main fault injection building blocks and the flow of models to perform safety assessment during simulation in early design phases of the V-cycle. The framework operates as follows. First, a workload generator generates the functional inputs to be applied to the SMUT. The workload generator consists of:

(i) selecting the SMUT,

(ii) choosing the robot type -*Robot selection*- and the environment scenario -*Operational Situation Selection*- from a catalogue, and

(iii) configuring fault injection experiments, i.e. creating the *fault list* and choosing read-outs or observation points (signal monitors).

Then, a *fault injector* uses both, the *fault list* and a fault model library implemented as C++ code templates, to create the saboteurs and generate as many test cases as the engineer needs. Saboteurs are extra components added as part of the model-based design for the sole purpose of Fault Injection experiments. Fault models are characterised by a type (e.g., *frozen*, *stuck at 0*, *delay*, *invert*, *oscillation* or *random*), target location, injection triggering (e.g., scenario position or time driven), and duration. In order to create a faulty SMUT, the *fault injector* injects an additional saboteur model block per fault entry from the *fault list*. Moreover, the injected block is filled with information coming from a fault model template library.

The SMUT (Papyrus4Robotics model) is extended with extra blocks called saboteurs. They reproduce a certain faulty behaviour of different components such as a sensor or an electronic control unit. Adding signal injectors or saboteurs at the inputs of the components together with read-out blocks or monitors at the outputs, establishes a viable solution to conduct complex fault injection campaigns. Fault models can thereby be selected by identifying potential prototypical failure modes (e.g., too high, too low or too late). Briefly speaking, the following information is considered:

- Where should the faults be injected?
- What is the most appropriate fault model representing the functional failure modes?
- How should the faults be triggered within the system?
- Where should the fault effect be observed?

By comparing a fault free simulation (golden SMUT) versus faulty ones (faulty SMUTs), tests can be automated and results obtained. The results of the simulation experiments complete the safety analysis and help dimensioning the safety concept by considering the system's fault tolerant time intervals. Thus,

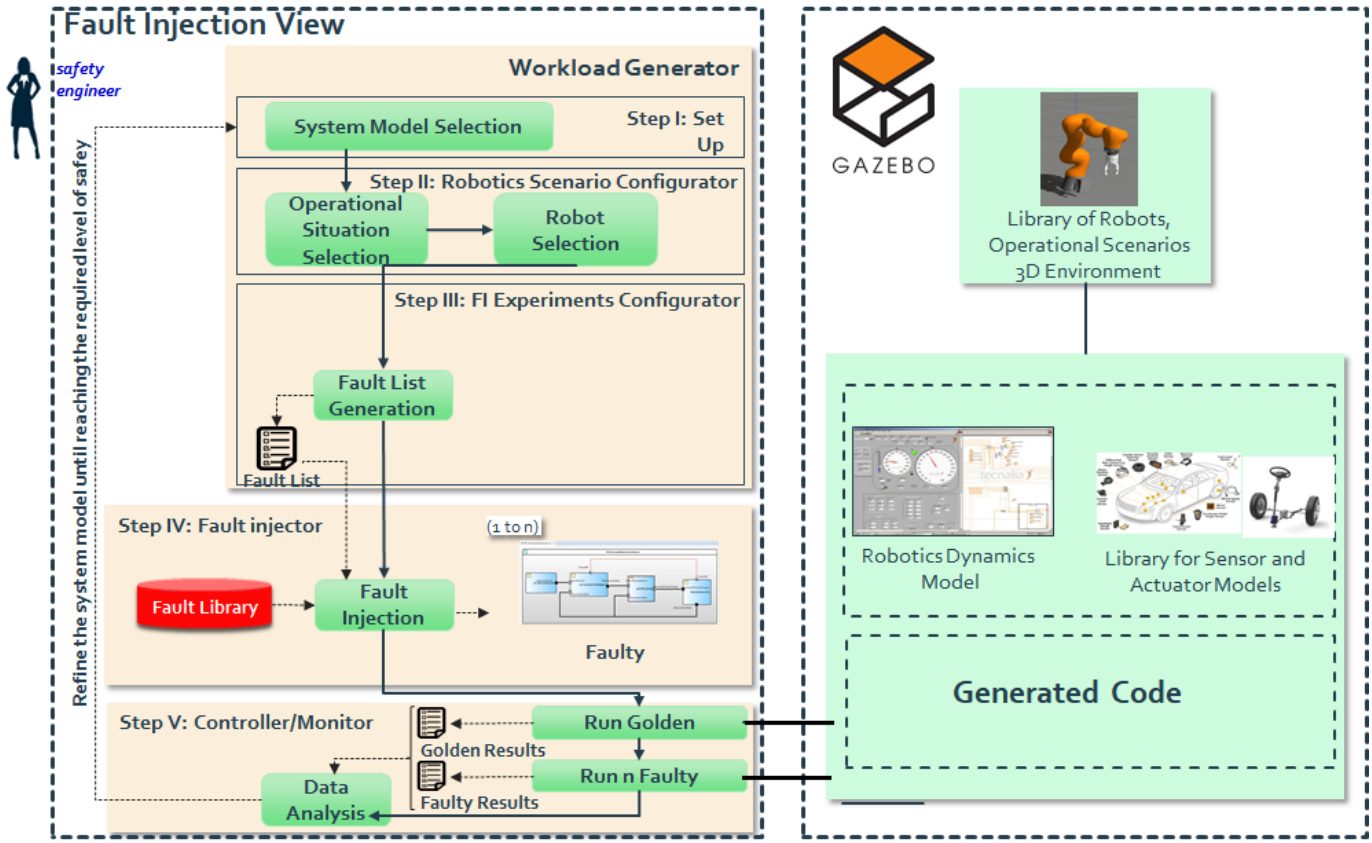

Fig. 3. The Fault Injection Framework

it assists in determining the required level of fault tolerance (e.g., redundancy or graceful degradation), identifying hazards (e.g., robot does not turn when it should) and ranking the failure modes with respect to fault occurrence.

Once a fault free version of the SMUT (golden) and at least one faulty SMUT are available, the simulation environment is invoked through the monitor. The Monitor not only runs experiments under the pre-configured environment scenario, but also compares and analyses the collected data. Regarding fault injection approaches, the solution developed in terms of the eITUS approach stands for a simulation-based fault injection framework for an early safety assessment. Combining simulation-based fault injection approaches and the inclusion of Gazebo for robot dynamics and environment simulation with the RobMoSys design platform, is one of the main goals of eITUS. Furthermore, Fault Injection can be applied for dimensioning monitoring functions by determining a system's maximum response time before a hazardous event occurs. The results of the simulation experiments will complete or verify the safety analysis and help dimensioning the safety mechanisms regarding the maximum tolerable time assigned to the safety monitor, which must handle the situation before the hazard occurs. Moreover, this approach will allow for an early dependability validation of the function and its fault tolerance.

IV. Case Study: Cartesian Impedance Control System in Torque Mode

The eITUS approach is applied to an industrial collaborative robot. Especially, a real-time Cartesian impedance Control System in torque mode is introduced as a use case scenario. The aim of this case study is to demonstrate the benefits of the eITUS outcomes, which have been focused on collaborative robotics, a field strongly relying on dependability and safety. More precisely, a prototype application of a collaborative robotics solution has been developed that includes a robotic arm manipulator. This case study has been developed following the approach of RobMoSys, where the complexity of robotic development is reduced by composing components and separating tasks that are executed at different levels of abstraction and by different roles, focusing on the safety related functionalities developed as part of the eITUS project via a safety analysis.

Fig. 4 complements the concept depicted in Fig. 3 by introducing the general eITUS process in the context of the described case study. Before starting the fault injection experiments, the golden system model, which represents a model without any faults in place, and its corresponding simulations, must be created.

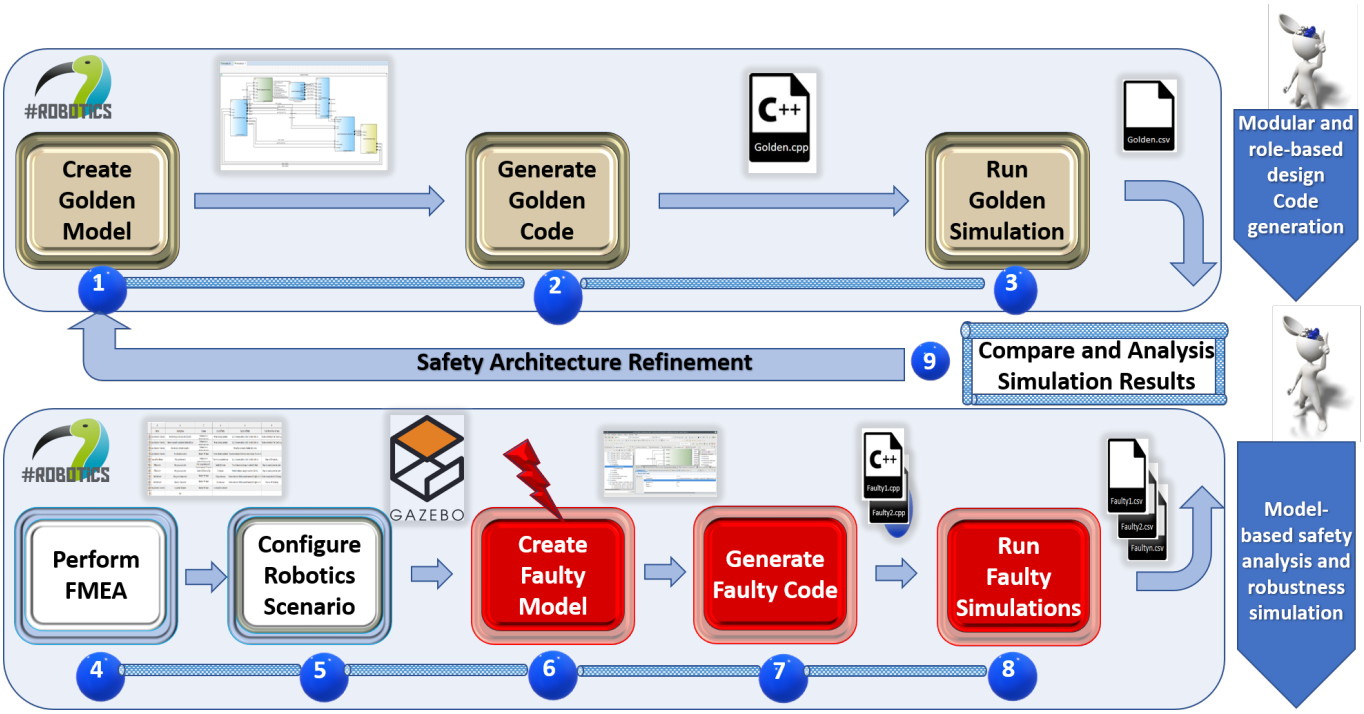

Fig. 4. eITUS Process

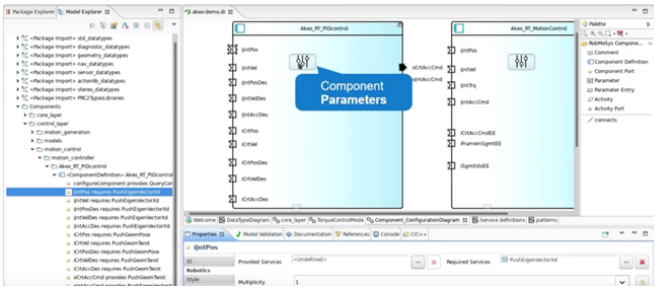

Fig. 5. Configuration of Components

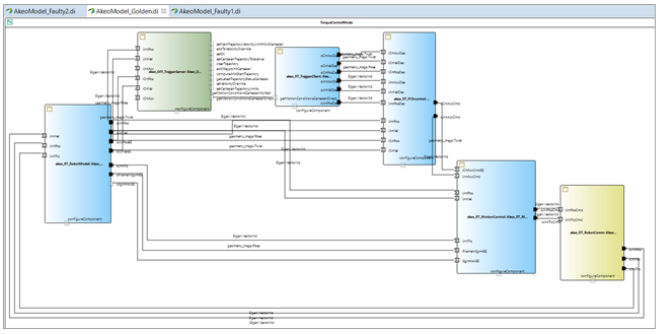

Fig. 6. Torque Control System modelled in Papyrus4Robotics

To start with, a component supplier manages the design and configuration of components as depicted in Fig. 5. Thereafter, a system builder designs the global architecture by assembling components and adjusting settings (cf. Fig. 6). After that, the controller code is generated, and the simulations are run in Gazebo.

Once the components and the system are designed, the safety engineer starts the FMEA process, which can be performed in a compositional way. First, FMEAs are carried out at component level. Then, the FMEA is developed at system level and the safety requirements are defined (cf. Fig. 7). Each failure mode set during this process will be represented as one fault model library in the configuration of the faulty system. In order to prevent a failure mode, a preventive action or safety requirement is defined by the safety engineer. These safety requirements will be the basis for the pass/fail criteria of the fault injection experiments.

The safety engineer specifies the safety requirements related to the controller via the safety requirements table. For example, as applies to this case study, “The velocity of the Robot arm must not be greater than 0.25 m/s”. The safety requirements are those requirements that are defined for the purpose of risk reduction.

After performing the traditional model-based safety analysis, fault injection experiments can be specifically set up to redefine and validate the aforementioned FMEA. Sometimes this process might also be known as Robustness Simulation. The safety engineer starts by selecting the system model and the robotics scenario, which includes the operational situation and the robot. After that, it is important to define the fault injection policy, which is referred to as the *fault list*. The golden model is extended with the different faults leading to the creation of the faulty controller (cf. Fig. 8).

		A	B	C	D
		Name	ESFCore::AbstractSElement:...	Causes	Local Effects
0	No sensor/encoder measu...	No sensor/encoder measurement in joint 1 x	N/A	Random Hardware failure	No sensor value
1	No sensor/encoder measu...	No sensor/encoder measurement in joint 1 y	N/A	Random Hardware failure	No Sensor Value
2	No sensor/encoder measu...	No sensor/encoder measurement in joint 1 z	N/A	Random Hardware failure	No Sensor Value
3	Internal wrong position ca...	Internal wrong position calculation (stuckat0)	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
4	Internal wrong position ca...	Internal wrong position calculation. Too High value.	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
5	Internal wrong position ca...	Internal wrong position calculation. Too Low value.	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
6	Internal wrong position ca...	Internal wrong position calculation. Out of range value.	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
7	Internal wrong position ca...	Internal wrong position calculation. Early value.	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
8	Internal wrong position ca...	Internal wrong position calculation. Late value.	N/A	Software error or Random Hardware fail...	Unwanted trajectory generator
9	Wrong constants value set	Wrong constants value set	N/A	Software error	Instability torque value
10	Wrong output value calcul...	Wrong output value calculated	N/A	Memory failure (bit flip)	Instability or Oscillation
11	Wrong output value calcul...	Wrong output value calculated	N/A	External interactions (Dynamic Perturbati...	Disturbances
12	Motion control omission	Motion control omission	N/A	Software error or Random Hardware fail...	Unwanted motor speed
13	Motion control comission	Motion control comission	N/A	Software error or Random Hardware fail...	Unwanted motor speed
14	Early Motion control	Early Motion control	N/A	Software error or Random Hardware fail...	Unwanted motor speed
15	Late Motion Control	Late Motion Control	N/A	Software error or Random Hardware fail...	Unwanted motor speed
16	Too High Value motion co...	Too High Value motion control	N/A	Software error or Random Hardware fail...	Unwanted motor speed
17	Too Low Value motion con...	Too Low Value motion control	N/A	Software error or Random Hardware fail...	Unwanted motor speed

Fig. 7. Torque Control: FMEA view

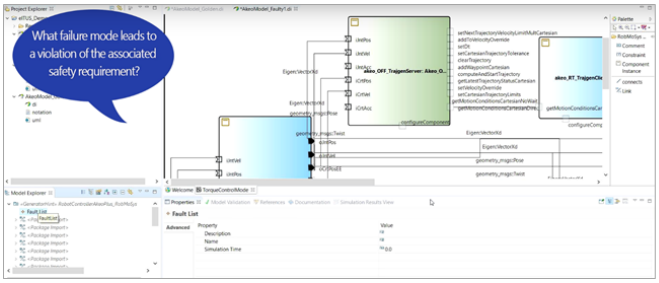

Fig. 8. Fault Injection View: Creation of the *Fault List*

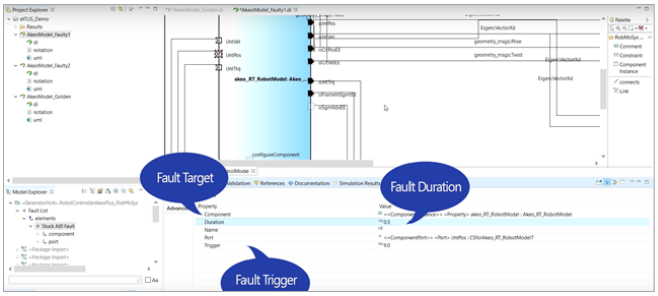

Fig. 9. Fault Injection View: configuration of a fault

This configuration process includes the definition of fault locations (Where to inject the fault?), fault injection times (When to trigger the fault?), fault durations (For how long the fault is present in the system?) and the fault model (How does the component fail?). The original system model is modified though the *fault injector* script according to the *fault list*. The *fault list* is used to produce a faulty model only in terms of reproducible and prearranged fault models. As illustrated in Fig. 9 fault models are characterised by a type (for example, omission, frozen, delay, invert, oscillation or random), target location, injection triggering (time driven), and duration. All this allows to:

- Exhaustively explore all possible behaviours of a system architecture with respect to some safety property of interest (e.g., the pre-defined safety requirement "The velocity of the Robot arm must not be greater than 0.25 m/s").
- Simulate the behaviour of system architectures early in the development process to explore potential hazards.

Afterwards, the read-out or observation points (signal monitors) are chosen and created. These blocks hold the information concerning the safety requirements or the pass/fail criterion of the tests, which will be used to compute and finalise the results (cf. Fig. 10). For instance, read-outs need to be logged in order to check what failure modes or fault models violate the aforementioned safety requirement: "The velocity of the Robot arm must not be greater than 0.25 m/s".

To aid model-based safety analysis and fault injection simulations, the original or golden system model must be extended by means of the *fault list*. Then, the generated code is instrumented through the *fault injector* script according to the *fault list*. For that, Xtend technology is used[1]. Out of these extended or faulty models the deployed code is to be generated, and the simulations are run.

To create the faulty code, the *fault injector* inserts an additional saboteur model block per fault entry from the *fault list* together with the associated fault models which are coded as templates in a fault library.

Once the faulty code is generated, steps 8 and 9 must be accomplished. The results of the fault injection experiments can be visually seen in the fault injection simulation results

[1]Xtend: https://www.eclipse.org/xtend/

Fig. 10. Simulation Results View

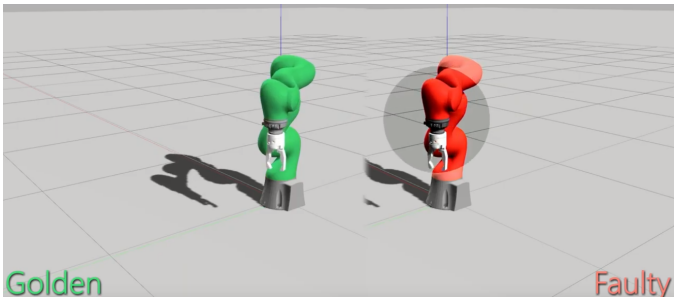

Fig. 11. Golden versus faulty in Gazebo.

view (cf. Fig. 10) or in the Gazebo simulation environment (cf. Fig. 11).

After analysing the effect of different failure modes in the system, Fig. 10 illustrates how the *stuck at 0* fault model (failure mode: no encoder measurement in axis X) violates the established safety requirement (velocity greater than 0.25 m/s) leading to a hazard. As a consequence, the architecture needs to be redefined (for example, by applying a 2-out-of-3 architecture for the encoder or including a plausibility check for the sensor) so that this failure mode is controlled to avoid the hazard to occur. Once the implementation of the eITUS Case Study has been performed, the Goal-Question-Metric approach (GQM) [21] has been used to evaluate the effectiveness of the proposed technologies. This approach defines a measurement model on three levels: conceptual level (Goal), operational level (Question) and Quantitative level (Metric).

eITUS benchmarking has been driven by the project goals: (i) demonstrate a gain for design efficiency of robotics systems by reducing safety assurance effort, and (ii) help to demonstrate a remarkable impact in the robotics community by increasing the interoperability of RobMoSys technologies with deployment and simulation technologies. These goals have been then decomposed in technical questions and questions in turn have been broken down into metrics.

The achievement of these goals has been assessed by comparing eITUS-supported practice regarding the current state of practice, in terms of metric measurements in the eITUS solution and expert estimations. The main results of the assessment provide an estimation of the potential savings with the eITUS solution. For example, the effort for reusing safety validation artefacts per component or the effort required in understanding safety-related decisions have notably decreased. For more information on detailed metrics please refer to the deliverable D2.2 of the eITUS project [22].

V. Related Work

Fault injection has been applied for evaluating and validating the dependability of safety-critical systems in different safety-critical domains [23] [18]. Functional safety standards such as IEC 61508 [24] mandate the use of this technique during different design phases, while the automotive functional safety standard ISO 26262 [25] highly recommends its usage in phases such as system or hardware development.

Juez et al. [26] presented a Simulink approach combined with a simulation-based fault injection technique and Vinter et al. [27] introduced some model-based approaches as FISCADE (A Fault Injection Tool For SCADE Models) fault injection tool that together with SCADE (Safety-Critical Application Development Environment) automatically replaces original operators with faulty operators. However, this tool does not consider the dynamics of the system. This topic is interesting when critical parameters such as Fault Tolerance

Time Interval (FTTI) are calculated, which are directly related with the controllability of the system.

Until recently, relatively little attention had been paid to safety and fault injection in the robotics community. Alemzadeh et al. [28] used a systems-theoretic hazard analysis technique (STPA) to identify the potential safety hazard scenarios and their contributing causes in RAVEN II, an open-source telerobotic surgical platform. Then, software-based fault injection was applied in order to measure the resilience of systems to the identified hazard scenarios by automatically inserting faults into different parts of the software. In the last decade model-based approaches have become prominent. In contrast to traditional methods, model-based techniques try to derive relationships between causes and consequences from some sort of model of the system. Once a product architecture model annotated with specific safety-related information has been derived, product safety analyses and assessment are performed with the support of a model-based tool. This topic of interest is broadly tackled in the literature. For example, HIP-HOPS [29] and Component Fault Trees [30] automatically generate traditional safety artefacts such as FTA out of extended system models (e.g., SysML, EAST-ADL and Matlab/Simulink).

AMASS [31] addressed different model-based safety and dependability analyses, for example, Sophia [32], Safety Architect [33], ConcertoFLA [34], and Medini Analyze [35]. Unfortunately, such solutions have not been so often applied in the robotics domain. Yakymets et al. [36] propose a methodology and associated framework for safety assessment of robotic systems during early phases of development. To do so, they follow a model-driven engineering approach to implement a preliminary safety assessment by using a quantitative and qualitative FTA. The used domain specific language is RobotML (an earlier robotic modeling language supported by a Papyrus customisation) and the framework includes facilities to automatically generate or manually construct fault trees and to make semantic connections with formal verification and FTA tools. However, the integration with simulation-based fault injection approaches is not tackled.

VI. Conclusion and Future Work

The ability of early identifying possible hazardous situations remains a contentious issue in the dependability community. The limitations of traditional safety analysis together with the risk of finding design issues in late stages make fault injection simulations an attractive solution. Moreover, the availability of tools is a key issue in order to leverage fault injection for the development of robotics system where safety plays a vital role.

Following a model-driven development approach and combining it with simulation-based fault injections and a virtual robot, poses as a promising practice to identify the potential safety hazard scenarios and to evaluate the controller's robustness to certain faults. We have presented the eITUS tool-assisted solution to perform fault injection campaigns in the context of RobMoSys.

For future development, a higher degree of automation between the different safety artefacts would be beneficial. Besides, on the path towards autonomous robots, the framework presented here could further be utilised to simulate error injections by generating erroneous patterns, such as flipped images from image sensors. By doing so, these input tests can evaluate the residual risk arising from real-life situations that could trigger a hazardous behaviour of the system when integrated in a robot. Moreover, extensions to consider not only safety but security could be investigated.

Acknowledgment

This work has been funded by the eITUS project (Experimental Infrastructure Towards Ubiquitously Safe Robotic Systems using RobMoSys). The eITUS Integrated Technical Project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 732410, in the form of financial support to third parties of the RobMoSys Project.

References

[1] "ISO 12100: Safety of machinery – General principles for design – Risk assessment and risk reduction," Standard, 2010.
[2] "ISO 12100: Safety of machinery – Safety-related parts of control systems," Standard, 2015.
[3] "IEC62061: Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems," Standard, 2005.
[4] "IEC 60812: Analysis techniques for system reliability procedure for failure mode and effects analysis (FMEA)," 2006.
[5] "IEC standard 61025 fault tree analysis," 2006.
[6] "eITUS, Experimental Infrastructure Towards Ubiquitously Safe Robotic Systems using Robmosys," https://robmosys.eu/e-itus/.
[7] "Robmosys, composable models and software for robotics systems," https://robmosys.eu/.
[8] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.
[9] "Robotics 2020, multi-annual roadmap for robotics in europe, call 2 ICT24 (2015) Horizon 2020, 2015."
[10] "Robot operating system (ROS)," http://www.ros.org/.
[11] "The open robot control software (OROCOS) project, http://www.orocos.org/ open source computer vision library (openCV)," http://opencv.org/.
[12] C. Schlegel, A. Steck, and A. Lotz, *Model-Driven Software Development in Robotics: Communication Patterns as Key for a Robotics Component Model*, 01 2011, pp. 119–150.
[13] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS Component Model: A Model-based Development Paradigm for Complex Robotics Software Systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13, 2013, pp. 1758–1764.
[14] B. Milville, B. Gradoussoff, and B. Morinière, "P-RC2: Platform for robot controller construction," in *11th National Conference on Software and Hardware Architectures for Robots Control & Quatrièmes Journées Architectures Logicielles pour la Robotique Autonome, les Systèmes Cyber-Physiques et les Systèmes Auto-Adaptables (SHARC 2016)*, Brest, FR, 2016.
[15] J. C. Laprie, A. Avizienis, and H. Kopetz, Eds., *Dependability: Basic Concepts and Terminology*. Berlin, Heidelberg: Springer-Verlag, 1992.
[16] J. Arlat, A. Costes, Y. Y. Crouzet, J.-c. Laprie, and D. Powell, "Fault injection and dependability evaluation of fault-tolerant systems," *IEEE Transactions on Computers*, vol. 42, 02 1970.
[17] J. Arlat, A. Costes, Y. Crouzet, J. C. Laprie, and D. Powell, "Fault injection and dependability evaluation of fault-tolerant systems," *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 913–923, Aug. 1993. [Online]. Available: https://doi.org/10.1109/12.238482

[18] H. Ziade, R. Ayoubi, and R. Velazco, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, pp. 171–186, 01 2004.
[19] R. Piscitelli, S. Bhasin, and F. Regazzoni, "Fault attacks, injection techniques and tools for simulation," in *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, April 2015, pp. 1–6.
[20] "Papyrus4robotics," https://robmosys.eu/wiki/baseline:environment_tools:papyrus4robotics.
[21] R. Solingen and E. Berghout, "The goal/question/metric method: A practical guide for quality improvement of software development," 01 1999.
[22] "D2.2 eITUS demonstrator," https://robmosys.eu/e-itus/.
[23] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, 1st ed. Springer Publishing Company, Incorporated, 2010.
[24] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems (E/E/PE, or E/E/PES)," https://www.iec.ch/functionalsafety/explained/.
[25] "ISO 26262 road vehicles- functional safety."
[26] G. Juez, E. A. Calonge, R. Lattarulo, A. Ruiz, J. Pérez, and H. Espinoza, "Early safety assessment of automotive systems using sabotage simulation-based fault injection framework," in *SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings*, ser. Lecture Notes in Computer Science, vol. 10488. Springer, 2017, pp. 255–269.
[27] J. Vinter, L. Bromander, P. Raistrick, and H. Edler, "FISCADE - a fault injection tool for SCADE models," 07 2007, pp. 1 – 9.
[28] H. Alemzadeh, D. Chen, A. Lewis, Z. Kalbarczyk, and R. K. Iyer, "Systems-theoretic safety assessment of robotic telesurgical systems," *CoRR*, vol. abs/1504.07135, 2015. [Online]. Available: http://arxiv.org/abs/1504.07135
[29] "HIP-HOPs," http://hip-hops.eu/.
[30] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Mhrle, A. Berres, M. Zeller, K. Hfig, and M. Rothfelder, "Advances in component fault trees," 06 2018.
[31] "AMASS ECSEL project," https://www.amass-ecsel.eu/.
[32] D. Cancila, F. Terrier, F. Belmonte, H. Dubois, H. Espinoza, S. Grard, and A. Cuccuru, "SOPHIA: a modeling language for model-based safety engineering," 2009.
[33] "Safety architect tool by ALL4TEC," https://www.all4tec.net/safety-architect.
[34] B. Gallina, Z. Haider, A. Carlsson, S. Mazzini, and S. Puri, "Multi-concern dependability-centered assurance for space systems via concertoFLA," in *23rd International Conference on Reliable Software Technologies - Ada-Europe 2018*, vol. 10873, June 2018. [Online]. Available: http://www.es.mdh.se/publications/5059-
[35] "Medini Analyze," http://www.medini.eu.
[36] N. Yakymets, S. Dhouib, H. Jaber, and A. Lanusse, "Model-driven safety assessment of robotic systems," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1137–1142, 2013.