# DECISIVE: Designing Critical Systems with Iterative Automated Safety Analysis

Ran Wei, Zhe Jiang*, Xiaoran Guo*, Ruizhe Yang*, Haitao Mei, Athanasios Zolotas and Tim Kelly

***Abstract*—Systems safety is becoming increasingly challenging due to the presence of ever-more complex applications. Safety analysis is an important aspect of Safety-Critical Systems Engineering (SCSE) to discover problems in system design that can potentially lead to hazards with risks that may lead to accidents. Performing safety analysis requires significant manual effort — its automation has become the research focus in the critical system domain due to the increasing complexity of systems and the emergence of open adaptive systems. In this paper, we propose a novel methodology in which automated safety analysis drives the design of safety-critical systems. We delve into the specifics of our approach and the supporting tools. Additionally, we discuss the method to integrate our approach into the current practice of SCSE. The experimental results reveal that the proposed approach with its supporting tool promotes the efficiency of safety analysis significantly, whilst maintaining high degrees of correctness, coverage and scalability.**

## I. Introduction

Safety-critical systems have stringent assurance and verification requirements that are essential to life-critical application scenarios, including medical, automotive, aerospace, and industrial automation [1]–[5]. In order to certify such systems, justifications are required to argue and demonstrate that they are acceptably safe to operate in defined operational contexts. To illustrate such justifications, practitioners typically list all the safety goals (and their relationships), the contextual information regarding system configuration, environments, etc. and most importantly, the evidence to substantiate that all safety goals are met. The above form a compelling argument regarding the safety of the system, which is typically organised in an *assurance case* [1]. Prior to certification, an assurance case must be rigorously, and often independently, evaluated to ensure that the arguments and evidence for safety are coherent and convincing [2]. Amongst the collection of evidence required to argue the safety of systems, safety analysis results hold a pivotal role.

Safety analysis is key in critical system engineering, primarily identifying design issues — mostly arising from system component failures — that can potentially induce hazards and lead to accidents. Once identified, these issues should be mitigated by improving the system design to enhance the reliability of (part of) the system that performs critical functions. In numerous industries, performing safety analysis is mandatory. Notable examples include the preliminary system safety assessment in SAE ARP4761 for the aerospace industry [6] and the safety analysis in ISO 26262 for the automotive industry [7]. As suggested in [8], safety analysis is required in the entire engineering lifecycle, i.e., from the earliest planning phase until the end of the development and verification phases.

Manual effort is heavily required in safety analysis techniques due to its highly subjective nature and the need for analysts' skills. However, the growing complexity of safety-critical systems, coupled with the emergence of *Open Adaptive Systems* [8], present considerable challenges for such manual approaches. Therefore, there's an imperative need for automated safety analysis methods to: (i) address the increasing complexity of critical systems, thus improving efficiency in system development; (ii) integrate into the system assurance process to enable the automated validation of system assurance cases; (iii) transition system assurance activities from design time to runtime (e.g. analysis and monitoring), guaranteeing assurance for the open adaptive systems with runtime uncertainties.

**Contributions.** In our previous work [4], we presented DECISIVE (DEsigning CrItical Systems with IteratiVe automated safEty analysis), a critical system design methodology, driven by automated, model-based safety analysis. We also presented SAME (Safety Analysis Management Environment), a tool prototype which exploits the benefits of Model-Based Systems Engineering (MBSE) to enable automated safety analysis by means of automated fault injection (within Matlab/Simulink). As SAME heavily relied on Matlab/Simulink, accompanied with SAME, we proposed SSAM (Structured System Architecture Metamodel), a modelling language which can be used to describe system components. On model instances that conform to SSAM, we developed a graph algorithm to perform safety analysis in an automated manner. However, SSAM in [4] was only able to create very simple block-based representations of systems. Since then, we have received rather a large number of requests to support DECISIVE and SAME on models defined in multiple platforms (e.g. Simulink, PTC Integrity Modeler [9], MagicDraw [10]). We motivate this work on the following identified requirements:

- **REQ1**: SSAM shall provide facilities to support engineering activities such as requirement elicitation, hazard identification, system design and failure modelling at multiple levels so that these models can be federated.
- **REQ2**: SSAM models shall contain the traceability to external, heterogeneous models, so that data contained in such models can be extracted by SAME and used in

Ran Wei is with the University of Cambridge, UK.
Zhe Jiang is with Southeast University, China.
Xiaoran Guo is with Specialised research, China.
Ruizhe Yang is with Dalian University of Technology, China.
*Corresponding authors.

the safety analysis. The users shall be able to define the traceability and rule to extract such data.
- **REQ3**: SAME shall provide a graphical user interface to support the above and existing functions.

In this paper, we revise SSAM and SAME in order to fulfil the requirements identified above. We discuss how the revised SSAM can be used in requirement engineering, hazard analysis and risk assessment as well as system design, and demonstrate how data from users' model can be fused into a SSAM model and be used in the automated safety analysis. We make the following additional contributions:

- A comprehensive modelling language (SSAM) which allows the users to create a) system safety requirement models; b) hazard analysis and risk assessment models; and c) block-based system component models on different levels of abstraction.
- A tested transformation algorithm to transform Simulink models to SSAM without information loss.
- A designed facility in SSAM, with support from SAME, to enable the traceability to heterogeneous models defined in different technologies, and federate such information, based on users' needs in the automated safety analysis.
- A graphical model editor in SAME for SSAM to support the existing functions, as well as the functions identified above.
- The integration of DECISIVE and the above into Safety Critical Systems Engineering processes, as well as an in-depth evaluation for our approach in terms of correctness, coverage, efficiency and scalability.

## II. Preliminaries and Motivation

### A. Safety-Critical Systems Engineering Lifecycle

The precise definition of a Safety-Critical System Engineering (SCSE) lifecycle, and particularly the terms used, depends on the respective application domain. For the sake of simplicity, we use the terms defined in ISO 26262 [7], which is the mandatory safety standard for the automotive industry.

The objective of SCSE is to ensure the 'freedom from unacceptable risk.' Risk typically refer to a combination of a) the probability of harm, and b) the severity of that harm. The harm is caused by *hazardous events* (when a hazard, a specific operational context and a specific configuration of the system coincide). The preliminary step in any lifecycle pertaining to SCSE involves the identification of these hazardous events and the assessment of their associated risks, which is typically referred to as Hazard Analysis and Risk Assessment (HARA).

Based on the findings from HARA, safety requirements are derived. Each safety requirement contains a functional part and an integrity level. The functional part specifies the functions that the system must (or must not) perform, whilst the integrity level specifies the degree of rigour necessary for the implementation of this requirement. Typical examples of integrity levels include Safety Integrity Levels (SILs) defined in IEC 61508 [11], and the Automotive Safety Integrity Levels (ASILs) defined in ISO 26262 [7].

Then, *safety analyses* are performed (taking available development artefacts as input) in order to identify potential causes of the identified system failures, safety requirements may be broken down into specific requirements based on the analysis results. A wide range of established analysis techniques has been developed [12]–[14], amongst which Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are the most widely used safety analysis techniques in practice.

With the analysis results, safety concepts can be derived [8][1]. Safety concepts include all relevant safety requirements and their allocation to functions and components. Based on the safety concept, an *assurance case* (or *safety case*) shall be developed [1], which forms the basis for certification. SCSE is incremental and iterative, when new hazards are identified, or system requirements are changed, every artefact along the process of SCSE shall be updated and re-validated to analyse the impact of all changes.

### B. Failure Modes, Effects, and Diagnostic Analysis (FMEDA)

Failure Modes and Effects Analysis (FMEA) is one of the most adopted techniques for safety analysis, which is a mandatory requirement in the development of safety-critical systems in various domains. Examples of these include EN 50128 for railway systems [3], DO-178C for avionics [15], and ISO 26262 for automotives [7]. The primary goal of FMEA is to comprehensively identify the Failure Modes (FMs) of components and subsequently analyse their potential effects on the overall system. This analysis helps determine the causes of potential hazards, with the ultimate aim to mitigate risks by improving system design [14].

**FMEA procedures.** Conducting an FMEA on a system involves the following steps:

- Step 1: decompose and partition the system into components concerning safety.
- Step 2: identify the function of each component, and its characteristics (safety-related or non-safety-related).
- Step 3: determine potential FMs of each safety-related component (and the probability distributions for the FMs). This can be obtained through the component manufacturer, or from certain documents (e.g. MIL-HDBK-338B).
- Step 4: for each FM of the safety-related component, analyse their impacts on the whole system.

In addition to FMEA, *Safety Mechanism*s (SMs) can also be deployed on components. SMs provide additional safety coverage (in terms of percentage by approximation) to decrease the probability of failure for components. FMEA with safety mechanism analysis is typically referred to as *Failure Modes, Effects and Diagnostic Analysis (FMEDA)*, therefore an additional step is needed:

- Step 5: apply SMs for the components, and quantitatively analyse the diagnostic coverage for the FMs.

In Table I, we illustrate an example of FMEDA on a Phase Locked Loop (PLL) using the above steps[2]. FMEDA is a quantitative analysis, as system architecture metrics can be

[1] Although the terminology of *safety concept* is defined in ISO 26262 [7], it is often used in different domains to signify the same notion

[2] *Comp.*: Component; *Char.*: Characteristic; *FM*: Failure Mode; *Dist.*: Distribution; *SMs*: Safety Mechanisms; *Cov.*: Coverage.

TABLE I
FMEDA ON PHASE LOCKED LOOP (PLL), DVF/IVF: DIRECTLY/INDIRECTLY VIOLATE SAFETY GOAL.

| Char. | FM | Impact | Dist | SMs | Cov. |
|---|---|---|---|---|---|
| safety-critical | lower frequency | DVF | 40.1% | time-out watchdog | 70% |
| | higher frequency | IVF | 28.7% | N/A | 0% |
| | jitter | DVF | 31.2% | dual-core lockstep | 99% |

derived from the results of FMEDA. Such metrics include Single Point Fault Metrics (SPFM) from the ISO 26262 standard, indicating the percentage of single point faults covered by a system design. In ISO 26262, four Automotive Safety Integrity Levels (ASILs) are defined (*ASIL-A* to *ASIL-D*), where *ASIL-A* is the least stringent level and *ASIL-D* the most stringent one). For different ASILs, ISO 26262 require different target SPFM values; for ASIL-B, the SPFM shall be $\geq 90\%$, $\geq 97\%$ for ASIL-C, and $\geq 99\%$ for ASIL-D. Details of SPFM calculation are discussed in Section V.

### C. The Need for Automation and Model Federation

Safety analysis such as FMEA often requires manual effort, as they are highly subjective and largely depend on the analysts' skills [8]. However, as the complexity of systems increases, manual safety analysis reaches its bottleneck and faces significant challenges [16]. In [4], we identified the need for automation for safety analysis in the sense that:

- Performing FMEA manually on complex systems is a lengthy, labour-intensive and error-prone process.
- Manual FMEA is not feasible for SCSE as SCSE is an iterative process, which may cause safety problems due to uncovered failure modes.
- The emergence of Robotics and Autonomous (RAS) with *Open* and *Adaptive* nature requires that some of the safety-related activities, including safety analysis, to be performed at runtime [8].

One current and pressing challenge that could be prioritised for automated safety analysis, is the lack of ability to federate information across multiple types of models defined in different platforms/technologies. For example, to perform FMEA, the failure rates, failure modes and their probability distribution is typically required. To perform FMEDA, information regarding applicable safety mechanisms for components is typically needed. On a higher level, components and their respective requirement shall typically link, and the failure modes of a component shall also be associated with identified hazards. On the SCSE level, FMEA results shall also be associated with evidence that supports safety goals. In the current state of practice, the above has not been properly addressed. In this work, we aim to provide a research direction to address the above challenges.

## III. APPROACH OVERVIEW

We now re-iterate our approach, DECISIVE (*DEsigning CrItical Systems with IteratiVe automated safEty analysis*), its processes are shown in Figure 1. There are five steps in DECISIVE, in which both system development artefacts (upper swim lane) and system assurance artefacts (lower swim lane) are derived. We discuss mainly DECISIVE with model-based support (as it promotes automation for the safety analysis process). However, it is worthy to note that DECISIVE may also be followed without model-based support (therefore, losing the benefit of automation).

In **Step 1**, the system (to be developed) is planned. First, the system definition is specified (which includes system boundaries, functions, running environments, etc.). From the system definition, the function requirements of the system are produced. Along with the definition of the system, Hazard Analysis and Risk Assessment (HARA) shall be performed, after which a hazard log will be produced.

In **Step 2**, the system is designed, based on the function requirements and hazard log in Step 1. The system design can be of any level (for ISO 26262, there are conceptual level, system level, hardware level and software level). In this step, system safety requirements are formed, which are then taken into consideration to produce a system architectural design.

In **Step 3**, reliability data related to each component of the system (e.g. the *failure mode*(s), the *failure mode distribution* and the probability of failure for each component) is aggregated into the system design. The reliability data can be obtained from various sources, such as standards (e.g. MIL-HDBK-338B) and component manufacturers, which form a *component reliability model*.

In **Step 4a**, the system design is evaluated based on each component's reliability data. In this paper, we focus on FMEA and its variants. The tasks in this step can be fully automated with our tool support, which automatically determines safety-related components and their failure modes. Afterwards, a *component safety analysis model* is produced, from which architectural metrics (such as Single Point Failures Metric – SPFM) can be derived. It is to be noted that this safety analysis model is optional – our tool support allows the architectural metrics to be calculated and an Excel-based FMEA table is always produced.

An optional **Step 4b** can be taken, in which further (in-place) refinement to the system design is possible. In this step, analysts can deploy *safety mechanisms* to components (e.g. from a *safety mechanism model*). Safety mechanisms can help diagnose failures of components, and can significantly decrease the probability of failures for components. Once safety mechanisms are deployed to components, analysts can go back to **Step 4a** and evaluate the system design, to check if the refined design meets the target safety integrity level. It is to be noted that **Step 4b** can only be taken if the safety mechanism deployment and Step 4a can be automated (which is achieved in our tool support). This means that analysts do *not* have to make actual changes to the system design – they can import the safety mechanism model which will be used in the evaluation of the system design to compute architectural metrics. This means they can enumerate and deploy the safety mechanisms to components until they find the best trade-off between safety and cost (which are often primary concerns for safety-critical systems engineering). Of course, chosen safety mechanisms need to be added to the system architectural

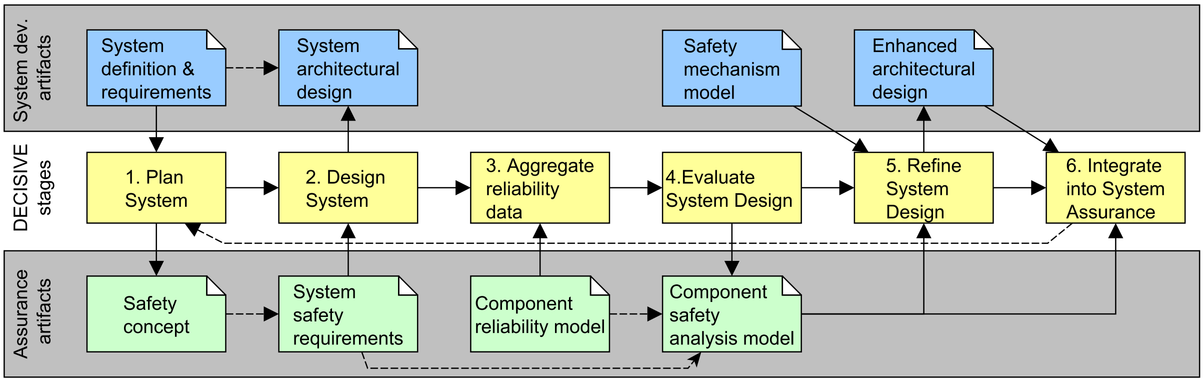

System dev. artifacts
System definition & requirements
System architectural design
Safety mechanism model
Enhanced architectural design
DECISIVE stages
1. Plan System
2. Design System
3. Aggregate reliability data
4.Evaluate System Design
5. Refine System Design
6. Integrate into System Assurance
Assurance artifacts
Safety concept
System safety requirements
Component reliability model
Component safety analysis model

Fig. 1. Stages and key artefacts of the DECISIVE methodology.

design in the next iteration of the DECISIVE process, with a proper change management process (e.g. as standardised in Clause 8 of ISO 26262 [7]), which is not the focus of the DECISIVE process).

In **Step 5**, once the system design is deemed acceptably safe, a safety concept can be synthesised. With the model-based approach, the safety concept may contain traceability among artefacts and may be taken as input for model-based safety management of the system in its entire development life cycle. Afterwards, all artefacts produced throughout the DECISIVE process can be integrated into the *System Assurance* process in a broader scope, in which such artefacts can be used to provide contextual and evidential information in a (presumably model-based) *Assurance Case* to argue the safety at the upmost system level.

DECISIVE is iterative as its name suggests, whenever there are changes to the system definition or system requirements, or when new hazards are identified, the DECISIVE process shall be repeated to determine the impacts of the changes. However, since steps that require heavy manual effort (Steps 3 and 4) can be automated, we argue that practitioners can focus more on planning and designing the system (Steps 1 and 2), and the whole design process is driven by automated safety analysis.

## IV. Tool Support

DECISIVE can be supported by model-based tools to fully exploit the benefit of automation brought by Model Based System Engineering (MBSE). In this section, we discuss our model-based tool support – Safety Analysis Management Environment (SAME).

In our previous work [4], SAME was designed to manage primarily Matlab/Simulink models in an automated manner. The automated failure injection algorithm and the automated execution of the Simulink models are organised in a workflow in the sense that *no* graphical user interface is provided for SAME. For SAME to be more generic, we proposed a simple metamodel, named Structured System Architecture Metamodel (SSAM), which uses simple blocks and relationships to represent system components, we also provided a proof-of-concept model transformation which is able to transform Simulink models to SSAM models and use graph algorithms to perform automated FMEA on the SSAM model.

In this paper, we revise SSAM and turn it into a comprehensive modelling language, so that it can be used to capture multiple aspects of the design process for safety critical systems (to address **REQ1** in Section I). We also design a facility for SSAM to trace to external, heterogeneous models (to address **REQ2** in Section I). We enhance SAME by providing a graphical model editor to better support DECISIVE (to address **REQ3** in Section I). In addition, we provide a comprehensive model-to-model transformation to demonstrate how Simulink models can be transformed into SSAM models with no information loss.

### A. Relevant Technologies

SAME leverages the following related technologies to support the automated safety analysis:

- *Eclipse Modelling Framework (EMF)*. EMF [17] is one of the most widely used modelling languages in the context of MBSE. EMF provides the *Ecore* modelling language which allows the rapid development of Domain Specific Languages (DSLs).
- *Eclipse Epsilon*. Epsilon [18] is an integrated model management platform which supports the automated management of models defined in arbitrary modelling technologies.
- *Eclipse Sirius*. Sirius [19] enables the users to create a graphical modelling workbench by leveraging Eclipse Modelling technologies (e.g. EMF). By defining *View-Points* in Sirius, users are able to create complex graphical modelling editors with complex functionalities.
- *Matlab/Simulink*. Matlab/Simulink provides a graphical block-based framework that enables the modelling, simulation and analysis of systems and supports model management operations like code generation and continuous model verification [20].

We make use of EMF to create a comprehensive *Structured System Architecture Metamodel* (SSAM), using which practitioners are able to create requirement packages, hazard logs and system designs on any level of abstraction. We then use Eclipse Sirius and SSAM to create a graphical model editor. In the editor, not only the users are able to create their system design models using SSAM, but they can also have a high

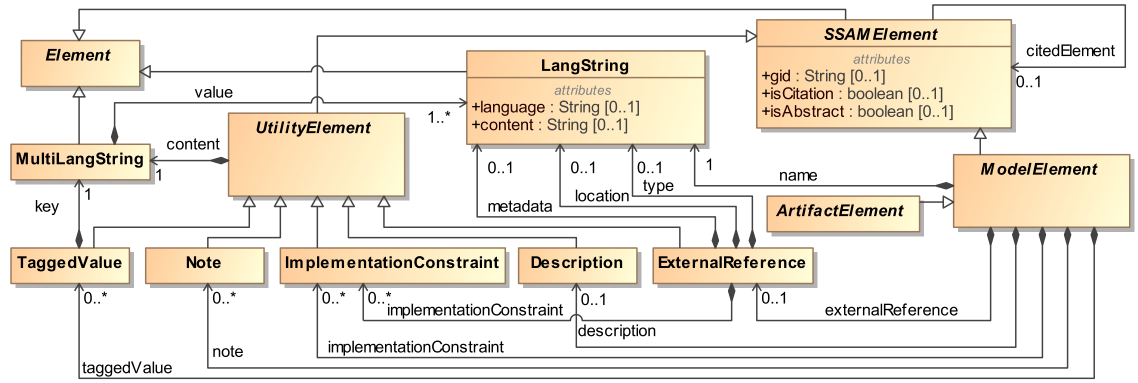


Fig. 2. Base Component of SSAM.

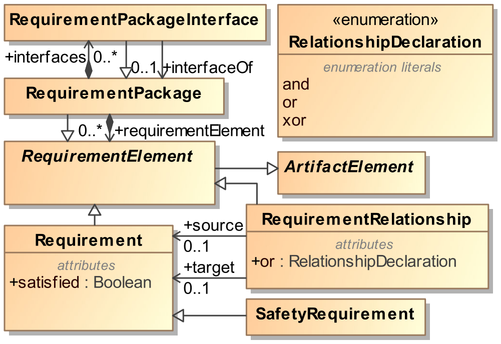


Fig. 3. Requirement Component of SSAM.

degree of traceability, both internal and external, as discussed in Section IV-B. For this purpose, Epsilon's EMC is exploited to create model drivers to access models defined in different frameworks. We then integrate the FMEA function for both SSAM and Simulink to SAME to support DECISIVE.

### B. SSAM and its tool support

With RQ1 and RQ2 in Section I in mind, we revise the Structured System Assurance Metamodel (SSAM). The aim is to design SSAM so that it becomes a generic and extensible metamodel that allows users to create models related to safety analysis (and transform their existing system design to SSAM), to be used for automated FMEA and to be integrated into the design process as well as the assurance process of safety-critical systems. The revision of SSAM focused on the following principles:

- **Extensibility**. Components of SSAM are extensible, SSAM contains a Base module, which allows the users to extend SSAM and adapt SSAM to their own needs with minimal efforts.
- **Modularity**. SSAM consists of a number of modules which can be used specifically for requirement engineering, hazard identification, system architecture design and failure modelling; they are organised into packages, so that they can exist independently, for reuse and interchange.
- **Traceability**. SSAM is designed to enable the traceability from SSAM models to (arbitrary) external and heterogeneous models (models defined in arbitrary technologies), so that a SSAM model can act as a federation model to integrate system information and be integrated into the process of SCSE.

*1) Base Module:* The base module of SSAM is shown in Figure 2. The base module is relatively complex since facilities are designed to promote extendability, modularity and traceability. The core SSAM element is *ModelElement*, where it has a *name* (of type *LangString*, which allows the users to specify a string as well as the language used), and a number of *UtilityElement*s, the important ones are:

- *ImplementationConstraint*, which allows the users to attach constraints (including machine-executable constraints) to the *ModelElement*;
- *ExternalReference*, which allows the *ModelElement* to refer to information that exists outside the SSAM model (e.g. to refer to a reliability model). In the *ExternalReference*, the users are able to specify: the *location*, *type* and the *metadata* of the external model (if it exists), and finally the (machine-executable) *implementationConstraint* which, when executed, are able to pull information from the external model.

A *ModelElement* is also able to "cite" another *ModelElement*, in the sense elements inside a SSAM model may have traceability to elements that may be organised in another package. With the above *UtilityElement*s, a *ModelElement* is able to provide multi-language support, as well as traceability to external models.

*2) Requirement Module:* The requirement module (that extends the Base module) is shown in Figure 3. Element *RequirementElement* is an abstract of requirement elements, including *Requirement*, *SafetyRquirement* and *RequirementRelationship*. *RequirementElement*s are organised in *RequirementPackage*s, which may have a number of *RequirementPackageInterface*, in the sense that requirements can be modular, reused and interchanged.

*3) Hazard Module:* The hazard module of SSAM is shown in Figure 4. Again, the hazard module extends the base module. Element *HazardElement* is the abstract for all hazard-

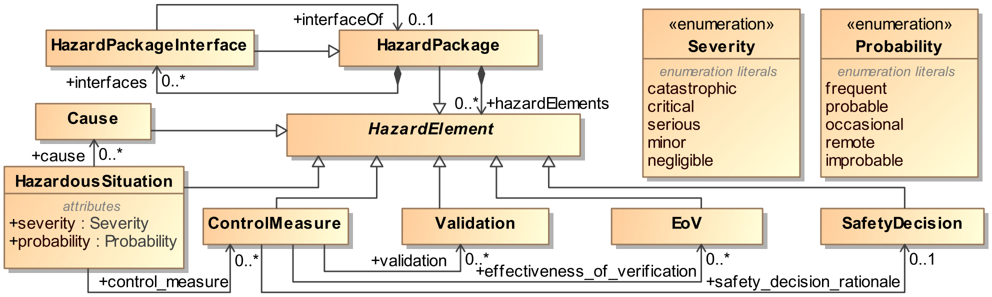


Fig. 4. Hazard Component of SSAM.

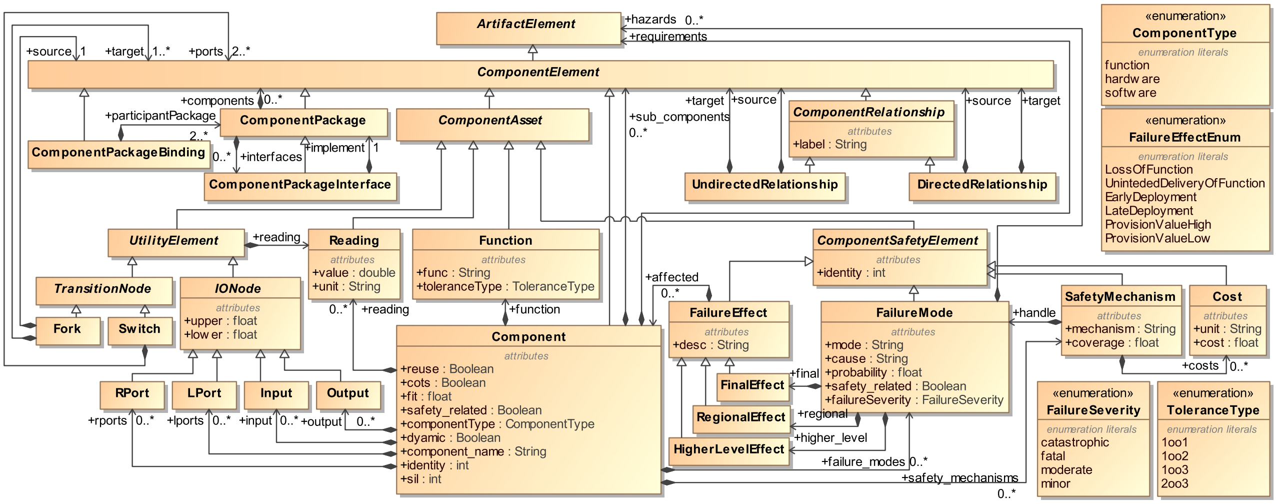


Fig. 5. Architecture Component of SSAM.

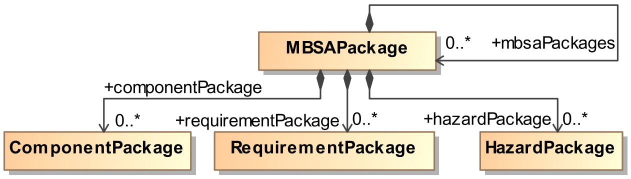


Fig. 6. MBSA Component of SSAM.

related elements, and *HazardElements* are organised in *HazardPackages*, which in turn may have a number of *HazardPackageInterfaces*. *HazardElements* allows the users to model:

- *HazardousSituation*, which may occur due to a *Cause*. A *HazardousSituation* may have a *severity* and a *probability*[3].
- *ControlMeasures* may be associated to *HazardousSituations* so that it can be mitigated to an acceptable safety level.
- A *ControlMeasure* may have a *SafetyDecision*, which provides the *safety decision rationale* to deploy such *ControlMeasure*.
- A *ControlMeasure* may also have a *Validation* plan and an *Effectiveness of Verification (EoV)* in the sense that the *ControlMeasure* is verified and validated so that it can mitigate the *HazardousSituation*.

*4) Architecture Module:* The architecture module of SSAM is shown in Figure 4. *ComponentElement* is the abstract for all architecture-related elements, and *ComponentElements* are organised in *ComponentPackages*, which in turn may have a number of *ComponentPackageInterfaces*. The *ComponentElement* allows the users to model:

- *Component*, which represents an atomic component in the user's systems. It may have a *FIT* (Failure-In-Time, $10^{-9}$ failures/hour). It also has a *safety integrity level*, which implies different levels of rigours for different application domains. The component may also have a *ComponentType*, which may be system, hardware or software. The *Component* may be *safety related*, indicating if any of the failure modes would cause a hazardous event. In addition, the *Component* may also be dynamic, which we will discuss briefly later.
- *ComponentRelationship*, which connects two *Components*. *Function*, which may have a *tolerance type*: 1oo1 (1 out of 1), 1oo2, 1oo3 or 2oo3.

[3] It is to be noted that SSAM does not adhere 100% to ISO 26262, to promote generality.

- *IONodes*, to capture the inputs and outputs of *Components*. They also may contain the values being passed and the lower and upper limits of the values.
- *FaiulreMode*, to capture the failure modes of a *Component*.
- *FailureEffect*, which allows the users to capture the effect of the failure.
- *FailureEffect* may be used to refer to another *Component* by using the "cite" reference as described in the base component.
- *SafetyMechanism*, to capture the safety mechanism that can be deployed on a *Component* to achieve diagnostic coverage.

*5) MBSA Module:* The Model-Based Systems Assurance (MBSA) module is shown in Figure 6. This package also extends the base package.

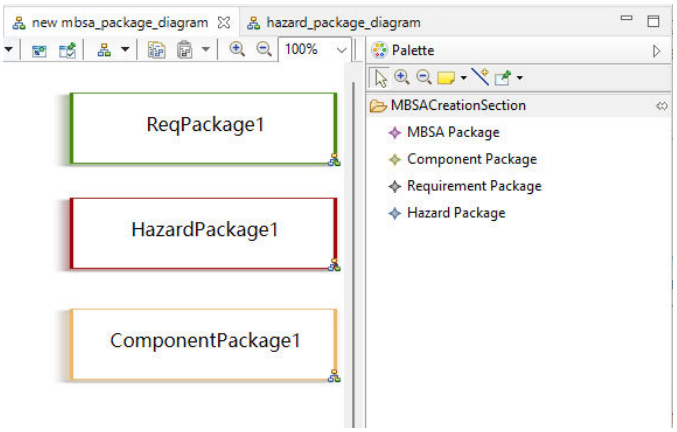
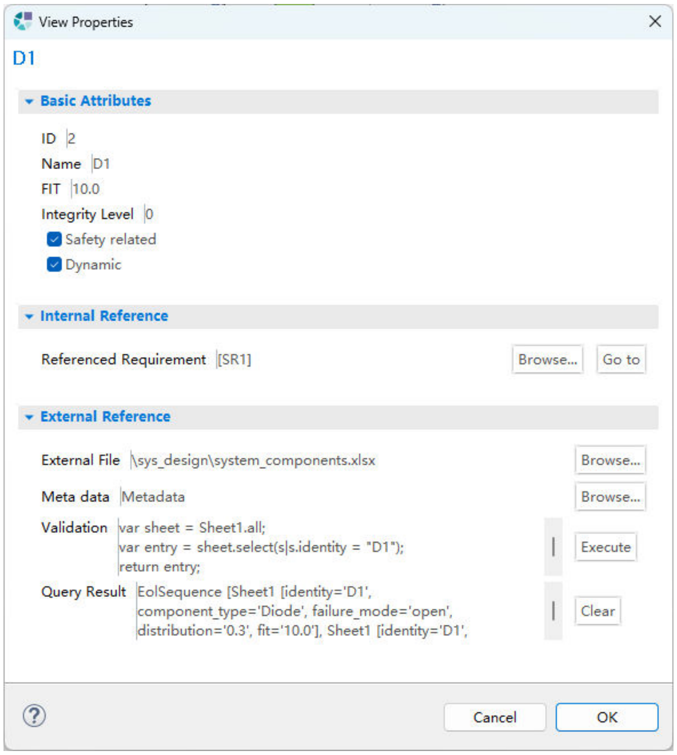


Fig. 7. MBSA editor for SSAM tool support.

*6) SAME with graphical editor:* To enable the users of SSAM to create content-rich models to design their systems or map their system designs to SSAM (in an automated manner), we developed a graphical modelling tool support for SSAM using Eclipse Sirius [19] and integrated it into SAME. The tool support provides hierarchical graphical editors, which allows the users to create different packages of SSAM discussed in previous sections. Figure 7 illustrates the graphical editor for *MBSAPackage*.

With the facilities in the Base component of SSAM, we are now able to extract and federate information across external and heterogeneous models. Figure 8 illustrates the property editor for a component named D1. The users may specify attributes of the component, such as the id, name, FIT, and integrity level of the Component. In the *Internal Reference* section, the users may also relate to the *Requirements* (which can be modelled in the Requirement editor) in the same SSAM model. In the *External Reference* section, the users may specify the location and the metadata of their own system design model/file, from which information can be extracted. In the *Validation* field, the users may specify the operation to be performed on their system design model. In the example in Figure 8, a script created using the Epsilon Object Language (EOL) is used to extract the information in the system model regarding component D1. In this way, information from external, heterogeneous models may be extracted in an automated manner, in order to perform automated safety analysis supported by SAME.



D1

Basic Attributes

ID 2
Name D1
FIT 10.0
Integrity Level 0
☑ Safety related
☑ Dynamic

Internal Reference

Referenced Requirement [SR1]

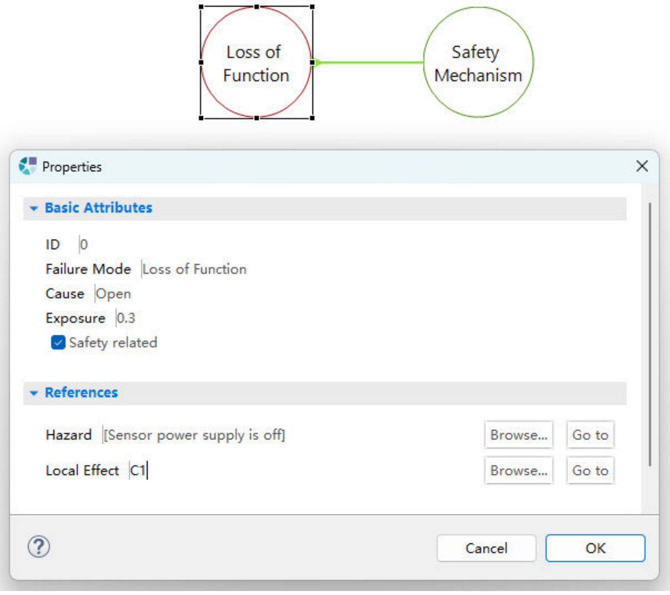
External Reference

External File \sys_design\system_components.xlsx
Meta data Metadata

Validation
```
var sheet = Sheet1.all;
var entry = sheet.select(s|s.identity = "D1");
return entry;
```

Query Result
```
EolSequence [Sheet1 [identity='D1',
component_type='Diode', failure_mode='open',
distribution='0.3', fit='10.0'], Sheet1 [identity='D1',
```

Fig. 8. Design editor for SSAM tool support.



Basic Attributes

ID 0
Failure Mode Loss of Function
Cause Open
Exposure 0.3
☑ Safety related

References

Hazard [Sensor power supply is off]
Local Effect C1

Fig. 9. Failure mode modelling.

The users may delve into a component and model the failure modes, as shown in Figure 9. Upon inspection of the failure modes, the users may model the Failure Mode type, its Cause and Exposure. In the *Reference* section, the users can associate Hazards (which can be modelled in the Hazard editor of SAME) that relate to the failure mode. In addition, the users may also refer to the affected components by this failure mode. If the users choose to do so, the automated FMEA would use this information and infer if the failure mode causes a single-point fault for the system. Again, in the *External Reference* section, the users may refer to an external model and extract failure mode data from their own files/models.

The hierarchical editors for Requirements modelling and Hazard modelling work in the same manner as the system design modelling editor, which we would not go into details. With the graphical editors, we support not only existing functions such as automated FMEA with Simulink models, we also support the in-detail modelling of requirements, hazards, as well as system design. We also provide the users with the flexibility to federate the information from their own models/files for automated FMEA.

Figure 12 in Section V shows the component system design editor, and the users may also model the IO nodes of the components, specifying their lower and upper limits. This is in place because the SSAM model not only can be used for static safety analysis, it can also be easily converted to a runtime monitoring algorithm[4]. In addition, we also implement an "import" function, which executes model-to-model transformation that transforms system architecture defined in arbitrary tools (e.g. Simulink) into SSAM in an automated manner. Within the system architecture editor, the users may invoke automated FMEA on any level (note that the *Component*s may be nested) and may choose to import a Simulink into the editor.

### C. Fitting into SCSE

The artefacts produced throughout the DECISIVE process can be naturally integrated into the System Assurance process – the FMEA results can be used as evidence to support the argument that the system design is acceptably safe, within an *assurance case*. Since FMEA is automated with the help of SAME, it automatically renders it possible for an assurance case to be automatically validated (provided that a model-based assurance case is in place).

In addition, we have to emphasise that DESICIVE shall be used alongside other guidelines in the development life cycle of SCSE. In particular, we find Clause 8 (Supporting Processes) of ISO 26262 to be most useful when it comes down to engineering management (e.g. safety requirement management, change management, configuration management, etc.), and highly recommend that practitioners that intend to use DECISIVE to follow such management guidelines.

To this extent, SSAM not only promotes the generality of our approach, but we argue that SSAM can also help with the integration of DECISIVE into SCSE. With SSAM and its graphical modelling tool in place, threes things are made possible: a) system designs defined in other modelling technologies can be transformed (in an automated manner) into SSAM models and the automated FMEA can be performed on such models; b) system design can also be done using SSAM and its graphical modelling tool, in which the information regarding the system can be extracted and federated from various sources; c) SSAM can be converted to a dynamic model, with runtime monitoring mechanism automated generated from SSAM (by declaring *Component*s as *dynamic*). Currently, the graphical tool for SSAM supports the extraction and federation of information defined using: Eclipse Modelling Framework, Matlab/Simulink, Cameo/MagicDraw Systems Modeller, XML, CSV, Excel, and other standards such as IFC.

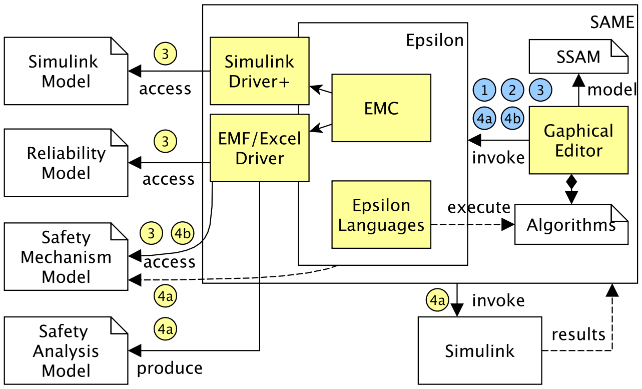


Fig. 10. Working process of SAME.

### D. Working Process

The working process of SAME in relation to the DECISIVE process is shown in Figure 10. The round shape with numbers in them rendered in yellow is the ones that relate to the DECISIVE steps for Simulink models, whereas the round shape rendered in blue is the ones for SSAM models. The blocks rendered in yellow are the main components of SAME. We first illustrate how SAME supports the automated FMEA on Simulink models and then we discuss how SSAM can be used for non-Simulink models.

*1) Support for Simulink:* For Simulink models, DECISIVE Steps 1 and 2 shall be performed independently, as they cannot be modelled using Simulink.

In DECISIVE Step 3, a separate Reliability Model needs to be provided so that SAME can keep track of the failure rate and the failure modes (and their probability distributions) of components. Since we leverage the extensible model drivers of Eclipse Epsilon, SAME supports Reliability Model defined in arbitrary modelling language. Reliability data can include, for each component: a) Failure-In-Time (FIT) data, which is the probability of failure for the component. Typically, 1 FIT is approximately $10^{-9}$ failures/hour; b) Failure modes and their probability distribution. Take the Resistor component type as an example, its failure modes can be *open* and *short* in an electrical system; associated with the failure modes are the probability distribution. For resistors, the probability distribution for failure modes (*open* and *short*) are 30% and 70%, respectively.

DECISIVE Step 4a is to evaluate system design, in SAME, we perform automated FMEA on the Simulink model with imported information through Step 3. The principle of the automated FMEA is based on failure injection [21], [22]. For our approach, the failure injection is performed automatically based on the failure modes of the components in the system

[4] By declaring them as *dynamic*, to be discussed in future work

design. The automated FME(D)A on Simulink models follow the steps below[5]:

1) **Initialise**: the values of the properties of all the components are recorded.
2) **Iterate Component**
   a) **Iterate Failure Modes**: For each component of the system, SAME looks for their failure modes in the *Reliability Model*. For a found failure mode, a failure is *injected* into the system.
   b) **Compare Results**: After a failure is injected, SAME invokes Simulink's simulate() function, and compares the readings of the voltage/current sensor with the readings before the failure injection. If the value differs by a threshold, SAME marks the failure mode of the chosen component as *safety-related*.
3) **Output**: After all components are iterated, SAME produces a *Component Safety Analysis Model*, this is what engineers are particularly familiar with – the FMEA result. In this step, it is also possible to compute the architecture metrics, such as the SPFM.

In DECISIVE Step 4b, if the system design does not meet the desired metrics (e.g. target ASIL), the design can be refined. For this purpose, safety mechanisms (e.g. watchdogs, redundancy) may be deployed on components. Safety mechanisms act as a means of rectification of failures for components when components fail, therefore improving the safety integrity level of the whole system.

However, deploying safety mechanisms for components manually would require the engineers to 1) look for safety-related components in the system; 2) find cost-effective safety mechanisms for the component; 3) deploy safety mechanisms and calculate system safety integrity level; 4) repeat previous steps if the target safety integrity level is not met. The above manual process is automated by SAME, which was discussed in our previous work.

*2) Support for Non-Simulink models:* As stated in Section I, there is a clear need for DECISIVE tool support for non-Simulink models. For this purpose, we make use of SSAM, which can be used in two ways to achieve the automated FMEA for non-Simulink models. First, the users may choose to transform their models (e.g. requirements, hazard logs, system designs) into SSAM models and perform the automated FMEA within SAME using the transformed model. To demonstrate this, we have implemented a transformation from the Simulink model to SSAM and tested its applicability on multiple Simulink models[6]. Alternatively, if the users lack tool support for requirements and hazard identification, they can create such SSAM models and map their own requirements/hazards into SSAM with the external traceability facility discussed in Section IV-B.

With SSAM in place, we naturally support DECISIVE Steps 1-3. One remark on Step 3 is that instead of importing reliability data from Excel, the users are free to model the failure modes of the *Components* using SAME, and are able to define the nature of the failure modes (e.g. loss of function). This is essential for the automated FMEA to be performed. For DECISIVE Step 4a, automated FMEA is performed following Algorithm 1. The algorithm is able to determine if a failure mode of the *Component* would render the end of a path (from the *Input* of the *Componeng* to the *Output* of the *Component*) unreachable (e.g. open for *Resistor*s), in order to determine safety-related failure modes.

In DECISIVE step 4b, the users may deploy *Safety Mechanism*s on their components and calculate the SPFM. Alternatively, the users may choose to extract information from their own Safety Mechanism Model and let SAME determine the solution for the target safety level and costs. If there are multiple options available, the users may be able to model a *cost* for each *Safety Mechanism* and provide weights on costs (if there are many) and ask SAME to search for the pareto front of viable solutions. The users can then choose the *Safety Mechanism*s that they see fit. Most importantly, with the *Safety Mechanism* in place, the changes in SSAM can be propagated back to the original model (e.g. back to the Simulink model).

**Algorithm 1:** Determining single point failures for SSAM models.

```
let component = the Component under analysis;
let paths = all possible paths containing nodes, between the
  input node and the output node of the Component;
for c in {all Component contained in component} do
    for fm in {all failure modes} do
        if fm is loss of function or similar nature then
            if c exists in all paths then
                mark fm as safety related;
            end
        end
        else
            provide a warning on fm;
        end
    end
    repeat this algorithm for c;
end
```

## V. Case Study

We now discuss a case study on which DECISIVE and SAME are applied. In this case study, we look into a simple power supply system for a proximity sensor, which is developed as a Safety Element out of Context (SEooC) as per ISO 26262. We first discuss how DECISIVE is followed by analysing a Simulink model, and then discuss the system model created using SSAM.

### A. Matlab

In DECISIVE step 1, we perform requirement elicitation and hazard identification. We select a top-level hazard *H1*: *The power supply fails unexpectedly* for consideration.

In DECISIVE step 2, we create a Simulink model based on the requirements and identified hazards, part of which is shown in Figure 11. In the system, *DC1* is a 5V direct current power source, *D1* is a diode, *L1* is an inductance, *C1* and *C2* are capacitors, *GND1* is a ground reference, *MC1* is a microcontroller, *CS1* is a current sensor. All other blocks are related

[5] For a more detailed description, please refer to our previous work [4]

[6] https://github.com/wrwei/DECISIVE/blob/main/org.eclipse.epsilon.simulink2ssam/test/simulink2dt.eol

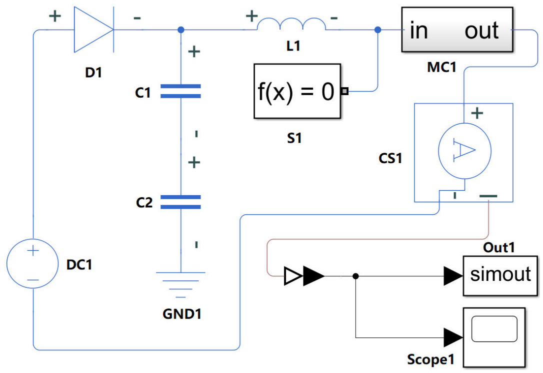


Fig. 11. A segment of the sensor power supply system designed using Simulink.

to simulation, *S1* is a solver configuration for the simulation, *Scope1* displays the signal coming out of *CS1*, and *Out1* is used to write the output of *CS1* into Simulink's workspace. We also specify that our safety requirement for hazard *H1* shall at least have a safety integrity level of ASIL-B (as per ISO 26262).

TABLE II
EXAMPLE COMPONENT RELIABILITY MODEL.

| Component | FIT | Failure_Mode | Distribution |
|---|---|---|---|
| Diode | 10 | Open | 30% |
| | | Short | 70% |
| Capacitor | 2 | Open | 30% |
| | | Short | 70% |
| Inductor | 15 | Open | 30% |
| | | Short | 70% |
| MC | 300 | RAM Failure | 100% |

In DECISIVE step 3, information from the *Reliability Model* is aggregated into the model. In our case study, we use one Excel spreadsheet containing reliability data, as shown in Table II. In the reliability model, the required information includes: 1) Component type; 2) FIT (failure-in-time, in $10^{-}9$ failures/hour); 3) Failure modes of components; 4) Probability distribution of the failure modes for components.

In DECISIVE step 4a, automated FMEA is performed following the steps discussed in Section IV-D. For our chosen top-level hazard (*H1*), we are interested in correct readings at CS1, and assume that DC1 is stable (i.e. over-voltage and under-voltage are not considered). Therefore, safety-related components are *D1*, *L1* and *MC1*, and the automated FMEA results produced by SAME supports this intuition (columns *Component*, *FIT*, *Safety_Related*, *Failure_Mode* in Table IV). In this step, we can also ask SAME to calculate the Single Point Fault Metric (SPFM), based on the following equation (where SR_HW denotes 'safety-related hardware', $\lambda_{\text{SPF}}$ denotes the failure rate of a component's failure mode that causes a single point fault, $\lambda$ denotes the total failure rate of a component):

$$\text{SPFM} = 1 - \frac{\sum_{\text{SR\_HW}}(\lambda_{\text{SPF}})}{\sum_{\text{SR\_HW}}\lambda} \qquad (1)$$

In our example, the calculated SPFM is 5.38%, much less than the required target value for ASIL-B (≥90%) in ISO 26262.

TABLE III
EXAMPLE SAFETY MECHANISM MODEL.

| Component | Failure_Mode | Safety_Mechanism | Cov. | Cost(hrs) |
|---|---|---|---|---|
| MCU | RAM Failure | ECC | 99% | 2.0 |

In DECISIVE step 4b, we import a *Safety Mechanism Model* to refine the existing design. The *Safety Mechanism Model* shall contain: 1) Component type; 2) Component failure mode; 3) Safety Mechanism for such failure mode; 4) Coverage of such safety mechanism for the failure mode. In our example, we store such information in another Excel spreadsheet, as shown in Table III, which contains only one entry: the safety mechanism for MCUs (Micro Controller Units such as MC1 in Figure 11). The type of safety mechanism that can be deployed on the MCU is Error Checking & Correction (ECC), which can cover 99% of RAM failures for MCUs. If the user chooses to deploy such a safety mechanism, a FMEDA result may be generated by SAME, as shown in Table IV. In our example, the analysis shows that safety-related failure modes of the components are: *D1*'s *open* failure mode, *L1*'s *open* failure mode, and *MC1*'s *RAM Failure* mode. Since we deployed ECC on *MC1*, and its diagnostic coverage is 99%, the FMEDA shows that *Single Point Failure Rate* for *MC1* is dropped to 3 FIT ($10^{-}9$ failures/hour).

We can now re-calculate SPFM with the refined architecture using Equation 1. This time it yields 96.77%, and achieves ASIL-B as per ISO 26262. We can then invoke a proper change management process (as per ISO 26262, but other standards may also be followed) to deploy the ECC on *MC1*. This also means that another iteration of DECISIVE shall be followed.

### *B. SAME and SSAM*

We repeat the above process in Matlab using SSAM with the SAME support.

In DECISIVE step 1, we create a *Requirement Package*, in which system requirements are recorded. Again, we select the top-level hazard *H1: The power supply fails unexpectedly*.

In DECISIVE step 2, we create the system design using the graphical editor provided by SAME, as shown in Figure 12, which is a 1-to-1 mapping to Figure 11. Again, the safety integrity level for *H1* is set to ASIL-B. Note that the SSAM model in this case is used as a mapping model, which is able to extract information only needed for FME(D)A from the system models.

In DECISIVE step 3, the information from the *Reliability Model* is aggregated into the SSAM model, using the external model traceability facility provided by SSAM discussed in Section IV-B6.

In DECISIVE step 4a, the automated FMEA is performed, as described in Section IV-D2. We provide a context menu entry which performs automated FMEA on SSAM models, to determine safety-related components.

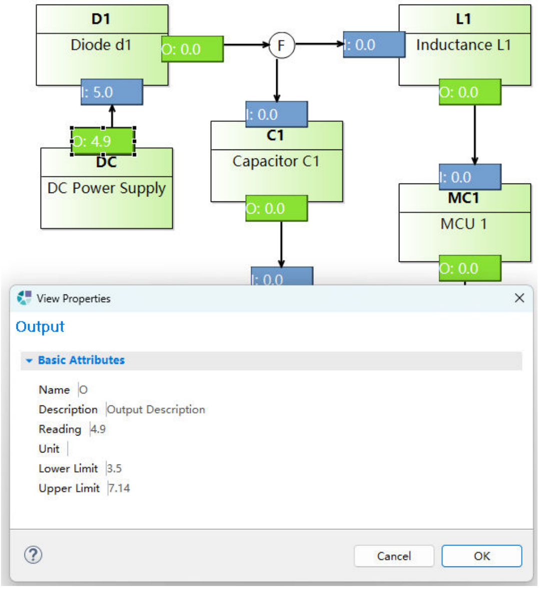
In DECISIVE step 4b, we model a safety mechanism (ECC as discussed in Section V-A) directly on *MC1*, and model the coverage by ECC, we are able to achieve the same SPFM of 96.77%, and the automated calculation of ASIL for the model yields ASIL-B.

TABLE IV
GENERATED FAILURE MODE AND EFFECT DIAGNOSTIC ANALYSIS (FMEDA).

| Component | FIT | Safety_Related | Failure_Mode | Distribution | Safety_Mechanism | SM_Coverage | Single_Point_Failure_Rate |
|---|---|---|---|---|---|---|---|
| D1 | 10 | Yes | Open | 30% | No SM | | 3 FIT |
| | | No | Short | 70% | | | |
| L1 | 15 | Yes | Open | 30% | No SM | | 4.5 FIT |
| | | No | Short | 70% | | | |
| MC1 | 300 | Yes | RAM Failure | 100% | ECC | 99% | 3 FIT |

### C. Integration to Assurance Case

With the current design, we integrate into the broader *System Assurance* process. For this purpose, we obtained a model-based assurance case management tool (named ACME), which is briefly discussed in [2] and created an assurance case module for our power supply system. In ACME, the users are able to create an *Artifact* class instance from the Structured Assurance Case Metamodel [2], by using which the information from external artefacts (such as Excel spreadsheets) can be extracted. In our example, we trace to our generated FMEDA result and store a query to calculate SPFM in the assurance case model, to check whether the SPFM meets the target ASIL value. In this case, when our design changes, it is reflected in the FMEDA result, which can in turn be automatically checked by ACME (by executing the query). In this way, it is possible to automate the evaluation of assurance cases.



Name O
Description Output Description
Reading 4.9
Unit
Lower Limit 3.5
Upper Limit 7.14

Fig. 12. Automated FMEA in SSAM with SAME support.

## VI. EVALUATION

The aim of our evaluation is to answer the following research questions:

**RQ1** (Correctness): Does DECISIVE, with model-based tool support SAME, produce correct FMEA results in an automated manner?

**RQ2** (Coverage): Does SAME cover all of Simulink's system design blocks? In its broader application, does SAME supported by the SSAM modelling language provide sufficient coverage to map system design information across application domains?

**RQ3** (Efficiency): Does DECISIVE, supported with SAME, increase the efficiency of developers for the design of safety-critical systems/components?

**RQ4** (Scalability): Does DECISIVE (supported with SAME), support the design of complex safety critical systems with a large number of model elements?

The evaluation set-up is as follows:

- **Evaluation Subjects**: we have chosen two systems[7], including a sensor power supply system (denoted as *System A*) with 102 elements in the design, and the main control unit (hardware and software) of an Autonomous Underwater Vehicle (AUV) (denoted as *System B*) with 230 elements in the design.
- **Participants**: two safety professionals (Participants A and B) with relatively the same level of expertise are asked to participate in the evaluation.
- **Platform**: we use SAME with JDK 1.8.0_301, Matlab 2018b and Epsilon 2.3.

### A. RQ1: Correctness

For correctness, we ask Participant *A* to perform FMEA manually on the evaluation subjects, and Participant *B* to use SAME[8] with automated support, and we compare the results yielded for both systems. For System A, we observe a 1.5% difference between the FMEA results generated by participants *A* and *B*. For System B, we observe a 2.67% difference between the results. This is typically due to the fact that FMEA is a highly subjective analysis technique, and the opinions on the effects of failing components may differ from the analysis algorithm in SAME. However, we also observe that the safety-related components for both System A and System B are all identified correctly by both participants, this provides us the confidence that SAME is able to produce correct analysis results.

### B. RQ2: Coverage

For coverage, we first evaluate the coverage for Simulink. Currently SAME supports the analysis of electrical systems built using Simulink's Simscape Foundation Library, which focuses on analogue circuit design, for proof-of-concept. There are some electrical elements that are not covered in SAME (e.g. complex microcontroller units), but work-arounds are devised for such elements – for elements not covered in

[7] Which we are not at liberty to disclose due to intellectual properties.
[8] A brief training on how to use SAME is provided

Simulink's Simscape, we create subsystems in Simulink and annotate them to be the desired elements. With the work-arounds solution, we are able to cover 100% of the evaluation subjects. Whilst it is possible to cover elements outside the Simscape library, we need to incorporate algorithms for identifying components and their failure modes in SAME for such components, which is an ongoing work. Then, we evaluate the coverage for SSAM for its ability to map system design across domains, we are able to map the conceptual function blocks for both System A and B, and we are also able to map both software and hardware blocks for both Systems A and B, this provides us with the confidence that DECISIVE and SAM are able to achieve a very high degree of coverage for both Simulink's Simscape library using Simulink, as well as all other types of system design using SSAM.

TABLE V
COMPARATIVE EXPERIMENT FOR EFFICIENCY EVALUATION.

| System | Participant | Time spent (minutes) | No. Iterations |
|---|---|---|---|
| A | A(Man.) | 505 | 5 |
| A | B(Auto.) | 62 | 2 |
| B | A(Man.) | 1143 | 6 |
| B | B(Auto.) | 105 | 3 |
| A | A(Auto.) | 57 | 6 |
| A | B(Man.) | 497 | 3 |
| B | A(Auto.) | 110 | 4 |
| B | B(Man.) | 1166 | 2 |

### C. RQ3: Efficiency

For efficiency, we ask participants *A* and *B* to follow the DECISIVE process to design both System A and System B in two settings. In the first setting, *Participant A* takes a completely manual process: to aggregate reliability data into the system, perform FMEA, search and deploy appropriate safety mechanisms, and come up with a design with a target safety level (in the case study, we aim at achieving ASIL-B, the requirements of which are described in ISO 26262) for both systems. *Participant B* follows DECISIVE with support of SAME, we provide a reliability model and a safety mechanism model, *Participant B* is asked to find a design with the same safety level for both systems. In the second setting, we ask *Participant A* to follow DECISIVE with automation support, and *Participant B* to take a manual process on both systems.

We compare the time it takes for both participants to complete their tasks, as shown in Table V. In the first setting, for *System A*, *Participant A* takes approximately 505 minutes to complete the system design (with three iterations), in which most of the time is spent on FMEA, deployment of safety mechanisms and change management. In contrast, *Participant B* takes 62 minutes (with six iterations), in which most of the time is spent on change management. For *System B*, *Participant A* takes approximately 19.05 hours (with two iterations), with a time distribution similar to *System A*. In contrast, *Participant B* takes 105 minutes (with five iterations). In the second setting, for *System A*, *Participant A* takes approximately 57 minutes (with 6 iterations), and *Participant B* takes 110 minutes (with 4 iterations). For *System B*, *Participant A* takes approximately 8.28 hours (with 3 iterations), and *Participant B* takes 19.43 hours (with 2 iterations).

Both participants state that the DECISIVE approach provides helpful guidance in designing the system. In this experiment, we observe approximately a tenfold increase in efficiency. We also observe that the complexity of the system is an affecting factor on how many iterations of the design process can be taken for manual efforts. However, this does not seem to affect the process with automation support.

### D. Scalability

Our last evaluation is on the scalability of SAME, although SAME is the secondary contribution of our work, we report our findings on the scalability of the tools. Our evaluation was performed on the premise that the majority of the models used in our development process are EMF models, with a mixture of Simulink models, Excel spreadsheets, formal models and JSON models. To evaluate the scalability of SAME, we selected 5 data sets, as shown in Table VI. It is to be noted that the in our end result systems, the maximum number of model elements we have in our collection of models was 5689 (Set3). We made duplicates of our models and put them together to form Set4 and Set5 to evaluate the scalability of SAME in different orders of magnitudes. We found that SAME suffered from scalability issues from Set4 and would not load Set5 due to memory overflow. This is typically caused by the fact that SAME needs to load EMF models in their entirety before any queries can be performed on them, which is an existing issue discovered in various studies [23]–[25]. However, this shall not be counted as a scalability problem with SAME, but rather a scalability problem with EMF. Currently, there are a number of frameworks that can be adopted directly to solve the scalability problem with EMF [23], [26]. Whilst we aim to improve SAME by addressing the scalability problem in EMF, we argue that SAME is scalable as long as the access (i.e. reading and storing) mechanism for the models is scalable.

TABLE VI
NORMALISED EFFICIENCY EXPERIMENT.

| Model | No. of Model Elements | Time taken for Evaluation(sec) |
|---|---|---|
| Set0 | 109 | 0.1 |
| Set1 | 269 | 0.2 |
| Set2 | 1369 | 0.8 |
| Set3 | 5689 | 4.1 |
| Set4 | 5689000 | 48.3 |
| Set5 | 568990000 | N/A |

## VII. RELATED WORK

A number of research look into the model-based approach for the safety analysis of safety-critical systems. In [27], a model-based system analysis approach using assume-guarantee compositional reasoning on AADL models is discussed. In this work, an extension to the AADL language is discussed and formal methods for analysis are presented. We would like to point out that, AADL models can also be transformed to SSAM and our approach can also be applied. In contrast to complete formal analysis, our approach is also applicable for runtime scenarios, where qualitative FMEA is important to determine if CPS can collaborate at runtime. In

[14], a model-based approach named HiP-HOPS (Hierarchically Performed Hazard Origin & Propagation Studies) and its supporting tool are presented, using which model-based Fault Tree Analysis can be generated. In addition, FMEA tables can be generated from the fault trees. In comparison, our generation of FMEA does not rely on the existence of a fault tree. In addition, SAME enables the automated transformation of system design models to SSAM not only from Simulink, but also other tools. In [28], the authors propose AltaRica, a family of event-based modelling languages, Altarica 3.0 focuses on deterministic or stochastic delays associated with events. AltaRica can be used for sequence diagram generation, markov chain generation, model checking, reliability allocation, stochastic simulator, as well as fault trees. Whilst AltaRica has a focus on probabilistic modelling and simulation, DECISIVE focuses more on the interoperability of system models among tools and the automated FMEA on such models. In [13], a domain-specific modelling framework (EAST-ADL) is presented, and evaluated using HiP-HOPS. In comparison, we provide a more comprehensive methodology for designing critical systems, which is driven by automated FMEA. In [29], the functional design phase, using SysML, is combined with commonly used reliability techniques (i.e. FMEA and construction of AltaRica Data-Flow models). In comparison, DECISIVE is applicable to any system design models not limited to SysML and considers the automated deployment of safety mechanisms.

## VIII. Summary and Future Work

In this paper, we answer to identified requirements from requests on tool support sent to us since our previous work. We revised the Structured System Assurance Metamodel (SSAM) and designed it to support modularity, extensibility and traceability. Based on the changes to SSAM, we also created a graphical editor for SAME, and provide hierarchical views for each aspect in the system design process. In our answer to REQ1, the revised SSAM covers requirement elicitation, hazard identification and system design, and enables the traceability inside SSAM models. In our answer to REQ2, the *Base* component of SSAM provides the facility for the traceability to external heterogeneous models, which is supported by graphical editors in SAME. In our answer to REQ3, the graphical user interface in SAME proves very useful in the modelling of systems for non-Simulink models.

In this paper, we also performed an in-depth evaluation of correctness, coverage, efficiency and scalability of the DECISIVE approach with SAME tool support. We find that there is an observable efficiency improvement due to automation, whilst maintaining a satisfactory level of correctness, coverage and scalability. We also discover that the DECISIVE process provides useful guidance on designing safety-critical systems. We also briefly tapped into how the DECISIVE methodology, as well as the models produced by SAME, can fit seamlessly into the process of Safety Critical Systems Engineering (SCSE). Throughout the modelling process of SSAM, we realised that the SSAM can also be converted into a dynamic model. By declaring a *Component* as *dynamic*, it is possible to generate Java facilities to receive runtime data for the component in a real time manner. We will investigate more in our future work.

In the future, we plan to: 1) enhance SAME to include the model-based support for Fault Tree Analysis (FTA) and how FTA and FMEA can be federated for quantitative system safety analysis; 2) provide support for system design in widely adopted languages such as SysML and AADL; 3) integrate a scalable model indexing (or model storage) framework into SAME to achieve scalability when accessing models defined in EMF, UML, and other technologies to achieve full scalability for SAME; and 4) investigate the automated generation of runtime monitoring framework for RAS with open and adaptive nature.

## References

[1] Timothy Patrick Kelly. *Arguing safety: a systematic approach to managing safety cases*. PhD thesis, University of York York, UK, 1999.

[2] Ran Wei, Tim P Kelly, Xiaotian Dai, Shuai Zhao, and Richard Hawkins. Model based system assurance using the structured assurance case metamodel. *Journal of Systems and Software*, 2019.

[3] Zhe Jiang, Shuai Zhao, Ran Wei, Dawei Yang, Richard Paterson, Nan Guan, Yan Zhuang, and Neil C Audsley. Bridging the pragmatic gaps for mixed-criticality systems in the automotive industry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(4):1116–1129, 2021.

[4] Ran Wei, Zhe Jiang, Xiaoran Guo, Haitao Mei, Athanasios Zolotas, and Tim Kelly. Designing critical systems with iterative automated safety analysis. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 181–186, 2022.

[5] Zhe Jiang, Ran Wei, Pan Dong, Yan Zhuang, Neil C Audsley, and Ian Gray. Bluevisor: Time-predictable hardware hypervisor for many-core embedded systems. *IEEE Transactions on Computers*, 71(9):2205–2218, 2021.

[6] SAE International. *Certification Considerations for Highly-integrated Or Complex Aircraft Systems*. SAE International, 1996.

[7] ISO. *ISO 26262: Road Vehicles - Functional Safety.* 2018.

[8] Mario Trapp, Daniel Schneider, and Peter Liggesmeyer. A safety roadmap to cyber-physical systems. In *Perspectives on the future of software engineering*, pages 81–94. Springer, 2013.

[9] David Norfolk. Ptc integrity modeler... a standards-based tool for systems and software engineering. *InDetail. Bloor. In: https://www. ptc. com/-/media/Files/PDFs/ALM/Integrity/PTC-Integrity-Modeler-Bloor-InDetail. pdf*, 2015.

[10] Halina Tańska. Enterprise architect and magic draw uml–comparing the abilities of case tools. *PUBLISHER UWM OLSZTYN 2009*, page 181, 2009.

[11] IEC. *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems.* 2010.

[12] Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. Fault tree analysis, methods, and applications, a review. *IEEE transactions on reliability*, 34(3):194–203, 1985.

[13] DeJiu Chen, Nidhal Mahmud, Martin Walker, Lei Feng, Henrik Lönn, and Yiannis Papadopoulos. Systems modeling with east-adl for fault tree analysis through hip-hops. *IFAC Proceedings Volumes*, 46(22):91–96, 2013.

[14] Yiannis Papadopoulos, Martin Walker, David Parker, Erich Rüde, Rainer Hamann, Andreas Uhlig, Uwe Grätz, and Rune Lien. Engineering failure analysis and design optimisation with hip-hops. *Engineering Failure Analysis*, 18(2):590–608, 2011.

[15] Stephen Jacklin. Certification of safety-critical software under do-178c and do-278a. In *Infotech@ Aerospace 2012*, page 2473. 2012.

[16] Saddek Bensalem, Chih-Hong Cheng, Wei Huang, Xiaowei Huang, Changshun Wu, and Xingyu Zhao. What, indeed, is an achievable provable guarantee for learning-enabled safety critical systems. *arXiv preprint arXiv:2307.11784*, 2023.

[17] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework.* Pearson Education, 2008.

[18] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. Eclipse development tools for epsilon. In *Eclipse Summit Europe, Eclipse Modeling Symposium*, volume 20062, page 200, 2006.

[19] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of dsm graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238. IEEE, 2014.

[20] Beatriz A Sanchez, Athanasios Zolotas, Horacio Hoyos Rodriguez, Dimitris Kolovos, Richard F Paige, et al. Runtime translation of ocl-like statements on simulink models: Expanding domains and optimising queries. *Software and Systems Modeling*, 2021.

[21] Anjali Joshi and Mats PE Heimdahl. Model-based safety analysis of simulink models using scade design verifier. In *International conference on computer safety, reliability, and security*, 2005.

[22] O Lisagor, JA McDermid, and DJ Pumfrey. Towards a practicable process for automated safety analysis. In *24th International system safety conference*. Citeseer, 2006.

[23] Konstantinos Barmpis and Dimitris Kolovos. Hawk: Towards a scalable model indexing architecture. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, pages 1–9, 2013.

[24] Ran Wei, Dimitrios S Kolovos, Antonio Garcia-Dominguez, Konstantinos Barmpis, and Richard F Paige. Partial loading of xmi models. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 329–339, 2016.

[25] Seyyed M Shah, Ran Wei, Dimitrios S Kolovos, Louis M Rose, Richard F Paige, and Konstantinos Barmpis. A framework to benchmark nosql data stores for large-scale model persistence. In *International Conference on Model Driven Engineering Languages and Systems*, pages 586–601. Springer, 2014.

[26] Antonio Garcia Dominguez, Konstantinos Barmpis, Ran Wei, and Richard Paige. Stress testing results for relational and graph-based stores. 2016.

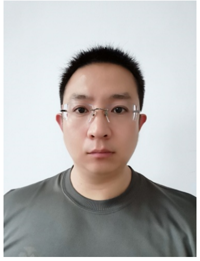
[27] Danielle Stewart, Michael W Whalen, Darren Cofer, and Mats PE Heimdahl. Architectural modeling and analysis for safety engineering. In *International Symposium on Model-Based Safety and Assessment*, pages 97–111. Springer, 2017.

[28] Tatiana Prosvirnova. *AltaRica 3.0: a model-based approach for safety analyses*. PhD thesis, Ecole Polytechnique, 2014.

[29] Pierre David, Vincent Idasiak, and Frederic Kratz. Reliability study of complex physical systems using sysml. *Reliability Engineering & System Safety*, 95(4):431–450, 2010.

**Dr Ran Wei** (Member, IEEE) is a Research Assistant Professor at the Department of Engineering, University of Cambridge. His research interests include Model Driven Engineering, High Integrity Systems Engineering, Model Based System Assurance and Model Based Digital Twin Generation and Maintenance. He can be reached at: rw741@cam.ac.uk.

**Prof Zhe Jiang** (Member, IEEE) is currently a Professor at Southeast University, China. He is broadly interested in computer architecture, micro-architecture, and design automation for emerging computing systems, with a particular focus on improving functional safety, security, reliability, and timing-predictability of automotive, cloud and embedded computing systems. He can be reached at: 101013615@seu.edu.cn.



**Dr Xiaoran Guo** is with the Specialised Services Research Department of China. His research interests include Model Based System Verification and Validation, Metrology and Instrumentation, and Safety Critical Systems Engineering. He can be reached at: vip850522@163.com.

**Ruizhe Yang** is a MSc student in Artificial Intelligence of Dalian University of Technology (DUT), China. His current research interests include model driven engineering, High Integrity Systems Engineering and Model-Based Digital Twin. He can be reached at: ruizheyang@mail.dlut.edu.cn.

**Dr Haitao Mei** received his Ph.D. degree from the Real-Time Systems Research Group at the University of York in 2018. His research interests are in real-time operating systems and programming languages, Big Data and real-time stream processing.

**Dr Athanasios Zolotas** is a Senior Lecturer in Software Engineering at the School of Computer Science and Mathematics at Liverpool John Moores University, UK. Athanasios received his EngD in Large-Scale Complex IT Systems from the University of York in 2017. His research interests are in model-driven engineering, big data analytics, safety critical systems and requirements engineering, while he is collaborating with leading companies in the aerospace and automotive domain such as Rolls-Royce and Volkswagen.

**Prof Rev. Tim Kelly** is a honorary professor of the High-Integrity Systems Engineering research group of the University of York. His research interests include system assurance cases, safety-critical systems engineering, and model based system assurance. He was one of the founding members of the GSN (Goal Structuring Notation) standard and is one of the leading contributors of SACM (Structured Assurance Case Metamodel).