An Open Modelling Approach for Availability and Reliability of Systems

Jussi-Pekka Penttinen[a,b,*], Arto Niemi[a,c], Johannes Gutleber[c], Kari T. Koskinen[a], Eric Coatanéa[a], Jouko Laitinen[a]

[a] *Tampere University of Technology, PO Box 527, FI-33101 Tampere, Finland*
[b] *Ramentor Oy, Hermiankatu 8 D FI-33720 Tampere, Finland*
[c] *CERN - The European Organization for Nuclear Research, CH-1211 Geneva, Switzerland*

Abstract

This paper introduces an Open Modelling approach for Availability and Reliability of Systems (OpenMARS), which is developed for risk and performance assessment of large and complex systems with dynamic behaviours. The approach allows for combining the most common risk assessment and operation modelling techniques. This ensures a high degree of freedom for the modeller to accurately describe the system without limitations imposed by an individual technique. OpenMARS uses a platform-independent tabular format to define the used modelling technique, to create the model structure, and to assign the parameter values. We developed the format to enable a straightforward manual model definition while maintaining database compatibility. This paper also presents our calculation engine for stochastic simulation-based analysis of OpenMARS models. Our intention is to use this approach as a basis for new software. We demonstrate the feasibility of OpenMARS with an example of a multi-state production process that is subject to failures. The example creates a comprehensive system model by combining interconnected failure logic, operation phase, and production function models. We believe that the advanced features of OpenMARS have wide ranging applications for analysis of reliability, performance, and energy efficiency of complex industrial processes.

*Corresponding author
Email addresses: jussi-pekka.penttinen@tut.fi (Jussi-Pekka Penttinen), jussi-pekka.penttinen@ramentor.com (Jussi-Pekka Penttinen)

Keywords: Risk assessment technique; System performance; Complex systems; Dynamic modelling; Fault tree analysis; Model data format

1. Introduction

Reliability, availability, and operational performance are integral factors to consider in system design and management. They provide an essential amount of information for risk-informed decision-making. Risk and performance analyses are used, for example, to compare design alternatives, to estimate the return of investment time, and to optimize maintenance of the system. Today's complex systems require sophisticated methods to analyse the effect of failures on overall system performance. This is especially true when considering dynamic interdependencies between failures, production and maintenance.

Modern reliability engineering still confronts challenges that relate to the representation of the system and quantification of the model [1]. The traditional methods are not always flexible enough to include all the needed details, which can lead to unrealistic simplifications. For example, fault tree analysis is one of the most prominent techniques in risk assessment, but without extensions it lacks the power to express essential dependability patterns, i.e. spare management, different operational modes, and dependent events [2]. Modern reliability engineering has tried to answer these challenges with model-based dependability assessment (MBDA) [3] and simulation-based analyses [4]. A related work section gives a brief overview of the latest techniques and compares OpenMARS to them.

We have also noted the limitations of the standard tools during our decade-long experience developing ELMAS [5] software. Various demanding use cases prompted us to add new advanced features for the modelling of complex relationships and dynamic operation changes [6, 7, 8, 9, 10]. The use of ELMAS for availability modelling of future circular colliders [11] in the European Organization for Nuclear Research (CERN) highlighted the challenge that the addition of customized domain-specific features required programming skills from the

modeller. This motivated CERN to launch a research and development (R&D) project in collaboration with Tampere University of Technology and Ramentor. The project goal was to create a new improved approach, which permits the inclusion of customised features with minimal programming needs during the modelling phase.

This paper introduces the result of the R&D project: an Open Modelling approach for Availability and Reliability of Systems (OpenMARS) [12]. The approach permits model definition with any of the most common risk assessment modelling techniques [13], such as fault tree analysis (FTA), reliability block diagram (RBD), Markov analysis, failure mode and effects analysis (FMEA), and Petri net. With OpenMARS a modeller can combine the most suitable techniques to accurately include all the details that affect the system behaviour. A model can consist of several sub-models that interact with each other. For example, a phase change within a dynamic operation model can update the failure rates of other models. OpenMARS also includes techniques to model mathematical and logic functions. With function models, the calculation of application-specific key performance indicators (KPIs), such as overall equipment effectiveness (OEE) [14], can be attached to the system dependability model. To make sure that OpenMARS is always applicable, we allow the modeller to extend the built-in features of the techniques for special needs. The approach is scalable, and open to support and combine additional modelling techniques.

This paper also introduces our model data format. A clear format is required to define models, to store the model information and to transfer the models between the software. As early as the 1970s, the developers of a FTA computer program had a high priority for developing an input format that was as simple as possible [15]. Recently, XML-based [16] model exchange formats have been developed for sharing of FTA [17] and Petri net [18] models between different software applications. Our format is platform-independent, human-readable and tabular. We chose tables as the basis of our format, because in our opinion an average modeller can understand tabular format easier than any

markup language. Tables are also a natural way to store models. For example, spreadsheet software and relational databases use tables to present data.

One of our key requirements is that the format is open, documented, and non-proprietary, which ensures long-term accessibility of the data. We also aimed at enabling easy manual definition of vast models and the model creation in a collaborative fashion. The manual model definition was inspired by a reliability study that analysed a large system with repetitive structures [19]. The collaborative development requires the ability to identify models [20], which is handled by including separate model information and change tables in the OpenMARS specification [12]. However, this will not be further expounded upon as this paper focuses on modelling.

We demonstrate the possibilities of the OpenMARS with an example case. It is a simplified reliability model of a multi-state industrial process, which can be used as a basis for individual cases. The example uses a fault tree model for system failures, a Markov model for processing phase changes, and a function model for production calculation. Our particle collider availability study [11] uses similar approach. It required including special modelling of production calculation and operation phase change logic in the base model.

The possibility to define the applied modelling techniques distinguishes OpenMARS from other approaches. In this paper we present the definition and combination of basic traditional techniques. The same methodology is applicable for inclusion of domain-specific customizations, which are often needed in actual cases. The definition of an arbitrary new modelling technique is also possible. With this OpenMARS aims for similar flexibility to modern MBDA techniques. However, with MBDA the model creation requires understanding the used high-level programming language. OpenMARS needs similar expertise only when the most suitable techniques are defined for certain type of cases. After that, tables with strict structure can be used for modelling without a need of expert knowledge in software engineering. With this approach OpenMARS aims for simple and clear model creation, which is as easy as modelling with basic traditional techniques, such as fault tree and Markov model.

Our end goal is to create user-friendly software that supports the advanced features of the OpenMARS approach. The discussion section explains the need for this type of software along with potential future improvements and applications of the approach. OpenMARS decouples the model from the calculation engine, which is used for obtaining the analysis results. Our own implementation of the calculation engine is based on the Monte-Carlo [21] method. It can be configured to analyse various modelling techniques and also to include user-defined special features. The simulation algorithm uses distributed processing architecture to permit efficient parallel calculation in a computing cluster.

2. Related Work

Fault tree analysis (FTA) [22], reliability block diagram (RBD) [23], Markov analysis [24], and Petri net [25] are examples of traditional formalisms for quantitative risk assessment. Recent research has proposed various extensions and generalizations to increase their expressive power and ease of use. Generalized stochastic Petri net (GSPN) [26], continuous-time Markov chains (CTMC) [27], extended stochastic Petri nets (ESPN) [28, 29], and semi-Markov process (SMP) [30] are examples of extensions that enhance the modelling power [31] of the traditional techniques. In contrast, a coloured Petri net (CPN) [32] is an extension that focuses on the practical use of the formalism instead of increasing its expressive power. Hybrid techniques have been created for situations where a single formalism is not the most practical for all parts of the model. For example, the RBD driven Petri net [33] and the conjoint system model (CSM) [34] both combine Petri net and RBD techniques.

Various software tools exist to enable efficient use of the modelling techniques. For example, CPN Tools [35] is for editing and simulation of CPN models, GRIF BStoK [36] for RBD driven Petri nets, and REALIST [37] for CSM. All of them share similarities with ELMAS [5] software, which was used as a basis for OpenMARS. All the tools use fixed modelling techniques and their combinations. ELMAS permits inclusion of user-defined code [7] to sup-

port modelling of domain-specific features. OpenMARS improves this flexibility further by enabling free definition and customization of modelling techniques.

Over the past twenty years, researchers have made continuous efforts to simplify the analysis process by automatically synthesising dependability related data from system models [38]. This has led to the emergence of the field of model-based dependability assessment (MBDA). While certain techniques focus on making the analysis process more manageable, other MBDA techniques have been developed to address the limitations of traditional techniques [39]. The field of MBDA encompasses a large variety of techniques, such as HiP-HOPS workbench [40], FPTN [41], FPTC [42], SAML [43], smartIflow [44], AltaRica [45], and Figaro [46].

The MBDA techniques can be classified according to different criteria. For example, the model provenance is a criterion that distinguishes MBDA techniques based on their relationship with the system design process [3]. Models can be defined either through extension to the design model, or as a standalone model without direct connections to design models. Creation of a dedicated dependability model requires more work, but it allows using the optimal level of abstraction and including only the needed details from reliability and risk analysis point of views.

The general underlying formalism and the types of analyses performed typically gravitate MBDA techniques towards two leading paradigms [39]. In failure logic synthesis and analysis (FLSA) the fault tree or other failure model is automatically constructed from the information stored in the system model. The other approach is behavioural fault simulation (BFS), where faults are injected into the model that simulates system behaviour.

One criterion for the classification is the type of relation modelling. Some MBDA techniques define basic relations, which are general directed connections between different type of model elements. These relations are similar to the connections that are used in traditional lower level formalisms. In advanced relation modelling the definition is made more in detail by creating connections between attributes of elements. This can be done by using, for example,

transitions and assertions [47]. Advanced relation modelling permits undirected connections, which help to keep a model structure close to reality [44]. The direction of an undirected connection is determined automatically, which makes the definition of the direction unnecessary for the model creator. Some techniques support both relation types to maintain closer compatibility with lower level formalisms. In that case, the basic directed connections can be seen as shortcuts that encapsulate certain more detailed relation definitions.

Table 1 presents the comparison of certain MBDA techniques. The classification is based on the previously mentioned criteria, and the use of an object-oriented paradigm, which was one criterion in [48]. The table includes a characterization of the approach that is presented in this paper. OpenMARS uses standalone models, which are dedicated to risk and performance analysis. Behavioural fault simulation can be made by using OpenMARS models. Basic modelling techniques, such as fault tree and Markov models, use directed connections in OpenMARS, but it is also possible to define a special modelling technique for the creation of undirected models. The object-oriented paradigm is used as a basis for OpenMARS models.

Table 1: Model-based dependability assessment (MBDA) formalisms

	Use of design models	**Underlying formalism**	**Relation modelling**	**Object-Oriented**
HiP-HOPS	yes	FLSA	basic	no
FPTN	standalone	FLSA	basic	no
FPTC	yes	FLSA	basic	no
SAML	yes	BFS	basic	no
smartIflow	yes	BFS	advanced	yes
AltaRica	standalone	BFS	advanced	yes
Figaro	standalone	BFS	both	yes
OpenMARS	standalone	BFS	both	yes

The comparison made with these four classification criteria shows that OpenMARS shares the most similarities with Figaro. However, we recognize two clear differences between OpenMARS and modelling language-based approaches. The first difference is the use of tabular model definition format, which based on our understanding is not used in any other approach. We decided to use tables because in our opinion an average modeller can understand tabular format easier than any markup language, such as XML[16]. Figaro uses Scala programming language [46] for model definition, which in our opinion creates a threshold for the modeller.

The other difference is the separation of modelling technique definition from model creation. OpenMARS includes several built-in techniques and permits the definition of a customized technique. All techniques are defined with the same tabular format, which allows using built-in techniques as a basis for a new technique. Based on our review, other approaches use only fixed techniques and possibly allow their combination. A more detailed comparison between the modelling features of OpenMARS and other MBDA techniques could be an issue of a separate publication. For example, such comparison has been made between AltaRica and SAML [48].

Future trends are likely to yield more robust integrations between existing paradigms and techniques [39]. SAML is an example of an integrative approach. Specifications can be written with various high-level tools, transformed into a SAML model, and verified using a selected verification tool. Similarly, guarded transition systems (GTS) [49] is a low-level formalism, which generalizes classical formalisms and also interprets the semantics of AltaRica and dynamic fault tree (DFT) [50] models. Open-PSA [17] and Petri Net Markup Language (PNML) [18] are similar model exchange formats for traditional techniques. We see that OpenMARS fits to this trend perfectly because it generalizes the traditional techniques and is also open for interacting extensions. For example, the creation of a modelling technique that is compatible with GTS would allow various GTS assessment tools for OpenMARS models. Conversely, it also adds our calculation engine as a tool for all compatible formalisms.

3. OpenMARS Methodology

This section provides an executive summary of the methodology behind the OpenMARS approach where the focus is on its basic principles and concepts. The OpenMARS specification [12] covers all features of the approach in greater detail. The OpenMARS modelling procedure has three main steps, which are illustrated in Table 2. Each step requires own type of knowledge and expertise.

Table 2: Steps of OpenMARS modelling procedure

	1) Select/declare techniques	**2) Create model structure**	**3) Assign parameter values**
Model	1a) Classes, 1b) Attributes	2a) Elements, 2b) Connections	3a) Assign element attribute values
Analyse	1c) Configure calculation engine		3b) Results from calculation engine
User	Software/reliability engineer	Reliability engineer, System expert	System expert
Tool	Tabular format, Java code	GUI, import tool, tabular format	GUI, import tool, tabular format

In a basic case, the first step is that a reliability engineer selects the most suitable modelling technique. If special domain-specific features are needed, an experienced user can customize existing techniques or create a new technique. This is done by declaring new model element classes and their attribute. Experience with programming and simulation algorithms is needed to configure the calculation engine for handling of the new special features.

The two following steps use the selected modelling techniques. The model structure is defined by creating the elements and by adding connections between them. The reliability engineer creates the model structure based on the information provided by system experts. If the model is simple, or if a helping GUI

can be used, system experts may create the model structure independently.

The model parametrization is made by assigning values for the attributes of the model elements. The system experts can compare different scenarios by updating the values before using simulation to calculate analysis results. Instead of manual definition, the model parametrization, and possibly also the model structure creation, can be made by importing the data automatically from external systems or databases.

3.1. Model Elements

The OpenMARS approach is based on an object-oriented paradigm. A model consists of *elements*, and each element has a *class*, which defines the individual *attributes* of the element. An attribute can be another element or a *primitive*. For example, strings and numbers are primitives, which store single parameter values, such as title, description, cost, duration, or colour. Elements are structured attributes that can contain multiple primitives. Four fundamental element classes form the basis of all modelling techniques. Each class of every technique inherits one of the following parent classes:

Node: Models the possible states of an item

Operator: Defines rules between state changes of different nodes

State: Models whether a particular set of circumstances is active

Transition: Defines the possible state changes

Each modelling technique has a catalogue of available classes, which extend the fundamental element classes based on the needs of the modelling technique. The common base classes make the combination of different techniques more straightforward. The goal is that most cases should be covered with the built-in modelling technique classes. To guarantee that the concept is always applicable, expert users can create model specific classes to customize the model to their special needs. The new classes can be defined to extend the attributes or overwrite the default values of an existing class.

As an example of how we implemented different techniques with element classes, Figure 1 and Table 3 show how the fundamental classes adapt to selected techniques. In the FTA technique, a fault node is a container for two states (*fault* and *normal*) and two transitions (*failure* and *restoration*). So, it forms a small Markov model where the active state element defines the fault node state. Fault nodes are connected with gate operators to define the logic rules, such as AND, OR, Vote, etc... A Markov node element is a container for a Markov model, which consists of user-defined states and transitions. Like fault nodes, the state of the Markov node is defined by the active state element. Petri nets consist of places, transitions, and arcs [25]. In OpenMARS a Petri net place is a node and its state is defined by the number of tokens in it. Petri net transitions are instances of an Operator class[1]. Arcs are formed by connecting places and transitions. A function modelling technique has value nodes, where the state is defined by the contained numeric value. Value nodes are connected with function operators, which can read and update the node values and other numeric attributes.

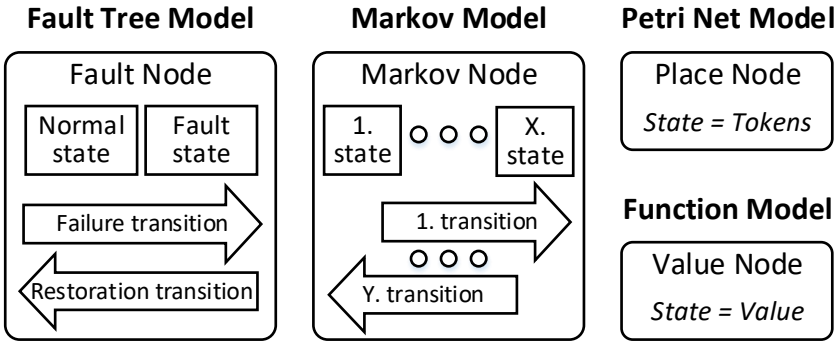

Figure 1: Examples of nodes in different modelling techniques

[1]Here confusion caused by the name of the Transition class and Petri net transitions is regrettable.

Table 3: The use of fundamental classes with different modelling techniques

Technique	Node	Operator	State	Transition
FTA	Fault	Gate	Normal, Fault	Failure, Restoration
Markov	Markov	-	User-defined	User-defined
Petri net	Place	Transition	-	-
Function	Value	Function	-	-

The model structure is formed by defining directed connections between elements. In OpenMARS, a connector has no parameters. An operator element is used to define the type of connection between nodes. For example, in FTA models the gate operators define the connections between fault nodes. Similarly, transition elements are used to define the connections between states. Figure 2 illustrates how gate operators are used in FTA and transitions in Markov modelling technique.

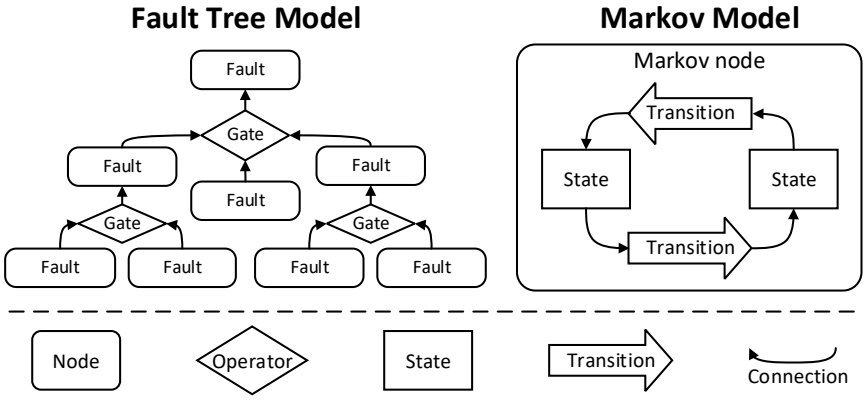

Figure 2: Connections in FTA and Markov modelling techniques

3.2. Special Features

The OpenMARS concept contains *folders*, which improve the handling of large models. Each folder is a container of nodes and operators. If a model

structure is created inside a folder, it is possible to create multiple instances of the folder where each contains the same structure. This helps the definition of models with repetition. Similarly, nodes are containers of states and transitions. It is possible to define the structure of states and transitions only once and create as many node instances as needed. For example, each fault node instance contains the two states and transitions.

The creation of multiple similar fault node instances follows the class-based paradigm, where the fault node is a class that is defined by the modelling technique. With the help of an array assignment, OpenMARS also supports a prototype-based [51] approach for the creation of several objects with similar contents. By adding an interval or a list inside square brackets as a suffix of the element or folder name, it is possible to create an array of similar objects. For example, definition 'pump[1-5]' creates five pumps. After this, it is possible to use the name 'pump' to make prototype-based definitions that consider all pumps. It is also possible to refer a set of pumps with 'pump[1-3]' or a specific one with 'pump[4]'. Furthermore, a comma separated list can be used instead of single values as a compact way to make multiple definitions just by using one table row. For example, 'valve, motor' can be used to make the same definition for both valve and motor.

The array assignment is also used for the definition of mode-dependent attributes. In OpenMARS, an array is actually an ordered map, which allows associating values to keys. Instead of using an attribute name, the key of the array assignment can be used to indicate the name of the mode when the attribute value is valid. For example, with *failure* transition it is possible to use definitions 'failure[prepare]' and 'failure[produce]' to have different failure distributions for preparation and production modes.

Unique Identifier (UID) is defined for each created folder and element. The UID is formed by combining the UID of the container with the name of the element by using a slash (/) symbol as a delimiter. The empty UID refers to the default base folder of the model, which is used if an element of a folder does not belong to any other container. With UID it is possible to refer any element

of the model, which also allows connecting sub-models that are possibly defined with different techniques.

We have also included a broadcasting system, which allows connecting separate models without a need to know the exact UIDs of the elements. OpenMARS uses radios and listeners for communication between distinct sub-models. A radio can be attached to transitions or states and set to broadcast messages on certain channels in defined situations. The radio broadcasts when the transition is triggered or when the state activates or ends. Similarly, listeners can be attached to both transitions and states. If the listener receives a signal in a certain channel, the transition or the state is activated. The channels form a simple and clear interface for combining separate sub-models to a comprehensive model. Figure 3 illustrates the connection between two simple models. A system fault causes a radio to broadcast on a waitStartChannel. The listener in the operation phases model receives this and changes to a *wait* phase. Similarly, the *operate* phase starts when the system returns to *normal* state.

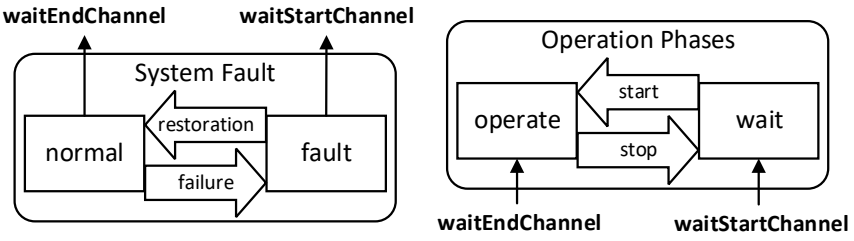

Figure 3: Examples of radios and listeners that connect separate models

OpenMARS allows modellers to define mode-dependent attribute values. Radios and listeners signal the currently active modes to elements, which update the attribute values based on the active mode. Radios emit the mode start or end messages, and listeners convey them to the elements which they are attached to. A practical example of this feature is the modelling of failure rates that change based on the operation mode. A modeller can assign higher failure rates

to the modes that are more demanding. Section 5 presents an example where a Markov model determines the active operation mode that is linked to the failure rates of a fault tree.

The function modelling technique of OpenMARS creates an environment for visual programming. The basic operators, such as addition and subtraction are built in, but modellers can also define functions freely with programming code. This is the key feature that enables the definition of custom KPIs. Programming code is required only when the KPI cannot be defined as a combination of basic operators.

3.3. Definition of Models with Tabular Format

The OpenMARS approach uses five different tables to define models: (i) Class, (ii) Attribute, (iii) Element, (iv) Connection, and (v) Value. The model definition procedure can be divided into three steps: modelling technique definition, model structure creation, and parameter value assignment. The first step differentiates OpenMARS from other approaches, which support only predefined fixed techniques. The first step can be skipped if built-in techniques are sufficient for the modelling of the case.

The *modelling technique definition* is made by using the first two tables. The Class table contains the following columns:

CLASS: The name of the introduced class

IS A: The name of the parent class that the new class extends

The Attribute table contains the following columns:

CLASS: The name of the class to which the new attribute is associated

TYPE: A class name that indicates the type of the new attribute

ATTRIBUTE: The name of the new attribute or an asterisk symbol (*) to define the class as a container for this type of object

The content of the first two tables is identical for all models that use the same modelling techniques. Sometimes the definition of a special tailor-made technique is needed, but usually already existing traditional techniques can be translated to the OpenMARS format and used directly. An expert user can use these tables to create model-specific classes. Each created class inherits the attributes of a built-in parent class and extends them with new attributes, which can be used to model special features.

Practical experience about the reliability modelling helps to select the most suitable techniques for solving the analysed problem. The selected modelling techniques provide a catalogue of classes for *model structure creation*. The second step of OpenMARS modelling procedure is made by using the next two tables. The Element table creates elements and folders of the model. It contains the following columns:

CONTAINER: The container of the new element

ELEMENT: The name of the new element

CLASS: The class of the new element

The Connection table defines the directed connections between elements. This forms the model structure, which is a finite directed graph [52] of elements. The correct direction of the connection is defined by the modelling technique. For example, in FTA the root faults are the sources that are connected to the top fault through gates and intermediate faults. The Connection table contains the following columns:

SOURCE: The source element of the connection

TARGET: The target element of the connection

The third step of OpenMARS modelling procedure is the *parameter value assignment*. The Value table is used for defining values for elements and classes. If an attribute value is defined for a class, it is the default value for all instances of that class. The Value table contains the following columns:

OBJECT: The element or the class for which the attribute value is defined

ATTRIBUTE: The name of the attribute

VALUE: The assigned value

4. Analysis of OpenMARS models

The OpenMARS models are decoupled from calculation. Potentially each modelling technique has various simulation tools or analytical solvers, which are created by different tool providers. This section presents our approach for a simulation-based analysis of OpenMARS models.

4.1. Dynamic compilation of the simulation algorithm

OpenMARS permits defining arbitrary modelling techniques. This unique feature creates a challenge for the used analysis tool, which must be highly configurable to support any modelling technique. Figure 4 illustrates our approach, in which the calculation engine, a simulator tool, and modelling technique-dependent configurations are dynamically compiled to a Java simulation program. The dynamic compilation ensures that only the needed procedures are included. This creates a minimal sufficient simulation algorithm, which increases the efficiency of the analysis process.

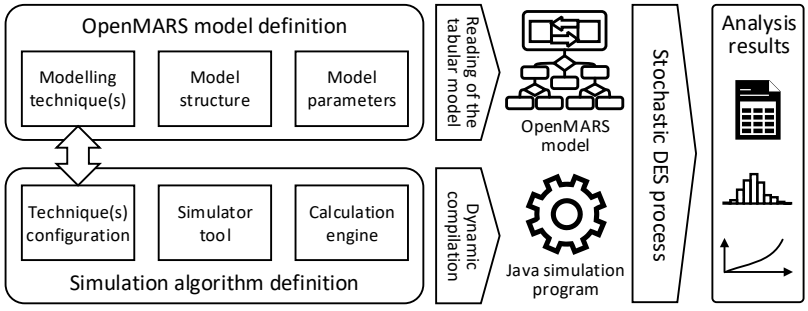

Figure 4: The dynamic compilation of the Java simulation program

The Java simulation program is built by using a template method pattern [53], which is an example of an inversion of control (IoC) design principle [54]. In object-oriented programming, the IoC is used to increase the modularity of the program. In traditional programming the custom code calls for static libraries, but with IoC, it is the generic framework that calls task-specific codes. Figure 5 illustrates how the extensible algorithm skeleton forms a framework that divides the stochastic discrete event simulation (DES) [55] process to separate phases. Each phase is a template method. The configuration of the simulator tool defines the Java code of each template method. This dynamically builds a Java object that implements the simulation algorithm.

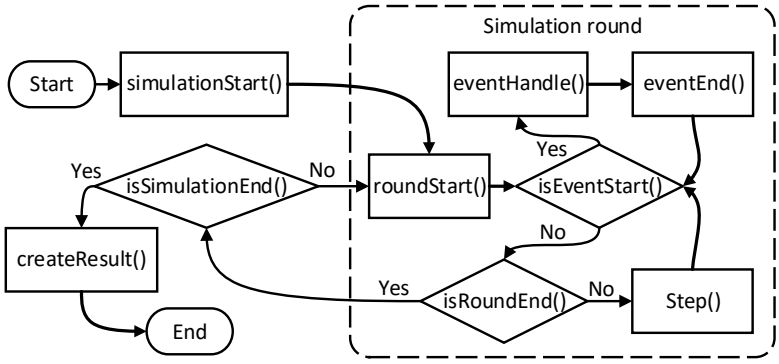

Figure 5: The DES algorithm skeleton divides the process to template methods

The model elements have also similar template methods, which are called by the simulator tool. The calculation engine translates each OpenMARS model element to a corresponding Java object. The classes of the Java objects implement the template methods with Java codes that are defined by the corresponding OpenMARS classes. The configuration uses the Value table to assign the Java codes for each OpenMARS class in tabular format. If a new class or modelling technique is added, a new configuration must be created to specify how the new elements are handled in the simulation.

By using the Class and Attribute tables the configuration declares the simulator tool and the simulation attributes of the model elements. An attribute is used either as a parameter or a variable. The parameters are defined by the model creator and only read during the simulation process. For example, *'rounds limit'* and *'simulation period'* are parameters of the simulator tool, which define the simulated time. The variable values are updated during the simulation progress. Variables can store the current status, collect statistics data or store analysis result values. For example, the simulator tool has status variables *'current round'* and *'current time'*. The *'cumulative action count'* of transitions and *'cumulative active time'* of states are examples of statistics variables.

4.2. Simulation process

During the DES process, the simulation algorithm calculates pre-defined number of rounds. Each round handles events and updates the status and statistics variables until the pre-defined simulation time period is reached. The model elements create initial events at the beginning of each round. An event has a time of occurrence and a target model element that handles the event. The events are stored within a chronologically ordered list. The handling of an event is the core of the simulation process. Each class can configure its own procedure to define how the event is handled. The handling can create new events, which are inserted to the list. An event can also set some events to wait, wake them up or remove them from the list. The first event of the events list is removed after it has been handled. Various analysis results are calculated based on statistics variables after all rounds are simulated.

The following list illustrates how the simulator tool handles the template methods that were shown in Figure 5. The list gives examples of the operations that the fundamental model element classes make during each template method.

simulationStart() method is called at the beginning of each simulation. Here a reset is made to the status variables that control the simulation process and to the statistics variables that collect the data. For example,

the simulator tool resets the *'current simulation round'*, the transitions the *'cumulative activation count'* and the states the *'cumulative active time'*.

isSimulationEnd() test is made after a simulation round has ended. Here the simulator tool increases the value of the *'current simulation round'* and compares it to the *'rounds limit'*.

roundStart() method is called before the handling of each round. Here the simulator resets the *'current time'*, which represents the simulation clock. The simulator tool also clears the events list. Each node element activates the initial state and creates the first state change event to the events list.

isRoundEnd() test is made before a simulation step. The currently simulated round is ended if the *'current time'* equals the *'simulation period'*.

isEventStart() test is made after each event handling and after taking a time step. A new event is handled if the *'current time'* equals the time of the first event.

eventHandle() method is called when starting an event handling. Here the simulator tool calls the template method of the event's target model element. The classes of the modelling techniques configure the procedures of event handling for each type of model element:

- A transition increases the *'cumulative activation count'* and creates a new immediate event for its target state. For example, in a fault tree a failure transition increases the number of failures count and creates an event for a fault state.
- A state creates an immediate event for its owner node to notice the state change, and calls the event creation method of its target transitions. For example, a fault state creates an event to its container node, and asks a restoration transition to create a new future event. The stochastic time of the future event can follow any distribution function, as presented in [12, Annex C].

- A node updates the active state and creates a new immediate event for its target operators. For example, a fault node creates an event for gate operators, which connect it to the nodes that represent its consequence faults.
- An operator checks the active states of its source nodes to decide if the target node state needs to be changed. For example, a fault tree gate uses a logic rule to decide when a source node state change affects to consequence fault nodes. OpenMARS fault tree implementation supports many gate operators, which are presented in [12, Annex D].

If the element that handled the event has a radio attached, new immediate activation events are created for all elements that listen to the same channel.

eventEnd() method is called after an event has been handled. Here the simulator tool removes the first event from the events list.

step() method is called if an event does not exist at the current time. Here the step is taken to the occurrence time of the next event or to the end of the simulation period if there aren't any events that occur before it. The simulator tool calculates a value for the *'step length'* status variable, which is added to the *'cumulative active time'* of each currently active state. After all elements have handled the step, the *'step length'* is added to the *'current time'* of the simulator tool.

createResult() method is called when all simulation rounds have been finished. Here the analysis results are calculated based on the statistics variables of the model objects. For example, the simulator tool calculates the product of *'current round'* and *'simulation period'* to get the *'total simulated time'* result value. Each state element adds the *'cumulative activation count'* variables of all its source transitions to get the *'number of state activations'* result value. It can be used together with the *'cumulative active time'* for the calculation of various other results. For example, the *'state probability'* is obtained by dividing the *'cumulative active time'* by the *'total simulated*

time' and the *'mean duration of an activation'* is obtained by dividing the *'cumulative active time'* by the *'number of state activations'*.

For the needs of large simulations, our calculation engine is suitable for deployment in a distributed processing environment. This enables an efficient parallel calculation in a cloud computing cluster. The parallelization is made by dividing the simulated rounds between processes before the simulationStart() method is called. Each process simulates the rounds independently. The statistics variables of all processes are combined to the root process before the createResult() method. After this, the analysis result creation can be done in the root process like in a basic single core simulation.

5. An Example of OpenMARS Modelling Procedure

This section demonstrates the application of OpenMARS with an example case, which resembles a multi-state industrial production process that is subject to failures. In our opinion, this simple case captures the core concepts of production process modelling. For example, our availability model for a particle collider uses a similar modelling approach [11]. The example model demonstrates the declaration and use of traditional techniques, which should be familiar to the reader. However, the approach would be the same for declaration of customized domain-specific techniques.

The example case is modelled by combining fault tree, Markov, and function techniques. The example illustrates how a modelling technique is declared, how a model is created by using it, and how different techniques are connected. The connections between separate parts of the model would make it challenging to define the model with a single traditional approach. The tabular format is used for all three OpenMARS modelling procedure steps. The model is analysed to study how much failures reduce the annual production. The analysis also compares the effect of different failure mitigation scenarios and changing the production parameters.

5.1. Case Description and Application of OpenMARS Modelling Steps

The KPI of the production process is the cumulative annual production. The process has an operation cycle that consists of two main phases: *prepare* and *produce*. During the *prepare* phase the process gets ready for the production, which occurs in the *produce* phase. The process has a time-dependent production rate, which is first a constant but starts to diminish after a certain time. Due to the diminishing production rate, a maximum length is defined for the *produce* phase to optimize the cumulative production. The end of the *produce* phase restarts the operation cycle from the *prepare* phase.

The process is subject to failures. The sources for system faults are a power input and two identical pumping units, which both have two redundant pumps. The *produce* phase is more demanding for the pumps, which makes them more likely to fail during the production. Each system fault interrupts the process and makes it wait until the fault is repaired. After the restoration a new operation cycle is started from the *prepare* phase.

The modelling of the example case can be divided to three parts: the system's failure logic, the process phases, and the production function. There exist basic built-in FTA, Markov and function modelling techniques [12], which are suitable for these needs. The *produce* phase of the Markov model activates the function model to calculate the cumulative production of an operation cycle. The calculation is made after the *produce* phase ends by using the length of the production as an input. The fault tree and the Markov model are connected in two ways. First, the active phase in the Markov model affects the failure rate of the pumps in the fault tree. This requires each phase change in the Markov model to send information to the fault tree. Second, if a system failure occurs or is restored, the Markov model changes the active phase. This requires the fault tree to send information to the Markov model. Figure 6 illustrates the OpenMARS model of the example case.

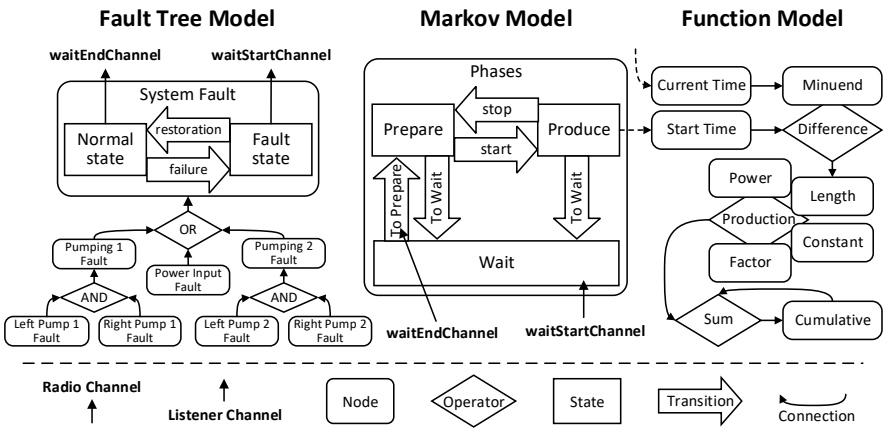

Figure 6: The example is a combination of fault tree, Markov, and function models

The following list describes how the example model can be created with the OpenMARS approach. The description follows the three main steps of the OpenMARS modelling procedure (see Table 2 in Section 3).

1) Select/declare techniques At first a user selects or declares the applied modelling techniques. This declaration step can be skipped in this case as the model uses pre-defined techniques. Regardless, Section 5.2 presents the class (step 1a) and attribute (step 1b) definition tables for declaring the applied built-in techniques. The way the declaration is made demonstrates the procedure of defining a customized modelling technique.

2) Create model structure The user defines the model structure by creating the elements (step 2a) and by adding the connections between them (step 2b). Section 5.3 presents the OpenMARS definition tables for creating the fault tree, Markov and function model structures. The combination of these sub-models forms the OpenMARS model of the example case.

3) Assign parameter values The user assigns values for the attributes of the model elements. In this case, the parameters are defined for the failure and restoration times, for the delays between the phase changes, and for the

production function. Also the channel names, which enable the communication between the sub-models, are defined as model parameters. Section 5.4 presents the OpenMARS definition tables for model parametrization (step 3a).

The analysis results of the example case are obtained with a stochastic simulation. The calculation engine compiles dynamically the simulation algorithm based on the configuration of the applied modelling techniques. The configuration defines the simulation algorithms of the model elements (step 1c). Section 4 presents the principles of this step, but the algorithm codes are not included in this paper. The user parametrizes the calculation engine by assigning the number of simulated rounds and the length of a simulation period. Section 5.5 presents the obtained analysis results (step 3b).

5.2. Declaration of the Modelling Techniques

The system model can be created by combining FTA, Markov, and function modelling techniques. This section shows how the Class and Attribute tables can be used to define these techniques. In a basic situation this step of the modelling could be skipped, because OpenMARS has these techniques built-in. By using the Class and Attribute tables it is also possible to declare tailor-made techniques for special needs.

Table 4 declares the classes of the three techniques. For example, FTA uses fault nodes that are connected with OR and AND gate operators. The transition classes are used both by the fault tree model to describe failure and restoration times, and by the Markov model to describe the transitions between process phases. Table 4 also illustrates the class inheritance. For example, the Operator is a parent of the Gate class.

Table 5 declares the attributes for the FTA and Markov modelling technique classes. The fault nodes have two states and transitions with pre-defined names. The Markov nodes are defined as containers for states and transitions by using the asterisk (*) symbol. The last two rows of the table declare the attributes for the transitions.

Table 4: The Class table for the example model

CLASS	**IS A**
Fault, Markov, Value	Node
Gate, Function	Operator
OR, AND	Gate
Addition, Subtraction, UserFunction	Function
ExpTransition, WeibullTransition, ConstantTransition	Transition

Table 5: The Attribute table for the FTA and Markov modelling technique classes

CLASS	**TYPE**	**ATTRIBUTE**
Fault	State	normal, fault
Fault	Transition	failure, restoration
Markov	State, Transition	*
ExpTransition, ConstantTransition	Number	mean
WeibullTransition	Number	scale, shape

Table 6 declares the attributes for the function modelling technique classes. The value nodes have a single numeric *value* attribute. A minuend is declared for subtraction, which allows defining the role of the operands. The container declaration is used for values of the user function, which allows a modeller to use any names for the parameter values of the function.

Table 7 declares the attributes for broadcasting messages between models. Radios and listeners are declared for all states and transitions. States have radios to send a message when the state starts and ends. Also folders can have a listener, which conveys the messages to the elements within the folder. Special wake and wait listeners are defined for elements and folders. They are used for activating and deactivating mode-dependent attributes.

Table 6: The Attribute table for function modelling technique classes

CLASS	TYPE	ATTRIBUTE
Value	Number	value
Subtraction	Value	minuend
UserFunction	Value	*
UserFunction	Text	code

Table 7: The Attribute table for connections between models

CLASS	TYPE	ATTRIBUTE
State, Transition	Name	radio, listener
State	Name	endRadio
Folder	Name	listener
Element, Folder	Name	wakeListener, waitListener

5.3. Creation of the Model Structure

This section shows how the example model structure is created by using the Element and Connection tables. The comprehensive model consists of three sub-models that are made using different techniques. Table 8 creates the fault tree model elements. The system contains two similar pumping units, which are in their own folders. This illustrates how the array definition can efficiently create similar structures. The pumping[1-2] creates two folders that can be referred to by setting the text 'pumping' as the container. This creates identical model structures within both folders. If needed, a specific folder could be identified by referring to it's number, for example, pumping[1].

Table 9 adds the connections of the fault tree model. The table illustrates how the array definition is a simple way to add multiple connections. A line with pumpingAND as a source and pumpingFault as a target creates a connection in both pumping folders. However, if a modeller wants to specify in which folder the connection is made, for example, (i) pumping[1]/pumpingAND to pump-

Table 8: The Element table for the fault tree model

CONTAINER	ELEMENT	CLASS
	system	Folder
system	systemFault, powerFault	Fault
system	systemOR	OR
system	pumping[1-2]	Folder
pumping	pumpingFault	Fault
pumping	pumpingAND	AND
pumping	pumpFault[left,right]	Fault

ing[1]/pumpingFault creates the connection only in one folder, and (ii) pumping[1]/pumpingAND to pumping/pumpingFault creates the connection from the pumpingAND in folder pumping[1] to pumpingFaults in both folders.

Table 9: The Connection table of the fault tree model

SOURCE	TARGET
systemOR	systemFault
powerFault, pumping/pumpingFault	systemOR
pumpingAND	pumpingFault
pumpFault	pumpingAND

Table 10 shows the creation of the Markov model. In this example the Markov node contains a state for each of the three process phases. The *start* and *stop* are defined as constant time transitions. In OpenMARS this means that the transition activates always after the set constant duration. The classes for the transitions toWait and toPrepare are not specified. Normally this means that the transition type will be defined in the Attribute table. However, in this case the definition of a specific transition class is not needed because the transitions are only triggered by radio messages from the fault tree model.

Table 11 adds the connections of the Markov model. Transition elements

Table 10: The Element table for the Markov model

CONTAINER	ELEMENT	CLASS
	phases	Markov
phases	prepare, produce, wait	State
phases	start, stop	ConstantTransition
phases	toWait, toPrepare	Transition

are added between the states. It is notable that the transition toWait has two sources, because the *wait* phase can be started from both *prepare* and *produce* phases.

Table 11: The Connection table of the Markov model

SOURCE	TARGET
prepare	start
start	produce
produce	stop
stop, toPrepare	prepare
wait	toPrepare
prepare, produce	toWait
toWait	wait

Table 12 shows the creation of the function model elements. The *output* folder contains several values and functions. It is notable that here the user-defined *production* function contains value nodes as it's attributes. They are used as parameters of the user-defined function.

Table 13 adds connections of the function model. The first two of them connect the simulated variable values to be the function operators. The simulator/currentTime measures the time in the simulation, and phases/produce/startTime the latest start time of the *produce* phase. The other connections define how the functions are combined to calculate the KPI.

Table 12: The Element table for the function model

CONTAINER	ELEMENT	CLASS
	output	Folder
output	difference	Subtraction
output	production	UserFunction
production	length, constant, factor, power	Value
output	sum	Addition
output	cumulative	Value

Table 13: The Connection table of the function model

SOURCE	TARGET
simulator/currentTime	output/difference/minuend
phases/produce/startTime	output/difference
difference	production/length
production, cumulative	sum
sum	cumulative

5.4. Model Parametrization

This section shows how the Value table is used to parametrize the example model. Table 14 defines the basic transitions of the fault tree and Markov models. By default all *failure* and *restoration* transitions are exponentially distributed and defined by using the *mean* parameter, which allows for defining the mean value directly for powerFault failures. In this example all restoration times have the same mean value, which is defined in one row. The last two rows define the constant durations for transitions of the Markov model.

Table 15 shows the definition of an operation mode-dependent failure rate for pumps. The first row defines the transition type for both modes. Here we use a Weibull distribution, but each mode can also have a different transition type. The Weibull *scale* parameter definition shows how the name of the mode

Table 14: The Value table to define the transitions of the example model

OBJECT	ATTRIBUTE	VALUE
Fault	failure, restoration	ExpTransition
powerFault/failure	mean	8760 h
Fault/restoration	mean	24 h
phases/start	mean	48 h
phases/stop	mean	24 h

is used as a key for the attribute value map. The Weibull *shape* parameter is the same for both modes, so it does not need to be mapped.

Table 15: The definition of the mode-dependent failure rate for pumps

OBJECT	ATTRIBUTE	VALUE
pumpFault	failure[prepare, produce]	WeibullTransition
pumpFault/failure[prepare]	scale	240 h
pumpFault/failure[produce]	scale	180 h
pumpFault/failure	shape	1.5

Table 16 shows how the model is set up to use the *prepare* mode values when the Markov model is in the *prepare* phase, and the *produce* mode values during the *produce* phase. Radios broadcast on certain channels when the *prepare* and *produce* phases start in the Markov model. The correct channels are defined for listeners, which activate and deactivate the modes.

The transition rates from the *prepare* and *produce* phases to the *wait* phase are not set. A connection with the fault tree model is required for the *wait* phase to activate. Table 17 shows how to define a situation where the top failure of the fault tree model starts and stops the *wait* phase of the Markov model. The radios and listeners are set up (i) to start the *wait* phase when the *fault* state starts, and (ii) to activate the transition to the *prepare* phase when the *normal* state starts.

Table 16: The definition of radios and listeners for mode changes

OBJECT	ATTRIBUTE	VALUE
phases/prepare	radio	prepStartChannel
phases/produce	radio	prodStartChannel
pumpFault	wakeListener[prepare]	prepStartChannel
pumpFault	waitListener[prepare]	prodStartChannel
pumpFault	waitListener[produce]	prepStartChannel
pumpFault	wakeListener[produce]	prodStartChannel

Table 17: The definition of connections between fault tree and Markov models

OBJECT	ATTRIBUTE	VALUE
systemFault/fault	radio	waitStartChannel
systemFault/normal	radio	waitEndChannel
phases/wait	listener	waitStartChannel
phases/toPrepare	listener	waitEndChannel

The attributes for the *production* function are defined in Table 18. In the first two rows a radio-listener pair is created to activate the function model in the *output* folder each time when the *produce* phase ends in the Markov model. In subsequent rows, the three input parameter values are defined for the user function. The last row shows how a value can also be read from an external file by using the URL of the location that contains the user-defined code.

The implementation of the OpenMARS calculation engine can define which programming languages are supported and how the code is executed during the simulation process. Listing 1 shows how our current Java-based implementation defines the function. The production of each operation cycle depends on the time spent in the *produce* phase. The production rate is stable up to a certain time constant and decreases after that. The attribute names of the function operator are used as variable names in the listing, which allows the calculation

Table 18: The Value table for the function model

OBJECT	ATTRIBUTE	VALUE
phases/produce	endRadio	functionChannel
output	listener	functionChannel
production/constant	value	10 h
production/factor	value	3
production/power	value	0.6
production/code	url	'url to the code'

engine to attach the user-defined code directly to the simulation algorithm.

Listing 1: The user defined code for the example production function

```
if (length < constant) {
  return factor * length;
}
return factor * (constant +
  Math.pow(length - constant + 1, power) - 1);
```

5.5. The Analysis Results

The analysis results are based on a one year period, which was simulated 10000 times. The following list presents the key results of our analysis:

- The mean number of failures is 36 and mean time to restoration (MTTR) is 12.5 hours, which causes 5.1% unavailability.
- The mean number of *prepare* phase starts is 136. The same value for *produce* phase is 110, which means that over a one year period, failures interrupted the operation cycle 26 times at the *prepare* phase.
- The mean production time of one operation cycle is 22.7 hours, which is 1.3 hours less than the optimal uninterrupted production period.

- The mean annual cumulative production is 4450. The 5% quantile for the production is 4240 and the 95% quantile is 4650.

By changing certain model parameters, we compare the current situation to four different scenarios: (i) the mean time to failure (MTTF) of the power input improves from 8760 to 17520 hours; (ii) the mode-dependency of the pump failure rate is eliminated, which means that the scale parameter of the Weibull distribution would be 240 hours during both *prepare* and *produce* phases; (iii) the mean time to restoration (MTTR) improves from 24 to 18 hours; and (iv) the length of the period where the production rate is a constant increases from 10 to 11 hours. Figure 7 shows the mean annual production in each scenario. Error bars illustrate the 90% confidence interval, which is obtained by calculating the 5% and 95% quantiles of the production.

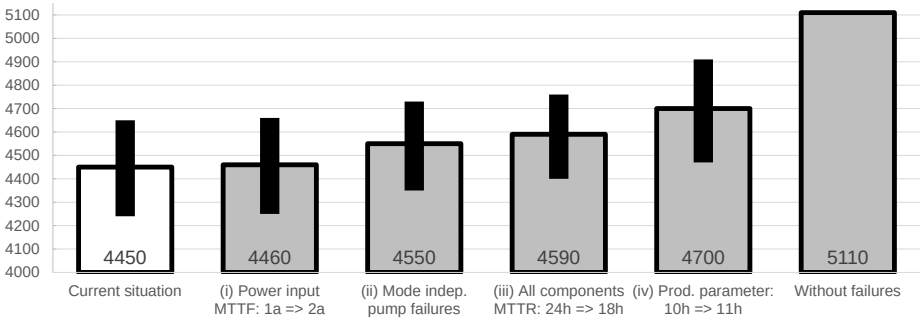

Figure 7: The mean annual production in four scenarios

The results show that the improvement of the power input reliability does not have a notable effect. The shorter restoration time improves the annual production a bit more than the elimination of mode-dependency of pump failures. A change of the parameter that defines the length of the constant production rate improves the KPI more than the changes of the failure and restoration times. For comparison, the current situation and a theoretical scenario of operation without any failures are included in the Figure 7. Because the lengths of the *prepare* and *produce* phases are constants, the optimal production result is without deviation.

Sensitivity analysis was made to study the effect of changing the maximum length of the production in a one operation cycle, which is used because of the diminishing production rate. Figure 8 shows the mean annual production with different maximum lengths of the *produce* phase. Also the 5% and 95% quantiles are included in the figure. The results show that minor changes to the current maximum length of 24 hours do not have a significant effect. The mean annual production is over 4400 with all maximum lengths that are between 18 and 40 hours.

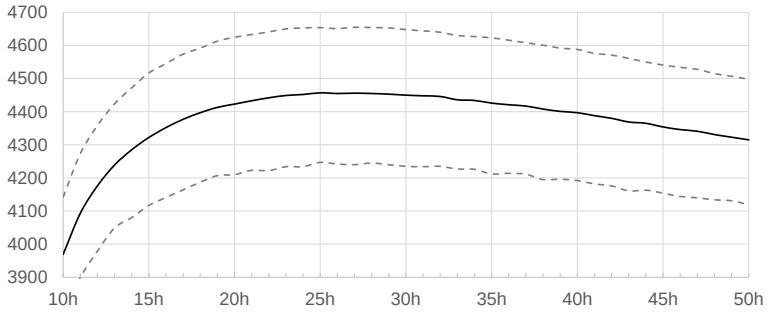

Figure 8: Sensitivity analysis to study the maximum production length parameter

6. Discussion

Section 5 showed how the core concepts of a multi-state reliability and performance model can be implemented utilizing the OpenMARS approach. We see a wide range of industrial applications for this kind of model. Our brief review shows that a combination of operations and reliability modelling has been made for (i) a nickel reduction plant production line [56], (ii) dynamic process simulation of a LNG fuel storage tanks risk assessment [57], (iii) extended warranty cost prediction [58], (iv) availability of offshore installation [59] and manufacturing lines [60]. Also, maintenance spare part circulation [61] could be modelled with OpenMARS.

We developed the OpenMARS to answer the needs we have encountered in various industry cases. For example, in cases from the metal industry, robotics,

and nuclear industry, we have required advanced features for (i) combination of FTA and FMEA analysis [8], (ii) multi-state modelling of partial process flows, (iii) dynamic rules for backup power supply use [9], and (iv) definition of exclusive stochastic consequences [10]. The extensive use of the advanced features was required in our particle collider availability model [11], which prompted us to develop OpenMARS. Like in the presented example, the model combines a fault tree model of failure logic, Markov models of operational cycles, and a function model for production calculation.

The OpenMARS development at CERN relates to an ambitious plan to build a 100 km long future circular collider (FCC), which would reach 7 times higher collision energies than the present large hadron collider (LHC) complex [62]. The FCC study was motivated to develop the collider operations model thanks to success of the LHC and HL-LHC availability studies [63, 64]. Accelerator reliability and performance models are also made for (i) the Tevatron hadron collider [65, 66], (ii) the planned International Linear Collider [67], (iii) the IFMIF fusion material test facility [68], and (iv) the European Spallation Source [69]. The reliability of particle accelerators have become consistently more important with increasing complexity of infrastructure and tightening user requirements [70]. For example, sustainable operations of accelerator-driven reactor applications depends highly on accelerator reliability [71].

A common feature of many of our reviewed studies[2] is that the model was developed for in-house software. This means that a major part of the operational reliability simulation project was spent developing software for that task. Several tools are available for standard applications, but for advanced tasks the lack of reasonably user-friendly software has hindered implementation of reliability methods [1]. In the current ELMAS version, some of these features could be added with user-specific Java snippets. This requires programming knowledge, which creates a threshold for a modeller to implement special features. The key motivation of our collaboration to develop the OpenMARS was

[2]In-house software or model is used at least in studies [56], [58] and [60]-[69].

to limit the need for programming in model development. This helps to shift the paradigm where the reliability engineer needs to have expert knowledge in software engineering in order to model complex system behaviours.

The applicability of OpenMARS can be extended to new situations by including new modelling techniques. One potential technique for implementation is functional-failure identification and propagation (FFIP) [72], which is proposed as a method for performing risk assessment on conceptual design information before the FTA top event and the mechanisms leading to it are known. Another possibility is to enable translation of FTA and RBD models to a Bayesian network [73], which can then include features like probabilistic gates, multi-state variables, uncertainty on model parameters, and dependence between components. Additionally, the implementation of dynamic flowgraph methodology (DFM) [74] would allow for producing a system model, which can be derived via algorithmic procedures to several timed fault trees. Outside the field of reliability modelling, for example, project evaluation and review technique (PERT) [75] for task management, and design structure matrix (DSM) [76] for requirement engineering could be implemented in OpenMARS.

There also exists alternative ways to implement the already built in techniques. For example, our current FMEA implementation is based on a German automotive standard [77], but other qualitative risk classification techniques could also be included. OpenMARS enables inclusion of domain-specific customizations, which helps to adapt the modelling process to the situation at hand. Similar approach is used in the field of systems engineering, where application specific tailoring of the processes leads to the best outcomes [78].

We developed a tabular model data format for OpenMARS to permit efficient definition of large models. Graphical user interface (GUI) can help the manual definition of OpenMARS models but with large and complex systems it can be challenging to handle the visualization of the model. For example, the graphical representation of a large PSB-RF model [19] with more than 2700 components cannot provide a single view to show the whole model structure. However, because the model consist of various similar structures and only 9 different type

of components, the definition of the model structure with OpenMARS table is simple [12, pp. 7–9]. The tabular format has a strict structure, which permits computer-supported model definitions. A future aim for OpenMARS is that various sources, such as management systems, are used for automatic creation of the model structures. With the non-proprietary tabular format we aim for open co-operation and creation of import tools that can interpret OpenMARS models from existing dependability software.

The OpenMARS specification [12] does not define tools for the analysis of the created models. The approach is decoupled from the calculation and open to be used with any analysis tool. Our implementation analyses the models with stochastic simulation. We see several ways to develop our calculation engine further, such as including the Latin hypercube sampling [79] for sensitivity analysis, and genetic algorithms [80] for optimization. We also consider the inclusion of analytical solvers, such as finding of minimal cut sets of fault trees [81], because they can reduce the calculation time in cases where their application is possible. Additionally, research is needed for efficient handling of continuous phenomena. Our calculation engine can currently combine models with discrete (e.g. FTA, Markov model) and continuous (e.g. function model) state spaces, but the simulation of continuously changing states has not yet been implemented.

7. Conclusions

This paper introduced the OpenMARS approach and summarised the basic concepts presented in our open specification document [12]. We focused on special features of OpenMARS by presenting how modelling techniques are defined, how dependability models are created and how custom KPIs are included. We demonstrated the potential of our approach in a simple example that captures the core concepts of production process modelling. In the example, the comprehensive model for risk and performance assessment is created by combining FTA, Markov, and function modelling techniques.

We see high potential for our approach in the operations and performance

modelling of industrial applications. In this field the lack of user-friendly tools has slowed down the application of this type of analysis. In many cases the accurate modelling of the application-specific features has required creating an in-house software. Our end goal is to develop a user-friendly tool that supports the advanced model definition features of the OpenMARS approach. We have already created a Monte-Carlo method based calculation engine, which we plan to employ in a distributed computing environment. With these tools OpenMARS will be the basis of a highly potent modelling and analysis environment.

8. Acknowledgements

This work and its achievements are part of the global FCC study [62] hosted by CERN. In 2017 the OpenMARS approach won the FCC study innovation award [82], which is given for advancements with high innovation capacity, high socio-economic impact potential or high relevance for the concepts of a frontier particle physics research. The FCC study has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 654305 (EuroCirCol). In Finland, this work has been funded by Tampere University of Technology and Ramentor.

References

[1] E. Zio, Reliability engineering: Old problems and new challenges, Reliab. Eng. Syst. Saf. 94 (2) (2009) 125–141. doi:10.1016/j.ress.2008.06.002.

[2] E. Ruijters, M. Stoelinga, Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools, Comput. Sci. Rev. 15-16 (2015) 29–62. doi:10.1016/j.cosrev.2015.03.001.

[3] O. Lisagor, T. Kelly, R. Niu, Model-based safety assessment: Review of the discipline and its challenges, in: The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety, IEEE, 2011, pp. 625–632. doi:10.1109/icrms.2011.5979344.

[4] A. K. Verma, S. Ajit, D. R. Karanki, Reliability and Safety Engineering, Springer, London, 2016, Ch. 4, pp. 123–159. doi:10.1007/978-1-4471-6269-8_4.

[5] Ramentor, Event Logic Modelling and Analysis Software (ELMAS) (Jun 2018).
URL http://ramentor.com/elmas

[6] S. Virtanen, P. Hagmark, J.-P. Penttinen, Modeling and analysis of causes and consequences of failures, in: RAMS '06. Annual Reliability and Maintainability Symposium, 2006., IEEE, 2006, pp. 506–511. doi:10.1109/rams.2006.1677424.

[7] J.-P. Penttinen, T. Lehtinen, Advanced fault tree analysis for improved quality and risk assessment, in: Proceedings of the 10th World Congress on Engineering Asset Management (WCEAM 2015), Springer International Publishing, 2016, pp. 471–478. doi:10.1007/978-3-319-27064-7_45.

[8] M. Tammi, V. Vuorela, T. Lehtinen, Advanced RCM industry case – Modeling and advanced analytics (ELMAS) for improved availability and cost-efficiency, in: Proceedings of the 10th World Congress on Engineering Asset Management (WCEAM 2015), Springer International Publishing, 2016, pp. 581–589. doi:10.1007/978-3-319-27064-7_58.

[9] D. Imran Khan, S. Virtanen, P. Bonnal, A. K. Verma, Functional failure modes cause-consequence logic suited for mobile robots used at scientific facilities, Reliab. Eng. Syst. Saf. 129 (2014) 10–18. doi:10.1016/j.ress.2014.03.012.

[10] S. Virtanen, J.-P. Penttinen, M. Kiiski, J. Jokinen, Application of design review to probabilistic risk assessment in a large investment project, in: Proceedings of the Probabilistic Safety Assessment and Management (PSAM) 12 Conference - Volume 9, 2016, pp. 200–213.
URL http://psam12.org/proceedings/paper/paper_319_1.pdf

[11] A. Niemi, A. Apollonio, J. Gutleber, P. Sollander, J.-P. Penttinen, S. Virtanen, Availability modeling approach for future circular colliders based on the LHC operation experience, Phys. Rev. Accel. Beams 19 (2016) 121003. doi:10.1103/PhysRevAccelBeams.19.121003.

[12] J.-P. Penttinen, A. Niemi, J. Gutleber, An open modelling approach for availability and reliability of systems, Specification CERN-ACC-2018-0006, CERN, Geneva (2018).
URL https://cds.cern.ch/record/2302387

[13] Risk management – Risk assessment techniques, Standard IEC/ISO 31010:2009, International Organization for Standardization, Geneva (Nov 2009).

[14] D. Stamatis, The OEE Primer: Understanding Overall Equipment Effectiveness, Reliability, and Maintainability, CRC Press, Boca Raton, 2011.
URL https://books.google.fi/books?id=8i7NBQAAQBAJ

[15] P. Chelson, Reliablity computation using fault tree analysis, Technical report NASA-CR-124740, NASA, Pasadena (1971).
URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19720005773.pdf

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible markup language (XML) 1.0 (fifth edition), Specification, World Wide Web Consortium (W3C) (2008).
URL http://www.w3.org/TR/xml/

[17] S. Epstein, A. Rauzy, Open-PSA Model Exchange Format, Specification 2.0.d-120-g703be91, The Open-PSA Initiative (2017).
URL https://open-psa.github.io/mef/

[18] L. Hillah, E. Kindler, F. Kordon, L. Petrucci, N. Tréves, A primer on the Petri net markup language and ISO/IEC 15909-2, in: Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools,

Department of computer science University of Aarhus, Aarhus, 2009, pp. 101–120. doi:10.7146/dpb.v38i590.7187.

[19] O. Rey Orozko, A. Apollonio, M. Jonker, M. Paoluzzi, Dependability studies for CERN PS booster RF system upgrade, in: Proc. of International Particle Accelerator Conference (IPAC'16), Busan, Korea, May 8-13, 2016, no. 7 in International Particle Accelerator Conference, JACoW, Geneva, 2016, pp. 4159–4162. doi:10.18429/JACoW-IPAC2016-THPOY030.

[20] G. Sirin, C. J. J. Paredis, B. Yannou, E. Coatanéa, E. Landel, A model identity card to support simulation model development process in a collaborative multidisciplinary design environment, IEEE Syst. J. 9 (4) (2015) 1151–1162. doi:10.1109/JSYST.2014.2371541.

[21] E. Zio, The Monte Carlo Simulation Method for System Reliability and Risk Analysis, Springer London, 2013. doi:10.1007/978-1-4471-4588-2.

[22] Fault tree analysis (FTA), Standard IEC 61025:2006, International Electrotechnical Commission, Geneva (Dec 2006).

[23] Reliability block diagrams, Standard IEC 61078:2016, International Electrotechnical Commission, Geneva (Aug 2016).

[24] Application of Markov techniques, Standard IEC 61165:2006, International Electrotechnical Commission, Geneva (May 2006).

[25] Analysis techniques for dependability – Petri net techniques, Standard IEC 62551:2012, International Electrotechnical Commission, Geneva (Oct 2012).

[26] M. Malhotra, K. S. Trivedi, Dependability modeling using Petri-nets, IEEE Trans. Reliab. 44 (3) (1995) 428–440. doi:10.1109/24.406578.

[27] V. G. Kulkarni, Continuous-Time Markov Models, Springer, New York, 2011, Ch. 4, pp. 85–145. doi:10.1007/978-1-4419-1772-0_4.

[28] J. B. Dugan, K. S. Trivedi, R. M. Geist, V. F. Nicola, Extended stochastic Petri nets: Applications and analysis, Tech. rep., Durham (1984).

[29] Y. Sugasawa, Q. Jin, K. Seya, Extended stochastic Petri net models for systems with parallel and cooperative motions, Comput. Math. Appl. 24 (1) (1992) 119–126. doi:10.1016/0898-1221(92)90236-B.

[30] R. Pyke, Markov renewal processes: Definitions and preliminary properties, Ann. Math. Stat. 32 (4) (1961) 1231–1242.
URL http://www.jstor.org/stable/2237923

[31] M. H. Everdij, H. A. Blom, Petri-nets and hybrid-state Markov processes in a power-hierarchy of dependability models, IFAC Proc. Vol. 36 (6) (2003) 313–318, iFAC Conference on Analysis and Design of Hybrid Systems 2003, St Malo, Brittany, France, 16-18 June 2003. doi:10.1016/S1474-6670(17)36450-9.

[32] K. Jensen, L. M. Kristensen, Coloured Petri Nets, Springer Berlin Heidelberg, 2009. doi:10.1007/b95112.

[33] J.-P. Signoret, Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, P. Thomas, Make your Petri nets understandable: Reliability block diagrams driven Petri nets, Reliab. Eng. Syst. Saf. 113 (2013) 61–75. doi:10.1016/j.ress.2012.12.008.

[34] P. Pozsgai, B. Bertsche, Conjoint modelling with extended coloured stochastic Petri net and reliability block diagram for system analysis, in: Probabilistic Safety Assessment and Management: PSAM 7 – ESREL '04 June 14–18, 2004, Berlin, Germany, Volume 6, Springer, London, 2004, pp. 1382–1387. doi:10.1007/978-0-85729-410-4_223.

[35] M. Westergaard, H. E. Verbeek, CPN Tools (Jun 2018).
URL http://cpntools.org/

[36] SATODEV, GRIF-Workshop BStoK module (Jun 2018).
URL http://grif-workshop.com/grif/bstok-module/

[37] F. Long, P. Zeiler, B. Bertsche, Modelling the production systems in industry 4.0 and their availability with high-level Petri nets, IFAC-PapersOnLine 49 (12) (2016) 145–150, 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016. doi:10.1016/j.ifacol.2016.07.565.

[38] S. Kabir, An overview of fault tree analysis and its application in model based dependability analysis, Expert Syst. Appl. 77 (2017) 114–135. doi:10.1016/j.eswa.2017.01.058.

[39] S. Sharvia, S. Kabir, M. Walker, Y. Papadopoulos, Model-based dependability analysis: State-of-the-art, challenges, and future outlook, in: I. Mistrik, R. Soley, N. Ali, J. Grundy, B. Tekinerdogan (Eds.), Software Quality Assurance, Morgan Kaufmann, Boston, 2016, pp. 251–278. doi:10.1016/B978-0-12-802301-3.00012-0.

[40] Y. Papadopoulos, J. A. McDermid, Hierarchically performed hazard origin and propagation studies, in: M. Felici, K. Kanoun (Eds.), Computer Safety, Reliability and Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 139–152.

[41] P. Fenelon, J. McDermid, An integrated tool set for software safety analysis, J. Syst. Softw. 21 (3) (1993) 279–290.

[42] M. Wallace, Modular architectural representation and analysis of fault propagation and transformation, Electron. Notes Theor. Comput. Sci. 141 (3) (2005) 53–71, proceedings of the Second International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures (FESCA 2005). doi:10.1016/j.entcs.2005.02.051.

[43] M. Gudemann, F. Ortmeier, A framework for qualitative and quantitative formal model-based safety analysis, in: 2010 IEEE 12th International Symposium on High Assurance Systems Engineering, 2010, pp. 132–141. doi:10.1109/HASE.2010.24.

[44] P. Hönig, R. Lunde, F. Holzapfel, Model based safety analysis with smartIflow, Inf. 8 (1) (2017) 7. doi:10.3390/info8010007.

[45] T. Prosvirnova, AltaRica 3.0: a Model-Based approach for Safety Analyses, Theses, Ecole Polytechnique (2014).
URL https://pastel.archives-ouvertes.fr/tel-01119730

[46] A. Pfeffer, Practical Probabilistic Programming, Manning Publications Co., Shelter Island, 2016.
URL https://www.manning.com/books/practical-probabilistic-programming

[47] M. Batteux, T. Prosvirnova, A. Rauzy, AltaRica 3.0 language specfication, Specification, AltaRica Association (2015).
URL https://www.openaltarica.fr/docs-downloads/

[48] M. Lipaczewski, F. Ortmeier, T. Prosvirnova, A. Rauzy, S. Struck, Comparison of modeling formalisms for safety analyses: SAML and AltaRica, Reliab. Eng. Syst. Saf. 140 (2015) 191–199. doi:10.1016/j.ress.2015.03.038.

[49] A. Rauzy, Guarded transition systems: A new states/events formalism for reliability studies, Proc. Inst. Mech. Eng., Part O: J. Risk and Reliab. 222 (4) (2008) 495–505. doi:10.1243/1748006xjrr177.

[50] A. Rauzy, C. Blriot-Fabre, Towards a sound semantics for dynamic fault trees, Reliab. Eng. Syst. Saf. 142 (2015) 184–191. doi:10.1016/j.ress.2015.04.017.

[51] J. Noble, A. Taivalsaari, I. Moore, Prototype-Based Programming: Concepts, Languages and Applications, Springer-Verlag, London, 1999.

[52] J. Bang-Jensen, G. Gutin, Digraphs: Theory, Algorithms and Applications, 2nd Edition, Springer-Verlag, London, 2009. doi:10.1007/978-1-84800-998-1.

[53] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, 1995.

[54] R. Johnson, B. Foote, Designing reusable classes, J. Object-Oriented Program. 1 (2) (1988) 22–35.
URL http://www.laputan.org/drc.html

[55] S. Robinson, Simulation: The Practice of Model Development and Use, John Wiley & Sons, 2004.

[56] J. Penttilä, V. Ikonen, Dynamic simulation of dependability of nickel reduction plant, in: ASME 2013 International Mechanical Engineering Congress and Exposition, Vol. 15, ASME, 2013, p. V015T12A012. doi:10.1115/imece2013-64308.

[57] Y. Noh, K. Chang, Y. Seo, D. Chang, Risk-based determination of design pressure of LNG fuel storage tanks based on dynamic process simulation combined with Monte Carlo method, Reliab. Eng. Syst. Saf. 129 (2014) 76–82. doi:10.1016/j.ress.2014.04.018.

[58] K. Mahlamäki, A. Niemi, J. Jokinen, J. Borgman, Importance of maintenance data quality in extended warranty simulation, Int. J. COMADEM 19 (1) (2016) 3–10.

[59] E. Zio, P. Baraldi, E. Patelli, Assessment of the availability of an offshore installation by Monte Carlo simulation, Int. J. Press. Vessel. Pip. 83 (4) (2006) 312–320. doi:10.1016/j.ijpvp.2006.02.010.

[60] W. Hopp, M. Spearman, Throughput of a constant work in process manufacturing line subject to failures, Int. J. Prod. Res. 29 (3) (1991) 635–655. doi:10.1080/00207549108930093.

[61] J. Laitinen, P.-E. Hagmark, Modeling of closed spare part circulation for better availability of aircraft, in: Advanced Reliability Modeling IV – Beyond the Traditional Reliability and Maintainability Approaches, McGraw Hill, Taipei, 2010, pp. 409–416.

[62] M. Benedikt, F. Zimmermann, Proton colliders at the energy frontier, Nucl. Instrum. Methods Phys. Res. Sect. A: Accel. Spectrom. Detect. Assoc. Equip.doi:10.1016/j.nima.2018.03.021.

[63] A. Apollonio, Machine protection: Availability for particle accelerators, Ph.D. thesis, Vienna University of Technology, Vienna (2015).
URL https://cds.cern.ch/record/2002820

[64] A. Apollonio, M. Jonker, R. Schmidt, B. Todd, S. Wagner, D. Wollmann et al., HL-LHC: Integrated luminosity and availability, in: IPAC2013: Proceedings of the 4th International Particle Accelerator Conference, JaCoW, Geneva, 2013, pp. 1352–1354.
URL http://accelconf.web.cern.ch/AccelConf/IPAC2013/papers/tupfi012.pdf

[65] E. McCrory, P. Lucas, A model of the Fermilab collider for optimization of performance, in: Proceedings Particle Accelerator Conference, Vol. 1, IEEE, 1995, pp. 449–451. doi:10.1109/PAC.1995.504687.

[66] E. McCrory, Monte Carlo of Tevatron operations, including the recycler, in: Proceedings of the 2005 Particle Accelerator Conference, IEEE, 2005, pp. 2479–2481. doi:10.1109/PAC.2005.1591151.

[67] T. Himel, J. Nelson, N. Phinney, M. Ross, Availability and reliability issues for ILC, in: 2007 IEEE Particle Accelerator Conference (PAC), IEEE, 2007, pp. 1966–1969. doi:10.1109/PAC.2007.4441325.

[68] E. Bargalló, P. Sureda, J. Arroyo, J. Abal, A. De Blas, J. Dies et al., Availability simulation software adaptation to the IFMIF accelerator facility RAMI analyses, Fusion Eng. Des. 89 (2014) 2425–2429. doi:10.1016/j.fusengdes.2013.12.004.

[69] M. Motyka, Impact of usability for particle accelerator software tools analyzing availability and reliability, Master's thesis, Blekinge Institute of Technology, Blekinge (2017).

URL http://bth.diva-portal.org/smash/record.jsf?pid=diva2%3A1104892&dswid=398

[70] L. Hardy, Accelerator Reliability – Availability, in: Proceedings of EPAC 2002, CERN, Geneva, 2002, pp. 149–153.
URL https://accelconf.web.cern.ch/accelconf/e02/PAPERS/WEXLA001.pdf

[71] L. Burgazzi, P. Pierini, Reliability studies of a high-power proton accelerator for accelerator-driven system applications for nuclear waste transmutation, Reliab. Eng. Syst. Saf. 92 (4) (2007) 449–463. doi:10.1016/j.ress.2005.12.008.

[72] S. Sierla, I. Tumer, N. Papakonstantinou, K. Koskinen, D. Jensen, Early integration of safety to the mechatronic system design process by the functional failure identification and propagation framework, Mechatron. 22 (2) (2012) 137–151. doi:10.1016/j.mechatronics.2012.01.003.

[73] H. Langseth, L. Portinale, Bayesian networks in reliability, Reliab. Eng. Syst. Saf. 92 (1) (2007) 92–108. doi:10.1016/j.ress.2005.11.037.

[74] C. J. Garrett, S. B. Guarro, G. E. Apostolakis, The dynamic flowgraph methodology for assessing the dependability of embedded software systems, IEEE Trans. Syst. Man Cybern.: Syst. 25 (5) (1995) 824–840. doi:10.1109/21.376495.

[75] J. Moder, C. Phillips, Project management with CPM and PERT, Van Nostrand Reinhold Inc, New York, 1964.

[76] S. Nonsiri, E. Coatanéa, M. Bakhouya, F. Mokammel, Model-based approach for change propagation analysis in requirements, in: 2013 IEEE International Systems Conference (SysCon), IEEE, 2013, pp. 497–503. doi:10.1109/SysCon.2013.6549928.

[77] Quality assurance before series production: System FMEA, no. 4 part 2 in Quality management in the automobile industry, Verband der Automobilindustrie e.V. (VDA), Frankfurt am Main, 1996.

[78] C. Haskins (Ed.), INCOSE Systems Engineering Handbook, 3rd Edition, International Council on Systems Engineering, 2006, Ch. 10, pp. 10.1–10.6.

[79] J. Helton, J. Johnson, C. Sallaberry, C. Storlie, Survey of sampling-based methods for uncertainty and sensitivity analysis, Reliab. Eng. Syst. Saf. 91 (10-11) (2006) 1175–1209. doi:10.1016/j.ress.2005.11.017.

[80] A. Alrabghi, A. Tiwari, State of the art in simulation-based optimisation for maintenance systems, Comput. Ind. Eng. 82 (2015) 167–182. doi:10.1016/j.cie.2014.12.022.

[81] W. Vesely, F. Goldberg, N. Roberts, D. Haasl, Fault Tree Handbook, U.S. Nuclear Regulatory Commission, Washington, D.C., 1981, Ch. XI, pp. XI–2–XI–5.
URL https://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/

[82] J.-P. Penttinen, A. Apollonio, A. Niemi, J. Gutleber, A scalable and open platform for complex system behaviour assessment, FCC week 2017, Poster session (2017).
URL https://indico.cern.ch/event/556692/contributions/2592536/