CERN-ACC-2018-0006
08-02-2018
arto.niemi@cern.ch

Report

An Open Modelling Approach for Availability and Reliability of Systems — OpenMARS

Jussi-Pekka Penttinen[1,2], Arto Niemi [1,3] and Johannes Gutleber[3]
[1]Tampere University of Technology, Tampere, Finland
[2]Ramentor, Tampere, Finland
[3]CERN, Geneva, Switzerland

Abstract

This document introduces and gives specification for OpenMARS, which is an open modelling approach for availability and reliability of systems. It supports the most common risk assessment and operation modelling techniques. Uniquely OpenMARS allows combining and connecting models defined with different techniques. This ensures that a modeller has a high degree of freedom to accurately describe the modelled system without limitations imposed by an individual technique. Here the OpenMARS model definition is specified with a tool independent tabular format, which supports managing models developed in a collaborative fashion. Origin of our research is in Future Circular Collider (FCC) study, where we developed the unique features of our concept to model the availability and luminosity production of particle colliders. We were motivated to describe our approach in detail as we see potential further applications in performance and energy efficiency analyses of large scientific infrastructures or industrial processes.

Keywords: Reliability, Availability, Modelling, Risk assessment, Fault tree analysis, Reliability block diagram, Markov analysis, Failure mode and effects analysis, Petri net

Contents

1 Introduction

The Open Modelling approach for Availability and Reliability of Systems (OpenMARS) allows to define models with any of the most common risk assessment modelling techniques [1], such as Fault Tree Analysis (FTA), Reliability Block Diagram (RBD), Markov analysis, Failure Mode and Effects Analysis (FMEA) and Petri Net (PN). Annex D describes presently defined techniques but our approach is scalable and open to support also additional modelling techniques. Uniquely our concept allows combining and connecting models created with different techniques, which gives modellers a high degree of freedom to accurately describe the details and behaviours of the system they are modelling. Our recognition of this need is backed by more than decade's worth of experience in concrete use cases that go beyond scope of traditional risk assessment techniques [2, 3].

In addition to reliability and availability various other Key Performance Indicators (KPI) exists for a system. For instance, industry uses Overall Equipment Efficiency (OEE) calculations [4] to link availability and goods production. This link is not always trivial. For example, our particle collider availability model takes into account operation time and operational cycle to calculate the collision production [5]. For such cases the OpenMARS approach allows to combine operation and production models to risk assessment modelling techniques. We also introduce a freely definable production function, which allows modellers to include programming code to define any application specific KPI and their interactions.

This document gives specification on how to define models in platform independent and human readable tabular format. Tables are natural way to store the models into a database. We see possibility to use the tables as an open interface. The tools can be implemented for creation, visual presentation and analysis of the OpenMARS models. Also, it is feasible to create tools which convert the model information between our format and other formats enabling shared use of different reliability software.

In OpenMARS a model consist of *elements* that have *classes*. The class defines which kind of *attributes* the element has. Each modelling technique has a catalogue of available classes and our goal is that the most cases should be covered with them. To guarantee that our modelling concept is always applicable, we allow expert users to extend the model specific classes to tailor them for their special needs.

Our approach uses five different tables to define models. Chapter 2 describes the two class tables used for modelling technique definition. Chapter 3 explains the next two tables used for the model elements and structure definition. The model parametrization is done in the attribute value table, which is described in Chapter 4. In addition, our specification recognizes the model identification and tracking as the necessities for collaborative work. These features are covered in Chapter 5.

2 Tables for Class Definition

The OpenMARS approach is based on fundamental classes, which are described in Chapter 2.1. The class definition table (Chapter 2.2) specifies the other classes and the class inheritance. Each non-fundamental class inherits some features of a fundamental class.

Each modelling technique comes with a catalogue of built in classes. Chapter 2.2.1 shows an example of the Fault Tree Analysis (FTA) classes. More examples for various modelling techniques are collected to Annex D. Usually these build in classes should be sufficient to build models with presently defined techniques, but for special cases OpenMARS allows creation of custom classes. This process is explained in Chapter 2.2.2.

Every OpenMARS class has attributes that describe the class and its behaviour. These attributes are introduced and defined in the class attribute table, which is explained in Chapter 2.3.

2.1 Fundamental Classes

The three main fundamental classes in OpenMARS are: Element, Property and Folder. The element class further divides to four fundamental classes: Node, Operator, State and Transition. They form a basis for every modelling technique, which makes the combination of different techniques more straightforward. Examples 1 and 2 describe how the fundamental classes are used with FTA.

Example 1. A node object contains states and transitions. For a basic fault node the states are: normal and fault. Only one of them can be active at a certain time. The transitions define when the state changes are made. The transition from normal state to fault is called failure and the transition from fault to normal is called restoration. Figure 1 shows the resulting fault node with these states and transitions. These states and transitions form a Markov model [6] that composes the fault node.

Fig. 1: A fault node containing normal and fault states that are connected with failure and restoration transitions

Example 2. In OpenMARS approach the operators model the relations between nodes. OR, AND, Vote and other gates define the relations between fault nodes in FTA. Figure 2 shows how fault nodes are connected to gate operators to form a fault tree.

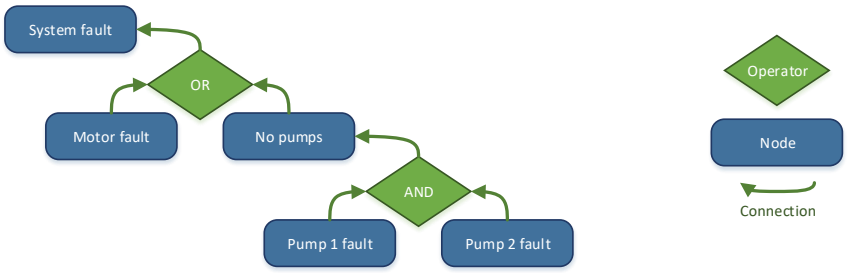

Fig. 2: An example fault tree model with five fault nodes and two gate operators

The folders improve the handling of large models. Figure 3 illustrates how a model can be divided in different system levels and how elements can be grouped in folders. Further benefits of folders are introduced in Chapter 3.

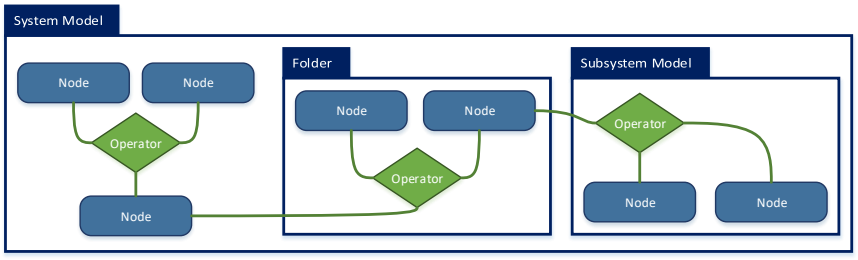

Fig. 3: Models can contain folders to hierarchically structure elements

The class defines what attributes an element has. An attribute can be another element or a property that stores a parameter value. A property can be for example a string or a number which are used for basic attributes such as title, description, cost or colour. The full list of property classes is defined in Annex B. Elements are structured attributes that have multiple properties. For example transitions defined in Annex C are attributes of nodes.

2.2 Class Definition Table

Each class used in the model is introduced class definition table, which also presents the inheritance between classes. The columns for the class definition table are CLASS and IS A. The CLASS column defines the name of the introduced class. IS A column defines the name of the class that the new class extends.

Following laws and a convention outline the proper use of class definition table:

Definition 3. The class names shall only consist of a-z, A-Z, 0-9 and underscore (_) characters.

Definition 4. The classes can be introduced irrespective of the order of the table rows. Thus, the inherited class can be introduced later than the inheritance is defined.

Definition 5. It is not allowed to introduce same class more than once.

Convention 6. Each class name starts with upper-case letter. [8]

Example 7. Table 1 shows how the fundamental classes from Chapter 2.1 are introduced.[1] Similarly with Java programming language the predefined Object class is a base for all classes in OpenMARS.

Table 1: The fundamental classes in class definition table

#	CLASS	IS A	COMMENT
1	Element	Object	Element is a model object which contains attributes
2	Property	Object	Properties are used as basic attribute values of elements
3	Folder	Object	Folders are for organizing the model elements
4	Node	Element	Node is an element which contains states and transitions
5	Operator	Element	Operators are elements which control node states and transitions
6	State	Element	State is an element which models whether a particular set of circumstances is active
7	Transition	Element	Transition is an element which defines when state changes are made

2.2.1 Examples of Build in Classes in the Class Definition Table

The FTA classes, introduced by Examples 1 and 2, are shown in Table 2. These classes form the basis for the Advanced FTA modelling technique, which is further defined in Annex D.

Table 2: Example of built in FTA classes in the class definition table

#	CLASS	IS A	COMMENT
1	Fault	Node	Node to model faults in FTA modelling technique
2	Gate	Operator	Logic operator in FTA modelling technique
3	OR	Gate	A logic operator which defines rule ”At least one”
4	AND	Gate	A logic operator which defines rule ”All”
5	Vote	Gate	A logic operator ”At least m” or ”Redundancy k out of n”, where m = n-k+1

Table 3 shows the property classes mentioned in Chapter 2.1 in the class definition table. Annex B contains the full property class documentation.

[1] Additional columns exist for the definition index and for the comments. The index number helps to refer a certain row and the comment gives a background information about the definition. The content of these two columns is not included in the defined model.

Table 3: Example of built in property classes in the class definition table

#	CLASS	IS A	COMMENT
1	String	Property	Unformatted text
2	Name	String	A String with only characters a-z, A-Z, 0-9 and _ allowed
3	Colour	String	Predefined colour name or a specification in a supported scheme
4	Boolean	String	Either true or false
5	Number	Property	Any number
6	Integer	Number	A whole number without fractional component (only integer literals allowed)
7	Probability	Number	A number with only a value between 0 and 1 allowed, with 0 and 1 included
8	Duration	Number	Time interval given with value and time unit (number can be calculated by multiplying the value with the factor that corresponds the time unit)

2.2.2 Model Specific Class Definition

An expert user can extend a built in class to create a model specific class. This can be needed to include a distinct feature that is not covered by existing classes. To ensure consistency of models the users cannot directly change the built in classes.

Example 8. Here a modeller chooses to introduce specific classes for mechanics and electronics related faults. They extend the fault node. Creating these new classes is shown in Table 4.

Table 4: Example of two user defined classes that extend the fault class

#	CLASS	IS A	COMMENT
1	MechanicsFault	Fault	Model specific node for mechanics faults
2	ElectronicsFault	Fault	Model specific node for electronics faults

2.3 Class Attribute Table

Attributes describe a class and its behaviour in a specific modelling technique. The objects of certain class can only contain attributes that are introduced for its class. Types and names of these attributes are defined in the class attribute table.

The class attribute table contains three columns: CLASS, TYPE and ATTR (attribute). The CLASS column defines the name of the class to which the attribute is associated. The TYPE column defines a class name that indicates the data format and range of the property or the structure of the attribute. The ATTR column defines the name of the introduced attribute.

Similarly with the class definition table, the class attribute table has an index and a comment column. Following laws and a convention outline the proper use of class attribute table:

Definition 9. The attribute names shall only consist of a-z, A-Z, 0-9 and underscore (_) characters.

Definition 10. It is not allowed to introduce same attribute more than once.

Convention 11. Each attribute name starts with lower-case letter. [8]

Example 12. Table 5 shows how to introduce attributes for the classes.

Table 5: Examples of introducing properties for the classes

#	CLASS	TYPE	ATTR	COMMENT
1	Node	String	title	The node title which can be shown in graphical user interface
2	Node	Colour	background	The background colour of the node shown in graphical user interface
3	Node	Name	initial	The name of the initial state of the node
4	Vote	Integer	atLeast	The number source node faults needed at least for target node fault

Folders and some elements are container objects that can contain other elements. In OpenMARS the contained elements are seen as attributes of the container. The container definition specifies the allowed classes for the contained elements. A container object can contain any number of elements of the specified class but they each need to have a distinct name. The container definition is made by giving the asterisk (*) symbol in the ATTR column.

Example 13. Table 6 shows definition that allows folders to contain any elements and subfolders, and a definition that allows nodes to contain only states and transitions.

Table 6: Container definitions for folders and nodes

#	CLASS	TYPE	ATTR	COMMENT
1	Folder	Element	*	Folders can contain any elements
2	Folder	Folder	*	Folders can contain also sub folders
3	Node	State	*	Nodes can contain states
4	Node	Transition	*	Nodes can contain transitions

3 Tables for Defining Model Elements and Structure

Each element and folder present in the model is defined in the element creation table, which is described in Chapter 3.1. The element connection table (Chapter 3.2) defines the relations between the elements. One of our goals for the table based model definition was to make defining vast models easy. Here we were motivated PSB RF-system reliability study, where the modelled system has multiple similar repetitive structures [7]. Chapter 3.3 introduces short-cuts that simplify the definition of repetitive structures. Furthermore, element tables are used to predefine class elements. This is explained in Chapter 3.4 that extends the information in Chapter 2.

3.1 Element Creation Table

The element creation table defines the elements and their location in a model. The element creation table has three columns: CONT (container), ELEMENT and CLASS. The CONT column defines element location, which is the container object (a folder or an element) that contains the specified element. The ELEMENT column defines the name of the created element and the CLASS column defines the name of the element class.

Following laws and a convention govern the proper use of the element creation table:

Definition 14. The element names shall only consist of a-z, A-Z, 0-9 and underscore (_) characters.

Definition 15. The elements can be created irrespective of the order of the table rows. Thus, the container object can be defined after it is used as a location for an element.

Definition 16. It is not allowed to create same element more than once.

Convention 17. Each element name should start with lower-case letter. [8]

Each folder and element has an Unique Identifier (UID). It is created by combining the UID of the container object with the name defined in the ELEMENT column, where the container object UID and the name are delimited by a slash (/) symbol. The empty value in the CONT column means that the default base folder of the model is used as a location for the created object.

Example 18. Table 7 shows how to create a folder to the default base folder. The created folder contains four elements. The resulting UIDs of the created objects are shown in the COMMENT[2] column. Figure 4 illustrates the created elements.

Table 7: Examples how create elements

#	CONT	ELEMENT	CLASS	COMMENT
1		system	Folder	A folder for the system model created to the default base folder, UID: /system
2	/system	systemFault	Fault	A fault of the system, UID: /system/systemFault
3	/system	systemOR	OR	A logic OR operator of the system, UID: /system/systemOR
4	/system	noPower	Fault	No power to the system, UID: /system/noPower
5	/system	motorFault	Fault	A motor fault of the system, UID: /system/motorFault

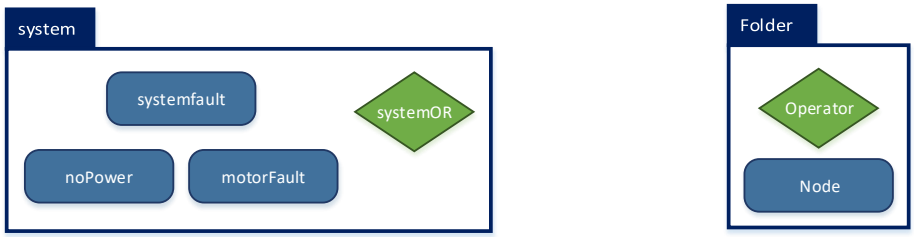

Fig. 4: The three fault nodes and an OR-gate are in a system folder

3.2 Element Connection Table

The element connection table defines the relations between elements. We implemented this as a set of ordered pairs, which is a straightforward technique to represent finite directed graphs [9]. The element connection table has two columns: SOURCE and TARGET. The correct direction of the connection is based on convention set up in the modelling technique.

Example 19. Table 8 shows how to add connections between fault tree elements created in Example 18. The chosen convention to fault tree is that the child fault is the SOURCE and the parent fault is the TARGET. Figure 5 illustrates the resulting model structure.

Table 8: Examples how to add connections between elements using element UID

#	SOURCE	TARGET	COMMENT
1	/system/systemOR	/system/systemFault	System OR gate connected to system fault
2	/system/noPower	/system/systemOR	No power connected to system OR gate
3	/system/motorFault	/system/systemOR	Motor fault connected to system OR gate

[2]Additional columns exist for the definition index and for the comments. The index number helps to refer a certain row and the comment gives a background information about the definition. The content of these two columns is not included in the defined model.

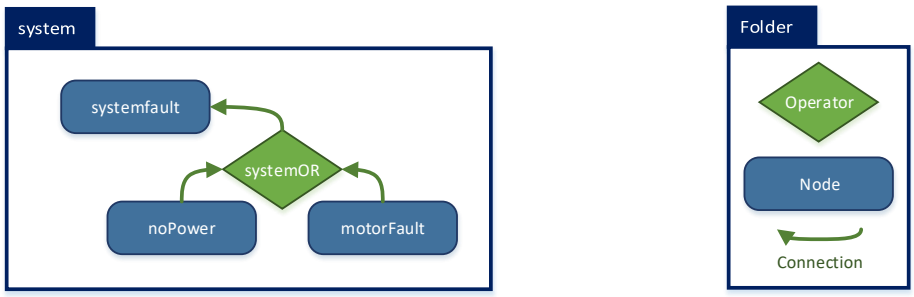

Fig. 5: The model structure with three fault nodes connected with one OR operator

Similarly with other tables, following law governs the proper use of the element connection table:

Definition 20. It is not allowed to add same connection more than once.

It is also possible to add a connection from or to an attribute of an element. This is needed for example with mathematical function models when defining the property value used as a parameter for the function (Annex D).

3.3 Methods to Ease Definition of Vast Models

3.3.1 Array and Reference Definitions in Element Creation Table

The array definition allows to create multiple similar elements to same container using only one table row. This definition is made by adding index inside square brackets as a suffix of the element name, which results as a syntax: *elementName[index]*. Table 9 shows examples on what can be used as an index.

Table 9: Examples on what can be used as array definition index

Description	Example
Integer	1
Name	first
List of integers	1,2,3
List of names	first, second, third
Integer interval	1-4 *(meaning: 1,2,3,4)*
Character interval	a-d *(meaning: a,b,c,d)*

A reference definition is created by using the container object name instead of UID in the CONT column. Example 21 shows how array and reference definitions can be used.

Example 21. The first row of Table 10 creates three pump folders with an array definition. By using the pump folder name as a reference definition the three fault nodes and an AND gate are created to all three folders. Figure 6 shows the resulting elements. Table 11 further highlights how to use reference definitions by showing different ways to refer the pumpFault nodes.

Table 10: Examples how to create multiple elements using array definition

#	CONT	ELEMENT	CLASS	COMMENT
6		pumps[1-3]	Folder	Three similar pump model folders
7	/pumps	pumpingFault	Fault	A pumping fault for each folder
8	/pumps	pumpingAND	AND	A logic AND operator of the pumping for each folder
9	/pumps	pumpFault[left,right]	Fault	Two similar pump faults for each folder

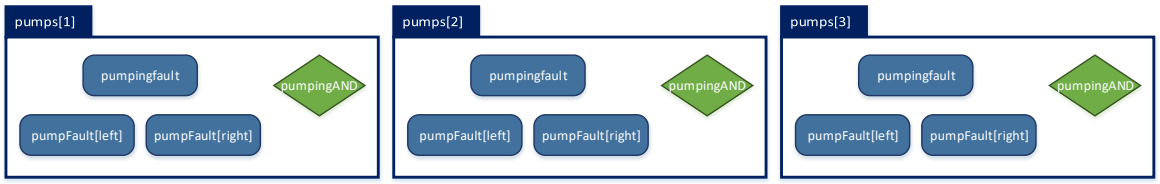

Fig. 6: The twelve elements in three folders created with only four definition rows

Table 11: Examples of available reference definitions

#	DEFINITION	COMMENT
1	/pumps[1]/pumpFault[left]	Element UID used to refer to certain node
2	/pumps[1]/pumpFault	Refers to both nodes in first pumps folder (2 exists)
3	/pumps[2-3]/pumpFault	Refers to both nodes in two last pumps folders (4 exists)
4	/pumps/pumpFault[left]	Refers to all left pump fault nodes in all pumps folder (3 exists)
5	/pumps/pumpFault	Refers all pump fault nodes in all pumps folders (6 exists)
6	pumps/pumpFault	Similar to #5, but refers to all pump fault nodes in every pumps folder in the project (not just the ones in the default folder)
7	pumpFault	Refers to every pump fault nodes in any project folder

3.3.2 Reference Definition in Connection Table

The reference definition (Chapter 3.3.1) can be used to efficiently add connections within containers. This simplifies connection definitions in vast models as single row can be used to define a connection in each container that has the same name.

Example 22. Table 12 extends Example 21 by defining the connections shown in the Figure 7. If both source and target elements contain reference definition, the connection is made only between elements in a common container object.

Table 12: Examples on using reference definitions with connections.

#	SOURCE	TARGET	COMMENT
4	pumpingAND	pumpingFault	Pumping AND gate connected to pumping fault
5	pumpFault	pumpingAND	Both pump faults connected to pumping AND gate

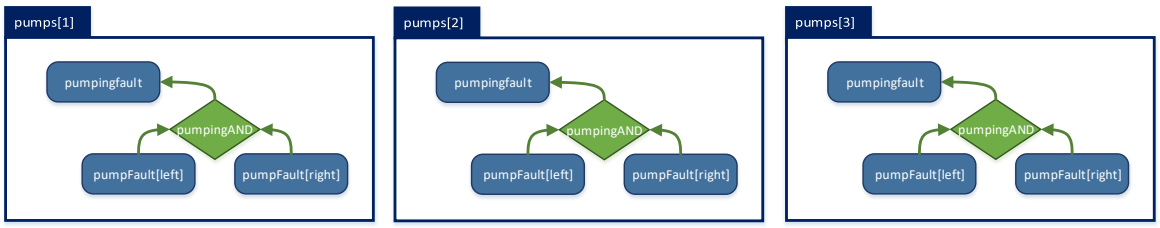

Fig. 7: Illustration on connections defined in Table 12

Example 23. The reference definition can be used also when connecting elements located in different folders, but the folder needs to be specified. Table 13 extends Examples 18 – 22 and shows how to connect elements in pumps folders to an OR-gate in system folder. Here it is notable that the definition without specified folders (*pumpingFault -> systemOR*) would not create any connections as the elements are in different containers. Figure 8 shows the resulting connections.

Table 13: Example how to use reference definition to add connection between elements from different folder

#	SOURCE	TARGET	COMMENT
6	/pumps/pumpingFault	/system/systemOR	All pumping fault nodes connected to system OR gate

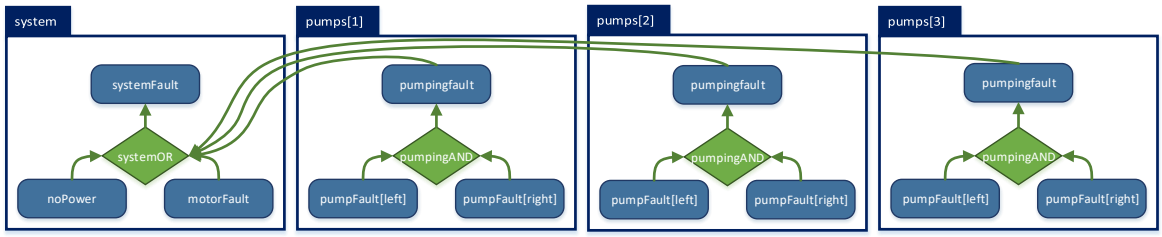

Fig. 8: Combination of models created to different folders

3.4 Predefined Class Elements

Element creation and connection tables can be used to further define element classes. The class elements created for each object of a specific class are defined by giving the class name in the CONT column of the element creation table. This is an optional way for defining class attributes (Chapter 2.3). Example 24 compares these two ways. The class element definition is allowed only if a class objects are defined as a containers for the created element (see Example 13).

Example 24. This example shows two ways to define elements contained in the Fault class. Figure 9 reintroduces the elements of the Fault class that were first shown in Example 1. Table 14 shows how the elements can be created in the class attribute table and Table 15 shows the same definitions in the element creation table. Using the element creation table for these definitions is allowed as the nodes are defined as containers for states and transitions.

Fig. 9: A fault node with normal and fault states and failure and restoration transitions

Table 14: Introduction of class elements attributes in the class attribute table

#	CLASS	TYPE	ATTR	COMMENT
10	Fault	State	normal	Normal state for all node elements of class Fault
11	Fault	State	fault	Fault state for all node elements of class Fault
12	Fault	Transition	failure	Transition which changes the node state from normal to fault
13	Fault	Transition	restoration	Transition which changes the node state from fault to normal

Table 15: Introduction of class elements in the element creation table

#	CONT	ELEMENT	CLASS	COMMENT
10	Fault	normal	State	Normal state for all node elements of class Fault
11	Fault	fault	State	Fault state for all node elements of class Fault
12	Fault	failure	Transition	Transition which changes the node state from normal to fault
13	Fault	restoration	Transition	Transition which changes the node state from fault to normal

Regardless on where elements are defined the element connection table is always used to add connections between class elements. Example 25 shows how the predefined connections for the Fault class are made.

Example 25. Table 16 shows how the connections are added between class elements of faults.

Table 16: Example on defining class element connections

#	SOURCE	TARGET	COMMENT
7	Fault/normal	Fault/failure	Failure transition starts from the normal state.
8	Fault/failure	Fault/fault	Failure transition leads to the fault state.
9	Fault/fault	Fault/restoration	Restoration transition starts from the fault state.
10	Fault/restoration	Fault/normal	Restoration transition leads to the normal state.

4 Attribute Value Table for Model Parameter Definition

The attribute value table defines the values for class and element attributes. The table consist of three columns: OBJECT, ATTR (attribute) and VALUE. The OBJECT column contains the name of the class or the element for which the attribute value is defined. The ATTR column contains the name of the attribute and the VALUE column the assigned value.

Annex B is dedicated for introducing the different data types. The attribute value table can contain also two optional columns: UNIT and FORMAT. In the UNIT column it is possible to define how quantities of the property are measured and the FORMAT column gives an indication in which format the value is provided. If these columns are omitted, a standard units and formats are assumed.

It is not allowed to add same attribute for same element more than once. However, in addition to specific element, a value can be defined for a class or a group of elements by using reference definition (Chapter 3.3). These levels form a hierarchy that permits overwriting values. This is highlighted by Example 26, which show definitions in different hierarchy levels. For more practical use case, Example 27 shows how failure and restoration times can be assigned.

Example 26. Table 17 shows examples how to define attribute values. The definitions are listed in overwrite order. In a fault tree this example would make fault nodes green and pumpFaults yellow, except for a one pumpFault, which would be red.

Table 17: Examples of property value definition highlighting the overwrite order

#	OBJECT	ATTR	VALUE	COMMENT
1	Node	background	blue	Default colour for all node objects
2	Fault	background	green	Overwrite of the default colour for fault objects
3	pumpFault	background	yellow	Overwrite of the colour for all the elements named pumpFault
4	/pumps[1]/pumpFault[left]	background	red	Overwrite of the colour for a specific pumpFault element

Example 27. This example shows how to define attribute values for failure and repair transitions that are attributes of the Fault class. Table 18 shows examples how to define the transitions of fault nodes. By default a failure transition uses exponential distribution, but the row 3 of Table 18 shows how to change this for specified nodes. Table 19 shows how to define values for the transition properties. The property data types and transitions are more formally addressed in Annexes B and C.

Table 18: Transition definition for fault nodes

#	OBJECT	ATTR	VALUE	COMMENT
1	Fault	failure	TransExp	By default the failure transition is exponential.
2	Fault	restoration	TransExp	By default the restoration transition is also exponential.
3	pumpFault	failure	TransWeibull	Weibull distribution is used for failure of all pump faults

Table 19: Adding property values for fault node transitions

#	OBJECT	ATTR	VALUE	COMMENT
4	motorFault/failure	mean	2a	Mean time to failure of exponential distribution
5	motorFault/restoration	mean	2d	Mean time to restoration of exponential distribution
6	pumpFault/failure	scale	10d	Shape parameter of Weibull distribution
7	pumpFault/failure	shape	1.5	Scale parameter of Weibull distribution
8	pumpFault/restoration	mean	5h	Mean time to restoration of exponential distribution

5 Model Information Tables

Every model carries information that helps tracing the model origins and evolution over time. In the OpenMARS approach the model table (Chapter 5.1) contains information about the model origin and the changes table (Chapter 5.2) about the revisions.

5.1 Model Table

The model table contains model meta data. On minimum a model should contain information listed by Table 20.

Table 20: Minimum information about a model

Item	Description
Name	A short and concise name, describing what this model contains
Identifier	A unique identifier (UID) of the model that is used to associate contents to a model. This identifier may be software or human generated.
Description	A textual description of the model, explaining the purpose and scope
Author	The name of the author along with sufficient information to contact the author (e.g. affiliation, address, phone, email)
Date	The creation date of the model

The model table consist of four columns: GROUP, PROPERTY, VALUE and COMMENT. Table 21 shows the recommended meta data groups for the GROUP column. Annex A lists the recommended meta data properties for each group. The PROPERTY column has the property name and the VALUE column defines the value for the property. An example on the model table format is given in Table 22.

Table 21: Recommended meta data groups

Group	Description
General	Information about the model
CM	Configuration and revision management related information
Access	Access management related information
Legal	Legal and license related information
Directive	Directives related to the model parsing and processing, e.g. include of other models

Table 22: Example of model meta information in the Model Table

#	GROUP	PROPERTY	VALUE	COMMENT
1	General	Name	Collider_Availability	General model for hadron collider availability
2	CM	Status	IN WORK	
3	Legal	Copyright	CERN	

5.2 Changes Table

In order to track the evolution of the model, each revision is documented as seen in Figure 10. A large model can contain models of individual subsystems and technical elements, which evolve over time. Figure 11 shows an example with individual revision information for a subsystem model. A model evolves over time. From case-to-case, a model revision may use the same or different revisions of a sub-system model.

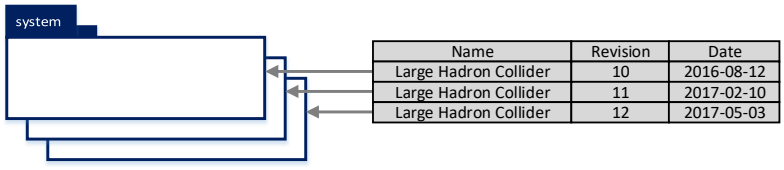

Fig. 10: A model evolves over time. Each revision needs to be tracked.

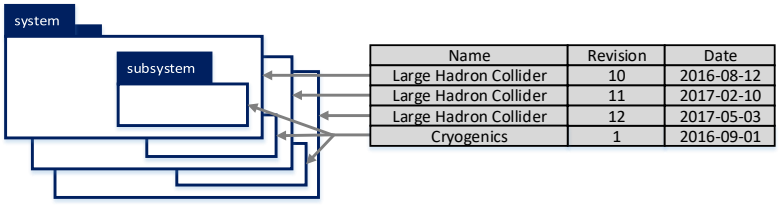

Fig. 11: A system model contains sub-system models that can have their own version histories.

The minimum revision information shown in Table 23. In the changes table this information is presented with four columns REV (revision), DATE, AUTHOR, STATUS and COMMENT.

Table 23: Minimum information about a revision

Item	Description
Revision	A unique revision identifier. This may be a running number, a version number or an arbitrary revision identifier possibly assigned also by a person.
Date	The date of the revision in format YYYY-MM-DD
Author	The author of this revision along with sufficient information to contact the author (e.g. affiliation, address, phone, email)
Status	Additional information that helps understanding, for which purpose this particular revision can be used. This information is usually subject to a configuration management plan that describes status and associated work flows. It is considered good practice, to distinguish at least between a revision, which is in work and a revision that is released for use by other persons.
Changes	A brief summary of the changes in this revision with respect to the preceding revision

Example 28. An example of revision table is shown in Table 24.

Table 24: An example of a revision table

#	REV	DATE	AUTHOR	STATUS	CHANGES
1	0.02b	2017-10-06	Author A	IN WORK	Other version on subsystem fault modelling based on alternative system implementation
2	0.02a	2017-10-05	Author B	IN WORK	Detailed to subsystem fault modelling
3	0.01	2017-10-02	Author A	IN WORK	Initial model

6 Acknowledgements

In 2017 the OpenMARS approach got the FCC Study Innovation Award, 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 infrastructure. The development of the concept was done within scope of an R&D project between CERN and Ramentor Oy. This work is a part of the global future circular collider study hosted by CERN. 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, the work for the FCC availability study has received funding from Tampere University of Technology and Ramentor Oy.

7 References

[1] European Committee for Electrotechnical Standardization, Risk management. Risk assessment techniques, EN 31010:2010, (2010).

[2] S. Virtanen, P.-E. Hagmark and J.-P. Penttinen, Proc. of the RAMS '06. Annual Reliability and Maintainability Symposium, (IEEE, 2006), pp. 506–511.

[3] J.-P. Penttinen and T. Lehtinen, Proc. of the 10th World Congress on Engineering Asset Management (WCEAM 2015), (Springer, Cham, 2016), pp. 471–478.

[4] T. Kanti Agustiady and E.-A. Cudney, *Total Productive Maintenance: Strategies and Implementation Guide* (CRC Press, Boca Raton, 2015), p. 111.

[5] A. Niemi *et al. Phys. Rev. Accel. Beams* **19** (2016) 121003.

[6] International Electrotechnical Commission, Application of Markov techniques, IEC 61165:2006, (2006).

[7] O. Rey Orozko *et al.* Proc. 7th Int. Particle Accelerator Conf. IPAC'16, (JaCoW, 2016), pp. 4159–4162.

[8] J. Gosling *et al.*, *The Java® Language Specification, Java SE 8 Edition*, (Oracle America, Inc, Redwood City, 2015), pp. 134–139.

[9] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, (Springer-Verlag, 2002), p. 2.

Appendices

A Annex A: Meta Data Group Properties for the Model Table

Tables A.1 – A.5 describe the recommended properties for each recommended meta data group.

Table A.1: Recommended meta data properties for general group

Group	Property	Type	Description
General	Name	Text	Concise name of this model that will be used as identifier. The name shall only consist of a-z, A-Z, 0-9 and underscore (_) characters. The name should be as short and precise as possible. For a comprehensive description, the purpose and scope properties should be used.
General	Purpose	Text	A comprehensive description of the model's purpose
General	Scope	Text	A comprehensive description of the model's scope
General	Documentation	Text	A text with a hyperlink (Text to Display and Address) pointing to further documentation
General	Author	Text	The full name of the author
General	Email	Text	The e-mail address of the author
General	Phone	Text	The telephone number of the author including international prefix
General	Tags	Text	A comma separated list of tags in name format (a-z,A-Z,0-9,_) that helps indexing and searching for this model. The use of whitespace characters in the comma separated list is not allowed.
General	Creation date	Text	Date in format YYYY-MM-DD
General	Creation time	Text	Time in format HH:MM
General	Organization	Text	The organization, who defined or owns this model (e.g. CERN)
General	Unit	Text	The organizational unit within the organization, who defined or owns this model
General	Country	Text	
General	Street	Text	
General	uid	Text	
General	Town	Text	
General	Zip	Text	

Table A.2: Recommended meta data properties for CM group

Group	Property	Type	Description
CM	Revision	Text	The revision identifier of this model. If the file contains only a subset of the model, the other subsets of the model in other files must carry the same revision identifier.
CM	Status	Text	The revision status. Different configuration management processes can be defined. The following status values are recommended: IN WORK, RELEASED, DEPRECATED
CM	Tag	Text	A readable revision tag to help finding back frequently used revision. As an example, the latest revision, which is in status IN WORK might be tagged HEAD or a particular released revision may carry an easy to remember name such as ALPHA_VERSION, BETA_VERSION.

Table A.3: Recommended meta data properties for access group

Group	Property	Type	Description
Access	Visibility	Text	
Access	ACL	Text	An ”Access Control List” property that can be used to define a role, group or person who can access this model using certain permissions.

Table A.4: Recommended meta data properties for legal group

Group	Property	Type	Description
Legal	Copyright	Text	
Legal	License	Text	
Legal	Disclaimer	Text	

Table A.5: Recommended meta data properties for directive group

Group	Property	Type	Description
Directive	Include	Text	URL to another model or model file that needs to be retrieved, parsed and processed before the contents of this model can be parsed and processed without error.
Directive	Use	Text	Indicate what to use from the included models. If nothing is indicated all linked models will be included entirely. Here it is possible to indicate paths to certain contained folders or elements according to the syntax used in the Class definition and Element definition tables.
Directive	Accept	Text	Indicate for a particular model, which version is accepted.

B Annex B: Property Data Types

This annex documents property data types, which are classes that are used for basic attribute values of elements. These classes are universally used in OpenMARS. Table B.1 lists the property data type classes. If needed, a user can define new classes that are based on the built in property data types.

Table B.1: Property classes which can be used for basic attribute types

#	CLASS	IS A	COMMENT
1	String	Property	Unformatted text
2	Text	String	Formatted text
3	UID	String	A string which is proper object UID
4	Name	String	A string with only characters a-z, A-Z, 0-9 and _ allowed
5	Colour	String	String that refers to predefined colour name or a specification in a supported scheme
6	URL	String	Reference to external location (as defined in [B1])
7	Boolean	String	Either true or false
8	Number	Property	Any number
9	Integer	Number	A whole number without fractional component
10	Probability	Number	A number with only a value between 0 and 1 allowed, with 0 and 1 included
11	Duration	Number	Time interval given with value and time unit (number can be calculated by multiplying the value with the factor that corresponds the time unit)
12	Rate	Duration	Value per time unit (duration can be calculated by dividing the time unit with the value)
13	Time	Number	Time in format YYYY-MM-DDThh:mm:ss (as defined in [B2])

B.1 String

String property value definition consist of zero or more characters. OpenMARS approach allows also String literals that are not enclosed in double quotes. Characters can be either a basic input character or an escape sequence (see [B3]). The property classes inherited from the String class can contain restrictions, which are defined in COMMENT column of Table B.1. Any string object can return a value, which is a sequence of characters.

B.2 Number

Number property value definition can be either an integer literal or a floating point literal. Integer literals are integer values that fit in 64 bits. Floating point literal values are represented using the IEEE 754 64-bit double-precision binary floating-point formats [B4]. The property classes inherited from the Number class can contain range or other restrictions, which are defined in COMMENT column of Table B.1. Any number object can return a value, which can be converted to 64 bit integer (long), 32 bit integer (int) or any other number format needed.

B.3 Duration and Rate

The Duration and Rate values require a time unit for defining the values. Following time units are used:

- s (second), factor: 1/3600
- m (minute), factor: 1/60
- h (hour), factor: 1
- d (day), factor: 24
- a (year, annus), factor: 8766

The factor shows the units magnitude compared to 1 hour that is the base duration in the OpenMARS. The time unit is defined in the UNIT column in the Attribute Value Table or in the VALUE column with the specified duration value. A Duration can be even defined with multiple value-unit pairs in the

VALUE column *(e.g. 3d 5h 12m)*. For Rate values the time unit is defined with a slash (/) character as a prefix for the time unit *(e.g. /h)*.

B.4 Array and Map Definition

Any property can be defined as an array or a map. Arrays are defined by adding square brackets as a suffix of the property name. The array value is defined as a comma separated list in the VALUE column.

A map is defined by adding a map key inside square brackets as a suffix of the property name. Note, also an array is considered as a map with index as a key. In context of arrays and maps, a single value definition is considered as the first value of an array definition or the value with empty string key of the map definition.

Example 29. Table B.2 shows examples of different value definitions.

Table B.2: Examples of array and map definitions

#	OBJECT	ATTR	VALUE	COMMENT
1	anObject	singleValue	5	Number 5 is put as single value to attribute singleValue
2	anObject	arrayValue[]	1,2,3	Three numbers are put as values to attribute arrayValue
3	anObject	arrayValue[4]	4	Number 4 is put as the fourth item in the arrayValue list
4	anObject	mapValue[someKey]	4	Number 4 is put attribute mapValue with the key *someKey*
5	anObject	mapValue[otherKey]	5	Number 5 is put attribute mapValue with the key *otherKey*

B.5 Defining Links to External Data

A property value can also be read from an external location. This is done by giving the Uniform Resource Locator (URL) for an attribute with name url. The urlQuery attribute can be used to define components of the URL (see [B1], Ch. 3.4), which give more information how the data is read.

Example 30. Table B.3 shows way how to link to external data in the attribute value table[3]. The rows 6 and 7 shows simple link and file location definitions. The rows 8 - 11 show how the initial URL can be extended with url components by using the urlQuery. The use of the urlQuery is not strictly necessary, the row 12 shows how similar information could be presented as an URL.

Table B.3: URL and format to read a property value from external location

#	OBJECT	ATTR	VALUE	COMMENT
6	xxx/failure/mean	url	http://www.cern.ch/.../data.xml	From xml file
7	xxx/restoration/mean	url	restorationtime.txt	From txt file
8	yyy/failure/scale	url	http://www.cern.ch/.../file.xlsx	From xlsx file
9	yyy/failure/scale/urlQuery	format	Excel	Define the file format
10	yyy/failure/scale/urlQuery	column	A	Define location
11	yyy/failure/scale/urlQuery	row	9	Define location
12	yyy/failure/shape	url	.../file.xlsx?format=Excel&cell=B2	With URL query

The OpenMARS specification gives a freedom for implementations to define the allowed urlQuery values and how they are interpreted. The URL properties could be used also to read array or map values by giving a range of cells in a spreadsheet or to run programs by giving a link to executable file that can return data.

[3]Here the reader is assumed to be familiar with Example 27 in Chapter 4.

B.6 References

[B1] T. Berners-Lee, R. Fielding and L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, (The Internet Society, 2005).

[B2] International Organization for Standardization, Data elements and interchange formats – Information interchange – Representation of dates and times, ISO 8601:2004, (2004).

[B3] J. Gosling *et al.*, *The Java® Language Specification, Java SE 8 Edition*, (Oracle America, Inc, Redwood City, 2015), Ch. 3.10.5, pp. 35–37.

[B4] IEEE Computer Society, IEEE Standard for Floating-Point Arithmetic, IEEE Std 754™-2008, (IEEE, New York, 2008).

C Annex C: Transitions

In OpenMARS stochastic durations such as state changes and delays are defined by transition classes. This annex introduces the classes listed in Table C.1 by using the notation in Table C.2. The list is not all inclusive. New class can be defined for any distribution function.

Table C.1: Transition classes, as a naming convention a prefix "Tr" is used with all transition class names.

#	CLASS	IS A	COMMENT
1	TrExp	Transition	Exponential distribution
2	TrConst	Transition	Constant transition time without deviation
3	TrWeibull	Transition	Weibull distribution
4	TrNorm	Transition	Normal distribution
5	TrLognorm	Transition	Log-normal distribution
6	TrProb	Transition	A probability to define if an event occurs or not. Mathematically this is similar to binomial trial.
7	TrHistory	Transition	Distribution class used when there is history data available

Table C.2: General notation in Annex C

Symbol	Description
F_X	Cumulative distribution function of transition X
Q_X	Generalized inverse distribution function of transition X
U	Next random event time from the distribution
t_c	Current simulation clock time
μ	Mean or expected value
σ	Standard deviation
u	Uniform random variable in the interval (0, 1)
$\exp(x)$	Natural exponential function e^x
$\ln(x)$	Natural logarithm function $\log_e(x)$

The transitions are defined as distributions for the transition duration. A cumulative distribution function

$$F_X(x) = \mathrm{P}(X \leq x) = u \tag{C.1}$$

gives the probability *u* that the transition *X* has occurred with a time value less than or equal to time *x*. To obtain results by using Monte-Carlo simulation it is necessary to generate random variables from the specified distribution. For this purpose methods such as inversion technique [C1] are used. The inversion technique requires generalized inverse distribution function

$$Q_X(u) = \inf_{x \in \mathbb{R}} \{F_X(x) \geq u\} \tag{C.2}$$

as the quantile function

$$F_X^{-1}(u) = Q_X(u), \text{ if } F_X \text{ is continuous and strictly monotonically increasing} \tag{C.3}$$

is not always defined. During the simulation process the random next event times are calculated by adding the transition time obtained from C.2 to current simulation clock time:

$$U(u) = t_c + Q_X(u) \tag{C.4}$$

C.1 Exponential Distribution (TrExp)

Exponential distribution assumes constant failure rate $1/\mu$. The mean value is used to define functions:

$$F_{exp}(x) = 1 - \exp\left(\frac{-x}{\mu}\right) \tag{C.5}$$

$$Q_{exp}(u) = -\mu \ln(1-u) \tag{C.6}$$

Table C.3 shows the property definition for the mean (μ).

Table C.3: Property of the exponential distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
1	TrExp	Duration	mean	The mean (μ) of the exponential distribution

C.2 Constant Distribution (TrConst)

For the constant distribution the transition time $Q_{const} = \mu$. It is the only property for the distribution, as shown in Table C.4.

Table C.4: Property of the constant distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
2	TrConst	Duration	mean	The constant transition time (μ)

C.3 Weibull Distribution (TrWeibull)

Weibull distribution is defined with scale parameter α and shape parameter β. Additionally, a user can define a failure free time or location γ (see pp. 41-42 in [C2]). It fixes the point in time from which the failures begin to occur.

$$F_w(x) = \begin{cases} 1 - \exp\left(- \left(\frac{x-\gamma}{\alpha}\right)^{\beta}\right) & \text{if } x \geq \gamma \\ 0 & \text{if } x < \gamma \end{cases} \tag{C.7}$$

$$Q_w(u) = \gamma - \alpha \ln(u)^{1/\beta} \tag{C.8}$$

Equations C.7 and C.8 change to functions for the basic two parameter Weibull distribution when the location parameter $\gamma = 0$. [C1] Table C.5 lists the properties for the Weibull distribution.

Table C.5: Properties of the Weibull distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
3	TrWeibull	Duration	scale	Scale parameter (α)
4	TrWeibull	Number	shape	Shape parameter (β)
5	TrWeibull	Duration	location	End of failure free time (γ)

C.4 Normal and Log-normal Distributions (TrNorm and TrLognorm)

Normal distribution is defined with mean μ and standard deviation σ parameters. The random variable is created by using the generalized Box-Muller transformation [C3], which uses two independent random variables:

$$Q_{norm}(u_1, u_2) = \mu + \sigma\sqrt{-2\ln u_1}\cos(2\pi u_2) \tag{C.9}$$

Table C.6 lists the properties for the normal distribution.

Table C.6: Properties of the normal distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
6	TrNorm	Duration	mean	Mean of the normal distribution (μ)
7	TrNorm	Number	deviation	Standartd deviation of the normal distribution (σ)

Equation C.9 is used for creating a log-normal random variable:

$$Q_{lognorm}(u_1, u_2) = \gamma + \exp(Q_{norm}(u_1, u_2)) \tag{C.10}$$

Possibility to add failure free time is implemented with location parameter γ, such that a failure cannot occur before time γ The scale parameter of the log-normal is given as a mean parameter of the normal distribution. Similarly, the shape parameter is given as a deviation parameter. Table C.7 lists the properties for the log-normal distribution.

Table C.7: Properties of the log-normal distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
8	TrLognorm	Duration	scale	Scale parameter of the log-normal distribution (μ)
9	TrLognorm	Number	shape	Shape parameter of the log-normal distribution (σ)
10	TrLognorm	Duration	location	Location parameter of the log-normal distribution (γ)

C.5 Immediate Probability Distribution (TrProb)

Immediate probability distribution is used for situations where an event either occurs immediately or does not occur at all. In mathematics this is referred as binomial trial. For example this has been used to model likelihood that a back-up power source starts during a power outage. The probability p defines how often the event is triggered:

$$Q_{prob}(u) = \begin{cases} 0 & \text{if } u \leq p \\ \infty & \text{if } u > p \end{cases} \tag{C.11}$$

Table C.8 shows the property defined for the immediate probability distribution.

Table C.8: Property of the trial distribution transition class

#	CLASS	TYPE	ATTR	COMMENT
11	TrProb	Probability	prob	Probability p that an immediate event is triggered

C.6 History Data Fitting (TrHistory)

In the OpenMARS there is possibility to import history data to be fitted to a distribution. The built in data matrix defines time, weigh and type parameters for rows in the data matrix. The type defines if the recorded history for an item ended in a failure or if the item was intact at the end of recording.

OpenMARS uses number 1 to denote failure and 0 to intact or censored record (see pp. 217-218 in [C2]). Table C.9 lists the properties used by the TrHistory class.

Table C.9: Properties of the history data fitting transition class

#	CLASS	TYPE	ATTR	COMMENT
12	TrHistory	Duration	data	List of history data values
13	TrHistory	Number	weight	Weights for history data values if needed (weight of each history data value is 1, if weights are not defined)
14	TrHistory	Number	type	Types for history data values if needed (type of each history data value is 1, if types are not defined)
15	TrHistory	Name	fitting	Name of the fitting used for the history

Example 31. Figure C.1 shows a result of a year long reliability test where two items survived the testing. Table C.10 presents the same history in a data matrix and Table C.11 in the attribute value table. The identical histories of items 4 and 5 are presented with one row by giving it weight 2.

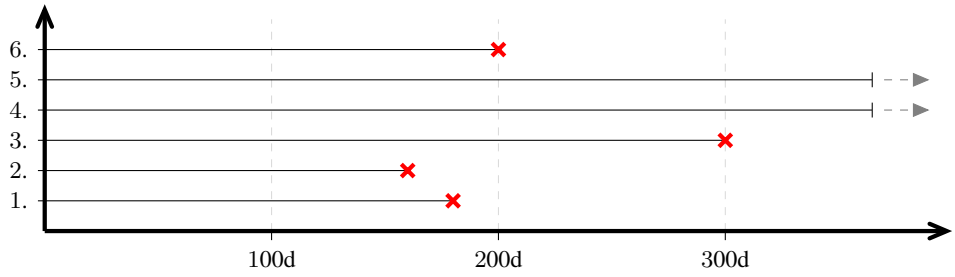

Fig. C.1: Result of a year long reliability test

Table C.10: The failure history from the Figure C.1 in a data matrix.

Item	Time	Weight	Type
1.	180d	1	1
2.	160d	1	1
3.	300d	1	1
4. & 5.	1a	2	0
6.	200d	1	1

Table C.11: Example history in the Attribute Value Table

#	OBJECT	ATTR	VALUE	COMMENT
1	exampleHist	data	180d, 160d, 300d, 1a, 200d	List of history duration values
2	exampleHist	weight	1,1,1,2,1	Weights for the history data values
3	exampleHist	type	1,1,1,0,1	Types for the history data values

With a two parameter Weibull fitting the example history data results in Weibull distribution with scale of 332 days and shape of 2.375 (see Ch. C.3).

C.7 References

[C1] P. Del Moral and S. Penev, *Stochastic processes: From applications to theory*, (CRC Press, Boca Raton, 2016), Ch. 4, pp. 71-98.

[C2] B. Bertsche, *Reliability in automotive and mechanical engineering*, (Springer-Verlag, Berlin, 2008), pp. 41-42.

[C3] G. Box and M. Muller, *Ann. Math. Stat.* **29** (1958), 610-611.

D Annex D: Modelling Techniques

This Annex introduces the inbuilt modelling techniques in OpenMARS and the associated classes for each technique. The present list of techniques shown in Table D.1. In future, new modelling techniques can be introduced and as well as alternative ways to define existing techniques.

Table D.1: The list of inbuilt modelling techniques of the OpenMARS approach

Modelling Technique	Chapter	Description
Advanced Fault Tree Analysis (FTA)	D.1	FTA of repairable systems with cost and maintenance modelling
Markov Analysis	D.2	Follows IEC 61165:2006 [D1]
Reliability Block Diagram (RBD)	D.3	Follows IEC 61078:2016 [D2]
OpenMARS	D.4	Provides tools to create composite models that combine and interconnect models defined with different techniques
Standard Fault Tree Analysis (FTA)	D.5.1	Follows EN 61025:2007 [D3]
Petri Net (PN)	D.5.2	Follows IEC 62551:2012 [D4]
Failure Mode and Effects Analysis (FMEA)	D.5.3	Follows VDA standard [D5]

D.1 Advanced Fault Tree Analysis (FTA) Modelling Technique

The advanced fault tree analysis (FTA) extends classical FTA by including repair, cost risk (Chapter D.1.1), maintenance action (Chapter D.1.2) and alternative consequence (Chapter D.1.3) modelling. Table D.2 lists the classes of the technique. An advanced FTA model consist of fault nodes with gate operators defining the connection logic.

Table D.2: Classes for advanced FTA modelling technique

#	CLASS	IS A	COMMENT
1	Fault	Node	Node with two states (normal, fault) and transitions (failure, restoration)
2	FaultTree	Fault	Top fault of the fault tree model, which contains all other faults and operators
3	Gate	Operator	Operator that handles Boolean logic rules
4	OR	Gate	The fault output occurs if any fault input occurs
5	AND	Gate	The fault output occurs only if all fault inputs occur
6	Vote	Gate	The fault output occurs only if at least m out of n fault inputs occur
7	XOR	Gate	The fault output occurs if exactly one fault input occurs
8	Limits	Vote	The fault output occurs only if at least m but no more than h fault inputs occur
9	Never	Gate	The fault output never occurs
10	Always	Gate	The fault output is always present and transmitted
11	Condition	Gate	The fault output occurs with certain probability if any fault input occurs
12	Delay	Gate	The output occurs with certain delay from any fault input
13	Ignore	Gate	The input elements of ignore gate are not taken into account, which can be useful for example to disregard undeveloped parts of tree in analyses

Table D.3 shows attributes of the advanced FTA modelling technique classes. The FaultTree class is used as the top node of the fault tree model. It is a fault that is also a container for faults, gates and folders of the model.

Table D.3: Attributes of for the advanced FTA classes

#	CLASS	TYPE	ATTRIBUTE	COMMENT
1	FaultTree	Fault	*	Fault tree can contain fault nodes
2	FaultTree	Gate	*	Fault tree can contain gate operators
3	FaultTree	Folder	*	Fault tree can contain folders
4	Vote	Integer	atLeast	The minimum number of inputs to create an output
5	Limits	Integer	atMost	The maximum number of inputs to create an output
6	Condition	Probability	prob	Probability that a fault output occurs after a fault input
7	Delay	Transition	delay	Delay after a fault input before a fault output is created

Table D.4 shows the predefined class elements for advanced FTA modelling technique. Table D.5 shows the connections of these class elements. As the Figure D.1 shows, all fault nodes contain normal and fault states that are connected with failure and restoration transitions.

Table D.4: Class elements for the advanced FTA modelling technique

#	CONT	ELEMENT	CLASS	COMMENT
1	Fault	normal	State	Normal state for all node elements of class Fault
2	Fault	fault	State	Fault state for all node elements of class Fault
3	Fault	failure	Transition	Transition which changes the node state from normal to fault
4	Fault	restoration	Transition	Transition which changes the node state from fault to normal

Table D.5: Class element connections for the advanced FTA modelling technique

#	SOURCE	TARGET	COMMENT
1	Fault/normal	Fault/failure	After normal state there can be failure transition
2	Fault/failure	Fault/fault	After failure transition there is fault state
3	Fault/fault	Fault/restoration	After fault state there can be restoration transition
4	Fault/restoration	Fault/normal	After restoration transition there is normal state

Fig. D.1: A fault node with normal and fault states and failure and restoration transitions

Example 32. As an example Figure D.2 illustrates a model structure created with the advanced FTA modelling technique. The 10 element creation table definition rows of Table D.6 create the 18 model elements. Altogether 17 connections have been added in 8 element connection table definition rows of Table D.7.

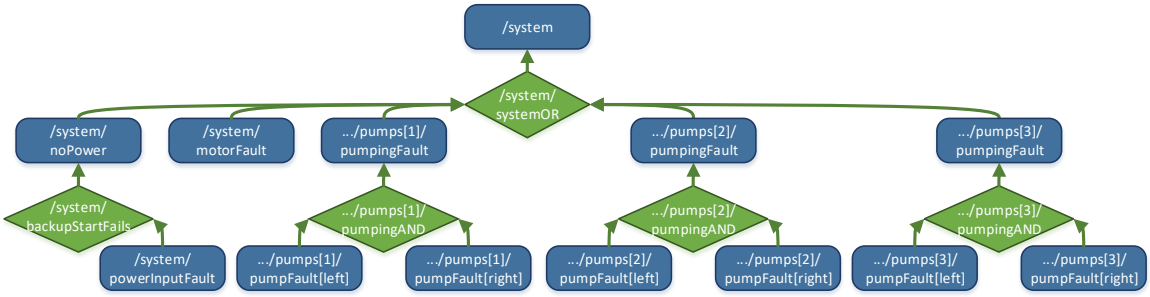

Fig. D.2: An example model structure created with advanced FTA modelling technique

Table D.6: The element creation of the advanced FTA modelling technique example

#	CONT	ELEMENT	CLASS	COMMENT
1		system	FaultTree	Fault tree of the system
2	system	systemOR	OR	Logic OR operator of the system
3	system	noPower	Fault	No power to the system
4	system	backupStartFails	Condition	Backup starts fails at certain probability
5	system	powerInputFaul	Fault	Power input fault of the system
6	system	motorFault	Fault	Motor fault of the system
7	system	pumps[1-3]	Folder	Three similar pump model folders
8	pumps	pumpingFault	Fault	A pumping fault for each folder
9	pumps	pumpingAND	AND	Logic AND operator of the pumping for each folder
10	pumps	pumpFault[left,right]	Fault	Two similar pump faults for each folder

Table D.7: The element connections of the advanced FTA modelling technique example

#	SOURCE	TARGET	COMMENT
1	system/systemOR	system	System OR gate connected to system fault tree top
2	noPower	systemOR	No power connected to system OR gate
3	backupStartFails	noPower	Backup fault causes no power
4	powerInputFault	backupStartFails	Backup is started after power input fault
5	motorFault	systemOR	Motor fault connected to system OR gate
6	pumps/pumpingFault	systemOR	All pumping faults connected to system OR gate
7	pumpingAND	pumpingFault	Pumping AND gate connected to pumping fault
8	pumpFault	pumpingAND	Both pump faults connected to pumping AND gate

D.1.1 Cost Risk Analysis

Costs can be associated to the number of occurrences of a transition or to the duration of a state. The cost risk analysis can assess both negative and positive impacts. Negative impacts can be the cost of repair each time a failure occurs or the loss of production associated to the system downtime. Positive impact can be for example revenue of the production.

The costs are defined by using the attributes introduced in Table D.8. Different type of risks can be modelled by using map definition (see Chapter B.4). The name of the cost type is given as an index when the value is defined. The estimation of the risk is the mean cumulative cost caused by the transition occurrences and the state durations during the studied analysis period.

Table D.8: Attributes for states to define cost risk

#	CLASS	TYPE	ATTR	COMMENT
1	Transition	Number	cost	A cost for each time the transition occurs
2	State	Rate	cost	A cost per time unit the state is active

Example 33. Table D.9 shows the attribute value table, which defines repair costs for motor and pump failures of Example 32. Also revenue of the uptime is defined for the normal state of the whole system. Optionally the UNIT column of attribute value table can be used to define the currency or other variable for the cost definitions.

Table D.9: Adding property values for cost risk

#	OBJECT	ATTR	VALUE	COMMENT
1	motorFault/failure	cost[repair]	1000	Each motor failure has a 1000 repair cost
2	pumpFault/failure	cost[repair]	800	Each pump failure has a 800 repair cost
3	system/normal	cost[revenue]	2000/h	The revenue is 2000 per an operation hour

D.1.2 Analysing the Impact of Maintenance Actions

Maintenance actions can be included in advanced FTA models. Table D.10 introduces the action classes that cover several types of maintenance operations. A fault node is container for maintenance actions as defined in Table D.11. In a model this describes how maintenance actions aims to reduce the likelihood of the failure modelled by the fault node.

Table D.10: Classes needed to include maintenance actions

#	CLASS	IS A	COMMENT
1	Action	Object	An object simulating the effects of a maintenance operation
2	Preventive	Action	Maintenance carried out at predetermined intervals intended to reduce the probability of failure. (In the simulation the change is permanent and the action interval does not influence the effect, as it is assumed that under the determined maintenance plan the failures occur less often.)
3	Inspection	Action	Inspection that can detect symptoms of developing fault and such avoid the occurrence of that failure
4	Improvement	Action	Maintenance carried out to reduce the degradation that could lead to a wear-out or ageing failure
5	Replacement	Action	Replace the element to be as good as new
6	Finding	Action	Detect and repair latent failures, which are not found in normal operations

Table D.11: The container definition which allows to add actions for fault nodes

#	CLASS	TYPE	ATTR	COMMENT
1	Fault	Action	*	Fault nodes can contain any actions

Model specific class definition (see Chapter 2.2.2) allows to create classes that include specified maintenance actions. All the instances of a fault class have the actions defined for the class. Example 34 shows how class definitions are used to add maintenance actions.

Example 34. Table D.12 defines model specific classes for pumps and motor faults. Table D.13 adds actions for the classes. Here pumps have preventive maintenance action and motors have inspection.

Table D.12: Model specific classes for certain component faults

#	CLASS	IS A	COMMENT
1	PumpFault	Fault	A class for pump faults
2	MotorFault	Fault	A class for motor faults

Table D.13: Predefined maintenance actions for model specific fault classes

#	CONT	ELEMENT	CLASS	COMMENT
1	PumpFault	preventive	Preventive	Preventive maintenance action to reduce likelihood of pump faults
2	MotorFault	inspection	Inspection	Inspection to detect motor developing motor faults

Table D.14 shows attributes of the action classes. All actions have a time interval and a cost. Different type of actions have their own attributes to define the effect of the maintenance action.

Table D.14: Attributes of the maintenance action classes

#	CLASS	TYPE	ATTR	COMMENT
1	Action	Transition	interval	The time for the maintenance interval (e.g. once per year)
2	Action	Number	cost	The cost of the action
3	Preventive	Number	effectFactor	The ageing of the node is changed according to the effect factor value when this preventive maintenance operation is active (e.g. with an effect factor 0.75 the element ages 75 % of the calendar time, thus the time to next failure grows by a factor of 1/0.75)
4	Inspection	Duration	symptom	Time before the failure when the detection is possible (during the symptom time the future failure can be avoided)
5	Inspection	Probability	prob	The probability that the inspection finds the failure if the action is made during the symptom time (by default inspection always detects the developing failure)
6	Improvement	Number	effectFactor	Age is used to measure degradation of an element as it e.g. affects MTTF of Weibull distribution. With effect factor 0.75 an element is 75 % ”younger” than before the improvement action.
7	Improvement	Duration	minAge	Minimum age to make the improvement
8	Replacement	Duration	minAge	Minimum age to make the replacement
9	Finding	Probability	prob	The probability to detect an existing latent failure during the failure finding action

Example 35. Table D.15 continues Example 34. It first defines that the actions have constant time interval for specific pump and motor elements. The pump will have a preventive maintenance action once a month, which reduces the failure rate by 50 %. The motor is inspected once per week. Here the symptom time is only a day so the inspection can only detect and stop about one out of seven failures.

Table D.15: The attribute values of the maintenance actions

#	OBJECT	ATTR	VALUE	COMMENT
1	Action	interval	TrConst	Constant time interval used for all actions
2	/somePump/preventive/interval	mean	30d	Preventive action is done once per month
3	/someMotor/inspection/interval	mean	7d	Inspection is done weekly
4	/somePump/preventive	effectFactor	0.5	Preventive action halves the failure rate
5	/someMotor/inspection	symptom	1d	Failure symptom time is only one day

D.1.3 Modelling Alternative Consequences

An event can have multiple consequences. For example a common cause fault results in several different faults due to same direct cause. This can be modelled with a fault tree. However, the issue comes when the consequences are exclusive and cannot be true at the same time. These cases are modelled with port elements that are introduced in Table D.16. Table D.17 describes the attributes for ports. A gate is a container for ports. A gate can have multiple ports that lead to alternative consequences. The ports can have a probability attribute or a delay transition.

Table D.16: The Port class in class definition table

#	CLASS	IS A	COMMENT
1	Port	Element	The class used for modelling alternative consequences

Table D.17: The Port class in class attribute Table

#	CLASS	TYPE	ATTR	COMMENT
1	Gate	Port	*	Gates are containers for ports
2	Port	Probability	prob	Probability (weight) for the port to activate (default is 1.0)
3	Port	Transition	delay	Delay between the event and the consequence. Only the fastest consequence of the same gate occurs (default is immediate transition)

The built in techniques in the advanced FTA modelling can model cases where: (i) each consequence has a probability to occur; (ii) each consequence has a stochastic transition and only the fastest transition occurs. As only one consequence can occur, sum of all port probabilities in one gate should be 100 %. If this is not the case, the values are considered as weights of the ports. A transition can model the delay between the event and the consequence. If ports in a same gate have both probability and delay attributes, by default the fastest transition occurs. However, if multiple transitions occur at the same time, the resulting consequence is decided stochastically based on probabilities.

The ports can be added to any gate element. Also, model specific class definition (see Chapter 2.2.2) with predefined class elements (see Chapter 3.4) can be used to create special gates with alternative consequences.

Example 36. Table D.18 shows how to add three alternative consequences for a gate. The target nodes of the ports can be added in the Connection Table. Table D.19 defines the probabilities of each consequence.

Table D.18: The definition of three alternative consequences for a gate

#	CONT	ELEMENT	CLASS	COMMENT
1	/someGate	minor	Port	Leads to a minor consequence
2	/someGate	average	Port	Leads to a average consequence
3	/someGate	major	Port	Leads to a major consequence

Table D.19: The probabilities of each alternative consequence

#	OBJECT	ATTR	VALUE	COMMENT
1	/someGate/minor	probability	0.6	60 % probability for the minor consequence
2	/someGate/average	probability	0.3	30 % probability for the average consequence
3	/someGate/major	probability	0.1	10 % probability for the major consequence

Negations can be modelled with a *not* port. The *not* port activates when the logic rule of the gate is false. Table D.20 introduces this element for all gates. The negation is implemented by connecting target nodes to the *not* port instead of the gate.

Table D.20: Predefined *not* port for gate operators

#	CONT	ELEMENT	CLASS	COMMENT
1	Gate	not	Port	A port for modelling negations

Example 37. Table D.21 shows how a NAND-gate is implemented by connecting the consequence to the *not* port of an AND-gate.

Table D.21: Example on how to implement a NAND-gate

#	SOURCE	TARGET	COMMENT
1	andGate/not	consequence	Connection from the *not* port of an AND-gate implements the NAND-gate

D.2 Markov Modelling Technique

OpenMARS implementation of Markov modelling technique follows the standard EN 61165 [D1]. The basis of the implementation is a Markov model node that contains the states and transitions of a single model. Table D.22 introduce the technique specific classes for Markov modelling technique. The initial active state of the model is defined with *initial* attribute introduced in Chapter's 2.3 Table 5. Example 38 shows how a Markov model is defined.

Table D.22: Classes for Markov modelling technique

#	CLASS	IS A	COMMENT
1	Markov	Node	Contains the states and transitions in a Markov model
2	StateMarkov	State	A state in Markov model

Example 38. Figure D.3 illustrates a phase model structure with three states created with the Markov modelling technique. The element creation table is shown in Table D.23 and the element connection table in Table D.24. Table D.25 shows how attribute values are defined to the *start* transition and sets the *prepare* as the initial phase. In this example defining the transition class for the *start* could have been avoided if the specific class had been defined in Table D.23.

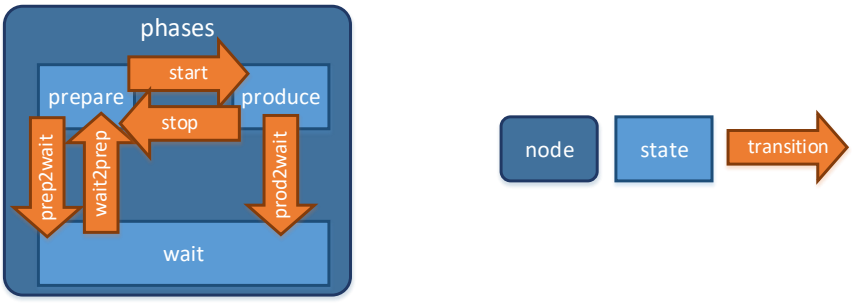

Fig. D.3: An example model structure created with Markov modelling technique

Table D.23: The element creation of the Markov modelling technique example

#	CONT	ELEMENT	CLASS	COMMENT
1		phases	Markov	Markov model node
2	phases	prepare	StateMarkov	A state of the Markov model node
3	phases	produce	StateMarkov	A state of the Markov model node
4	phases	wait	StateMarkov	A state of the Markov model node
5	phases	start	Transition	A transition of the Markov model node
6	phases	stop	Transition	A transition of the Markov model node
7	phases	prep2wait	Transition	A transition of the Markov model node
8	phases	prod2wait	Transition	A transition of the Markov model node
9	phases	wait2prep	Transition	A transition of the Markov model node

Table D.24: The element connections of the Markov modelling technique example

#	SOURCE	TARGET	COMMENT
1	prepare	start	From state to transition
2	start	produce	From transition to state
3	produce	stop	From state to transition
4	stop	prepare	From transition to state
5	prepare	prep2wait	From state to transition
6	prep2wait	wait	From transition to state
7	produce	prod2wait	From state to transition
8	prod2wait	wait	From transition to state
9	wait	wait2prep	From state to transition
10	wait2prep	prepare	From transition to state

Table D.25: Attributes for the *start* transition in the example

#	OBJECT	ATTR	VALUE	COMMENT
1	phases	initial	prepare	When simulation begins the prepare state is active
2	phases	start	TransExp	The start transition is exponential
3	start	mean	2d	Mean time to transition

D.3 Reliability Block Diagram (RBD) Modelling Technique

Table D.26 introduces the OpenMARS implementation of Reliability Block Diagram (RBD) technique that follows the standard IEC 61078:2016 [D2]. In OpenMARS RBD is a fault node that contains other fault nodes that form the diagram. The RBD contains predefined start and end nodes. The RBD node is at the normal state if there is a path from start to end that does not contain failed fault nodes. Otherwise the RBD is in the fault state. Parallel and serial fault logics are modelled by connecting the fault nodes in the diagram to each other to form the logics. However, the n/m redundancy structures are modelled with a specific operator. Table D.26 show the classes and Table D.27 the attributes of the RBD modelling technique classes.

Table D.26: Classes for Reliability Block Diagram (RBD) modelling technique

#	CLASS	IS A	COMMENT
1	RBD	Fault	RBD is a fault node that implements the fault and normal states with a diagram
2	RBD_Vote	Operator	An operator for the voting logic in the RBD

Table D.27: Attributes for the Reliability Block Diagram (RBD) modelling technique class

#	CLASS	TYPE	ATTR	COMMENT
1	RBD	Fault	start	Predefined start node for the diagram
2	RBD	Fault	end	Predefined end node for the diagram
3	RBD	Fault	*	The RBD can contain any number of intermediate fault nodes between predefined start and end nodes
4	RBD	RBD_Vote	*	The RBD can contain any number of vote operators
5	RBD_Vote	Integer	atLeast	The number of input connections required for a path

Example 39. Figure D.4 shows a reliability block diagram (RBD) forming a bridge circuit structure. (This example is taken from Chapter 5.2 in [D3].) The elements of the example structure are defined in Table D.28 and the connections in Table D.29.

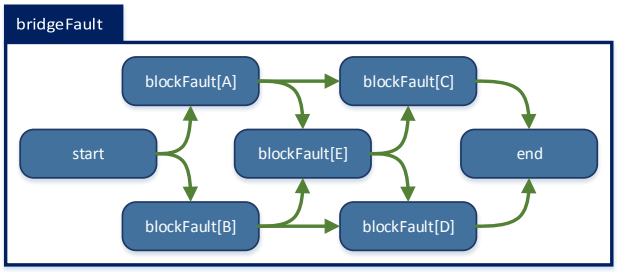

Fig. D.4: Bridge structure in a reliability block diagram

Table D.28: The element creation of the reliability block diagram example

#	CONT	ELEMENT	CLASS	COMMENT
1		bridgeFault	RBD	Reliability block diagram of bridge fault
2	bridgeFault	blockFault[A-E]	Fault	Block faults A,B,C,D and E

Table D.29: The element connections of the reliability blcok diagram example

#	SOURCE	TARGET	COMMENT
1	start	blockFault[A,B]	The start is connected to blocks A and B
2	blockFault[A,B]	blockFault[E]	Blocks A and B are connected to the block E
3	blockFault[E]	blockFault[C,D]	The block E is connected to blocks C and D
4	blockFault[A]	blockFault[C]	The block A is connected to the block C
5	blockFault[B]	blockFault[D]	The block B is connected to the block D
6	blockFault[C,D]	end	Blocks C and D are connected to the end

Example 40. This example shows how the n out of m logic can be used in RBD. Table D.30 introduces the five fault nodes that are connected to a vote operator. The connections are added in Table D.31. Here it is important that the vote operator is the target for the nodes. Lastly, Table D.32 defines the property value for the logic voting operator, which defines the required number of elements in the normal state.

Table D.30: The element creation of the RBD voting example

#	CONT	ELEMENT	CLASS	COMMENT
1		voteExample	RBD	Voting example RBD
2	voteExample	itemFault[1-5]	Fault	Five item faults
2	voteExample	logic	RBD_Vote	Logic vote operator

Table D.31: The element connections of the RBD voting example

#	SOURCE	TARGET	COMMENT
1	start	itemFault[1-5]	Start connected to all item faults
2	itemFault[1-5]	logic	Item faults connected to logic operator
3	logic	end	Logic operator connected to end

Table D.32: The property value definition of the RBD voting example

#	OBJECT	ATTR	VALUE	COMMENT
1	/voteExample/logic	atLeast	3	At least 3 out of 5 required

D.4 OpenMARS Modelling Technique

OpenMARS modelling technique provides tools to create composite models that combine and interconnect models defined with different techniques. Chapter D.4.1 presents radios and listeners that are used for relaying messages and commands between models. Chapter D.4.2 introduces the concept of modes where a model can have different sets of values based on defined modes. Additionally OpenMARS technique has tools for defining mathematical and logic operators. In OpenMARS these are called as functions and they are presented in Chapter D.4.3.

This document does not fully explore the possibilities that are achievable with OpenMARS tools. This document presents an example that interconnects a fault tree with a Markov chain and examples of function models. Regardless with the presented tools, models done with additional techniques could be connected to each other.

D.4.1 Radios and Listeners

Radios and Listeners enable communication between models. This allows creating composite models comprised of interconnected sub-models defined with different techniques. Radios broadcast messages at certain channel in defined situations. They can be attached to transitions or states. The radio broadcasts when the transition is triggered or when the state activates or ends. Action listeners can be attached also to both transitions and states. If listener receives a signal in certain channel, the transition or the state is activated. The attributes to define the radios and action listeners are shown in Table D.33.

Table D.33: Radios and action listeners are added as attributes of states and transitions

#	CLASS	TYPE	ATTR	COMMENT
1	State	Name	radio	List of radio channels that broadcast when the state activates
2	State	Name	endRadio	List of radio channels that broadcast when the state ends
3	Transition	Name	radio	List of radio channels that broadcast when the transition is triggered
4	State	Name	listener	List of listener channels that activate the state
5	Transition	Name	listener	List of listener channels that trigger the transition

In a Markov model[4] only one state can be active at certain time. Example 41 illustrates the transition logics of the active state.

Example 41. Figure D.5 illustrates the rules for the action listeners:

1) The state can be activated as a transition exist from the active state.

2) The transition can be triggered as the source of the transition is active.

3) The state can not be activated as no direct transition exists from the active state to the listener state.

4) The transition can not be triggered as the source state is not active.

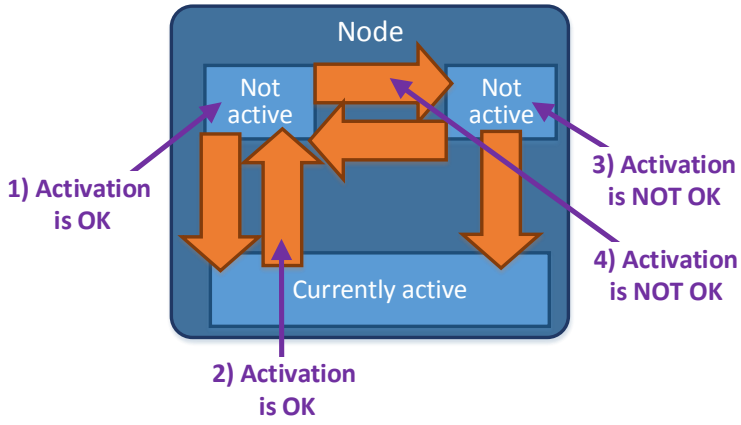

Fig. D.5: Allowed and not allowed activations of states and transition based on the position of the listener

Example 42. This example shows how radios and listeners can be used to connect models. Here Example 38 is extended by radio-listener connection that sets the phases Markov model to the wait state when the top node of the fault tree (system) is in the fault state. Figure D.6 illustrates the model structure and Table D.34 shows the attributes for the radios and listeners. The first radio is set to transmit a message in the waitStartChannel when the fault state start in the system node. The second radio sends a message when the normal state starts. The Markov model has listeners for these channels. The wait state starts when a failure occurs and the prepare state starts when the system is back to the normal state. The presented solution is not the only way to reach similar functionality.

Table D.34: The definition of the radios and listeners for the example

#	OBJECT	ATTR	VALUE	COMMENT
1	system/fault	radio	waitStartChannel	Send a message when the fault state starts
2	system/normal	radio	waitEndChannel	Send a message when the normal state starts
3	phases/wait	listener	waitStartChannel	Activates the wait state when message is received
4	phases/wait2prep	listener	waitEndChannel	Activate the transition from wait to prepare

[4] A node can be seen to contain a Markov model, see Example 1.

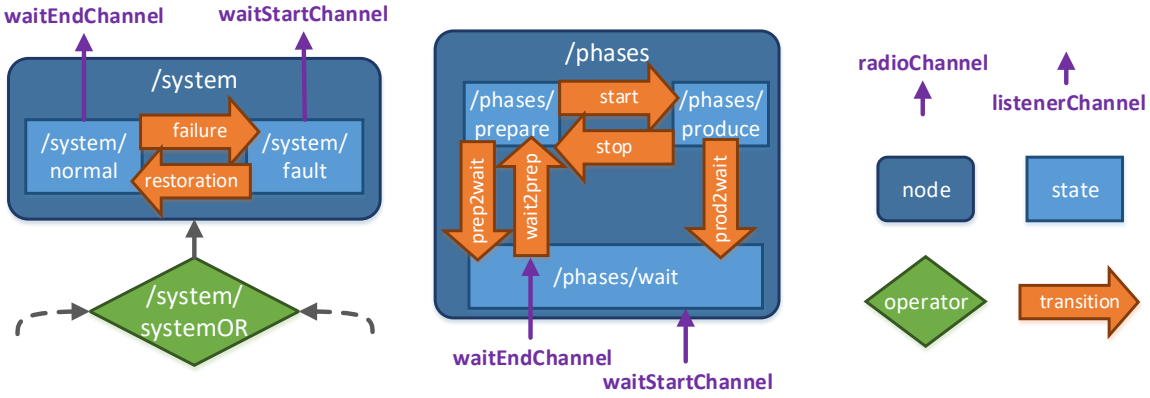

Fig. D.6: An example of connecting models with radios and listeners

D.4.2 Mode Dependent Properties

A system can operate in different modes that can affect the system behaviour. For example certain mode can be more demanding and thus system failure is more likely this specific mode. In the OpenMARS the mode activations and waits are handled with listeners that wait specific radio messages. As shown in Table D.35, a mode listener can be an attribute of an element or a folder. If a mode listener is defined for a container, a mode change affects all the contained elements.

Table D.35: Mode listeners can be added to folders or elements

#	CLASS	TYPE	ATTR	COMMENT
1	Element	Name	wakeListener	A map of modes (key) connected to channel (values). Receiving a message in certain channel sets the channel specific modes to activate
2	Folder	Name	wakeListener	A map of modes (key) connected to channel (values). Receiving a message in certain channel sets the channel specific modes to activate (affects all the elements in the folder)
3	Element	Name	waitListener	A map of modes (key) connected to channel (values). Receiving a message in certain channel sets the channel specific modes to wait
4	Folder	Name	waitListener	A map of modes (key) connected to channel (values). Receiving a message in certain channel sets the channel specific modes to wait (affects all the elements in the folder)

Example 43 show how the mode dependent behaviour is implemented by mapping the attribute values based on modes. The attribute values from the active modes are used in the simulation.

Example 43. Table D.36 shows examples how to define mode dependent failure rates for the motorFault in Example 32. The mode changes are connected to the prepare and produce states of the Markov model shown in Example 42. Table D.37 shows how to add the mode listeners to start and stop the modes when broadcast is received in specified channels.

Table D.36: Example how add to mode dependent property values

#	OBJECT	ATTR	VALUE	COMMENT
1	motorFault/failure[prep]	mean	6a	Mean time to failure the prepare mode
2	motorFault/failure[prod]	mean	2a	Mean time to failure in the produce mode

Table D.37: Example how add to mode listener

#	OBJECT	ATTR	VALUE	COMMENT
3	motorFault	wakeListener[prep]	prepStartChannel	Wakes the prep mode when a message is received in the prepStartChannel
4	motorFault	wakeListener[prod]	prodStartChannel	Wakes the prod mode when a message is received in the prodStartChannel
5	motorFault	waitListener[prep]	prodStartChannel	Sets the prep mode to wait when a message is received in the prodStartChannel
6	motorFault	waitListener[prod]	prepStartChannel	Sets the prod mode to wait when a message is received in the prepStartChannel

D.4.3 Modelling Logical and Mathematical Functions

OpenMARS contains tools to create an environment for visual programming. Table D.38 introduces the classes used for modelling logical and mathematical function models and Table D.39 introduces the attributes for these classes. In addition to the built in functions, OpenMARS allows users to freely define functions with programming code. The functions are connected to values, which can be constant input values, results or other functions. Connection to and from a function can be made also from any element attribute.

Table D.38: Function classes

#	CLASS	IS A	COMMENT
1	Value	Node	A node which state is the contained number value
2	Function	Operator	An operator to update value nodes
3	Addition	Function	Predefined sum function (+), note empty sum is 0
4	Subtraction	Function	Predefined difference function (−)
5	Multiplication	Function	Predefined product function (∗), note empty product is 1
6	Division	Function	Predefined quotient function (/)
7	UserFunction	Function	A function defined by user provided a code text attribute
8	Random	Value	A random value between 0 and 1

Table D.39: Attributes for the function classes

#	CLASS	TYPE	ATTR	COMMENT
1	Value	Number	value	The value attribute in the value element
2	Subtraction	Addition	minuend	Subtraction = **minuend** − subtrahend. If two or more values are connected to minuend it is sum of the connected values. If no value is connected to minuend this returns opposite number or additive inverse of the subtrahend.
3	Division	Multiplication	divident	Division = **divident**/divisor. If two or more values are connected to divident, it is the product of connected values. If no value is connected to divident this returns the multiplicative inverse of the divisor.
4	UserFunction	Value	*	User function can contain freely defined input values
5	UserFunction	Text	code	Function code

Example 44. Figure D.7 illustrates a function model structure. The elements of the function model are defined in Table D.40. The constant input values and a result value are connected to functions in Table D.41. The model implements the equation:

$$\text{result} = \frac{\text{duration}[1] + \text{duration}[2]}{\text{divisor}} \tag{D.1}$$

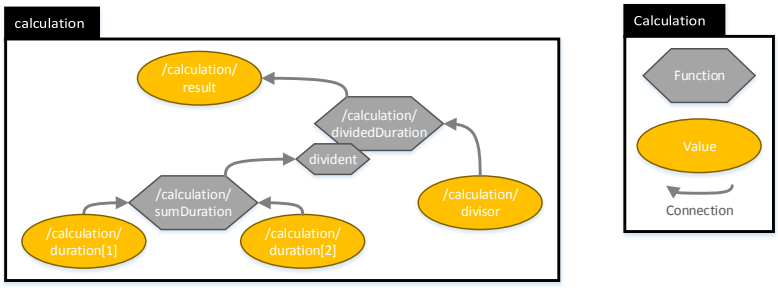

Fig. D.7: An example of a function model

Table D.40: The element creation of the function model example

#	CONT	ELEMENT	CLASS	COMMENT
1		calculation	Folder	Simple calculation example
2	calculation	result	Value	Result of the calculation
3	calculation	dividedDuration	Division	Quotient calculation
4	calculation	sumDuration	Addition	Sum calculation
5	calculation	divisor	Value	Divisor value
6	calculation	duration[1,2]	Value	Duration values

Table D.41: The element connections of the function model example

#	SOURCE	TARGET	COMMENT
1	dividedDuration	result	Result value is sum / divisor
2	sumDuration	dividedDuration/divident	The sum is connected to the divident of the division
3	divisor	dividedDuration	The divisor is connected directly to the function
4	duration	sumDuration	The sum is duration[1] + duration[2]

Example 45. This example shows how a user defined function can be included in a model. The example function is Equation 11 from [D6]:

$$L_{int}(t_f) = \begin{cases} L_p t_f, & : t_f \leq t_{lv} \\ L_p t_l v + L_p \tau_L (\exp(-t_{lv}/\tau_L) - \exp(-t_f/\tau_L)), & : t_f \geq t_{lv} \end{cases} \quad \text{(D.2)}$$

It models how much integrated luminosity the LHC can produce based on time length of the fill t_f, peak value of instantaneous luminosity L_p, luminosity lifetime τ_L and luminosity levelling time t_{lv}.

Table D.42 shows the element creation of the example function. The inputs are a map of number values, which all need to be introduced. The element connections are made in Table D.43. There is only one input variable coming from the simulation that needs to be connected to the function. The other input values are constants that in this example are defined as attributes of the function.

The code attribute needs to be defined for the user function. It can be written as a text in the attribute table or the code can be stored in an external file (see Chapter B.5). The example code implementing Equation D.2 is shown in Listing 1. Here Java is used as the programming language. However, available programming languages for user code is not set by this specification and is free to be determined by the implementation.

Table D.42: The element creation of the user defined function example

#	CONT	ELEMENT	CLASS	COMMENT
1		luminosity	Folder	Luminosity calculation example
2	luminosity	production	Result	Resulted luminosity production
3	luminosity	intLumi	UserFunction	Function of the luminosity production of a fill
4	intLumi	fillTime	Value	Fill length obtained from the simulation
5	intLumi	levellingTime	Value	Constant levelling time defined by the user
6	intLumi	peakLumi	Value	Constant peak luminosity defined by the user
7	intLumi	lumiLifetime	Value	Constant luminosity lifetime defined by the user

Table D.43: The element connections of the user defined function example

#	SOURCE	TARGET	COMMENT
1	simulator/fillTime	luminosity/intLumi/fillTime	The simulated parameter connected to the function
2	intLumi	production	The return value connected to a result element

Listing 1: The user defined code for the integrated luminosity function

```
1 if (fillTime <= levellingTime) {
2   return peakLumi * fillTime;
3 }
4 double levelled = peakLumi * levellingTime;
5 double decay = peakLumi * lumiLifetime * (Math.exp(-levellingTime /
6   lumiLifetime) - Math.exp(-fillTime / lumiLifetime));
7 return levelled + decay;
```

D.5 Other Modelling Techniques

This chapter lists examples of supported modelling techniques that follow specified standards. This shows that the OpenMARS approach can be applied to various techniques and this list can be extended if need to use other techniques emerges.

D.5.1 Standard Fault Tree Analysis (FTA) Modelling Technique

Tables D.44 and D.45 list the classes and Table D.46 the attributes for a modelling technique that follows the standard EN 61025:2007 [D3]. This technique can be used only for analysis of non-repairable systems. Advanced FTA technique suitable for repairable systems is presented in Chapter D.1.

Table D.44: Event classes for standard FTA modelling technique

#	CLASS	IS A	COMMENT
1	FT_Event	Node	Node with two states: Normal and Fault
2	FT_Basic	FT_Event	The lowest level event for which probability of occurrence or reliability information is available
3	FT_Cond	FT_Event	Event that is a condition of occurrence of another event when both have to occur for the output to occur
4	FT_Dorm	FT_Event	A primary event that represents a dormant failure; an event that is not immediately detected but could, perhaps, be detected by additional inspection
5	FT_Undev	FT_Event	A primary event that represents a part of the system that is not yet developed
6	FT_House	FT_Event	Event which has happened, or will happen with certainty
7	FT_Zero	FT_Event	Event which cannot happen

Table D.45: Gate classes for standard FTA modelling technique

#	CLASS	IS A	COMMENT
8	FT_Gate	Operator	Operators for traditional fault tree analysis modelling technique
9	FT_Trf	FT_Gate	Transfer gate indicating that this part of the system is developed in another part or page of the diagram
10	FT_TrfOut	FT_Trf	Out means that the same gate developed in this place will be used elsewhere
11	FT_TrfIn	FT_Trf	In means that the develop gate is elsewhere
12	FT_OR	FT_Gate	The output event occurs if any of the input events occur
13	FT_AND	FT_Gate	The output event occurs only if all of the input events occur
14	FT_Vote	FT_Gate	The output occurs if m or more inputs out of a total of n inputs occur
15	FT_Not	FT_Gate	The output event occurs only if the input event does not occur
16	FT_XOR	FT_Gate	The output event occurs if one, but not the other inputs occur
17	FT_NOR	FT_Gate	The output occurs if none of the input events occur (FT_Not with > 1 inputs)
18	FT_NAND	FT_Gate	The output occurs if at least one of the input events does not
19	FT_Inhibit	FT_Gate	The output occurs only if both of the input events take place, one of them conditional
20	FT_PAND	FT_Gate	The output event occurs only if the input events occur in sequence from left to right
21	FT_Seq	FT_PAND	The output event (failure) occurs only if all input events occur in sequence from left to right. This gate is identical to the PAND gate if the number of inputs to the PAND gate is not limited to 2 as done by some analysts
22	FT_Spare	FT_Gate	The output event will occur if the number of spare components is less than the number required

Table D.46: Attributes of the classes for standard FTA modelling technique

#	CLASS	TYPE	ATTR	COMMENT
1	FT_Event	State	normal	Normal state which means that the event did not occur
2	FT_Event	State	fault	Fault state which means that the event has occurred
3	FT_Basic	Probability	prob	Probability of the node to be in Fault state
4	FT_Vote	Integer	atLeast	Number of inputs needed at least for output to occur
5	FT_Tfr	Name	link	Name that connects the transfer gate to other transfer gates

Example 46. Standard fault tree analysis (FTA) model can used to model the bridge circuit model (See Example 39). The elements of the standard FTA model are defined in Table D.47 and the connections in Table D.48.

Table D.47: The element creation of the standard FTA example

#	CONT	ELEMENT	CLASS	COMMENT
1		bridgeFault	FT_Event	Standard fault tree event of bridge fault
2		bridgeOR	FT_OR	All situations that cause the bridge fault
3		blocks[AB,CD,AED,BEC]	FT_Event	Block fault situations
4		blocksAND[AB,CD,AED,BEC]	FT_AND	AND gate for the block fault situations
5		blockFault[A-E]	FT_Basic	Basic events for block faults A,B,C,D and E

Table D.48: The element connections of the standard FTA example

#	SOURCE	TARGET	COMMENT
1	bridgeOR	bridgeFault	Bridge OR connected to bridge fault
2	blocks[AB,CD,AED,BEC]	bridgeOR	All fault situations connected to bridgeOR
3	blocksAND[AB]	blocks[AB]	AND gate connected to fault situation
4	blocksAND[CD]	blocks[CD]	AND gate connected to fault situation
5	blocksAND[AED]	blocks[AED]	AND gate connected to fault situation
6	blocksAND[BEC]	blocks[BEC]	AND gate connected to fault situation
7	blockFault[A,B]	blocksAND[AB]	Block faults connected to AND gate
8	blockFault[C,D]	blocksAND[CD]	Block faults connected to AND gate
9	blockFault[A,E,D]	blocksAND[AED]	Block faults connected to AND gate
10	blockFault[B,E,C]	blocksAND[BEC]	Block faults connected to AND gate

D.5.2 Petri Net (PN) Modelling Technique

Table D.49 lists classes for Petri net (PN) modelling technique. The definitions follow the standard EN 62551:2012 [D4]. Petri net consist of places, transitions and arcs. In OpenMARS arcs are the connections between places and transitions. This works in normal cases where one token is removed from and added to places when a transition is fired. For defining the number of added and removed tokens OpenMARS uses stream elements that have weight of the arc as an attribute. The inhibitor arc is implemented with specific stream class.

Table D.49: Classes for Petri net (PN) modelling technique

#	CLASS	IS A	COMMENT
1	PN_Place	Node	Node which state is the number of PN markings (tokens)
2	PN_Transition	Operator	A connection from a PN_Place to other PN_Place
3	PN_Stream	Element	An element to define weight for upstream and downstream of transitions
4	PN_Inhibitor	PN_Stream	An element to define not logic for upstream of transitions

Petri nets follow [D4], such that a transition is ”valid” (or ”enabled”) when all the following conditions are true: (i) the upstream places have at least a number of token equal to the weight of the corresponding upstream arc; (ii) the upstream places have a number of token strictly lower than the weight of the corresponding inhibitor arc; (iii) the predicates are ”true”.

Table D.50 shows attributes of the classes for the PN modelling technique. Predicates and assertions can be included for transitions by defining them in string format. Map definitions can be used if more than one predicates or assertions are needed. However, the functional implementation of predicates and assertions is not specified by this document.

Table D.50: Attributes for Petri net (PN) modelling technique classes

#	CLASS	TYPE	ATTR	COMMENT
1	PN_Place	Integer	tokens	The number of PN markings
2	PN_Transition	Name	predicate	Predicate string of map of predicates
3	PN_Transition	Transition	delay	As soon as a transition becomes valid its attached delay distribution is used to evaluate when it is going to be fired
4	PN_Transition	Name	assertion	Assertion string or map of assertions
5	PN_Stream	Integer	weight	Upstream and downstream can be weighted (default weight is 1 if not defined)

Example 47. Here an order management process is modelled with a Petri net. In the process an order is closed when it is paid and delivered. A goal of ten completed orders per day is set to demonstrate the weight modelling. The Petri net model elements of an order management process are defined in Table D.51 and the connections in Table D.52. The arcs of the order process have the default weight 1 so places can be directly connected to transitions. The stream element is needed to define the weight for the daily goal transition. Figure D.8 illustrates the resulting Petri net model.

Table D.51: The element creation of the Petri net model example

#	CONT	ELEMENT	CLASS	COMMENT
1		orderStart	PN_Place	Has token whenever order needs to be started
2		fillOrder	PN_Transition	Order process is stated
3		shipOrder	PN_Place	The product is sent to customer
4		sendInvoice	PN_Place	The invoice is sent to customer
5		payment	PN_Transition	The customer pays the invoice
6		acceptPayment	PN_Place	Made payment is accepted
7		closeOrder	PN_Transition	Close the order when all ready
8		orderFinished	PN_Place	The order process has ended
9		dailyGoalWeight	PN_Stream	Stream element needed to define weight
10		dailyGoal	PN_Transition	Check when daily goal is achieved
11		success	PN_Place	Success after daily goal

Table D.52: The element connections of the Petri net model example

#	SOURCE	TARGET	COMMENT
1	orderStart	fillOrder	Start the order process
2	fillOrder	shipOrder	Product sending is started
3	shipOrder	closeOrder	Product needs to be sent before order can be closed
4	fillOrder	sentInvoice	Invoice sending is started
5	sentInvoice	payment	Wait for payment after invoice has been sent
6	payment	acceptPayment	Accept after customer has made the payment
7	acceptPayment	closeOrder	Payment needs to be accepted before order can be closed
8	closeOrder	orderFinished	End the order process finally
9	orderFinished	dailyGoalWeight	Connect to stream element after order finish
10	dailyGoalWeight	dailyGoal	Stream connected to transition
11	dailyGoal	success	Success after daily goal

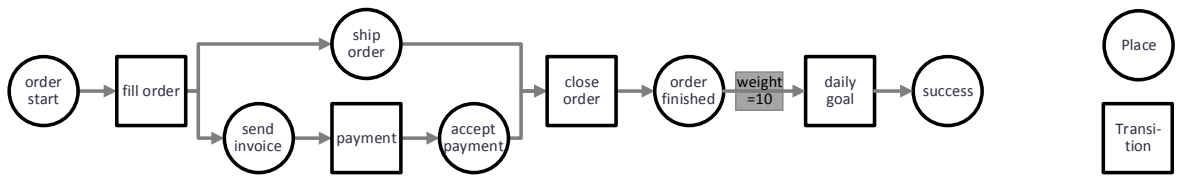

Fig. D.8: An example of Petri net model

D.5.3 Failure Mode and Effects Analysis (FMEA) Modelling Technique

OpenMARS Failure Mode and Effects Analysis (FMEA) follows the standard [D5]. Table D.53 introduces the FMEA attributes that can be added to any fault node. The standard [D5] allows multiple recommended actions for identified failures. To model this a list of recommendations and improved result values can be formed.

Table D.53: Attributes for FMEA modelling technique

#	CLASS	TYPE	ATTR	COMMENT
1	Fault	Text	severity	Text to describe severity
2	Fault	Text	occurrence	Text to describe occurrence rate
3	Fault	Text	detection	Text to describe detectability
4	Fault	Integer	sev	Integer between 0 and 10 to rate the severity
5	Fault	Integer	occ	Integer between 0 and 10 to rate the occurrence likelihood
6	Fault	Integer	det	Integer between 0 and 10 to rate the detectability
7	Fault	Integer	rpn	The risk priority number (RPN) is a product of sev, occ and det
8	Fault	Text	recommend	Text to describe recommended action to reduce the risk priority
9	Fault	Text	responsibility	Person responsible for the recommendation and the actions
10	Fault	Time	date	Target date to complete the recommended actions
11	Fault	Integer	improvedSev	The reduced severity if recommended mitigation is applied
12	Fault	Integer	improvedOcc	The reduced occurrence likelihood recommended action is applied
13	Fault	Integer	improvedDet	The improved detectability if recommended action is applied
14	Fault	Integer	improvedRpn	The updated priority number if recommended action is applied

D.6 References

[D1] International Electrotechnical Commission, Application of Markov techniques, IEC 61165:2006, (2006).

[D2] International Electrotechnical Commission, Reliability block diagrams, IEC 61078:2016, (2016).

[D3] European Committee for Electrotechnical Standardization, Fault Tree Analysis (FTA), EN 61025:2007, (2007).

[D4] International Electrotechnical Commission, Analysis techniques for dependability - Petri net techniques, IEC 62551:2012, (2012).

[D5] Verband der Automobilindustrie e.V. Quality assurance before series production, vol. 4, Part 2: System FMEA Failure Mode and Effects Analysis, 1st ed. (VDA, Frankfurt am Main, 1996).

[D6] J. Wenninger, Simple models for the integrated luminosity, CERN-ATS-Note-2013-033 PERF, (CERN, Geneva, 2013).

E Annex E: Analysis of OpenMARS Models

This annex defines tool classes for mathematical analyses of OpenMARS models. This document introduces two simulation based analysis tools. However, there is no limitations for creating analytical calculation tool for obtain results or solving model with specific techniques. Table E.1 introduces the tool classes that are described more in detail in following chapters of this annex.

Table E.1: Classes for tools to analyse OpenMARS models

#	CLASS	IS A	COMMENT
1	Tool	Element	Class for tools used to analyse OpenMARS models
2	Simulator	Tool	Basic probability simulator tool
3	DES	Simulator	Discrete event simulator (DES) tool
4	Sensitivity	Tool	Sensitivity analysis tool

E.1 Basic Probability Simulator Tool

Table E.2 lists the attributes used with the the basic probability simulator tool. The tool considers only the immediate probability transitions and elements that do not contain time durations. Thus, this tool can be used for example to analyse models created with standard FTA modelling technique. The basic analysis result is the failure probability of each node in the model. The results is calculated based on the eventCount variables of nodes and currentRound variable of the simulator.

Table E.2: Attributes used with the basic probability simulator tool

#	CLASS	TYPE	ATTR	COMMENT
1	Node	State	currentState	The currently active state
2	Transition	Integer	eventCount	The number of transition events triggered during the simulation
3	Simulator	Integer	maxRounds	The maximum number of simulated rounds
4	Simulator	Integer	currentRound	The current simulate round

E.2 Discrete Event Simulator (DES) Tool

The discrete event simulator (DES) extends the basic probability simulator by introducing the simulation period. During each simulation round the node state change events are simulated until the defined simulation period ends. Table E.3 lists the attributes for the DES tool that are not included in the basic probability simulator.

Table E.3: Attributes of the discrete event simulator (DES) tool

#	CLASS	TYPE	ATTR	COMMENT
5	State	Duration	stateTime	The time the state is active during the simulation
6	DES	Duration	period	The length of the simulation period of each round
7	DES	Duration	currentTime	The current time in the simulation
8	DES	Duration	nextTime	The next time that will be handled in the simulation
9	DES	Duration	stepLength	The length of the currently made simulation step

The basic results are the mean number of events in the simulation period and the mean time spend in certain state. A more extensive implementation of this tool can add new result features such as parameter end value distribution and parameter monitoring during a simulation round.

E.3 Sensitivity Analysis Tool

Table E.4 lists attributes of the sensitivity analysis tool. Any simulator or other analyser tool can be used with the sensitivity analysis tool. During the analysis defined parameter or parameters are gradually changed from the minimum to the maximum value based the set number of steps. A result is stored for each simulated value combination. Multidimensional sensitivity analyses can be created by defining a map for parameter UIDs, min and max values and steps.

Table E.4: Attributes of the sensitivity analysis tool

#	CLASS	TYPE	ATTR	COMMENT
1	Sensitivity	UID	analyser	The UID of the tool used for sensitivity analysis
2	Sensitivity	UID	parameter	The UID of the property value used as sensitivity parameter
3	Sensitivity	Property	parameterMin	The minimum value of sensitivity parameter
4	Sensitivity	Property	parameterMax	The maximum value of sensitivity parameter
5	Sensitivity	Integer	parameterSteps	The number of steps for the sensitivity parameter
6	Sensitivity	UID	result	The UID of the property value used as sensitivity result