

# VALU3S

*Verification and Validation of Automated Systems' Safety and Security*

## Final Detailed Description of Improved Process Workflows

<b>Document Type</b>	Report
<b>Document Number</b>	D4.8
<b>Primary Author(s)</b>	Thomas Bauer (FRAUNHOFER IESE)
<b>Document Date</b>	2022-10-25
<b>Document Version</b>	1.0 Final
<b>Dissemination Level</b>	Public (PU)
<b>Reference DoA</b>	2022-03-03
<b>Project Coordinator</b>	Behrooz Sangchoolie, <a href="mailto:behrooz.sangchoolie@ri.se">behrooz.sangchoolie@ri.se</a> , RISE Research Institutes of Sweden
<b>Project Homepage</b>	<a href="http://www.valu3s.eu">www.valu3s.eu</a>
<b>JU Grant Agreement</b>	876852



This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.



### **Disclaimer**

The views expressed in this document are the sole responsibility of the authors and do not necessarily reflect the views or position of the European Commission. The authors, the VALU3S Consortium, and the ECSEL JU are not responsible for the use which might be made of the information contained in here.

## Project Overview

Manufacturers of automated systems and the manufacturers of the components used in these systems have been allocating an enormous amount of time and effort in the past years developing and conducting research on automated systems. The effort spent has resulted in the availability of prototypes demonstrating new capabilities as well as the introduction of such systems to the market within different domains. Manufacturers of these systems need to make sure that the systems function in the intended way and according to specifications which is not a trivial task as system complexity rises dramatically the more integrated and interconnected these systems become with the addition of automated functionality and features to them.

With rising complexity, unknown emerging properties of the system may come to the surface making it necessary to conduct thorough verification and validation (V&V) of these systems. Through the V&V of automated systems, the manufacturers of these systems can ensure safe, secure and reliable systems for society to use since failures in highly automated systems can be catastrophic.

The high complexity of automated systems incurs an overhead on the V&V process making it time-consuming and costly. VALU3S aims to design, implement, and evaluate state-of-the-art V&V methods and tools in order to reduce the time and cost needed to verify and validate automated systems with respect to safety, cybersecurity and privacy (SCP) requirements. This will ensure that European manufacturers of automated systems remain competitive and that they remain world leaders. To this end, a multi-domain framework is designed and evaluated with the aim to create a clear structure around the components and elements needed to conduct V&V process through identification and classification of evaluation methods, tools, environments, and concepts that are needed to verify and validate automated systems with respect to SCP requirements.

In VALU3S, 13 use cases with specific safety, security and privacy requirements will be studied in detail. Several state-of-the-art V&V methods will be investigated and further enhanced in addition to implementing new methods aiming for reducing the time and cost needed to conduct V&V of automated systems. The V&V methods investigated are then used to design improved process workflows for V&V of automated systems. Several tools will be implemented supporting the improved processes which are evaluated by qualification and quantification of safety, security and privacy as well as other evaluation criteria using demonstrators. VALU3S will also influence the development of safety, security and privacy standards through an active participation in related standardisation groups. VALU3S will provide guidelines to the testing community including engineers and researchers on how the V&V of automated systems could be improved considering the cost, time and effort of conducting the tests.

VALU3S brings together a consortium with partners from 10 different countries, with a mix of *industrial partners* (25 partners) from automotive, agriculture, railway, healthcare, aerospace and industrial automation and robotics domains as well as leading *research institutes* (6 partners) and *universities* (10 partners) to reach the project goal.

## Consortium

RISE RESEARCH INSTITUTES OF SWEDEN AB	RISE	Sweden
STAM SRL	STAM	Italy
FONDAZIONE BRUNO KESSLER	FBK	Italy
KNOWLEDGE CENTRIC SOLUTIONS SL - THE REUSE COMPANY	TRC	Spain
UNIVERSITA DEGLI STUDI DELL'AQUILA	UNIVAQ	Italy
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO	ISEP	Portugal
UNIVERSITA DEGLI STUDI DI GENOVA	UNIGE	Italy
CAMEA, spol. s r.o.	CAMEA	Czech
IKERLAN S. COOP	IKER	Spain
R G B MEDICAL DEVICES SA	RGB	Spain
UNIVERSIDADE DE COIMBRA	COIMBRA	Portugal
VYSOKE UCENI TECHNICKE V BRNE - BRNO UNIVERSITY OF TECHNOLOGY	BUT	Czech
ROBOAUTO S.R.O.	ROBO	Czech
ESKISEHIR OSMANGAZI UNIVERSITESI	ESOGU	Turkey
KUNGLIGA TEKNISKA HOEGSKOLAN	KTH	Sweden
STATENS VAG- OCH TRANSPORTFORSKNINGSINSTITUT	VTI	Sweden
UNIVERSIDAD DE CASTILLA - LA MANCHA	UCLM	Spain
FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	FRAUNHOFER	Germany
SIEMENS AKTIENGESELLSCHAFT OESTERREICH	SIEMENS	Austria
RULEX INNOVATION LABS SRL	RULEX	Italy
NXP SEMICONDUCTORS GERMANY GMBH	NXP-DE	Germany
PUMACY TECHNOLOGIES AG	PUMACY	Germany
UNITED TECHNOLOGIES RESEARCH CENTRE IRELAND, LIMITED	UTRCI	Ireland
NATIONAL UNIVERSITY OF IRELAND MAYNOOTH	NUIM	Ireland
INOVASYON MUHENDISLIK TEKNOLOJI GELISTIRME DANISMANLIK SANAYI VE TICARET LIMITED SIRKETI	IMTGD	Turkey
ERGUNLER INSAAT PETROL URUNLERI OTOMOTIV TEKSTIL MADENCILIK SU URUNLER SANAYI VE TICARET LIMITED STI.	ERARGE	Turkey
OTOKAR OTOMOTIV VE SAVUNMA SANAYI AS - OTOKAR AS	OTOKAR	Turkey
TECHY BILISIM TEKNOLOJILERI DANISMANLIK SANAYI VE TICARET LIMITED SIRKETI - TECHY INFORMATION TECHNOLOGIESAND CONSULTANCY LIMITED COMPANY	TECHY	Turkey
ELECTROTECNICA ALAVESA SL	ALDAKIN	Spain
INTECS SOLUTIONS SPA	INTECS	Italy
LIEBERLIEBER SOFTWARE GMBH	LLSG	Austria
AIT AUSTRIAN INSTITUTE OF TECHNOLOGY GMBH	AIT	Austria
E.S.T.E. SRL	ESTE	Italy
NXP SEMICONDUCTORS FRANCE SAS	NXP-FR	France
BOMBARDIER TRANSPORTATION SWEDEN AB	BT	Sweden
QRTECH AKTIEBOLAG	QRTECH	Sweden
CAF SIGNALLING S.L	CAF	Spain
MONDRAGON GOI ESKOLA POLITEKNIKOA JOSE MARIA ARIZMENDIARRIETA S COOP	MGEP	Spain
INFOTIV AB	INFOTIV	Sweden
BERGE CONSULTING AB	BERGE	Sweden
CARDIOID TECHNOLOGIES LDA	CARDIOID	Portugal

## Executive Summary

This deliverable is part of WP4, which focusses on designing and implementing tailored V&V process workflows and documents the final results from Task 4.2 “Initial detailed description of improved process workflows”. Task 4.2 developed specific workflows and solution patterns for verification and validation that address the challenges and goals stated by the industrial use cases. The goal is to bring partner-specific and tool-specific workflows and contributions into a holistic and integrated verification and validation process.

Within the scope of Task 4.2 and deliverable D4.8, KPI-4 of the project proposal is being addressed, which deals with the development of at least 13 novel tailored V&V workflows that will improve the time and cost of V&V processes. Finally, 42 workflows have been modelled for the VALU3S use cases.

As a result, the deliverable D4.8 describes the final results from the analysis and modelling activities of the verification and validation workflows. It is an update of D4.6 [1], which contained the intermediate version of the V&V workflows. Parts of the content of the document remain unchanged compared to D4.6, while other sections have been newly created.

Parts of the workflows reference virtual validation and virtual prototyping solutions, which have been addressed in D4.3 [2]. The workflows contained in D4.8 apply tools that are being developed in Task 4.3 and described in D4.9 - *Final implementation of V&V tools suitable for the improved process workflows*.



## Contributors

Matt Luckcuck	NUIM	Thomas Bauer	FRAUNHOFER IESE
Georgios Giantamidis	UTRCI	Metin Ozkan	ESOGU
Stylianos Basagiannis	UTRCI	Unai Muñoz	IKER
Gürol Çokünlü	OTOKAR	Xabier Mendialdua	IKER
Muharrem Saral	OTOKAR	José Proença	ISEP
Ömer Şahabaş	OTOKAR	Martin Hrubý	BUT
Rupert Schlick	AIT	Manuel Schmidt	NXP
José Luis de la Vara	UCLM	Giovanni Giachetti	UCLM
Luis Alonso	TRC	Arturo García	UCLM
Hamid Ebadi	INFOTIV	Martin Karsberg	INFOTIV
Ugur Yayan	IMTGD	Mustafa Karaca	IMTGD
Thorsten Tarrach	AIT	Jack Jensen	BERGE
Aleš Smrčka	BUT	Thanh Bui	RISE
Lukáš Maršík	CAMEA	Joakim Rosell	RISE
Mateen Malik	RISE	Peter Folkesson	RISE
Cem Baglum	IMTGD	Alim Kerem Erdogmus	IMTGD
Marie Farrell	NUIM	Bernhard Fischer	SIEMENS
Oisin Sheridan	NUIM	Martin Matschnig	SIEMENS
Rosemary Monahan	NUIM	Ricardo Ruiz	RGB
Beáta Davidová	ROBO	Aitor Agirre	MGEP
Katia Di Blasio	INTECS	Mikel Aldalur	IKER
Maytheewat Aramrattana	VTI	Juan Manuel Morote	UCLM
Bernd Bredehorst	PUMACY	Zain Shahwar	PUMACY
Sina Borrami	Alstom	Wolfgang Herzner	AIT

## Reviewers

Mustafa Karaca	IMTGD	2022-04-06
Alim Kerem Erdogmus	IMTGD	2022-04-05; 2022-04-14; 2022-09-30
Cem Baglum	IMTGD	2022-04-05; 2022-04-14; 2022-10-03
Wolfgang Herzner	AIT	2022-04-04; 2022-04-19; 2022-10-07; 2022-10-13
Ali Sedaghatbaf	RISE	2022-04-05; 2022-04-14; 2022-10-04; 2022-10-13
Behrooz Sangchoolie	RISE	2022-04-24; 2022-10-25
Bob Hruska	LLSG	2022-10-14

## Revision History

Version	Date	Author (Affiliation)	Comment
0.1	2022-09-26	Thomas Bauer (FRAUNHOFER IESE)	First version created
0.2	2022-10-12	Thomas Bauer (FRAUNHOFER IESE)	Chapter 2 updated; workflow figures in chapter 3 updated; comments from 1 <sup>st</sup> review incorporated;
0.3	2022-10-20	Thomas Bauer (FRAUNHOFER IESE)	Workflow descriptions updated; comments from 2 <sup>nd</sup> review incorporated;
0.4	2022-10-24	Behrooz Sangchoolie (RISE)	Review of the first final draft of the deliverable while making minor formatting changes and leaving additional comments to be addressed.
0.5	2022-10-25	Thomas Bauer (FRAUNHOFER IESE)	Final format changes; conclusion updated
0.6	2022-10-25	Behrooz Sangchoolie (RISE)	Review of the final draft of the deliverable while making minor formatting changes.
1.0	2022-10-25	Behrooz Sangchoolie (RISE)	Report to be submitted.



# Table of Contents

Table of Contents.....	9
List of Figures .....	12
List of Tables .....	16
1 Introduction .....	21
1.1 Scope.....	21
1.2 Document Structure .....	21
2 V&V Workflow Modelling Languages .....	23
2.1 Diagram Types.....	23
2.1.1 V&V Method Definition .....	23
2.1.2 V&V Workflow Definition .....	24
3 VALU3S V&V Workflows.....	27
3.1 V&V Workflow of UC1 CAMEA.....	27
3.1.1 Artifacts used in UC1_CAMEA .....	28
3.1.2 V&V Workflows of V&V of Machine Learning-Based Systems Using Simulators .....	29
3.1.3 V&V Workflows of Model-Based Threat Analysis .....	33
3.1.4 V&V Workflows of Assessment of Implementation of Network Communication .....	36
3.2 V&V Workflow of Use Case 2 ROBO .....	39
3.2.1 Artifacts used in UC2_ROBO.....	40
3.2.2 V&V Workflows of UC2 ROBO V&V Workflow .....	42
3.2.3 V&V Workflows of Simulated Fault-Injection of a Network Link .....	43
3.2.4 V&V Workflows of Simulation-Based Fault and Attack Injection at System-level Improved.....	46
3.2.5 V&V Workflows of UC2 Daily Regression Test .....	50
3.3 V&V Workflow of Use Case 3 NXP .....	52
3.3.1 Artifacts used in UC3_NXP.....	52
3.3.2 V&V Workflows of Use Case 3 Radar Systems for ADAS.....	54
3.3.3 V&V Workflows of Doppler Division Multiplexing Access (DDMA) .....	57
3.4 V&V Workflow of Use Case 4 PUMACY .....	59
3.4.1 Artifacts used in UC4_PUMACY .....	61
3.4.2 V&V Workflows of Combined Virtual Validation and Failure Detection Diagnosis .....	62
3.4.3 V&V Workflows of Failure Detection Diagnosis.....	63
3.4.4 V&V Workflows of Virtual Validation .....	65



3.5	V&V Workflow(s) of UC5 UTRCI .....	68
3.5.1	Artifacts used in UC5_UTRCI.....	70
3.5.2	V&V Workflows of Verifying and Refactoring Formalised Requirements .....	71
3.5.3	V&V Workflows of Model-implemented Fault and Attack Injection with Pre-Injection Analysis .....	73
3.5.4	V&V Workflows of SiLVer (SimuLation-based Verification) .....	75
3.6	V&V Workflow of Use Case 6 ESTE .....	78
3.7	V&V Workflow of Use Case 7 ALDAKIN.....	79
3.7.1	Artifacts used in UC7_ALDAKIN .....	79
3.7.2	V&V Workflows of MGEP V&V Workflow .....	80
3.8	V&V Workflow of Use Case 8 RGB.....	85
3.8.1	Artifacts used in UC8_RGB .....	88
3.8.2	V&V Workflows of Tailored Model-Based Assurance and Certification .....	90
3.8.3	V&V Workflows of Model Based Safety Analysis FLA.....	92
3.8.4	V&V Workflows of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis .....	94
3.8.5	V&V Workflows of Extended Knowledge-Centric System Traceability Management .....	96
3.8.6	V&V Workflows of Single Experiment .....	97
3.8.7	V&V Workflows of TC Automated Experimenting.....	99
3.8.8	V&V Workflows of TC Management.....	100
3.9	V&V Workflow of Use Case 9 CAF.....	102
3.9.1	Artifacts used in UC9_CAF .....	103
3.9.2	V&V Workflows of Overall UC9 Method .....	103
3.9.3	V&V Workflows of Simulation-Based V&V of Computer Vision System .....	105
3.10	V&V Workflow of Use Case 10 BT .....	108
3.10.1	Artifacts used in UC10_BT .....	110
3.10.2	V&V Workflows of UC10 Overall Method .....	111
3.10.3	V&V Workflows of Model Checking Families of Real-Time Specifications.....	114
3.10.4	V&V Workflows of Optimize Fault Injection Experiments Using Model-Based Mutation Testing .....	116
3.10.5	V&V Workflows of Behaviour-Driven Model Development and Test-Driven Model Review . .....	118
3.11	V&V Workflow of Use Case 11 OTOKAR.....	120
3.11.1	Artifacts used in UC11_OTOKAR.....	122

3.11.2	V&V Workflows of Model-Based Formal Specification and Verification of Robotic Systems ..	123
3.11.3	V&V Workflows of Penetration Testing.....	127
3.11.4	V&V Workflows of Simulation-Based Verification.....	129
3.11.5	V&V Workflows of Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks .....	131
3.12	V&V Workflow of Use Case 13 SIEMENS .....	135
3.12.1	Artifacts used in UC13_SIEMENS.....	136
3.12.2	V&V Workflows of UC13 - SIEMENS.....	138
3.12.3	V&V Workflows of Model-Based Mutation Testing.....	141
3.12.4	V&V Workflows of Monitoring Enriched Test Execution .....	143
3.12.5	Mutation-Driven Model-Based Test Case Generation.....	144
3.13	V&V Workflow of Use Case 14 CARDIOID .....	146
3.13.1	Artifacts used in UC14_CARDIOID.....	148
3.13.2	V&V Workflows of Biometric Model Performance and Privacy Validation.....	150
3.13.3	V&V Workflows of Hardware in the Loop Validation & Verification.....	152
3.13.4	V&V Workflows of Safe Generation and Instrumentation of Runtime Verification Architectures.....	153
3.13.5	V&V Workflows of Software-Implemented Fault Injection .....	155
3.13.6	V&V Workflows of Verification of Driver Monitoring Models .....	156
4	Conclusion.....	159
	References .....	161

## List of Figures

Figure 2.1 Visual representation of the Method definition diagram and its elements structure .....	23
Figure 3.1 Method Definition of V&V of machine learning-based systems using simulators defined for UC1_CAMEA .....	27
Figure 3.2 Method Definition of Model-Based Threat Analysis defined for UC1_CAMEA.....	28
Figure 3.3 Method Definition of Assessment of implementation of network communication defined for UC1_CAMEA .....	28
Figure 3.4 Workflow Definition diagram of V&V of machine learning-based systems using simulators - Workflow used in UC1_CAMEA.....	31
Figure 3.5 Workflow Definition diagram of Model-Based Threat Analysis - Workflow used in UC1_CAMEA .....	35
Figure 3.6 Workflow Definition diagram of Assessment of implementation of network communication used in UC1_CAMEA.....	37
Figure 3.7 Method Definition of ROBO V&V defined for UC2_ROBO .....	39
Figure 3.8 Method Definition of Simulated fault-injection of a network link defined for UC2_ROBO .	39
Figure 3.9 Method Definition of ComFASE_RISE_VTI_Improve defined for UC2_ROBO.....	40
Figure 3.10 Workflow Definition diagram of UC2_ROBO Workflow used in UC2_ROBO .....	42
Figure 3.11 Workflow Definition diagram of Simulated fault-injection of a network link used in UC2_ROBO .....	45
Figure 3.12 Workflow Definition diagram of Simulation-Based Fault and Attack Injection at System-level with additional fault and attack models valid for multiple inter-vehicle communication (IVC) layers used in UC2_ROBO.....	48
Figure 3.13 Workflow Definition diagram of Regression tests workflow used in UC2_ROBO.....	51
Figure 3.14 Method Definition of Use Case 3 defined for UC3_NXP .....	52
Figure 3.15 Workflow Definition diagram of d Use Case 3 Radar systems for ADAS used in UC3_NXP .....	55
Figure 3.16 Workflow Definition diagram of Doppler Division Multiplexing Access (DDMA) used in UC3_NXP .....	58
Figure 3.17 Method Definition of Combined Virtual Validation and Failure Detection Diagnosis defined for UC4_PUMACY.....	59
Figure 3.18 Method Definition of Failure Detection Diagnosis defined for UC4_PUMACY .....	60
Figure 3.19 Method Definition of Virtual Validation defined for UC4_PUMACY .....	60
Figure 3.20 Workflow Definition diagram of Combined Virtual Validation and Failure Detection Diagnosis used in UC4_PUMACY.....	62
Figure 3.21 Workflow Definition diagram of Failure Detection Diagnosis used in UC4_PUMACY .....	64
Figure 3.22 Workflow Definition diagram of Virtual Validation used in UC4_PUMACY.....	66
Figure 3.23 Method Definition of SILVER_UTRCI defined for UC5_UTRCI .....	68
Figure 3.24 Method Definition of MIFI_MIAI_RISE (pre-injection) defined for UC5_UTRCI.....	69
Figure 3.25 Method Definition of Verifying and Refactoring Formalised Requirements defined for UC5_UTRCI .....	69

Figure 3.26 Workflow Definition diagram of Verifying and Refactoring Formalised Requirements used in UC5_UTRCI.....	71
Figure 3.27 Workflow Definition diagram of MIFI_MIAI_RISE used in UC5_UTRCI .....	74
Figure 3.28 Workflow Definition diagram of SiLVer (SimuLation-based Verification) used in UC5_UTRCI..	76
Figure 3.29 Method Definition of UC6_ESTe defined for UC6_ESTe .....	78
Figure 3.30 Method Definition of MGEP-2 UC7 defined for UC7_ALDAKIN.....	79
Figure 3.31 Method Definition of MGEP V&V Workflow used in UC7_ALDAKIN .....	83
Figure 3.32 Method Definition of TC Management defined for UC8_RGB .....	85
Figure 3.33 Method Definition of TC Automated experimenting for UC8_RGB .....	85
Figure 3.34 Method Definition of Single Experiment for UC8_RGB.....	86
Figure 3.35 Method Definition of Model based Safety Analysis FLA defined for UC8_RGB .....	86
Figure 3.36 Method Definition of Tailored Model-based Assurance and Certification defined for UC8_RGB .....	87
Figure 3.37 Method Definition of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis defined for UC8_RGB .....	87
Figure 3.38 Method Definition of Extended Knowledge-Centric System Traceability Management defined for UC8_RGB .....	88
Figure 3.39 Model-Based Assurance and Certification used in UC8_RGB .....	91
Figure 3.40 Workflow Definition diagram of Model Based Safety Analysis FLA used in UC8_RGB .....	93
Figure 3.41 Workflow Definition diagram of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis used in UC8_RGB.....	95
Figure 3.42 Workflow Definition diagram of Extended Knowledge-Centric System Traceability Management used in UC8_RGB .....	96
Figure 3.43 Workflow Definition diagram of Single Experiment used in UC8_RGB.....	98
Figure 3.44 Workflow Definition diagram of TC Auto Experiment used in UC8_RGB.....	99
Figure 3.45 Workflow Definition diagram of TC Management used in UC8_RGB .....	100
Figure 3.46 Method Definition of Overall UC9 workflow defined for UC9_CAF .....	102
Figure 3.47 Method Definition of Simulation based V&V of Computer Vision System defined for UC9_CAFs.....	102
Figure 3.48 Workflow Definition diagram of UC9_VV_Method used in UC9_CA .....	104
Figure 3.49 Workflow Definition diagram of Simulation based V&V of Computer Vision used in UC9_CAF_Submethods.....	106
Figure 3.50 Method Definition of UC10 Overall Method defined for UC10_BT .....	108
Figure 3.51 Method Definition of Model Checking Families of Real Time Systems - Method defined for UC10_BT.....	109
Figure 3.52 Method Definition of Optimize Fault Injection Experiments Using Model-Based Mutation Testing defined for UC10_BT .....	109
Figure 3.53 Behaviour-driven model development and test-driven model review.....	110
Figure 3.54 Workflow Definition diagram of UC10 Overall Method Workflow used in UC10_BT .....	112
Figure 3.55 Workflow Definition diagram of Model Checking Families of Real Time Systems used in UC10_BT.....	115
Figure 3.56 Workflow Definition diagram of Optimize Fault Injection Experiments Using Model-Based Mutation Testing used in UC10_BT.....	117

Figure 3.57 Workflow Definition diagram of Behaviour-driven model development and test-driven model review used in UC10_BT.....	118
Figure 3.58 Method Definition of UC11_OTOKAR_2_Penetration_testing defined for UC11_OTOKAR .....	120
Figure 3.59 Method Definition of Model-Based Formal Specification and Verification of Robotic Systems defined for UC11_OTOKAR.....	121
Figure 3.60 Method Definition of Simulation-based Verification defined for UC11_OTOKAR.....	121
Figure 3.61 Method Definition of Vulnerability Analysis of Cryptographic Modules Against Hardware-Based Attacks defined for UC11_OTOKAR.....	122
Figure 3.62 Workflow Definition diagram of MBF used in UC11_OTOKAR.....	125
Figure 3.63 Workflow Definition diagram of Penetration Testing used in UC11_OTOKAR.....	128
Figure 3.64 Workflow Definition diagram of Simulation-based Verification - Workflow used in UC11_OTOKAR .....	130
Figure 3.65 Workflow Definition diagram of Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks used in UC11_OTOKAR .....	133
Figure 3.66 Method Definition of UC13 - SIEMENS defined for UC13_SIEMENS.....	135
Figure 3.67 Method Definition of Model-Based Mutation Testing defined for UC13_SIEMENS.....	135
Figure 3.68 Method Definition of Monitoring Enriched Test Execution defined for UC13_SIEMENS	136
Figure 3.69 Method Definition of Mutation-Driven Model-Based Test Case Generation defined for UC13_SIEMENS.....	136
Figure 3.70 Workflow Definition diagram of UC13 - SIEMENS used in UC13_SIEMENS .....	139
Figure 3.71 Workflow Definition diagram of Model-Based Mutation Testing - Workflow used in UC13_SIEMENS.....	142
Figure 3.72 Workflow Definition diagram of «Method» Monitoring Enriched Test Execution used in UC13_SIEMENS.....	143
Figure 3.73 Workflow Definition diagram of Mutation-Driven Model-Based Test Case Generation used in UC13 .....	144
Figure 3.74 Method Definition of Biometric Model Performance and Privacy Validation defined for UC14_CARDIOID.....	146
Figure 3.75 Method Definition of Hardware in the Loop Validation & Verification defined for UC14_CARDIOID.....	147
Figure 3.76 Method Definition of Verification of Driver Monitoring Models defined for UC14_CARDIOID.....	147
Figure 3.77 Method Definition of Safe Generation and Instrumentation of Runtime Verification Architectures defined for UC14_CARDIOID.....	148
Figure 3.78 Method Definition of Software-Implemented Fault Injection defined for UC14_CARDIOID .....	148
Figure 3.79 Workflow Definition diagram of Biometric Model Performance and Privacy Validation used in UC14_CARDIOID.....	151
Figure 3.80 Workflow Definition diagram of Hardware in the Loop Validation & Verification used in UC14_CARDIOID.....	152
Figure 3.81 Workflow Definition diagram of Safe Generation and Instrumentation of Runtime Verification Architectures used in UC14_CARDIOID.....	154



Figure 3.82 Workflow Definition diagram of Software Implemented Fault Injection used in UC14\_CARDIOID ..... 156

Figure 3.83 Workflow Definition diagram of Verification of Driver Monitoring Models used in UC14\_CARDIOID ..... 157



## List of Tables

Table 2.1 VVML Method Definition Elements .....	24
Table 2.2 VVML Workflow Definition Elements .....	25
Table 3.1 List of artifact types used in UC1_CAMEA .....	29
Table 3.2 List of activities performed by V&V of machine learning-based systems using simulators ..	32
Table 3.3 List of activities performed by Model-Based Threat Analysis .....	36
Table 3.4 List of activities performed by Assessment of implementation of network communication .	37
Table 3.5 List of artifact types used in UC2_ROBO .....	40
Table 3.6 List of activities performed by UC2 ROBO V&V Workflow .....	43
Table 3.7 List of activities performed by Simulated fault-injection of a network link.....	45
Table 3.8 List of activities performed by Simulation-Based Fault and Attack Injection at System-level Improved .....	49
Table 3.9 List of activities performed by UC2 Daily regression test .....	51
Table 3.10 List of artifact types used in UC3_NXP .....	52
Table 3.11 List of activities performed by Use Case 3 Radar systems for ADAS .....	56
Table 3.12 List of activities performed by Doppler Division Multiplexing Access (DDMA) .....	58
Table 3.13 List of artifact types used in UC4_PUMACY .....	61
Table 3.14 List of activities performed by UC4_Combined Virtual Validation and Failure Detection Diagnosis .....	63
Table 3.15 List of activities performed by Failure Detection Diagnosis .....	64
Table 3.16 List of activities performed by Virtual Validation .....	66
Table 3.17 List of artifact types used in UC5_UTRCI .....	70
Table 3.18 List of activities performed by Verifying and Refactoring Formalised Requirements .....	72
Table 3.19 List of activities performed by Model-implemented fault/attack injection with pre-injection analysis .....	75
Table 3.20 List of activities performed by SiLVer (SimuLation-based Verification).....	77
Table 3.21 List of artifact types used in UC7_ALDAKIN .....	79
Table 3.22 List of activities performed by MGEP V&V Workflow .....	84
Table 3.23 List of artifact types used in UC8_RGB .....	88
Table 3.24 List of activities performed by Tailored Model-Based Assurance and Certification .....	91
Table 3.25 List of activities performed by Model Based Safety Analysis FLA .....	94
Table 3.26 List of activities performed by Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis.....	95
Table 3.27 List of activities performed by Extended Knowledge-Centric System Traceability Management .....	97
Table 3.28 List of activities performed by Single Experiment.....	98
Table 3.29 List of activities performed by TC Automated Experimenting.....	99
Table 3.30 List of activities performed by TC Management.....	100
Table 3.31 List of artifact types used in UC9_CAF .....	103
Table 3.32 List of activities performed by Overall UC9 Method .....	104
Table 3.33 List of activities performed by Simulation based V&V of Computer Vision System .....	106



Table 3.34 List of artifact types used in UC10_BT .....	110
Table 3.35 List of activities performed by UC10 Overall Method .....	112
Table 3.36 List of activities performed by Model Checking Families of Real-Time Specifications.....	115
Table 3.37 List of activities performed by Optimize Fault Injection Experiments Using Model-Based Mutation Testing .....	117
Table 3.38 List of activities performed by Behaviour-driven model development and test-driven model review .....	119
Table 3.39 List of artifact types used in UC11_OTOKAR .....	122
Table 3.40 List of activities performed by Model-Based Formal Specification and Verification of Robotic Systems .....	126
Table 3.41 List of activities performed by Penetration Testing.....	128
Table 3.42 List of activities performed by Simulation-based Verification.....	130
Table 3.43 List of activities performed by Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks.....	133
Table 3.44 List of artifact types used in UC13_SIEMENS .....	137
Table 3.45 List of activities performed by UC13 - SIEMENS .....	140
Table 3.46 List of activities performed by Model-Based Mutation Testing.....	142
Table 3.47 List of activities performed by Monitoring Enriched Test Execution .....	143
Table 3.48 List of activities performed by Mutation-Driven Model-Based Test Case Generation.....	145
Table 3.49 List of artifact types used in UC14_CARDIOID .....	149
Table 3.50 List of activities performed by Biometric Model Performance and Privacy Validation .....	151
Table 3.51 List of activities performed by Hardware in the Loop Validation & Verification .....	153
Table 3.52 List of activities performed by Safe Generation and Instrumentation of Runtime Verification Architectures .....	154
Table 3.53 List of activities performed by Software-Implemented Fault Injection .....	156
Table 3.54 List of activities performed by Verification of Driver Monitoring Models .....	157



## Acronyms

ACC	Adaptive Cruise Control
ADAS	Advanced Driver-Assisted System
COTS	Commercial off-the-shelf
CV	Computer Vision
DSL	Domain-Specific Language
FDD	Failure Detection Diagnosis
ISO	International Organization for Standardization
KPI	Key Performance Indicator
MBT	Model-based Testing
ML	Machine Learning
NMT	NeuroMuscular Transmission
RADAR	RAdio Detection And Ranging
SCP	Safety, Cybersecurity, and Privacy
SoC	System on a Chip
SUT	System under Test
VaV	Verification and Validation; used for names in tools, where the symbol '&' cannot be used
V&V	Verification and Validation
VVML	Verification and Validation Workflow Modelling Language
WP	Work Package



# 1 Introduction

The efficient conducting of software development and quality assurance activities in complex projects require their systematic description and modelling (including their sub-activities, execution steps, and work products that they process and produce) and the provision of appropriate tool support for executing the activities.

In WP4, a generic V&V workflow design approach and modelling language has been developed to easily visualize V&V-oriented workflows in industrial use cases and concrete tool chains and facilitate the understanding, analysis, and improvement of these workflows. The solution has paved the way towards the efficient evaluation and optimization of V&V workflows and tool chains for selected quality properties. The development of the V&V workflow design approach has been performed in close connection with the V&V method library to support the systematic description, extension, and gap analysis of V&V methods. In the next step, the V&V workflows will be transferred to the web-based repository as project artifacts.

## 1.1 Scope

Within VALU3S, WP4 deals with the design and implementation of tailored V&V process workflows in different industrial domains. Task 4.2 develops specific workflows and solution patterns for verification and validation that address the challenges and goals stated by the industrial use cases. The goal is to bring partner-specific and tool-specific workflows and contributions into a holistic and integrated verification and validation process. One solution item of Task 4.2 is the systematic and tool-supported analysis and modelling of V&V workflows. A further strategy is the virtualization of the V&V process by exploitation models, prototypes, and digital twins for dedicated product and process aspects to improve and accelerate quality assurance processes.

The outcomes presented in D4.8 focus on the final detailed description of improved process workflows. The document describes the tool-supported modelling language VVML (Verification and Validation Modelling Language) and the final set of V&V workflows for the VALU3S use cases. The results and information from the interim version of V&V workflows (D4.6 [1]) have been used as inputs.

## 1.2 Document Structure

This document is structured as follows: The modelling language for verification and validation workflows VVML is introduced in Chapter 2, which is slightly updated from D4.6 [1]. The final set of V&V workflows designed for VALU3S use cases are presented in Chapter 3, which contains an adapted structure and new content. Most of the content has been generated from Enterprise Architect, which was used as the modelling framework for V&V workflows. Chapter 4 provides a summary and conclusion of the work done and results achieved in Task 4.2.



## 2 V&V Workflow Modelling Languages

In modelling languages such as UML, it is possible to represent the same idea in many ways. While the flexibility that the language offers, is a positive aspect, it also brings problems in communicating ideas effectively. Not everyone is a UML expert or knows every feature that the modelling language offers. By creating a DSL which clearly specifies what diagrams and elements can be used in creating a V&V method definition or its workflow, everyone follows a common standardized language. Modelling V&V workflows falls into a specialized domain that requires a tailored modelling approach for activity models. To meet such requirements, it was decided to develop a UML profile for V&V Modelling Language – shortly *VVML profile* – introducing a set of model constructs and deploy the UML profile with other extension mechanisms as a modelling framework enabling rapid modelling of V&V workflows. The tool environment, in which VVML is implemented, is Enterprise Architect (EA), a UML modelling tool by Sparx Systems.

In the following, essential VVML aspects are introduced to ease reading of the diagrams used in Chapter 3. A detailed description of VVML, in particular rules and guidelines for its usage will be given in an own VVML-handbook, which is currently being prepared and will be made available to the community at the end of the VALU3S project.

### 2.1 Diagram Types

Two diagram types are distinguished in VVML, corresponding to two modelling levels:

- V&V Method Definition
- V&V Workflow Specification

#### 2.1.1 V&V Method Definition

This diagram type serves for specifying global properties of a VVML method applied in the project. Figure 2.1 shows an example of a method definition diagrams with its main elements.

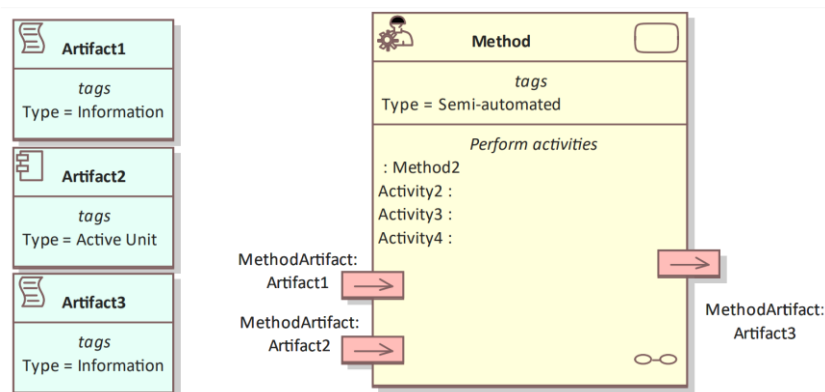







Figure 2.1 Visual representation of the Method definition diagram and its elements structure

The V&V method definition enables the design of the base elements of the workflow. The modelling element *Method* is a unit that represents a process workflow dedicated to a specific V&V phase. It has a defined method type, which is used to represent the automation level. Three automation levels are considered here: automated, semi-automated, or manual. An *Artifact* is an object that is exchanged between methods and its environment (or activities within methods, see next clause). It has a dedicated type and represents either an information object or an active unit, i.e., program code or executable.

Every method owns a set of *MethodArtifacts*, which represent the method interfaces for the artifacts that they consume or produce. Table 2.1 describes the elements of VVML method definition diagrams.

Table 2.1 VVML Method Definition Elements




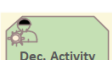





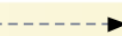
Element	Description
Method	<p>Represents a high-level definition of a «Method». It organizes and specifies the participation of subordinate behaviours, such as sub-Activities, to reflect the control and «Artifact» flow of a process.</p> <p>The icon in the top left corner indicates the automation level which can be one of the following:</p> <ul style="list-style-type: none"> <li>Manual , Automated , Semi-automated </li> </ul> <p>A method has parameter, called «MethodArtifacts» (which are of type «Artifact»).</p> <p>Besides MethodArtifacts, in the body of the method box the activities contained in the methods workflow are listed, as well as other methods called within its workflow.</p>
MethodArtifact	<p>A «MethodArtifact» is an «Artifact» exposed to the method environment with a defined flow direction (in or out). This element indicates which artifacts are provided or required by a method.</p>
Artifact	<p>Represents a data object, document either produced (output) or consumed (input) or a functional (active) unit used by a «Method» / «Activity» (depending on the level of abstraction). The icon in the top left corner indicates the artifact type which can be one of the following:</p> <ul style="list-style-type: none"> <li>Information , Active Unit </li> </ul>

### 2.1.2 V&V Workflow Definition

The actual implementation of the workflow within a V&V method (V&V Workflow) is specified by the V&V workflow definition diagram. Its main purpose is to organize and specify the composition of activities, to reflect their sequential dependencies and the internal flow of artifacts while executing the method. Table 2.2 presents the elements of the V&V workflow definition and implementation.



Table 2.2 VVML Workflow Definition Elements

Element	Symbol	Description
Start Workflow		Node that initiates the beginning of a workflow
Stop Workflow		Node that indicates the end of a workflow
Activity		Atomic action that is not further decomposed into steps
Decision Activity		Combination of Activity and Gateway. Appropriate if the decision needs input and/or activity, but produces no output besides the decision.
Call Behaviour		Invocation of another method (with another method workflow diagram)
Activity Artifact		Activity interface for its input and output artifacts
Gateway		Branching of sequence flow based on condition
Fork / Join		Enables parallel sub-paths of sequence and artifact flows
Sequence Flow		Sequential connection of VVML activities
Artifact Flow		Exchange of artifacts between activities or from/to method interfaces

A workflow defines Control Flows and Artifact Flows. A Control Flow is defined by sequences of Activities that are executed in a defined order. Branches in the Control Flow are supported by Gateways. Quasi parallel execution is realized by Fork and Join Elements. Start and End Nodes indicate beginning and ending of a workflow. Activities can exchange Artifacts through their interfaces, which define the Artifact Flow of the workflow. The internal Artifact Flow is defined between activities, whereas the external Artifact Flow is defined from the method interfaces to the activities for method inputs or from the activities to the method interfaces for method outputs.

For examples on how V&V-workflows are modelled with these elements, see the Chapter 3.



### 3 VALU3S V&V Workflows

This chapter elaborates on the interim set of V&V workflows that have been modelled for industrial use cases in VALU3S. Partners directly modelled their V&V workflows in Enterprise Architect and generated the content for the following sections. Due to the difference in focus and technical setup of the use cases, the workflow figures and descriptions differ in the level of detail. Note that the provider of use case 12 left the consortium in the end of 2020. The following subchapters are entitled after the use cases, with the name of the use case provider in square brackets.

#### 3.1 V&V Workflow of UC1 CAMEA

UC1\_CAMEA package contains the following workflow(s):

- V&V of machine learning-based systems using simulators
- Model-Based Threat Analysis
- Assessment of implementation of network communication

Figure 3.1 shows the V&V of machine learning-based systems using simulator Method Definition diagram type of the V&V workflow UC1\_CAMEA.

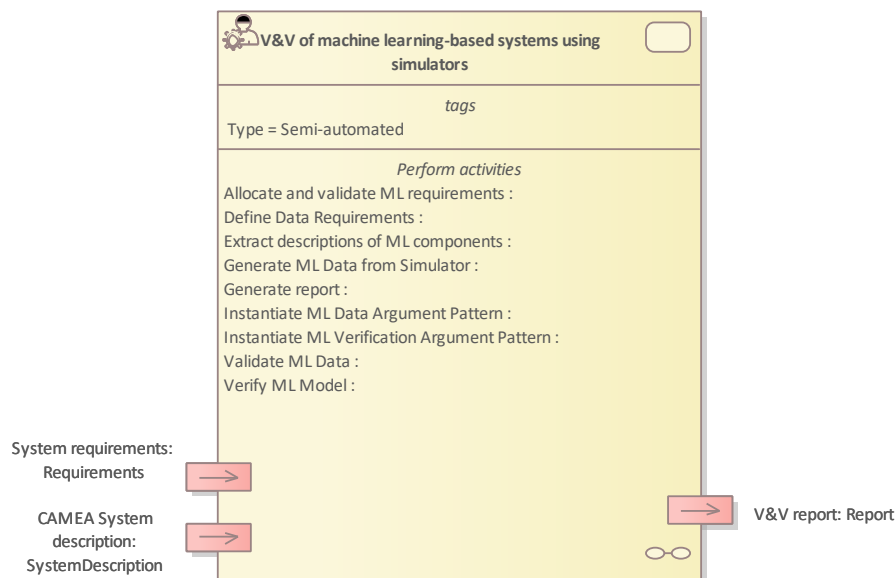


Figure 3.1 Method Definition of V&V of machine learning-based systems using simulators defined for UC1\_CAMEA

Figure 3.2 shows the Model-Based Threat Analysis Method Definition diagram type of the V&V workflow UC1\_CAMEA.

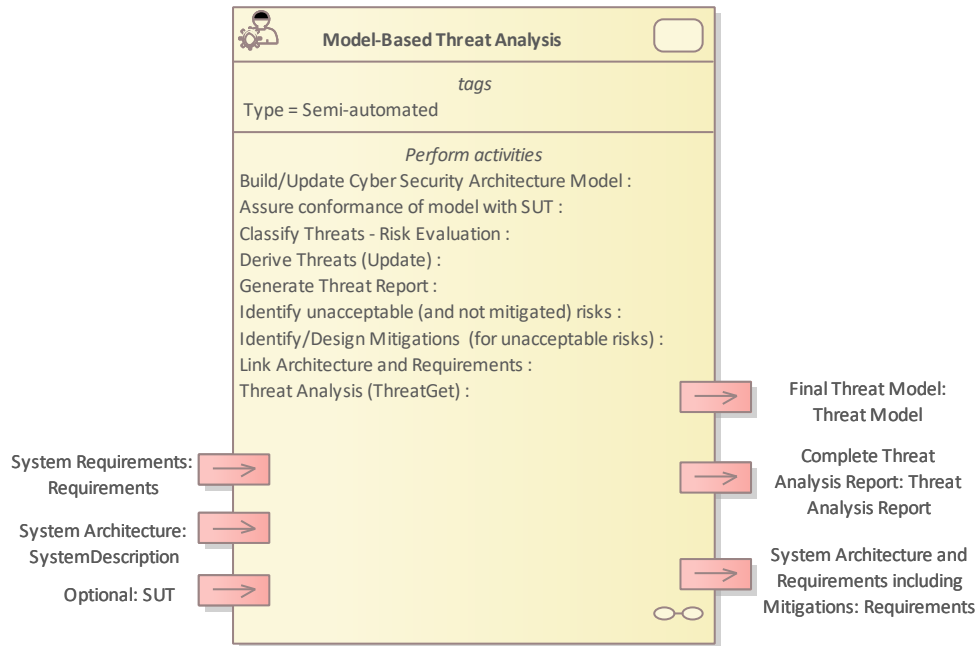


Figure 3.2 Method Definition of Model-Based Threat Analysis defined for UC1\_CAMEA

Figure 3.3 shows the Assessment of implementation of network communication Method Definition diagram type of the V&V workflow UC1\_CAMEA.

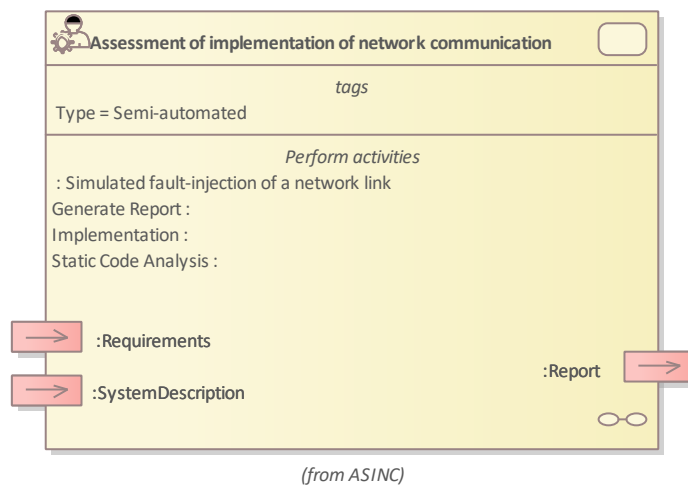


Figure 3.3 Method Definition of Assessment of implementation of network communication defined for UC1\_CAMEA

Details on the workflow(s) are given in the following subsections.

### 3.1.1 Artifacts used in UC1\_CAMEA

Table 3.1 lists the artifacts used for the workflow(s) defined for UC1\_CAMEA.

Table 3.1 List of artifact types used in UC1\_CAMEA

Name	Description
Report (Information)	A written document that consolidates analysis results from an activity
Requirements (Information)	A requirement describes a condition or capability to which a system/component must conform
SUT (Active Unit)	Principally, this method is intended to be executed before the SUT is designed. If, however, it exists already, it should be modified according to the findings during method execution.
System Description (Information)	Descriptions and architecture of a system
Threat Analysis Report (Information)	The output of the method is a report that describes the output of the analysis.
Threat Model (Information)	The model is an architecture diagram of the system.

### 3.1.2 V&V Workflows of V&V of Machine Learning-Based Systems Using Simulators

Machine learning is a critical enabling technology for many of the highly automated applications today. Typical examples include intelligent transport systems (ITS) where ML solutions are used to extract a digital representation of the traffic context from the highly dimensional sensor inputs. Unfortunately, the ML models are opaque in nature (stochastic and data driven with limited output interpretability), while functional safety requirements are strict and require a corresponding safety case. Furthermore, development of systems that rely on deep learning introduces new types of faults. To meet the increasing needs of trusted ML-based solutions, numerous V&V approaches have been proposed. Simulators can be used to support system testing as part of V&V of SCP requirements. An ideal simulator to test perception, planning and decision-making components of an autonomous system must realistically simulate the environment, sensors and their interaction with the environment through actuators. Simulated environments bring several benefits to V&V of ML-based systems, particularly when (i) data collection or data annotation is difficult, costly or time consuming, (ii) real-world testing is endangering human safety, (iii) coverage of collected data is limited, and (iv) Reproducible and scalability are important.

The major bulk of system-level testing of autonomous features in the automotive industry is carried out through on-road testing or using naturalistic field operational tests. These activities, however, are expensive, dangerous, and ineffective. A feasible and efficient alternative is to conduct system-level testing through computer simulations that can capture the entire self-driving vehicle and its operational environment using effective and high-fidelity physics-based simulators. There is a growing number of public-domain and commercial simulators that have been developed over the past few years to support realistic simulation of self-driving systems, e.g., TASS/Siemens PreScan [3], CARLA [4], LGSVL [5], and BeamNG [6]. Simulators will play an important role in the future of automotive V&V, as simulation is recognized as one of the main techniques in ISO/PAS 21448 [7]. As the possible input space when testing automotive systems is practically infinite, attempts to design test cases for comprehensive testing over the space of all possible simulation scenarios are futile. Hence, search-based software testing has been advocated as an effective and efficient strategy to generate test scenarios in simulators. Another line of

research proposes techniques to generate test oracles, i.e., mechanisms for determining whether a test case has passed or failed. Related to the oracle problem, several authors proposed using metamorphic testing of ML-based perception systems, i.e., executing transformed test cases while expecting the same output. Such transformations are suitable to test in simulated environments, e.g., applying filters on camera input or modifying images using generative adversarial networks.

Inspired by AMLAS [8] process, the method "V&V of machine learning-based systems using simulators" is designed to work with ITS surveillance domain. The process starts with allocating the system requirements to ML-component requirements, and subsequently requirements for verification data. The Scenario Generator tool, realizing different algorithms (such as search-based testing) to generate test case scenarios that can later be imported into a realistic simulator (that can either be open-source solutions or proprietary ones). The simulator will then synthesize sensor responses of these scenarios to build the verification data that fulfill the data requirements. The V&V results of the process consist of test cases results and instantiated arguments.

Figure 3.4 shows the workflow specification diagram of V&V of machine learning-based systems using simulators.

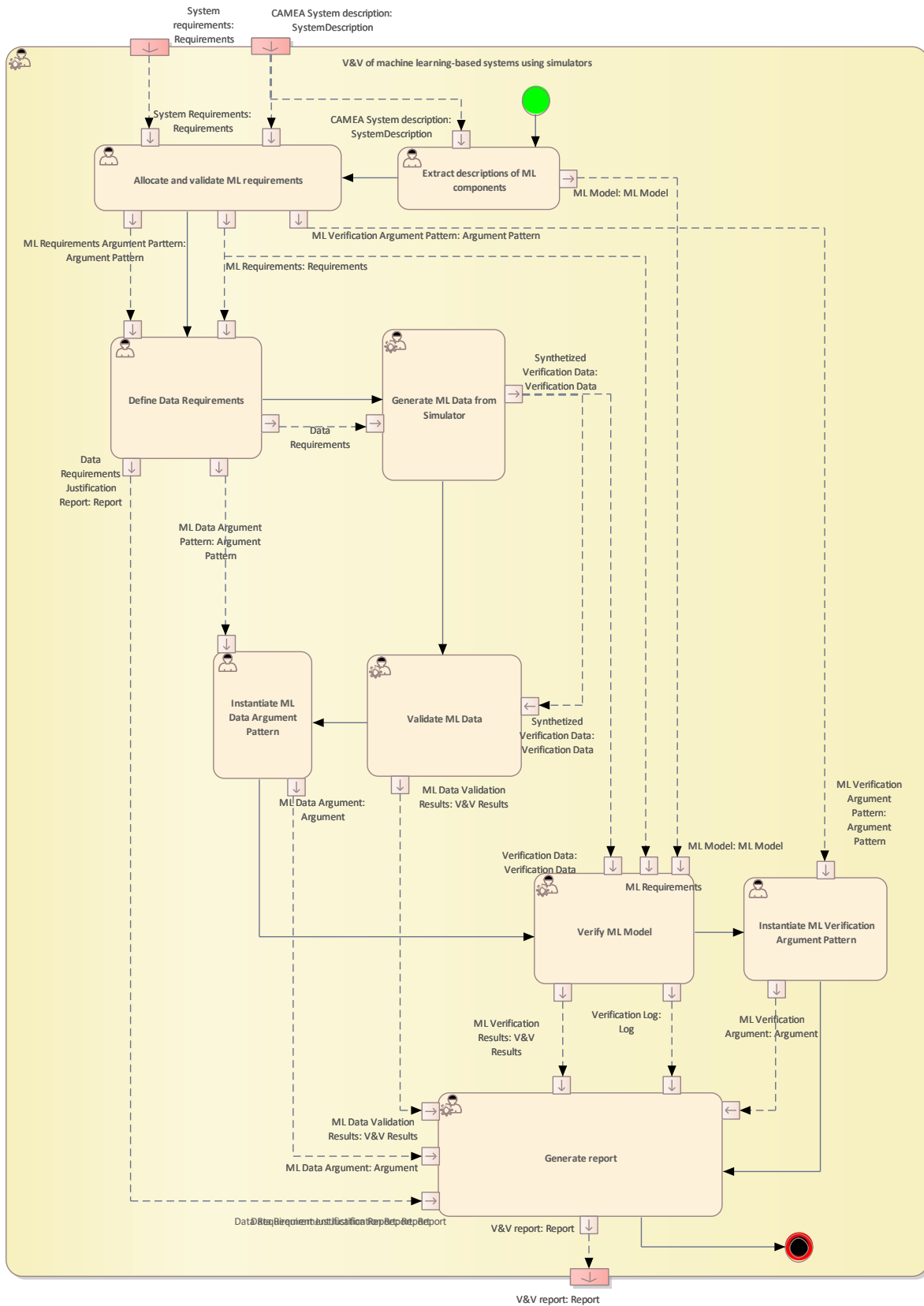


Figure 3.4 Workflow Definition diagram of V&V of machine learning-based systems using simulators - Workflow used in UC1\_CAMEA

Table 3.2 lists the activities of the workflow V&V of machine learning-based systems using simulators.

Table 3.2 List of activities performed by V&V of machine learning-based systems using simulators

Name	Type	Description
Allocate and validate ML requirements	Manual	(a) Allocate ML Requirements from System Requirements; (b) Validate the ML requirements against the system architecture and operational environment (such as sensor mounting positions, operational weather conditions...); (c) Formulate Argument Patterns with GSN structure, where the top claim is that the allocated ML requirements are satisfied in the defined environment.
Define Data Requirements	Manual	Develop data requirements, specify the required characteristics of synthesized data must have to ensure that the ML model meets the allocated requirements. These characteristics include relevance, completeness, accuracy and balance of data. The data requirements shall also include made assumptions regarding system operation environments.
Extract descriptions of ML components	Manual	Extract description, roles, and scope of ML components within the system and related interfaces.
Generate ML Data from Simulator	Semi-automated	<p>This activity takes as input the data requirements and uses the simulator to synthesize datasets meeting these requirements. Generated data includes separate datasets: Test data and Verification data.</p> <p>The simulator (Berge simulator) will be developed/configured against the system description and the operational environment, which include the following:</p> <ul style="list-style-type: none"> <li>- 3D scene description and required parameterization (e.g. number of lanes/direction etc.)</li> <li>- Sensor specification and parameterized mounting positions (to ensure that the simulator outputs accurately simulate sensor responses as in real world settings)</li> <li>- Parameterized lighting and weather conditions</li> <li>- Ability to run pre-defined scenarios from generated scenario scripts.</li> </ul> <p>Scenario Generator will generate traffic scenario scripts containing vehicle trajectories complying with data requirements. The generated trajectories will be used as input to the simulator to generate the datasets.</p>
Generate report	Manual	This activity consolidates different output artifacts of previous activities into a V&V report



Name	Type	Description
Instantiate ML Data Argument Pattern	Manual	This activity takes as input the ML Data Argument Pattern and other data related artifacts to create ML Data Argument
Instantiate ML Verification Argument Pattern	Manual	This activity creates ML Verification Argument from the Verification Argument Pattern and previously generated artifacts.
Validate ML Data	Semi-automated	The ML data validation activity checks that the generated data sets are sufficient to meet the ML data requirements. The results of the data validation activity will be explicitly documented. Data validation considers the relevance, completeness, and balance of the data sets. Discrepancies identified between the data generated and the ML data requirement will be justified. These justifications will be captured as part of the data validation results.
Verify ML Model	Semi-automated	This activity takes as input the ML requirements, the Verification Data and the ML model. The verification may consist of two sub-activities: test-based verification and formal verification. For each ML requirement, at least one activity shall be undertaken. Verification results for each requirement will be recorded in the ML verification results. Verification log with document the verification measures to ensure that the data used in verification was not exposed to the development team (independent from the development activities)

### 3.1.3 V&V Workflows of Model-Based Threat Analysis

Model-based threat analysis is a threat modelling approach that utilizes STRIDE model, which categorizes different types of threats and simplifies the overall security conversations. It serves as means to analyse systems for threats as well as failures, and consists of three major components:

1. A *system model* represents the system under consideration in its current status. This means that the approach can be applied during the design phase where assumptions about the future system are driving development, as well as during the implementation phase which reveals shortcomings of the planned system and therefore results in an adaption of the system. Moreover, model-based threat analysis can also be applied during the operational phase when the system is already running. A component may fail and, therefore, requires replacement. The system model is based on a data flow diagram. It holds all known security attributes of system components as well as the connections between them.
2. A *threat model* represents a digital twin of known threats. It is constituted of rules that allow for a later analysis of the system model. These rules are anti-patterns, which are basically system

configurations that are considered insecure and should therefore not be contained within the system under consideration.

3. A *threat analysis engine* enables an automated analysis of the system. It compares each rule with the system model to detect potentially insecure configurations and consequently threats the system under consideration may be affected by.

The whole threat modelling process results in a catalogue depicting threats that the system suffers from and, consequently, require treatment. The current rule sets were derived from UNECE WP29 [9], ETSI and the ITU. The tool used is ThreatGet [10]. The described approach is an iterative process which allows for consecutive analysis of the system with applied security measures that serve as threat mitigations.

Figure 3.5 shows the workflow specification diagram of Model-Based Threat Analysis.

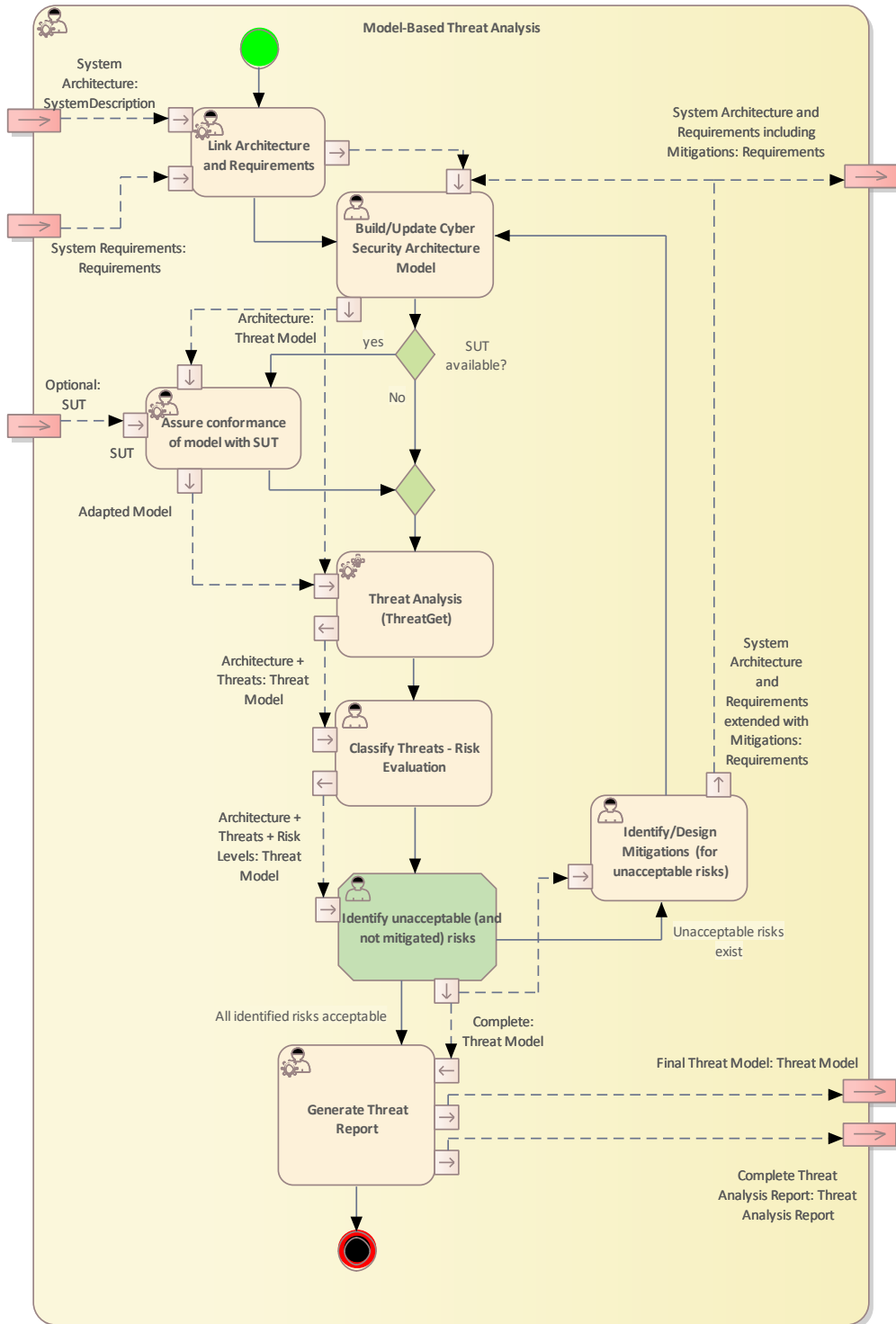


Figure 3.5 Workflow Definition diagram of Model-Based Threat Analysis - Workflow used in UC1\_CAMEA

Table 3.3 lists the activities of the workflow Model-Based Threat Analysis.

Table 3.3 List of activities performed by Model-Based Threat Analysis

Name	Type	Description
Assure conformance of model with SUT	Semi-automated	If the SUT exists already, the architecture is checked against it for conformance.
Build/Update Cyber Security Architecture Model	Manual	For the target system (SUT), an architecture model is developed containing relevant elements and considering security requirements. In iterations, it is adapted according to mitigation proposals. (For ThreatGet, a proper editor exists.)
Classify Threats - Risk Evaluation	Manual	Identified threats are evaluated with respect to their criticality. Mitigation means of previous iterations are considered.
Generate Threat Report	Semi-automated	Resulting test report is produced.
Identify unacceptable (and not mitigated) risks	Manual	If unacceptable risks were identified, they must be treated.
Identify/Design Mitigations (for unacceptable risks)	Manual	For unacceptable risks, mitigation means are elaborated.
Threat Analysis (ThreatGet)	Automated	The architecture model is examined with respect to vulnerabilities against (cyber) attacks, using e.g., threat models.

### 3.1.4 V&V Workflows of Assessment of Implementation of Network Communication

Assessment of implementation of network communication consists of two analyses: static code and dynamic analysis. At first, the code which deals with communication within the given system (e.g., connection of the camera to the cloud) must be implemented according to feature requests or bug reports. The code is then inspected in static code analysis.

Static code analysis uses either general analysers which are available in well-known static analysis frameworks (for instance, but not limited to, Infer or Frama-C). The targets for the analyses are general software quality issues like memory related bugs, synchronization bugs, or general software weaknesses. Static code analysis can also incorporate purpose-specific analysers which focus on, e.g., performance or cyber-security related problems. The results from static code analyses can be used not just by developers to fix the code but it can sometimes be used to locate possible weakness which should be further inspected by dynamic analysis during runtime. The execution of static analysis is fully automated, but the results must be processed manually.

Dynamic analysis of a design and an implementation of communication of the system is based on simulated fault-injection of a network link. The method requires one to clearly specify the communication nodes, communication parts, and prioritize which parts of communication are sensitive on communication link reliability, stability, and speed. The method incorporates a tool which can

automatically introduce faults on selected network flows which simulate connection loss, connection delays, or man-in-the-middle attacks. The activity ends with generation of the overall report of the assessment.

Figure 3.6 shows the workflow specification diagram of Assessment of implementation of network communication.

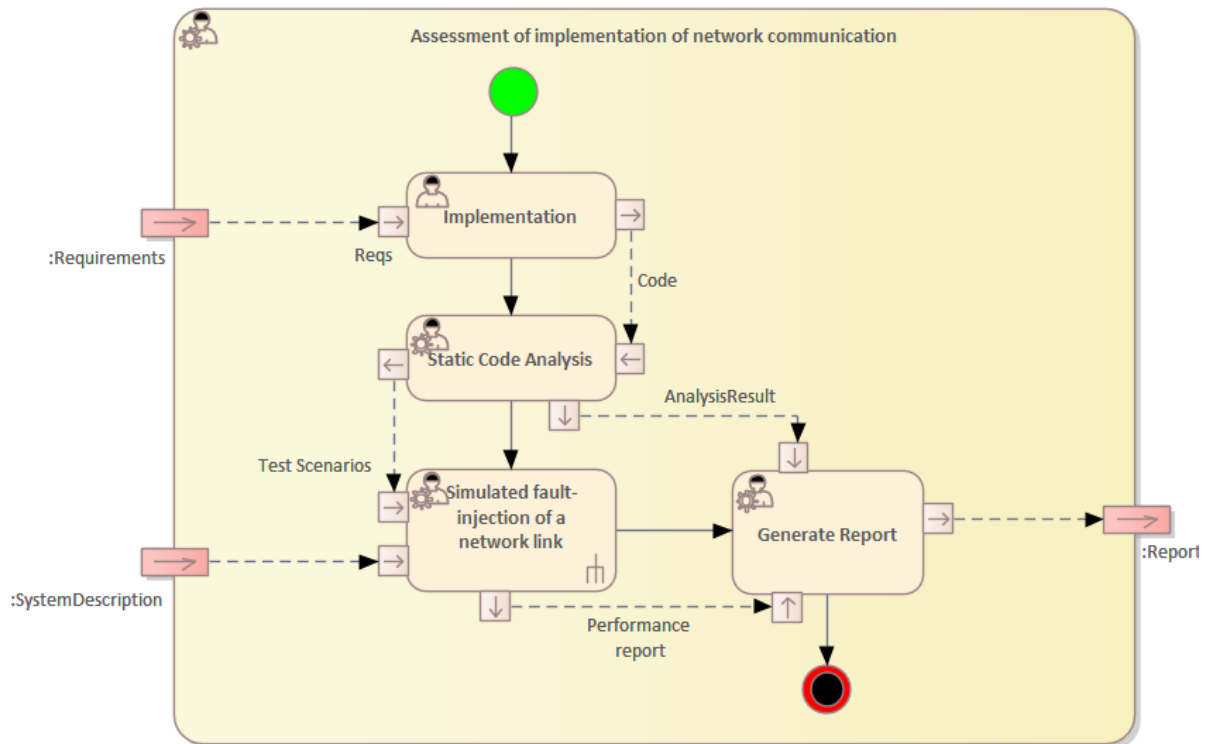


Figure 3.6 Workflow Definition diagram of Assessment of implementation of network communication used in UC1\_CAMEA

Table 3.4 lists the activities of the workflow Assessment of implementation of network communication.

Table 3.4 List of activities performed by Assessment of implementation of network communication

Name	Type	Description
Simulated fault-injection of a network link (CallBehavior, see Section 3.2.3)	Semi-automated	Implementation of the system under test. The implementation is based on the feature requests (e.g., requirements), or bug reports if a new version of the system is required.
Generate Report	Semi-automated	Merging of the reports from static and dynamic analyses.
Implementation	Manual	Implementation of the system under test. The implementation is based on the feature requests (e.g., requirements), or bug reports if a new version of the system is required.



Name	Type	Description
Static Code Analysis	Manual	Static code analysis incorporates either general or purpose-specific analyses. The analysis itself should be done fully automatically.

## 3.2 V&V Workflow of Use Case 2 ROBO

UC2\_ROBO package contains the following workflow(s):

- UC2 ROBO V&V Workflow
- Simulated fault-injection of a network link
- Simulation-Based Fault and Attack Injection at System-level Improved
- UC2 Daily regression test

Figure 3.7 shows the ROBO V&V Method Definition diagram type of the V&V workflow UC2\_ROBO.

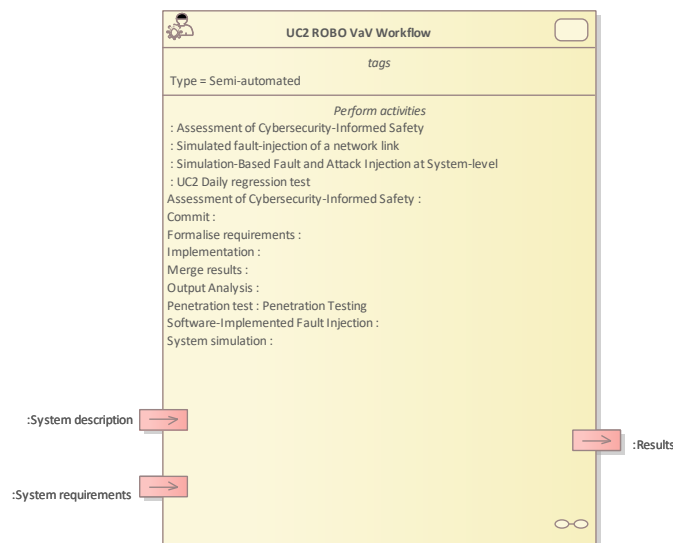


Figure 3.7 Method Definition of ROBO V&V defined for UC2\_ROBO

Figure 3.8 shows the Simulated fault-injection of a network link Method Definition diagram type of the V&V workflow UC2\_ROBO.

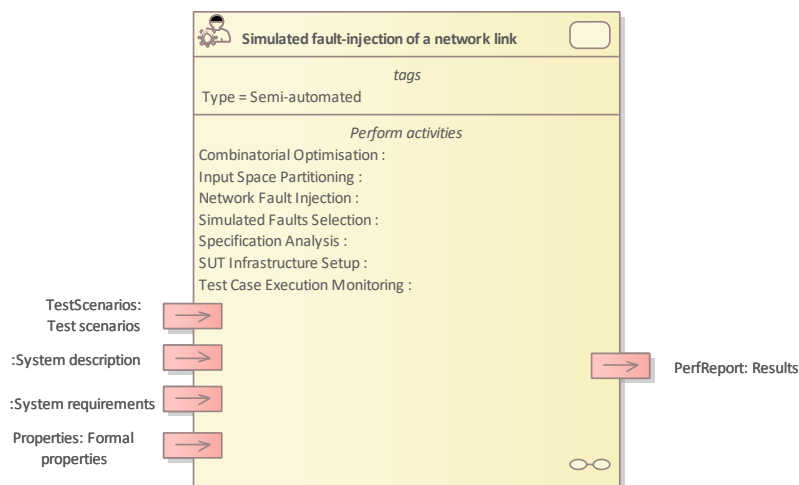


Figure 3.8 Method Definition of Simulated fault-injection of a network link defined for UC2\_ROBO

Figure 3.9 shows the ComFASE\_RISE\_VTI\_Improved Method Definition diagram type of the V&V workflow UC2\_ROBO.

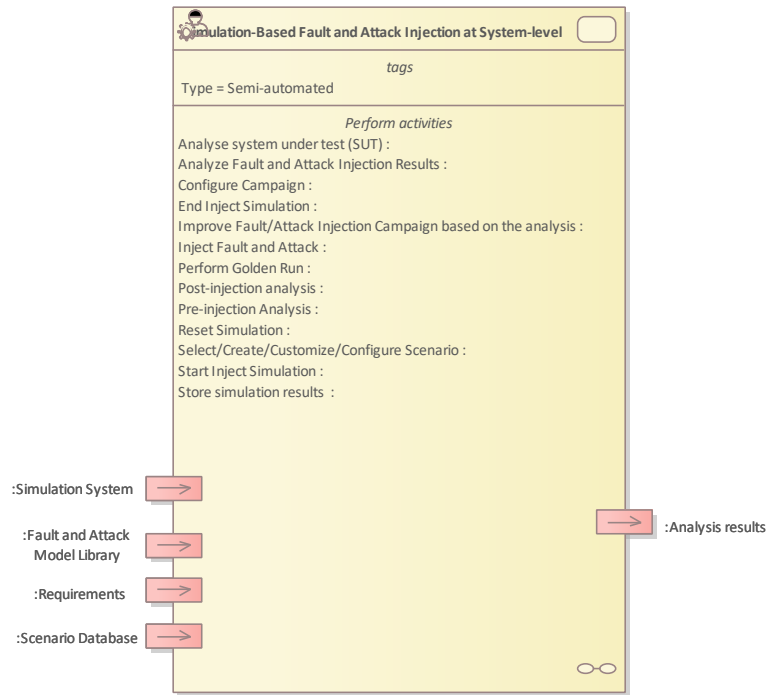


Figure 3.9 Method Definition of ComFASE\_RISE\_VTI\_Improve defined for UC2\_ROBO

Details on the workflow(s) are given in the following subsections.

### 3.2.1 Artifacts used in UC2\_ROBO

Table 3.5 lists the artifacts used for the workflows defined for UC2 by ROBO.

Table 3.5 List of artifact types used in UC2\_ROBO

Name	Description
Analysis results (Information)	This is the database where the <b>analysis results</b> are stored. The analysis results are based on the data logged in the <b>output database</b> .
Fault and attack injected System model (Active Unit)	<b>Fault and Attack injected system models</b> are the representation of the impact that can be caused due to the actual faults and attacks injected in the real environment of the system under test.
Fault and Attack Model Library (Active Unit)	The <b>fault and attack model library</b> is another input to the execution flow. The library stores all the faults and attack models the user could select and inject into the target system.
Formal properties (Information)	Formalized properties of a system (e.g., functional requirements).



Name	Description
Information Gathering (Information)	Information Gathering Description
Inject Fault and Attack (Active Unit)	This is the activity in which the selected faults and attacks are injected into the system under test
List of issues (Information)	List of issues from prevision iteration
List of methods (Information)	List of methods used in V&V process.
Output Database (Information)	This is the <b>output database</b> where all the results of the test campaign are stored where is later used for analysis.
Passed/Failed tests (Information)	Results of test cases marked as passed or failed.
Post Exploitation (Information)	Post exploitation description
Report (Information)	Providing a detailed report of strategies to improve your security
Requirements (Information)	<b>Requirements</b> is yet another input to the execution flow allowing the user to configure the tests and analyze the results.
Reset Simulation (Information)	This artifact represents resting the simulation in case of any errors such as, if the simulation cashes.
Results (Information)	Results from all used methods.
Scenario Database (Active Unit)	The <b>scenario database</b> has a set of scenarios that are input to the ComFASE execution flow. Each scenario defines the road attributes, vehicle maneuvers and their interactions.
Simulation System (Active Unit)	A <b>simulation system</b> is the input to the execution flow. This is the system under test where the fault and attacks are injected to analyze the behavior of the system to test the cybersecurity and safety attributes of the system.
Start Inject Simulation (Active Unit)	This activity represents the start of fault and attack campaign.
Store Simulation Result (Active Unit)	A <b>simulation system</b> is the input to the execution flow. This is the system under test where the fault and attacks are injected to analyze the behavior of the system to test the cybersecurity and safety attributes of the system.
System description (Information)	Description of the verified and validated system. The description may include diagrams, architecture, and/or its behavior.
System requirements (Information)	High-level requirements of the developed or improved system.
Test scenarios (Information)	High-level description of test scenarios and test cases.
Vulnerability Analysis (Information)	Vulnerability description

### 3.2.2 V&V Workflows of UC2 ROBO V&V Workflow

ROBO V&V process starts with logical system description and set of requirements. In case of the transmission line reliability this might consist of the communication protocols and error handling procedures. System requirements describe constraints that needs to be validated, e.g., state of the system after connection timeout.

In the first phase of a V&V process, simulation of the system is performed, and potential critical faults are identified with methods from our partners (Sim-based Fault and Attack Injection at System Level and Assessment of Cybersecurity-informed safety). If simulation results pass all requirements, then changes of the system are implemented and validated in the daily regression tests.

After passing regression tests, the system is tested with model-based methods from our partners (Software implemented fault injection and penetration tests). For these methods exists testing setup of a whole system. Results of all the methods are passed as output of the V&V process and used to alter system description.

Figure 3.10 shows the workflow specification diagram of UC2 ROBO V&V Workflow.

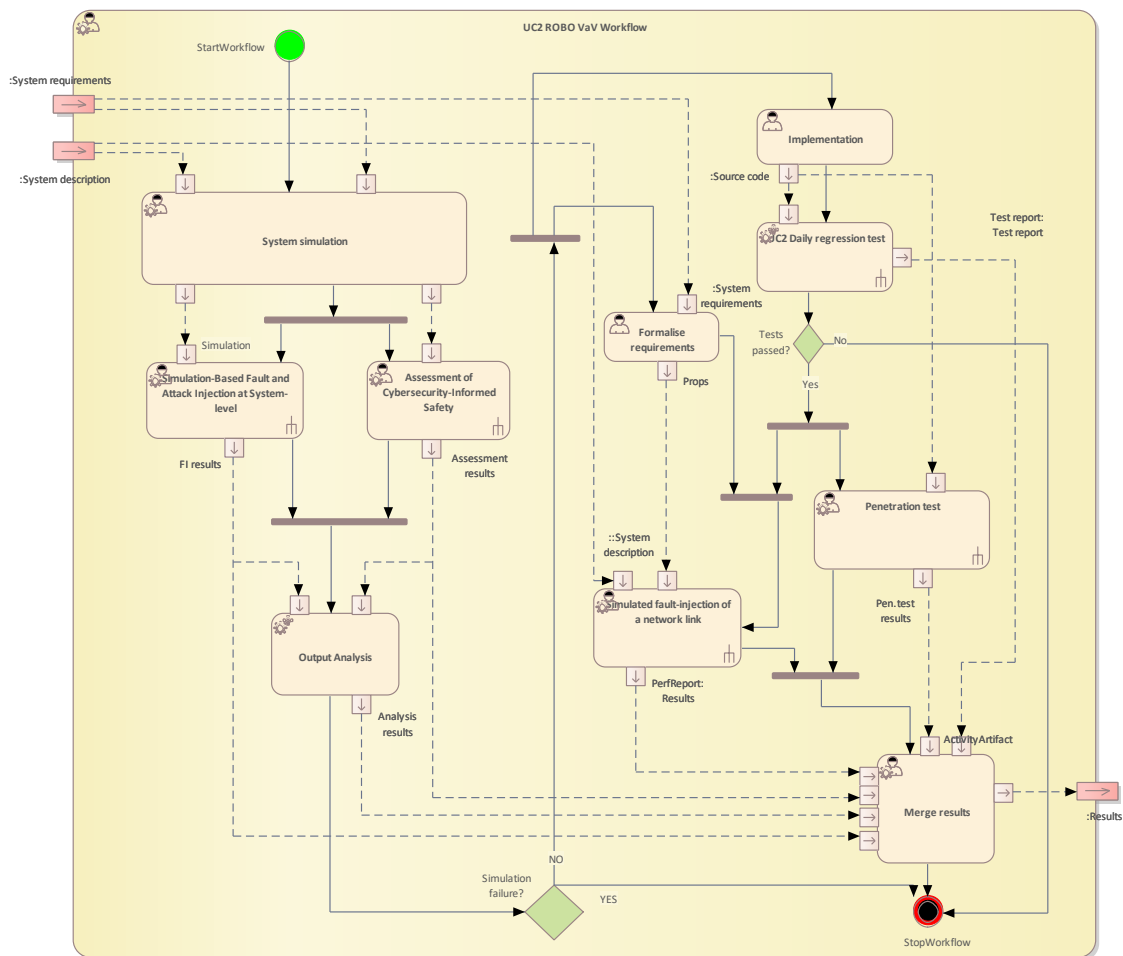


Figure 3.10 Workflow Definition diagram of UC2\_ROBO Workflow used in UC2\_ROBO

Table 3.6 lists the activities of the workflow UC2 ROBO V&V Workflow.

Table 3.6 List of activities performed by UC2 ROBO V&V Workflow

Name	Type	Description
Activity: Simulated fault-injection of a network link (CallBehavior)	Semi-automated	Implemented system is dynamically analyzed aiming at network communication and overall performance. The method checks if a developed system fulfills safety and cybersecurity properties under different (faulty) network conditions. The method results with a report about the safety/cybersecurity performance of a system.  The method needs safety/cybersecurity requirements to be monitored during system execution. The requirements must be formalised in order to be run-time verified.
Activity: Simulation-Based Fault and Attack Injection at System-level Improved (CallBehavior)	Semi-automated	Simulation-Based Fault and Attack Injection at System-level
Activity: Assessment of Cybersecurity-Informed Safety (CallBehavior)	Semi-automated	Assessment of Cybersecurity-Informed Safety
Activity: UC2 Daily regression test (CallBehavior)	Automated	Continuous tests run either periodically or based on events of continuous development (e.g. a push of a new commit to the repository).
Formalise requirements	Manual	System requirements are formalised in order to be used for run-time verification of the developed system.
Implementation	Manual	Implementation (initial or incremental) of new features.
Merge results	Semi-automated	A simple activity that merges results from different analysis methods.
Output Analysis	Automated	Analysis of outputs from previous activities
Penetration test: Penetration Testing (CallBehavior)	Semi-automated	Penetration tests of a system. Note that penetration tests should take into account not just modified code but also dependent code, since the whole architecture or hidden behaviour may be affected by the code change.
System simulation	Semi-automated	

### 3.2.3 V&V Workflows of Simulated Fault-Injection of a Network Link

The activity starts with manual analysis of specification of the system under test (SUT). All the requirements are considered with respect to system description in order to select possible performance/security risks stemming from unexpected (temporary or permanent) network conditions.

These cases are described in a formal way from different points of view of which network conditions are needed to be examined and how they relate to each other. The number of all possible combinations of different network parameters will possibly be high so an optimised plan is required.

The optimisation of the fault injection plan (i.e., the plan which different faults aka. network conditions are needed to be examined) is performed by combinatorial testing techniques (e.g., pair-wise testing). Combinatorial optimisation is fully automated.

The optimised fault-injection plan needs to be performed. The next set of faults are selected (Simulated Faults Selection) and configured (Network Fault Injection) in simulated network link(s), The test case is setup (SUT Infrastructure Setup) with re-linking the original network link with simulated (fault-injected) network link and the test case is executed while the SUT performance is automatically monitored. If all the faults and their combinations are examined, the aggregated performance report is being transferred as a result of the method.

Figure 3.11 shows the workflow specification diagram of Simulated fault-injection of a network link.

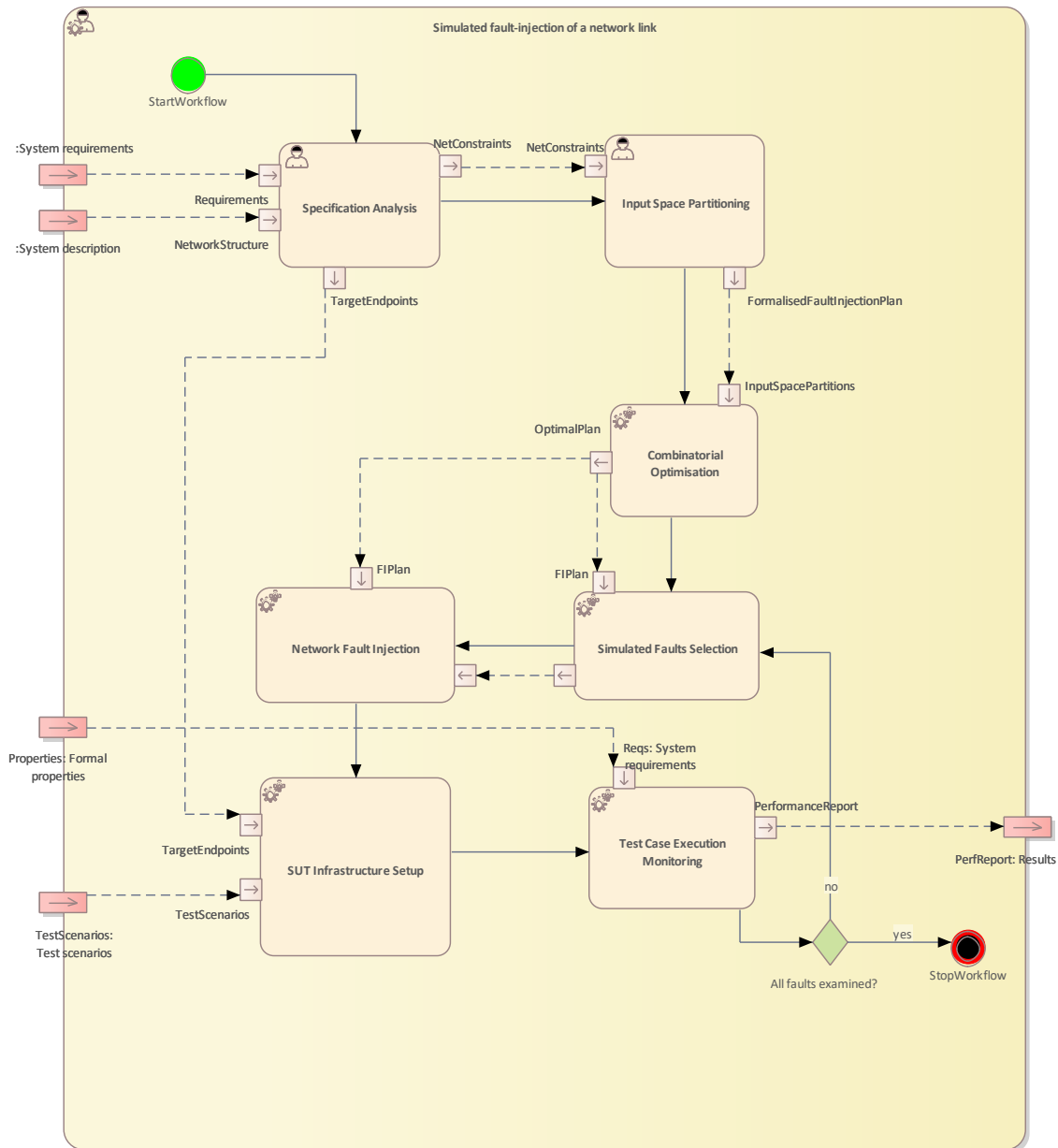


Figure 3.11 Workflow Definition diagram of Simulated fault-injection of a network link used in UC2\_ROBO

Table 3.7 lists the activities of the workflow Simulated fault-injection of a network link.

Table 3.7 List of activities performed by Simulated fault-injection of a network link

Name	Type	Description
Combinatorial Optimisation	Automated	Optimisation using combinatorial testing method will be used (e.g., pair-wise coverage) to provide reduced number of test conditions. Optimisation can be done automatically using Combine tool (developed by BUT).

Name	Type	Description
Input Space Partitioning	Manual	Network conditions are categorised, partitioned, and formalised as constraints for combinatorial optimisation.
Network Fault Injection	Automated	Selected faults are injected to the network infrastructure. Faults are generated dynamically during system under test (SUT) execution. The tool netloiter developed in BUT can be used as automatic network fault injector.
SUT Infrastructure Setup	Automated	SUT infrastructure is setup. This contains main and supplemental nodes (mocks or stubs can be used), network connection between nodes together with network fault injector.
Simulated Faults Selection	Automated	Network communication faults are selected for a single test in order to systematically cover all network faults.
Specification Analysis	Manual	Requirements are considered with respect to system description in order to select possible performance/security risks stemming from unexpected (temporary or permanent) network conditions.
Test Case Execution Monitoring	Automated	Execution of test case is monitored to provide the test passed/failed feedback.

### 3.2.4 V&V Workflows of Simulation-Based Fault and Attack Injection at System-level Improved

The workflow describes the application of the V&V method for Simulation-based fault and attack injection. This method allows fault- and attack-injection mechanisms to evaluate the system's cybersecurity and safety properties e.g., by using simulator control commands during target system simulations.

The workflow has four inputs: scenario database, test requirements, fault and attack model library and a target simulation system. These inputs are described below.

The **Scenario database** has a set of scenarios that are inputs to the ComFASE execution flow. Each scenario defines e.g., a network of roads, vehicle manoeuvres and their interactions. From the scenario database, a specific scenario can be selected based on the **test requirements**.

An intrinsic part of this simulation method is its **fault and attack model library**, which stores all the faults and attacks that could be modelled by the fault and attack injector. The user could then select a model from the library and inject that into the target system.

The list of **test requirements** is another input to the execution flow allowing the user to configure the tests and analyse the results. The requirements come from the test department, the stakeholders, or the customers.

The analyse **system under test activity** determines the fault and attack space structure and the details of the system under test, such as the core components of the system, working methodology, protocols used, and vulnerabilities if any. This information about the system could be used to perform pre-injection analysis to reduce the fault and attack space. Following are the activities within the method (i.e., Simulation-Based Fault and Attack Injection at System-level Improved).

After analysing the system, the scenario is selected and customized to perform the test campaigns.

After customizing a scenario, the **golden run** can be performed. The data logged during the golden run may serve as a reference for analyses of the injections. As part of the activities performed within a golden run, detailed traffic data generated from the simulation system is logged in a database for offline analysis.

After executing the golden run activity, an **attack injection campaign** would need to be configured. This activity consists of setting parameters such as the attack's type, value, initiation time, and duration for the specific scenario selected. The result of this activity is a test campaign that consists of a set of attack injection experiments.

After the attack injection campaign is configured, the **attack injection experiments** are performed in the target system according to the detailed configurations set in the **test campaign**. In case of simulation crash during an experiment run, results are stored in a database for result analysis. The simulation is then reset, and the **test campaign** continues by conducting the next experiment planned. In case of no simulation crashes, the results of the experiments, including the traffic data logged during the simulation are stored in a database for result analysis. Once the entire test campaign is finished, the results stored in the database are analysed to evaluate the impact of attacks on the system.

Figure 3.12 shows the workflow specification diagram of Simulation-Based Fault and Attack Injection at System-level Improved.

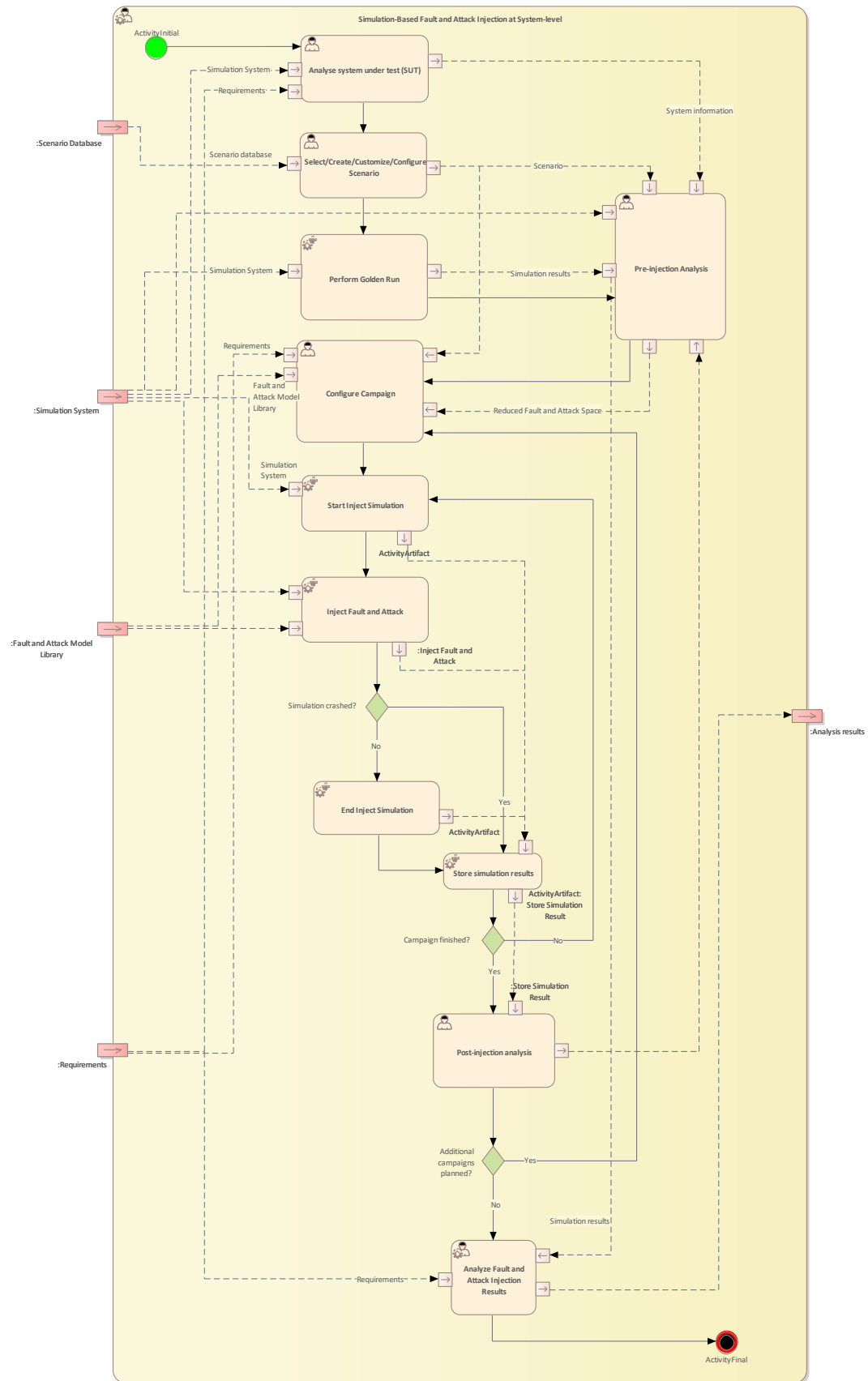


Figure 3.12 Workflow Definition diagram of Simulation-Based Fault and Attack Injection at System-level with additional fault and attack models valid for multiple inter-vehicle communication (IVC) layers used in UC2\_ROBO



Table 3.8 lists the activities of the workflow Simulation-Based Fault and Attack Injection at System-level Improved.

Table 3.8 List of activities performed by Simulation-Based Fault and Attack Injection at System-level Improved

Name	Type	Description
Analyse system under test (SUT)	Manual	The system under test is analyzed in this activity. The analysis covers the information that can be gathered about the system such as the component and sub-components of the system, working methodology and any vulnerabilities of the system that are known.
Post-injection analysis	Manual	<p>The pre- and post-injection analysis helps to understand the system and evaluate the system while it is under the influence of faults and attacks. Moreover, the analysis also helps to reduce the fault and attack space for the next campaigns.</p> <p>The <b>Post Injection Analysis</b> is mainly performed on the stored simulation results of the fault and attack injection campaign.</p> <p>The post analysis results are stored in the result analysis database.</p> <p>The result database has not only the post injection analysis results from the current from the current test campaigns, but it also contains the post injection analysis of the previous test campaigns which can be used to improve the upcoming campaigns to validate the requirements.</p> <p>The next test campaign can be optimized by analyzing the previous test campaign results.</p>
Analyse Fault and Attack Injection Results	Semi-automated	In the <b>Analyse Fault and Attack Injection Results</b> activity, the monitored data obtained during simulations of the golden run and faulty run is analysed based on the initial test requirements to report the fault and attack injection results of the campaign performed.
Configure Campaign	Manual	In the <b>Configure Campaign</b> activity, the test campaign is configured by setting parameter values such as fault and attack start time, end time, duration, and values.
End Inject Simulation	Automated	The <b>End Inject Simulation</b> activity represents the end of fault and attack injection campaign.
Inject Fault and Attack	Automated	<p>The <b>Inject Fault and Attack</b> activity represents the stage of the test campaign where the faults and attacks are injected in the system under test (SUT).</p> <p>This step activates after each experiment is finished until the whole test campaign is finished.</p> <p>In case of a simulation crash during the campaign run, the results are logged, and the simulation also resets.</p>

Name	Type	Description
Perform Golden Run	Automated	Golden run is an experiment which is free from any faults or attacks. This defined the ground truth of the system and can be used to analyze fault and attack injection results.
Pre-injection Analysis	Manual	<p>The <b>Pre injection analysis</b> is used to reduce the fault and attack space to reduce the number of experimentation efficiency without compromising the quality of the results. The system information gathered during the system analysis stage is initially used to perform pre-injection analysis. Later the results from the post injection analysis can improve the pre injection analysis for the next campaign.</p> <p>The inputs for the pre injection analysis are,</p> <ol style="list-style-type: none"> <li>1- The results from the analysis performed on the system under test.</li> <li>2- The scenario that is selected based on the test needs.</li> <li>3- Simulation results from the golden run.</li> <li>4- The analysis results output from the post injection analysis.</li> </ol> <p>The fault and attack injection test campaign can then be optimized based on the above points which could help to reduce the fault and attack space.</p>
Select/Create/Customize/Configure Scenario	Manual	Here the test scenario is selected after the system analysis. The selected scenario can be further enhanced and configured in this stage if needed.
Start Inject Simulation	Automated	The <b>Start Inject Simulation</b> activity represents the start of fault and attack injection campaign.
Store simulation results	Automated	This activity (i.e., <b>Store Simulation Results</b> ) represents the logging of the fault and attack injection test campaign. The simulation results log includes the simulation results together with the test setup such as when the faulty simulation run was initiated and ended.

### 3.2.5 V&V Workflows of UC2 Daily Regression Test

The workflow describes the daily regression test. Continuous tests run either periodically or based on events of continuous development (e.g., a push of a new commit to the repository).

Once a new commit has been made, the system is automatically built (a program for remote station and vehicle). Both programs are deployed and validated (with some sanity tests). The prescribed test scenarios are run and evaluated. If an error occurs, the issue will be automatically reported (either for a developer or to an issue tracker system - depends on where the tests are executed).

Figure 3.13 shows the workflow specification diagram of UC2 Daily regression test.

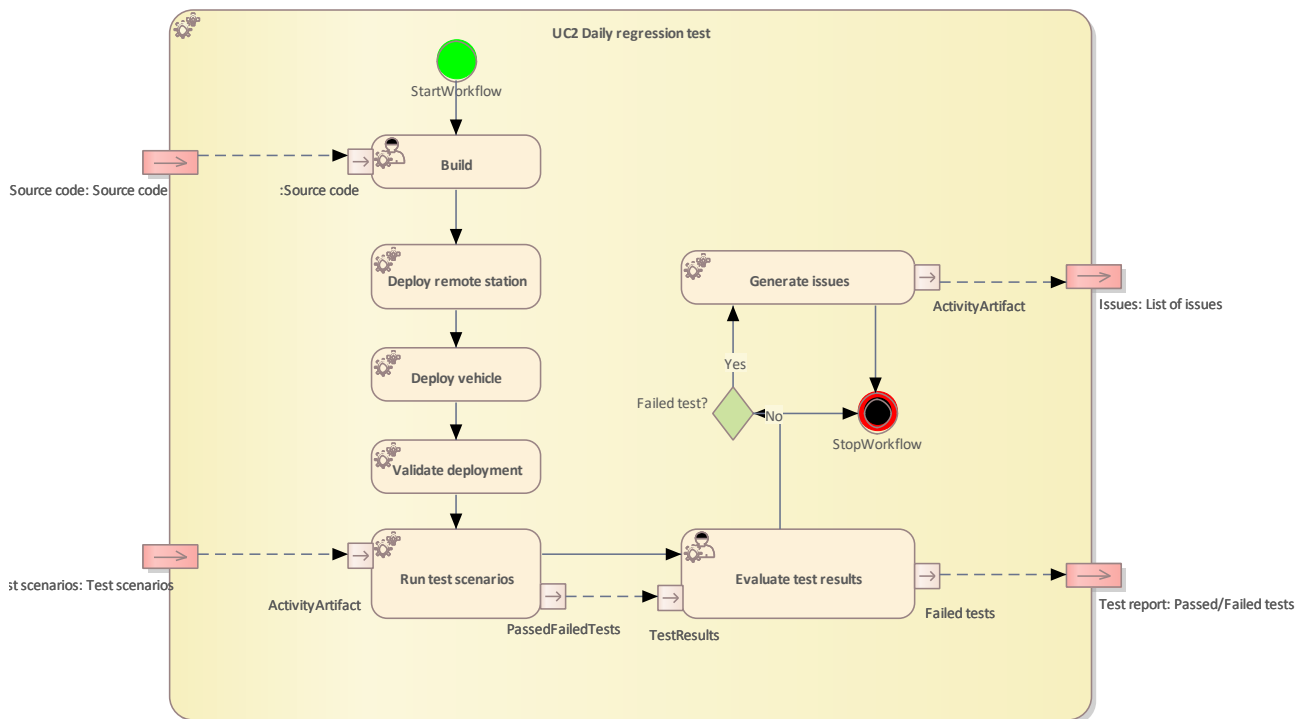


Figure 3.13 Workflow Definition diagram of Regression tests workflow used in UC2\_ROBO

Table 3.9 lists the activities of the workflow UC2 Daily regression test.

Table 3.9 List of activities performed by UC2 Daily regression test

Name	Type	Description
Build	Semi-automated	Building a system under test from a fresh commit.
Deploy remote station	Automated	Uploading a built remote station program.
Deploy vehicle	Automated	Uploading a new control system of a vehicle.
Evaluate test results	Semi-automated	Checking the results of tests.
Generate issues	Automated	Generation of issues from backtraces and logs.
Run test scenarios	Automated	Execution of test scenarios.
Validate deployment	Automated	Sanity tests whether deployment is correct.

### 3.3 V&V Workflow of Use Case 3 NXP

UC3\_NXP package contains the following workflow:

- Use Case 3 Radar systems for ADAS

Figure 3.14 shows the Use Case 3 Radar systems for ADAS Method Definition diagram type of the V&V workflow UC3\_NXP.

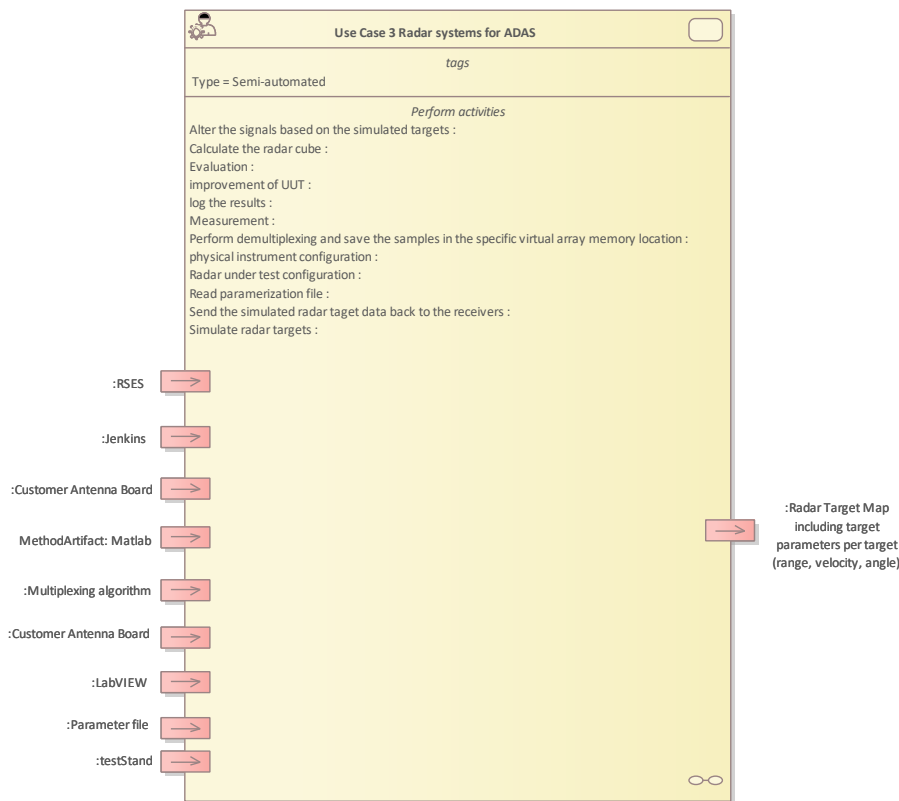


Figure 3.14 Method Definition of Use Case 3 defined for UC3\_NXP

Details on the workflows are given in the following subsections.

#### 3.3.1 Artifacts used in UC3\_NXP

Table 3.10 lists the artifacts used for the workflow(s) defined for UC3\_NXP.

Table 3.10 List of artifact types used in UC3\_NXP

Name	Description
CouchBase (Active Unit)	Couchbase Server is an open-source, distributed (shared-nothing architecture) multi-model NoSQL document-oriented database software package optimized for interactive applications. These applications may serve many concurrent users by creating, storing, retrieving, aggregating,

Name	Description
	manipulating and presenting data. In VALU3S it is used as smart test evaluation data base.
Customer Antenna Board (Information)	The customer antenna board is the radar system used for testing
data log (Information)	The data are logged in the database
Jenkins (Active Unit)	Jenkins is an open source automation server which enables developers to reliably build, test, and deploy software. Allows automated test execution and sandbox testing of newly deployed SoC software execution. This reduces test time and improves low cost error detection.
LabVIEW (Active Unit)	LabVIEW is a graphical programming system from National Instruments. The acronym stands for "Laboratory Virtual Instrumentation Engineering Workbench". The first version was released in 1986 for Macintosh computers. Today, the development environment is also available for Windows and Linux.
MATLAB (Active Unit)	MATLAB is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.
Multiplexing algorithm (Active Unit)	Multiplexing algorithm enables the method DDMA
Parameter file (Active Unit)	Describes the requirements for testing such as: target parameters, test cases as well as scenarios
Python (Active Unit)	Python is a general-purpose, commonly interpreted, higher-level programming language. It claims to promote an easy-to-read, concise programming style. For example, blocks are structured by indentations rather than curly braces.
Python (Active Unit)	Python is a general-purpose, commonly interpreted, higher-level programming language. It claims to promote an easy-to-read, concise programming style. For example, blocks are structured by indentations rather than curly braces.
Radar Target Map including target parameters per target (range, velocity, angle) (Information)	The final output of the radar measurements. The comparison between the expected and the real map indicates the quality of the unit under test.
RSES (Active Unit)	A target simulator for radar targets in real world driving scenarios.
System Test Box (Active Unit)	Is a test set up for laboratory
testStand (Active Unit)	TestStand is a test management software suite from National Instruments. TestStand is a software framework that adds

Name	Description
	value to test software developers. One of the most important features it provides is a consistent look and feel for test operators.
Unit under test (Active Unit)	The unit under test is the chip which' quality is measured.

### 3.3.2 V&V Workflows of Use Case 3 Radar Systems for ADAS

The "Radar system for ADAS" workflow tackles the need for new V&V methods and tools required due to the higher complexity of modern ADAS systems. Some challenges in the V&V process cannot be addressed with traditional methods.

One of the challenges is the necessity to include the verification and validation at system level, which should cover the design and production phase of the IC components. The system-level validation must include the interaction between the semiconductors and all the peripherals around the sensor to grant the safety and reliability of the final system.

Thus, the environmental peripherals are introduced in the designed method to test the chip on a system level and under simulated real-world conditions. To ensure reliable and valid testing, the chip-specific thresholds are defined in advance, and all relevant test setups are listed in the test manual. After the workflow is finished, the IC performance protocol highlights the IC performance.

The above-described activities lead to overall four artefact inputs and one output. Overall, the programmed test environment, a test manual, predefined test parameters, a radar multi-target simulator and the system test box are the requirements for the use case.

The input artefacts in the workflow defined for this use case are used to define test cases and design a test plan for consequent and reliable test procedures following the test manual. The test plan is then executed, and deviations from the expected thresholds are highlighted. Consequently, the chip is then debugged until the performance in all test cases is appropriate. The final results are then captured in the IC performance measurement protocol.

Figure 3.15 below shows the workflow specification diagram of Use Case 3 Radar systems for ADAS.

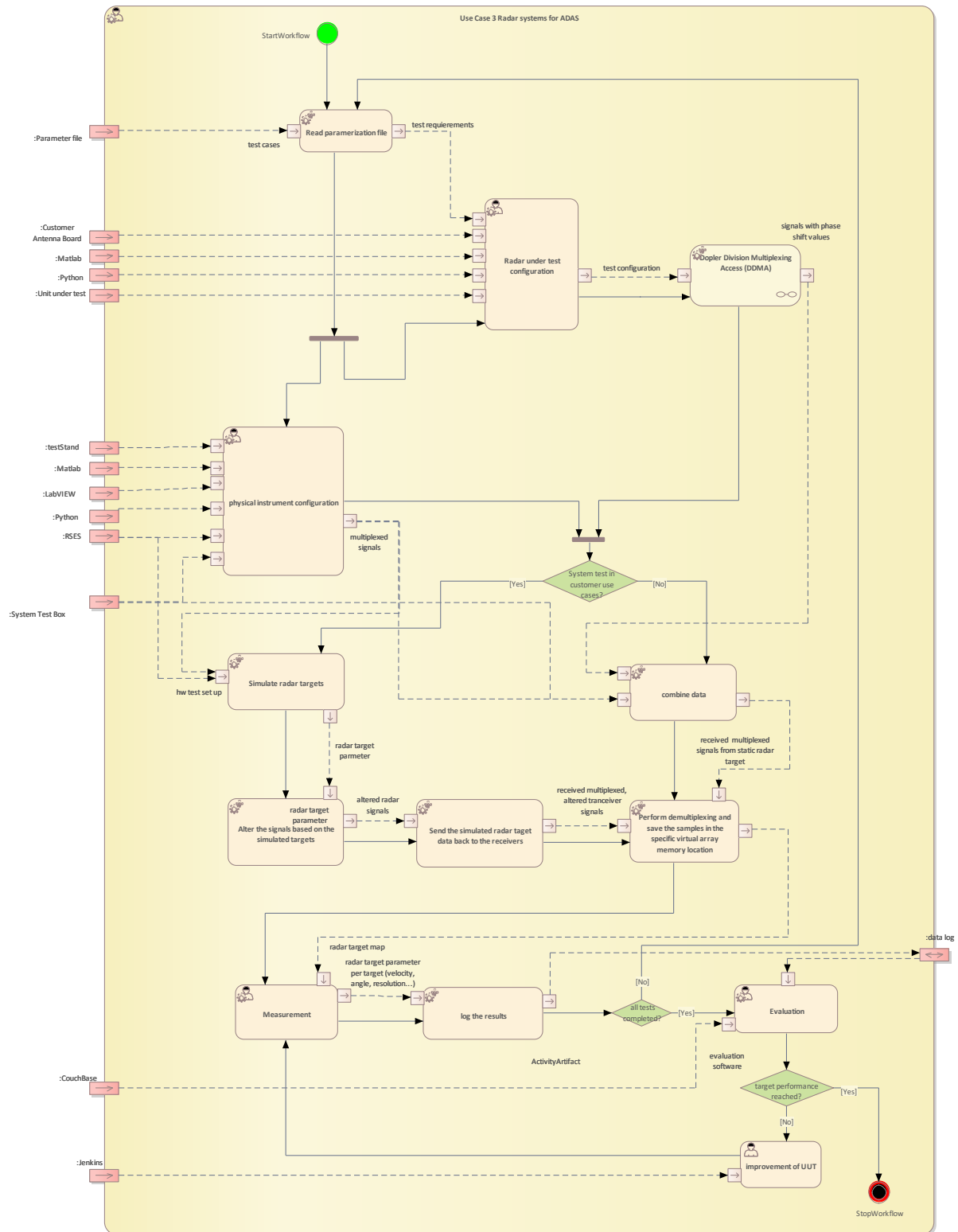


Figure 3.15 Workflow Definition diagram of Use Case 3 Radar systems for ADAS used in UC3\_NXP

Table 3.11 lists the activities of the workflow Use Case 3 Radar systems for ADAS.

Table 3.11 List of activities performed by Use Case 3 Radar systems for ADAS

Name	Type	Description
Alter the signals based on the simulated targets	Automated	The RSES receives the radar target parameter as well as the multiplexed transceiver signals. The signals are consequently altered in a way as they would appear in the real-world perception of radar targets (e.g. when driving in a city).
Calculate the radar cube	Automated	The radar cube is a 3D point cloud of the received signals which mirrors the perceived world
Evaluation	Semi-automated	The actual test results are compared against the target performance parameters of the radar chip.
Measurement	Semi-automated	The performance of the radar chip in the radar system is measured. Velocity, angle, resolution and fault detection are among the most important performance parameters. The target definition enables us to analyse radar imperfections in linearity behaviour, signal to noise ratio, signal to spurious ratio.
Perform demultiplexing and save the samples in the specific virtual array memory location	Automated	Demultiplexing enables the receiving chip to draw a radar target map in form of a point cloud based on multiple simultaneously received signals.
Radar under test configuration	Semi-automated	An UUT control enables the flawless execution of different test cases in a single set up. The control is composed on the HW set up (radar under test+ customer evaluation board) and a software set-up which is developed with MATLAB and Python. The device parameters are set using firmware commands.
Read parameterization file	Automated	Instructions regarding test cases and target parameters (velocity, angle, radar cross section.
Send the simulated radar target data back to the receivers	Automated	After simulating the radar targets, the simulated signals are sent back to the receivers.
Simulate radar targets		Different radar targets are simulated based on the tests' requirements. They do not only vary in the number of targets, but they can also be moving targets or moving ego.
improvement of UUT	Manual	Actions are derived from the identified performance gaps by the evaluation. The hardware and software of the tested radar (unit under test) get iteratively improved until the target parameters are reached. Jenkins as continuous integration tool supports the software improvement.
log the results	Automated	The tests results are documented and saved for later use. The generated data is logged in a database and can be reused.



Name		Type	Description
physical configuration	instrument	Semi-automated	In the improved workflow, two new HW set-ups are used to enable radar system testing. The RSES or the STB can be used for system tests but have different use cases. The STB measures the signals of one target in a laboratory environment. The RSES can be used to simulate multiple and moving targets as well as ego movement, which comes along with a new array of possible faults which traditionally could only be detected by system applicants such as OEMs. Both radar system test set-ups are controlled by a newly developed control software which is set up by a tool combination (Python, MATLAB, LabVIEW, TestStand).
Doppler Multiplexing Access (DDMA)	Division	Automated	DDMA helps with parallelization of different signals simultaneously, more closely described done in 3.3.3.

### 3.3.3 V&V Workflows of Doppler Division Multiplexing Access (DDMA)

Integrated radar circuits usually have multiple transmitting and receiving units serving multiple channels. Conventionally those are used sequentially to update the radar map. With using these methods, we can test the performance of these channels simultaneously. The phase of the transmitted signals is being changed by a phase shifter at the end of the transmitting chain to make the signals orthogonal (invisible) to each other. Hence, the signals of all transmitters can be separated by the receiver in the detection map. This saves customers time and improves the quality of the delivered chips because they can be already validated at system level early in the supply chain. Last but not least, quality is also improved because parasitic effects between channels are characterized that would otherwise be invisible if the channels were tested sequentially.

Figure 3.16 shows the workflow specification diagram of Doppler Division Multiplexing Access (DDMA).

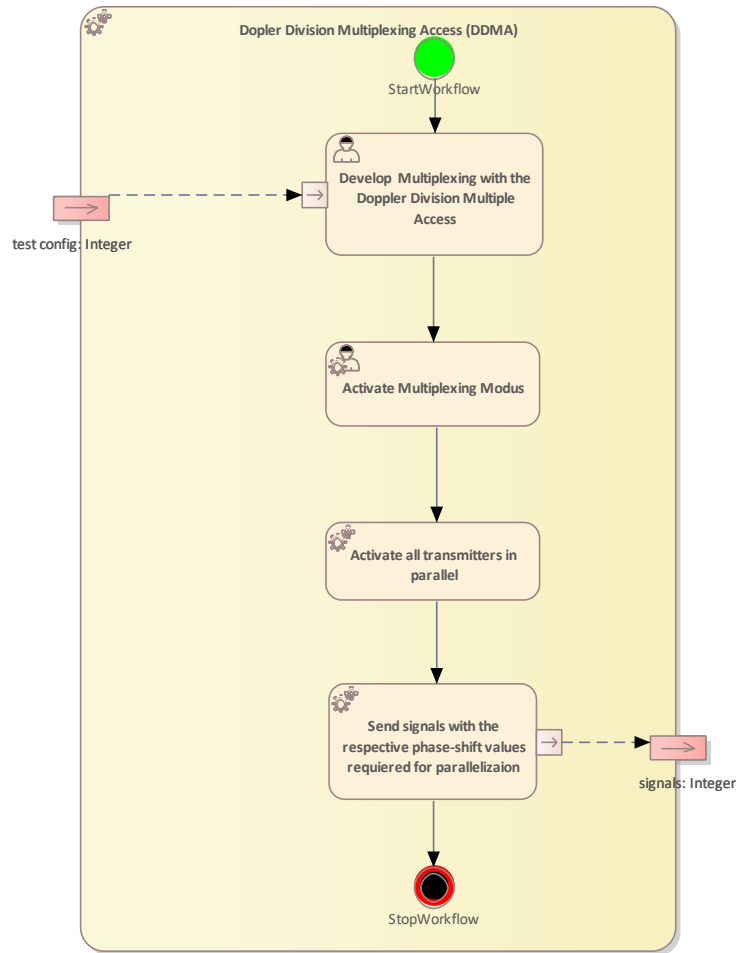


Figure 3.16 Workflow Definition diagram of Doppler Division Multiplexing Access (DDMA) used in UC3\_NXP

Table 3.12 lists the activities of the workflow Doppler Division Multiplexing Access (DDMA).

Table 3.12 List of activities performed by Doppler Division Multiplexing Access (DDMA)

Name	Type	Description
Activate Multiplexing Modus	Semi-automated	Multiplexing modus gets activated to enable the sending of signals in parallel
Activate all transmitters in parallel	Automated	The difference to time-based multiplexing is the parallelization of the sending and perception of signals simultaneously without interference. Consequently, all transmitters get activated in parallel here.
Develop Multiplexing with the Doppler Division Multiple Access	Manual	The phase of the transmitted signals is being changed by a phase shifter at the end of the transmitting chain to make the signals orthogonal (invisible) to each other.
Send signals with the respective phase-shift values required for parallelization	Automated	The altered signals are sent back and all transmitters can be separated by the receiver in the detection map.

### 3.4 V&V Workflow of Use Case 4 PUMACY

UC4\_PUMACY package contains the following workflows:

- UC4\_Combinded Virtual Validation and Failure Detection Diagnosis
- Failure Detection Diagnosis
- Virtual Validation

Figure 3.17 shows the Combined Virtual Validation and Failure Detection Diagnosis Method Definition diagram type of the V&V workflow UC4\_PUMACY.

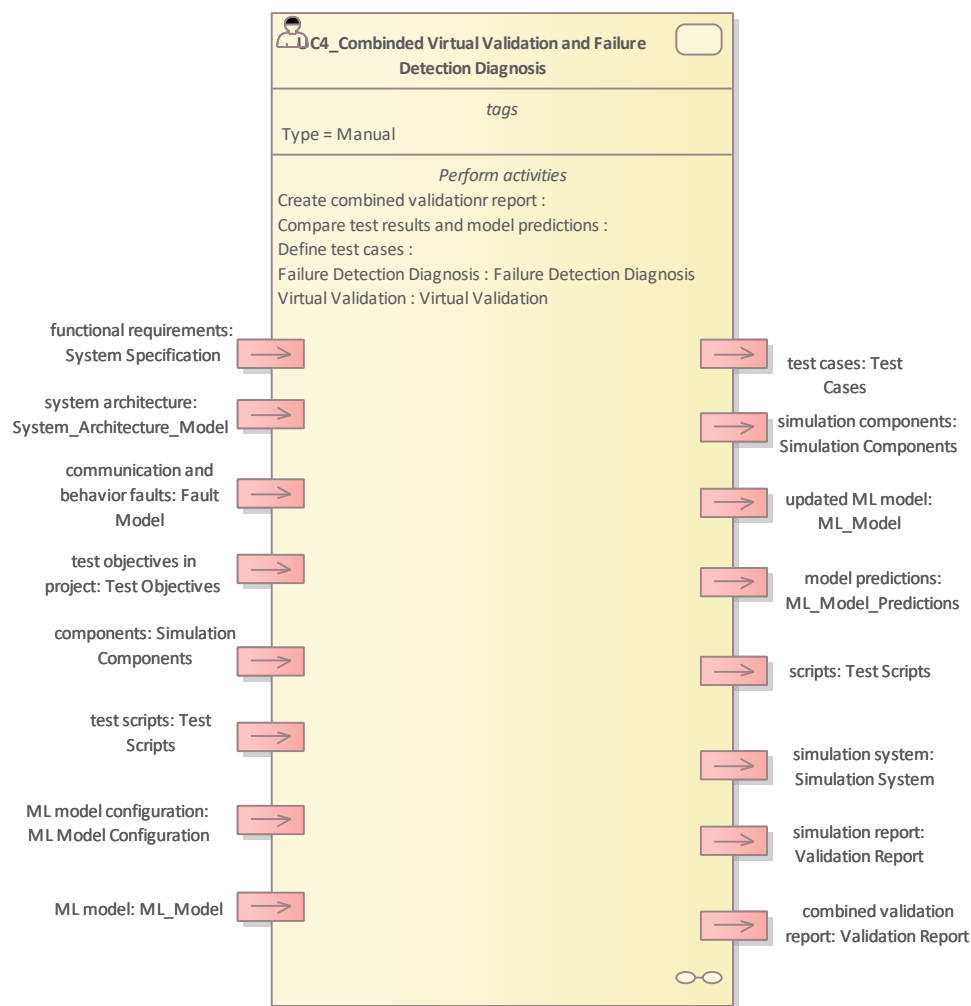


Figure 3.17 Method Definition of Combined Virtual Validation and Failure Detection Diagnosis defined for UC4\_PUMACY

Figure 3.18 shows the Failure Detection Diagnosis Method Definition diagram type of the V&V workflow UC4\_PUMACY.

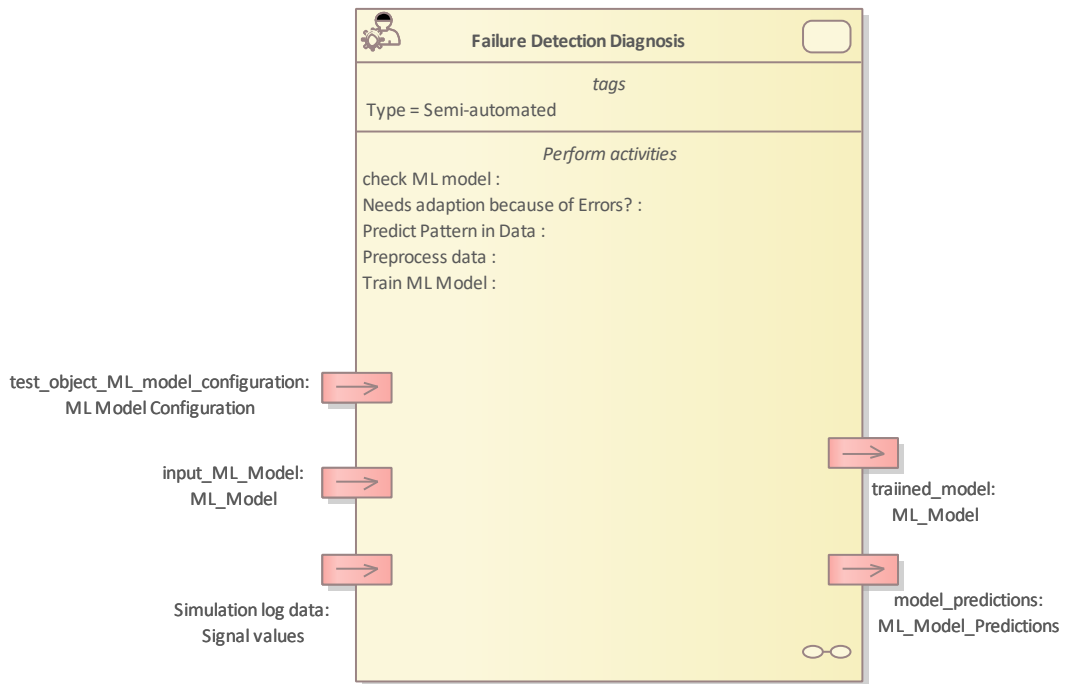


Figure 3.18 Method Definition of Failure Detection Diagnosis defined for UC4\_PUMACY

Figure 3.19 shows the Virtual Validation Method Definition diagram type of the V&V workflow UC4\_PUMACY.

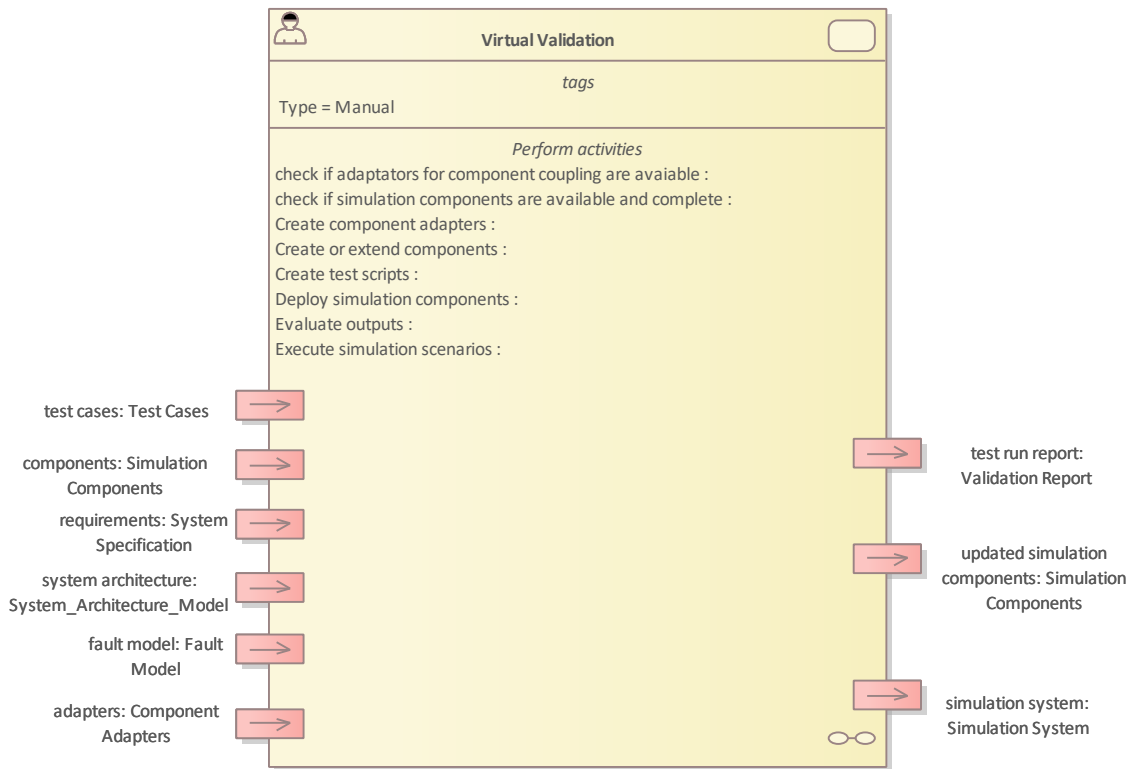


Figure 3.19 Method Definition of Virtual Validation defined for UC4\_PUMACY

Details on the workflows are given in the following subsections.

### 3.4.1 Artifacts used in UC4\_PUMACY

Table 3.13 lists the artifacts used for the workflows defined for UC4\_PUMACY\_NEW.

Table 3.13 List of artifact types used in UC4\_PUMACY

Name	Description
Component Adapters (Active Unit)	All simulation components are created.
Fault Model (Information)	If trained machine learning model is not accurate to predict the failures, it is considered as a Fault Model.
ML Model Configuration (Information)	This step checks if there is any new feature or sensor added into the dataset. With the addition new sensor variable in existing dataset, machine learning model must be configured and retrained.
ML_Model (Information)	After pre-processing, raw data is converted into a suitable format which fed into machine learning model for training. A deep learning model LSTM (Long Short-Term Memory) is used to predict the anomalies in an assembly process. An LSTM network is well-suited to learn from experience to classify, process and predict time series events when there are very long-time lags of unknown size between important events.
ML_Model_Predictions (Information)	This is the output of trained ML model which has been deployed for predictions. The results from trained ML model is used for the evaluation of Virtual Failure Detection & Diagnosis.
Simulation Components (Active Unit)	Check if simulation components are available and complete.
Simulation System (Active Unit)	Execution of simulation scenario.
System Specification (Information)	List of system specification is created and defined here.
System_Architecture_Model (Information)	System architecture is defined here.
Test Cases (Information)	List of test cases are defined here.
Test Objectives (Information)	List of test objectives are defined here.
Test Scripts (Active Unit)	Create and run python-based test scripts for fault injection
Validation Report (Information)	After fault injection, verification and validation report is generated for fault diagnosis.

### 3.4.2 V&V Workflows of Combined Virtual Validation and Failure Detection Diagnosis

The combined method of Data Analytics/ML and Virtual Validation is designed and developed to detect failures in the Simulation. ML-Pipeline will be used as enhancement / improvement of the “Failure Detection and Diagnosis (FDD)” Method that is applied in UC4 as part of the tool chain.

Figure 3.20 shows the workflow specification diagram of Combined Virtual Validation and Failure Detection Diagnosis.

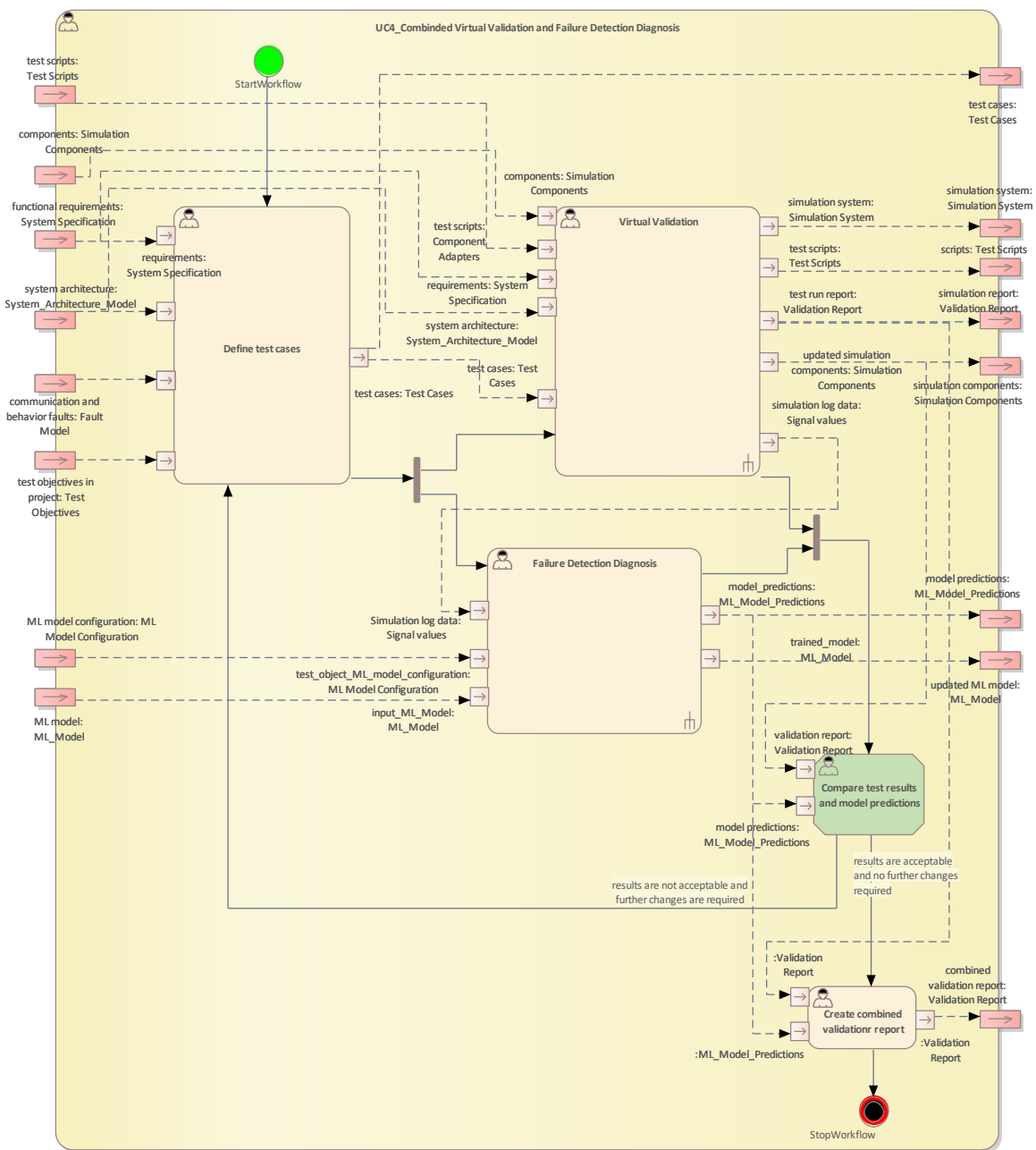


Figure 3.20 Workflow Definition diagram of Combined Virtual Validation and Failure Detection Diagnosis used in UC4\_PUMACY

Table 3.14 below lists the activities of the workflow Combined Virtual Validation and Failure Detection Diagnosis.

Table 3.14 List of activities performed by UC4\_Combined Virtual Validation and Failure Detection Diagnosis

Name	Type	Description
Compare test results and model predictions	Manual	Results from the sub-methods (virtual validation and failure detection diagnosis) are compared and used to take decisions on possible changes of test cases and test artifacts in subsequent test runs.
Create combined validation report	Manual	Validation report is created based on the results from the evaluation and comparison of both sub-methods (virtual validation and failure detection diagnosis)
Define test cases	Manual	Test cases are designed based on the input information from the development process (requirements, architecture, fault model)
Failure Detection Diagnosis: Failure Detection Diagnosis (CallBehavior )	Manual	The data analytics/ML tool mainly focuses on predicting and detecting the occurrence of failures in advanced in an assembly process. The raw data will be collected from CIROS simulation and failure detection model will be trained and deploy for failure prediction & diagnosis.
Virtual Validation: Virtual Validation (CallBehavior )	Manual	A virtual simulation environment and dedicated test cases are constructed and used to run and evaluate validation scenarios.

### 3.4.3 V&V Workflows of Failure Detection Diagnosis

The data analytics/ML tool mainly focuses on predicting and detecting the occurrence of failures in advance in an assembly process. The raw data will be collected from CIROS simulation and failure detection model will be trained and deployed for failure prediction & diagnosis.

Figure 3.21 shows the workflow specification diagram of Failure Detection Diagnosis.

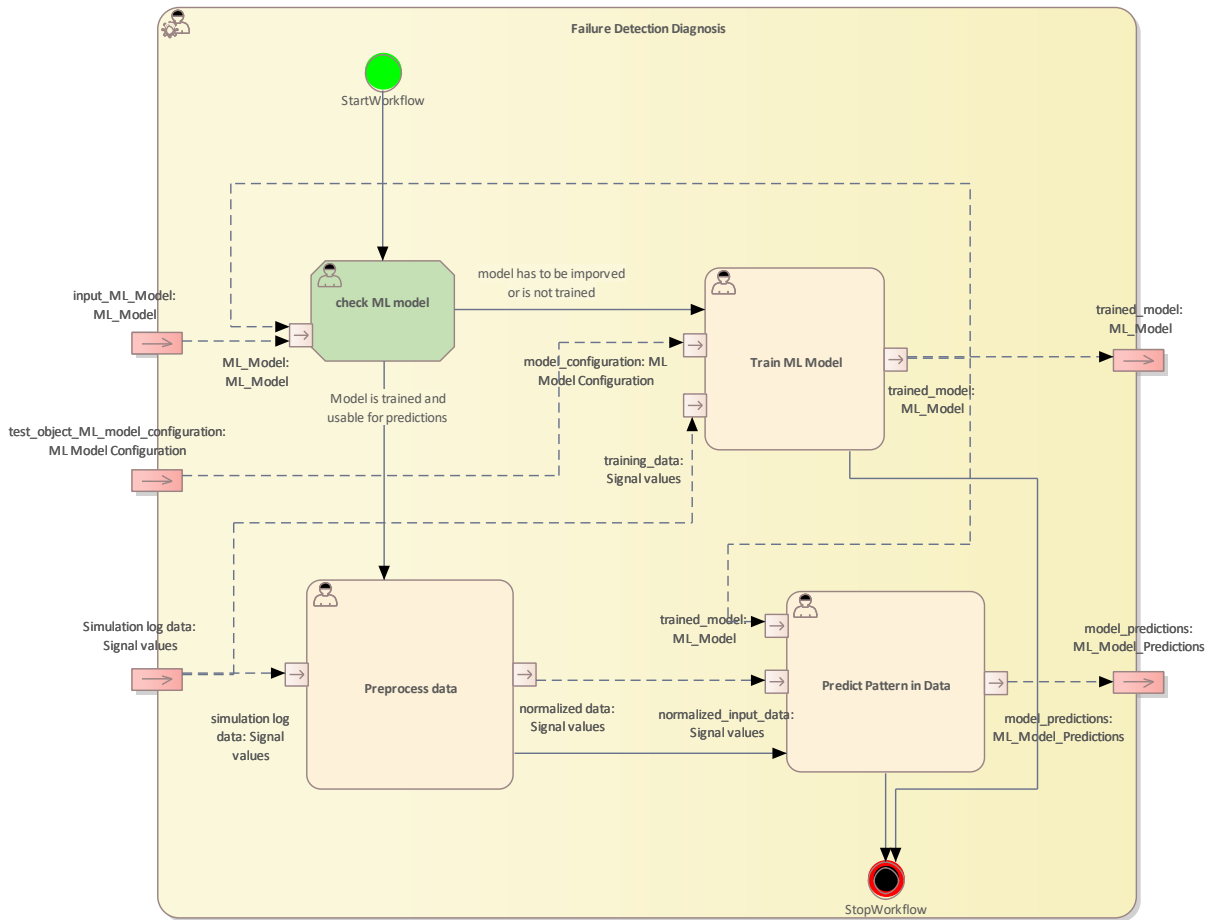


Figure 3.21 Workflow Definition diagram of Failure Detection Diagnosis used in UC4\_PUMACY

Table 3.15 lists the activities of the workflow Failure Detection Diagnosis.

Table 3.15 List of activities performed by Failure Detection Diagnosis

Name	Type	Description
Predict Pattern in Data	Manual	After machine learning model training, the trained model is deployed for failure detection. The LSTM model is trained in such a way that it takes a single sample and predicts the next future value. A threshold is set in a way that if the predicted output value from LSTM model overshoots the threshold, an alert signal has been generated in advance.
check ML model	Manual	The data from Simulation consists of important features/attributes. This step checks if there is any new feature or sensor added into the dataset. With the addition new sensor variable in existing dataset Machine Learning model must be retrained.



Name	Type	Description
Pre-process data	Manual	Stream of raw data collected from the CIROS simulation has to be pre-processed. Pre-processing involves: <ul style="list-style-type: none"> <li>• Data cleaning</li> <li>• Data segmentation</li> <li>• Feature correlation</li> <li>• Feature reduction</li> <li>• Data scaling</li> </ul>
Train ML Model	Manual	After pre-processing, raw data is converted into a suitable format which fed into machine learning model for training. Since our data is a stream of time series, deep learning model LSTM (Long Short-Term Memory) is used to predict the anomalies in an assembly process. LSTM networks are a type of recurrent neural network which are capable of learning patterns or order dependence in sequence. An LSTM network is well-suited to learn from experience to classify, process and predict time series events when there are very long-time lags of unknown size between important events.

### 3.4.4 V&V Workflows of Virtual Validation

Virtual validation uses a fully virtual setup, i.e., virtual components and a virtual environment, to check specific properties of the system under test through simulation. A virtual simulation environment and dedicated test cases are constructed and used to run and evaluate validation scenarios. The simulation scenarios comprise a set of simulation components, which are connected by dedicated adapters. Missing or incomplete components and adapters are created or extended in separate steps. Simulation components and adapters are deployed to defined execution nodes and connected to the FERAL framework [11]. FERAL executes the simulation scenarios and controls the simulation components and the data flow between them. Log data from simulation runs is collected and provided. A validation report is created after the execution and evaluation of all simulation scenarios.

Figure 3.22 shows the workflow specification diagram of Virtual Validation.

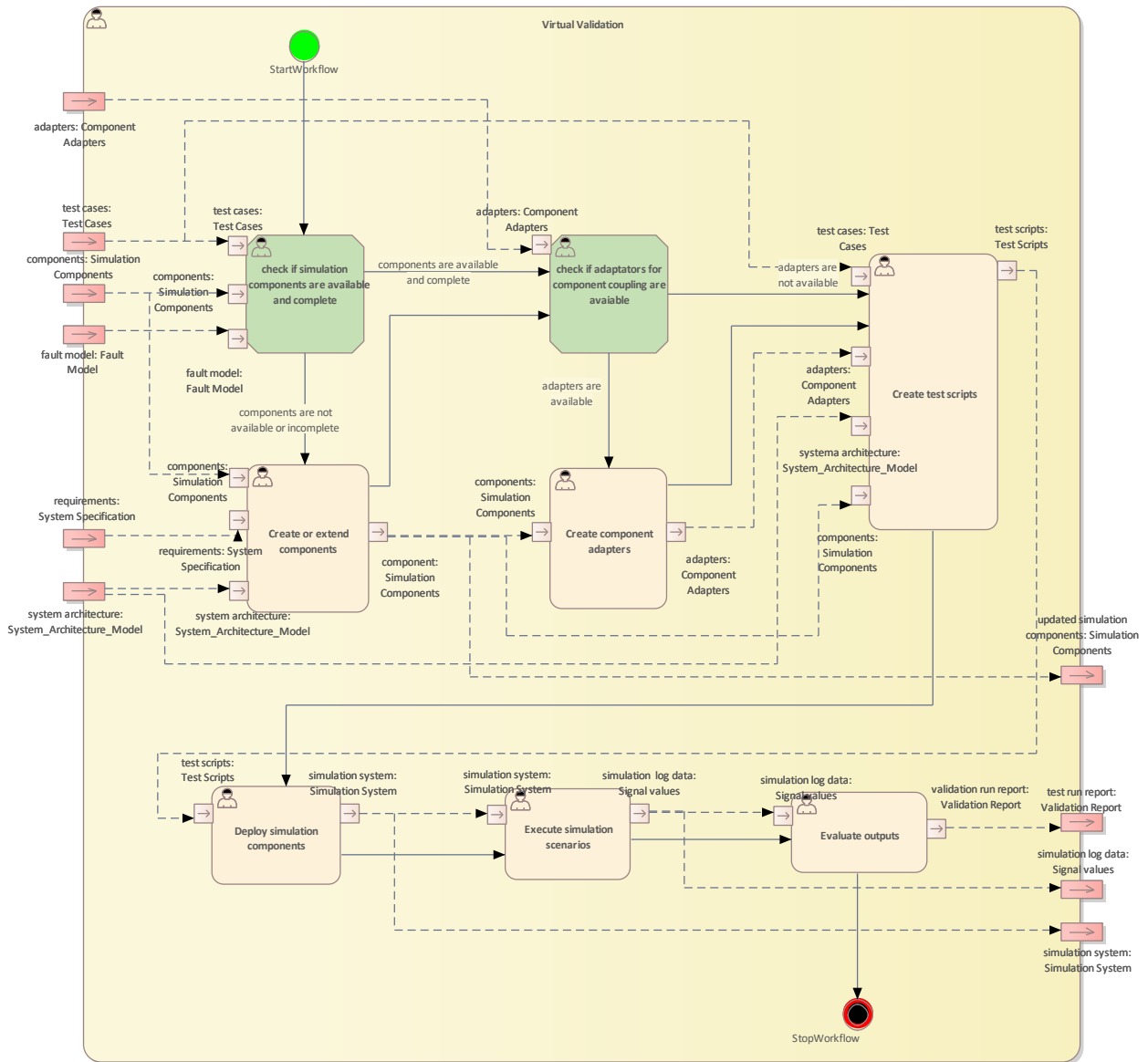


Figure 3.22 Workflow Definition diagram of Virtual Validation used in UC4\_PUMACY

Table 3.16 below lists the activities of the workflow Virtual Validation.

Table 3.16 List of activities performed by Virtual Validation

Name	Type	Description
Create component adaptors	Manual	Missing adapters for connecting components to the FERAL framework are created.
Create or extend components	Manual	Missing components for creating a virtual simulation scenario in FERAL are created. For this activity external simulation, modelling, or programming tools can be used.
Create test scripts	Manual	Validation scenarios are implemented as test scripts, which will be executed by the FERAL

Name	Type	Description
Deploy simulation components	Manual	Simulation components are deployed to their execution nodes and connected to the FERAL framework.
Evaluate outputs	Manual	Outputs from the validation runs are evaluated by dedicated log components.
Execute simulation scenarios	Manual	Simulation scenarios are executed within the FERAL framework.
check if adaptors for component coupling are available	Manual	Check if adaptors for connecting components to FERAL are available
check if simulation components are available and complete	Manual	Check if all required components of the simulation scenario are available.

### 3.5 V&V Workflow(s) of UC5 UTRCI

UC5\_UTRCI package contains the following workflows:

- SiLVer (SimuLation-based Verification)
- Model-implemented fault/attack injection with pre-injection analysis
- Classical Formal Verification Driven by Formal Requirements

Figure 3.23 shows the SILVER\_UTRCI Method Definition diagram type of the V&V workflow UC5\_UTRCI.

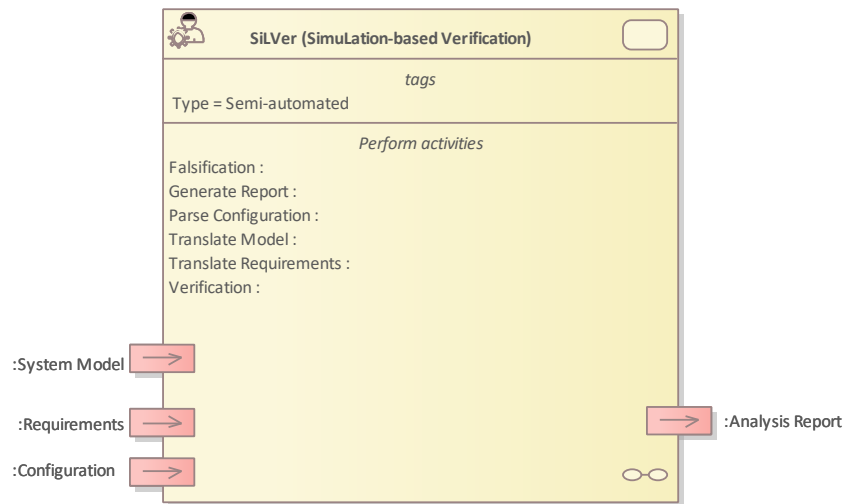


Figure 3.23 Method Definition of SILVER\_UTRCI defined for UC5\_UTRCI

Figure 3.24 shows the MIFI\_MIAI\_RISE (pre-injection) Method Definition diagram type of the V&V workflow UC5\_UTRCI.

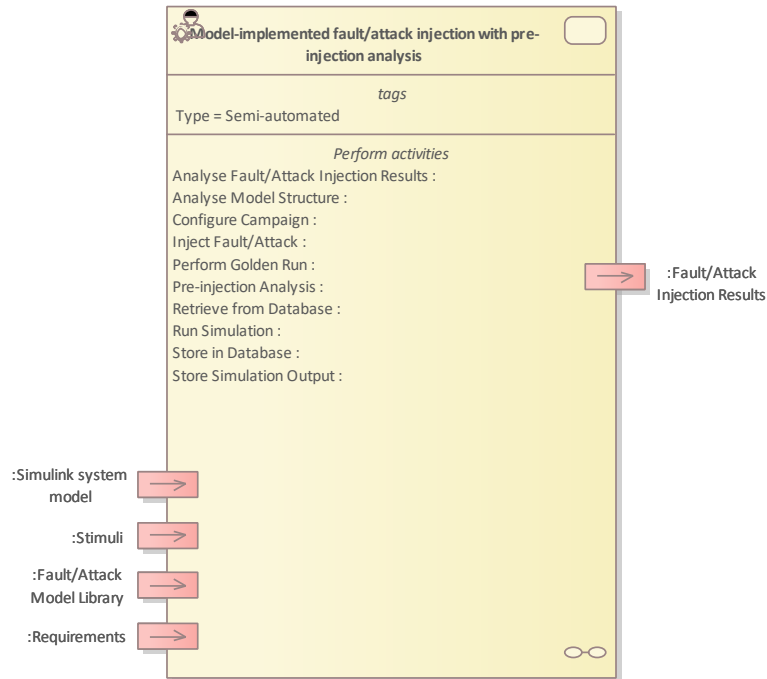


Figure 3.24 Method Definition of MIFI\_MIAI\_RISE (pre-injection) defined for UC5\_UTRCI

Figure 3.25 shows the Verifying and Refactoring Formalised Requirements Method Definition diagram type of the V&V workflow UC5\_UTRCI.

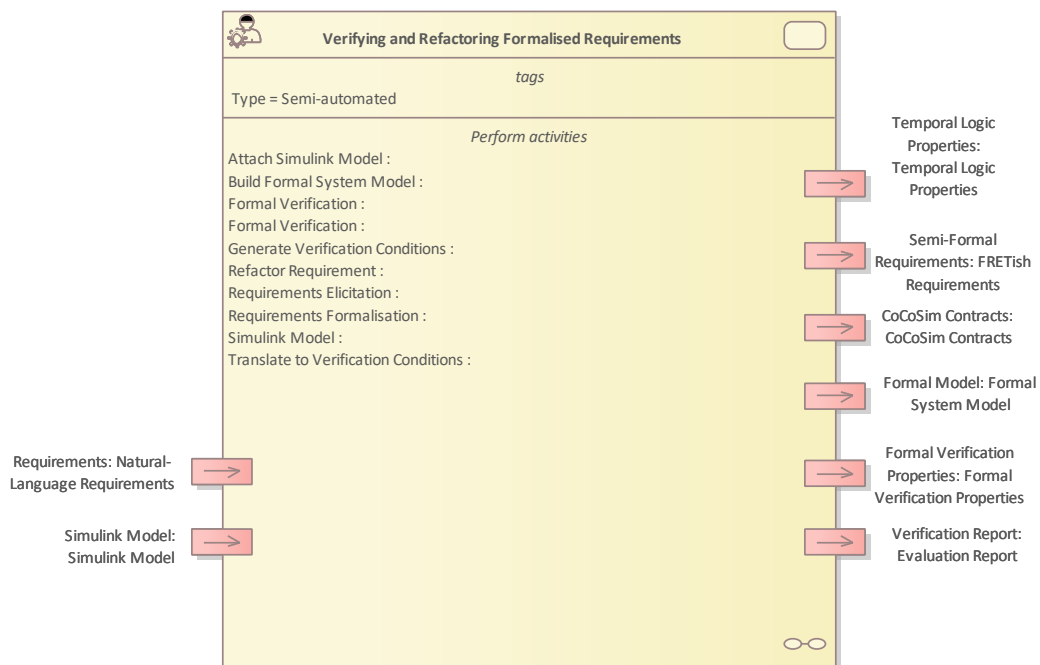


Figure 3.25 Method Definition of Verifying and Refactoring Formalised Requirements defined for UC5\_UTRCI

Details of the workflows are given in the following subsections.

### 3.5.1 Artifacts used in UC5\_UTRCI

Table 3.17 lists the artifacts used for the workflows defined for UC5\_UTRCI.

Table 3.17 List of artifact types used in UC5\_UTRCI

Name	Description
Analysis Report (Information)	Textual report based on system analysis results. Optionally, includes plots.
CoCoSim Contracts (Active Unit)	Verification contracts, written in the contract language CoCoSpec [12], derived from the system's requirements.
Configuration (Information)	Set of YAML configuration files containing information such as whether to perform verification or falsification, whether to plot the results, etc.
Evaluation Report (Information)	The output of the verification step, formatted as produced the verification tool(s) that were used. This is likely to be the verification conditions and if they passed (or counterexamples if they failed), but will depend on the tools and methods used by the verifier.
Fault/Attack Injection Results (Information)	Results of V&V of the target Simulink system model.
Fault/Attack Model Library (Information)	Library of fault- and attack Simulink models that can be injected into the target Simulink system model.
Formal System Model (Active Unit)	A formal model of the system, derived from the requirements and other design documents. This will be written in the formal language of choice by the verifier.
Formal Verification Properties (Active Unit)	Formalised properties that the system or design should be checked for. They are derived from the requirements and written in the formal language of choice of the verifier.
FRETish Requirements (Active Unit)	The semi-formalised requirements written in the controlled natural language FRETish [13] (which is the input language to the tool FRET).
Natural-Language Requirements (Information)	The system's requirements, written in natural-language. These may be improved or added to during the requirement elicitation activity.
Requirements (Information)	V&V requirements of the target Simulink system model evaluated.
Simulink system model (Active Unit)	V&V target Simulink system model evaluated.
Stimuli (Information)	Input to the target Simulink system model during simulation.
System Model (Information)	A Matlab Simulink model of the system's design.
Temporal Logic Properties (Active Unit)	Temporal logic properties automatically generated from the FRETish requirements.

### 3.5.2 V&V Workflows of Verifying and Refactoring Formalised Requirements

The method takes the system’s natural-language requirements and a Simulink diagram of the system’s design as input, and ultimately produces formal verification results (usually either that a requirement-check passes, or a counterexample that shows the events leading to a violation of the requirement). Other outputs are produced and re-consumed within the method but are also available for use outside the method. These artefacts include the semi-formal and temporal-logic versions of the requirements, other formal models of the system, etc.

Figure 3.26 shows the workflow specification diagram of Verifying and Refactoring Formalised Requirements.

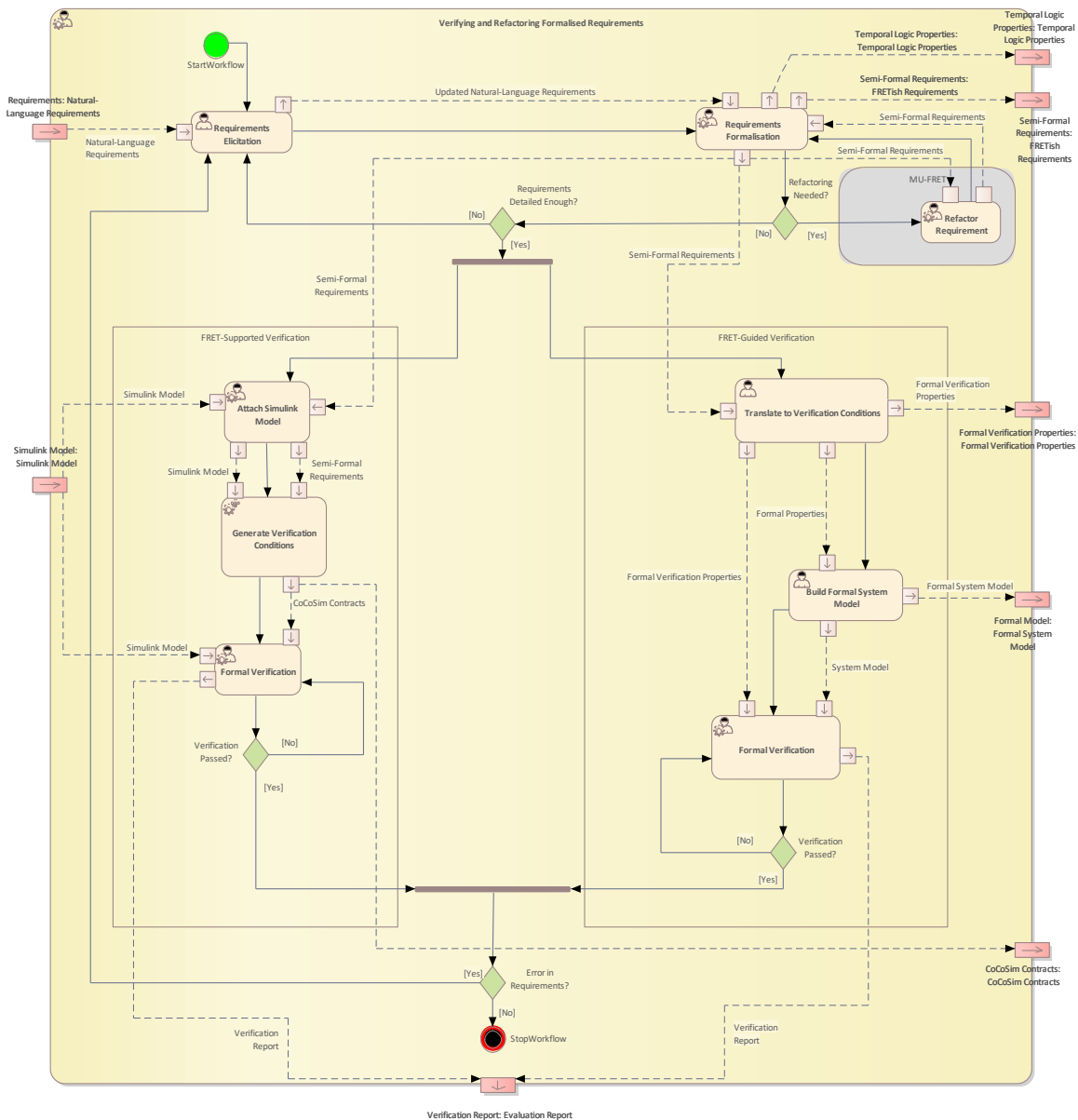


Figure 3.26 Workflow Definition diagram of Verifying and Refactoring Formalised Requirements used in UC5\_UTRCI

Table 3.18 lists the activities of the workflow Verifying and Refactoring Formalised Requirements

Table 3.18 List of activities performed by Verifying and Refactoring Formalised Requirements

Name	Type	Description
Attach Simulink Model	Semi-automated	Within MU-FRET, add the Simulink that describes the system's design.
Build Formal System Model	Manual	Using the available information about the system (various forms of requirements, possibly even design diagrams) build a formal model of the system in the verifier's formal language of choice. For example, in our work with Event-B, this step involves constructing an Event-B specification that describes the system.
Formal Verification	Semi-automated	(Right-hand side of Figure 3.26) Perform the formal verification of the Formal Properties (verification conditions) against the formal system model. For example, in our work with Event-B, this step uses the Rodin tool to prove the invariants and other constrains.
Formal Verification	Semi-automated	(Left-hand side of Figure 3.26) Run the formal verification of the CoCoSim contracts on the Simulink diagram. This happens in Simulink, using the CoCoSim plugin.
Generate Verification Conditions	Automated	Export the verification conditions, based on the formalised requirements, for the attached Simulink diagram. This outputs contracts in CoCoSpec, a contract language for CoCoSim.
Requirements Elicitation	Manual	Using the Natural-Language version of the system's requirements, and potentially the currently formalised version of the requirements, elicit either new requirements or more detail about the existing requirements.
Requirements Formalisation	Semi-automated	Formalise the natural language requirements. Outputs requirements in FRETish (a controlled natural language for describing requirements) and Temporal Logic.
Translate to Verification Conditions	Manual	Manually produce the verification conditions, from the FRETish version of the requirements (or using the Temporal Logic properties directly). This produces formal properties in the verifier's language of choice. For example, in our work with Event-B, this step involves creating invariants and other properties that the Event-B specification must obey.
Refactor Requirement	Semi-automated	Apply a refactoring to a requirement. Refactoring is the process of reorganising the requirements without changing their behaviour.



Name	Type	Description
		This activity also formally verifies that the new requirement and the original requirement have the same behaviour.

### 3.5.3 V&V Workflows of Model-implemented Fault and Attack Injection with Pre-Injection Analysis

The workflow describes the application of the V&V methods MIFI (Model-Implemented Fault Injection with Pre-injection Analysis) and MIAI (Model-Implemented Attack Injection with Pre-injection Analysis). These methods allow fault- and attack injection mechanisms implemented as model blocks to be injected into simulated target system models to evaluate the impact of faults and cybersecurity attacks on target systems at early development phases.

The workflow has four inputs: Simulink system model, Stimuli, Fault/Attack Model Library and Requirements. There is also one output: the Fault/Attack Injection Results. The sequence of main activities included in the workflow is as follows:

- Analyse Model Structure: The target Simulink system model structure is analysed to determine the fault/attack space.
- Perform Golden Run: The target Simulink system model is simulated with the chosen Stimuli and monitored to obtain the nominal (fault/attack free) behaviour.
- Pre-injection Analysis: The target Simulink system model fault/attack space structure and nominal behaviour is analysed to reduce the fault/attack space.
- Configure Campaign: The campaign of fault- and attack injections to conduct on the target Simulink system model is configured. The target system Requirements and Stimuli to use for the simulations determines the set of faults and attacks to inject including their locations, activation times and durations.
- Inject Fault/Attack: The faults and attacks are injected into the target Simulink system model according to the campaign configuration.
- Run Simulation: The target Simulink system model injected according to the campaign configuration is simulated with the chosen stimuli and the target system behaviour is monitored.
- Analyse Fault/Attack Injection Results: The monitored data obtained during simulations of the golden run and fault injected target Simulink system model is analysed based on the Requirements to report the Fault/Attack Injection Results of the campaign performed.

Figure 3.27 shows the workflow specification diagram of Model-implemented fault/attack injection with pre-injection analysis.

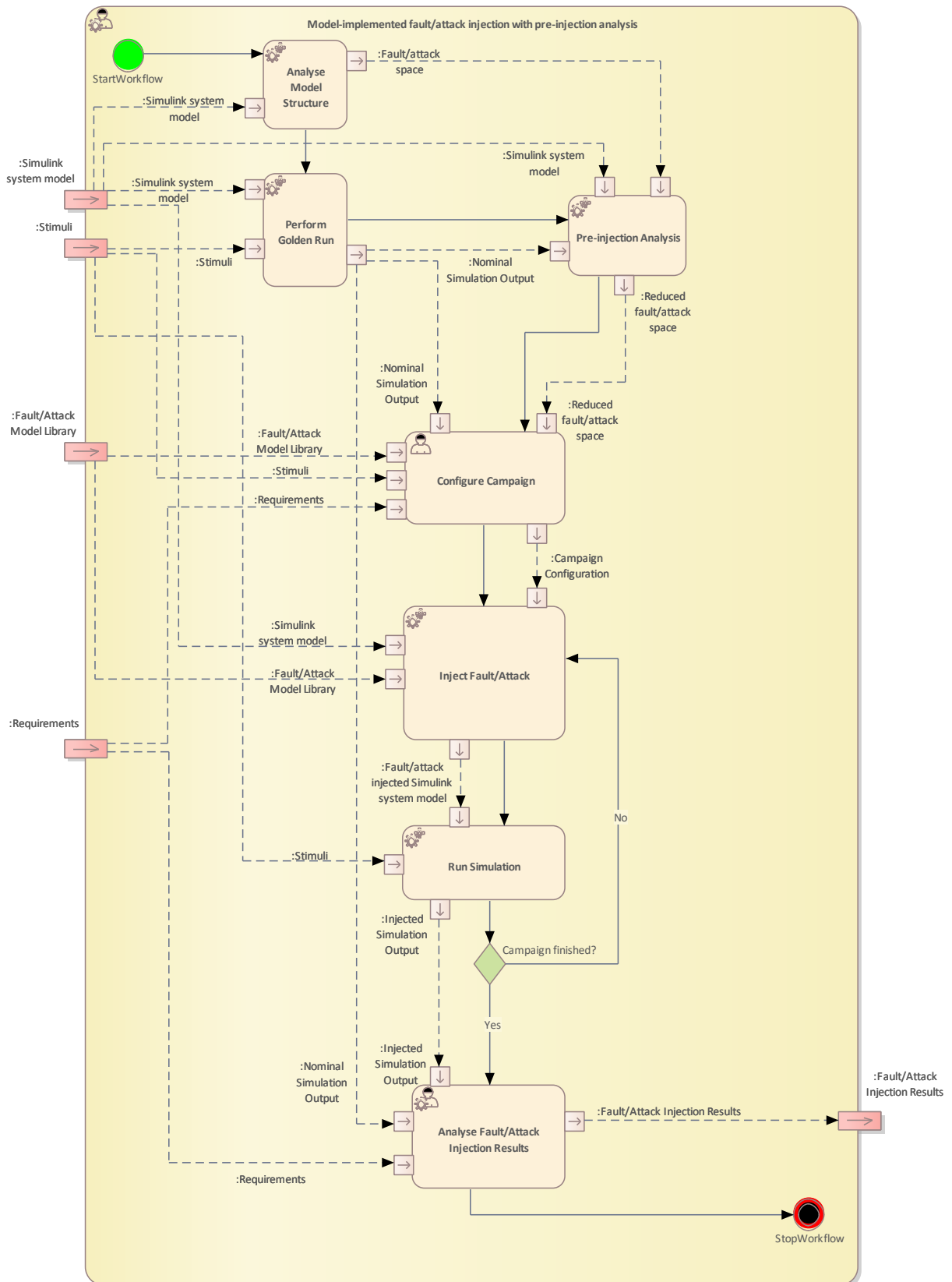


Figure 3.27 Workflow Definition diagram of MIFI\_MIAI\_RISE used in UC5\_UTRCI

Table 3.19 lists the activities of the workflow Model-implemented fault/attack injection with pre-injection analysis.

Table 3.19 List of activities performed by Model-implemented fault/attack injection with pre-injection analysis

Name	Type	Description
Analyse Fault/Attack Injection Results	Semi-automated	Analyse Fault/Attack Injection Results: The monitored data obtained during simulations of the golden run and fault injected target Simulink system model is analysed based on the Requirements to report the Fault/Attack Injection Results of the campaign performed.
Analyse Model Structure	Automated	Analyse Model Structure: The target Simulink system model is analysed to determine the fault/attack space structure.
Configure Campaign	Manual	Configure Campaign: The campaign of fault- and attack injections to conduct on the target Simulink system model is configured. The target system Requirements and Stimuli to use for the simulations determines the set of faults and attacks to inject including their locations, activation times and durations.
Inject Fault/Attack	Automated	Inject Fault/Attack: The faults and attacks are injected into the target Simulink system model according to the campaign configuration.
Perform Golden Run	Automated	Perform Golden Run: The target Simulink system model is simulated with the chosen Stimuli and monitored to obtain the nominal (fault/attack free) behaviour.
Pre-injection Analysis	Automated	Pre-injection Analysis: The target Simulink system model fault/attack space structure and nominal behaviour is analysed to reduce the fault/attack space.
Run Simulation	Automated	Run Simulation: The target Simulink system model injected according to the campaign configuration is simulated with the chosen stimuli and the target system behaviour is monitored.

### 3.5.4 V&V Workflows of SiLVer (SimuLation-based Verification)

The workflow is a semi-automated approach for determining whether a system model conforms to a given set of requirements. It supports both verification and falsification of the given system model w.r.t. requirements. Falsification is conducted by Monte-Carlo simulation runs, while verification is essentially symbolic simulation of the system model using Affine arithmetic. System model and requirement monitors are expected to be provided as C++ code. This enables analysis throughout the design cycle (even when moving to implementation). Currently, there is support for automatic generation of C++ code from specific, parametrized requirement / system templates. Once the system

model and requirements are provided, depending on the choice (verification or falsification) made by the user (via the configuration file), the appropriate algorithm is run, generating a report with analysis results.

Figure 3.28 shows the workflow specification diagram of SiLVer (SimuLation-based Verification).

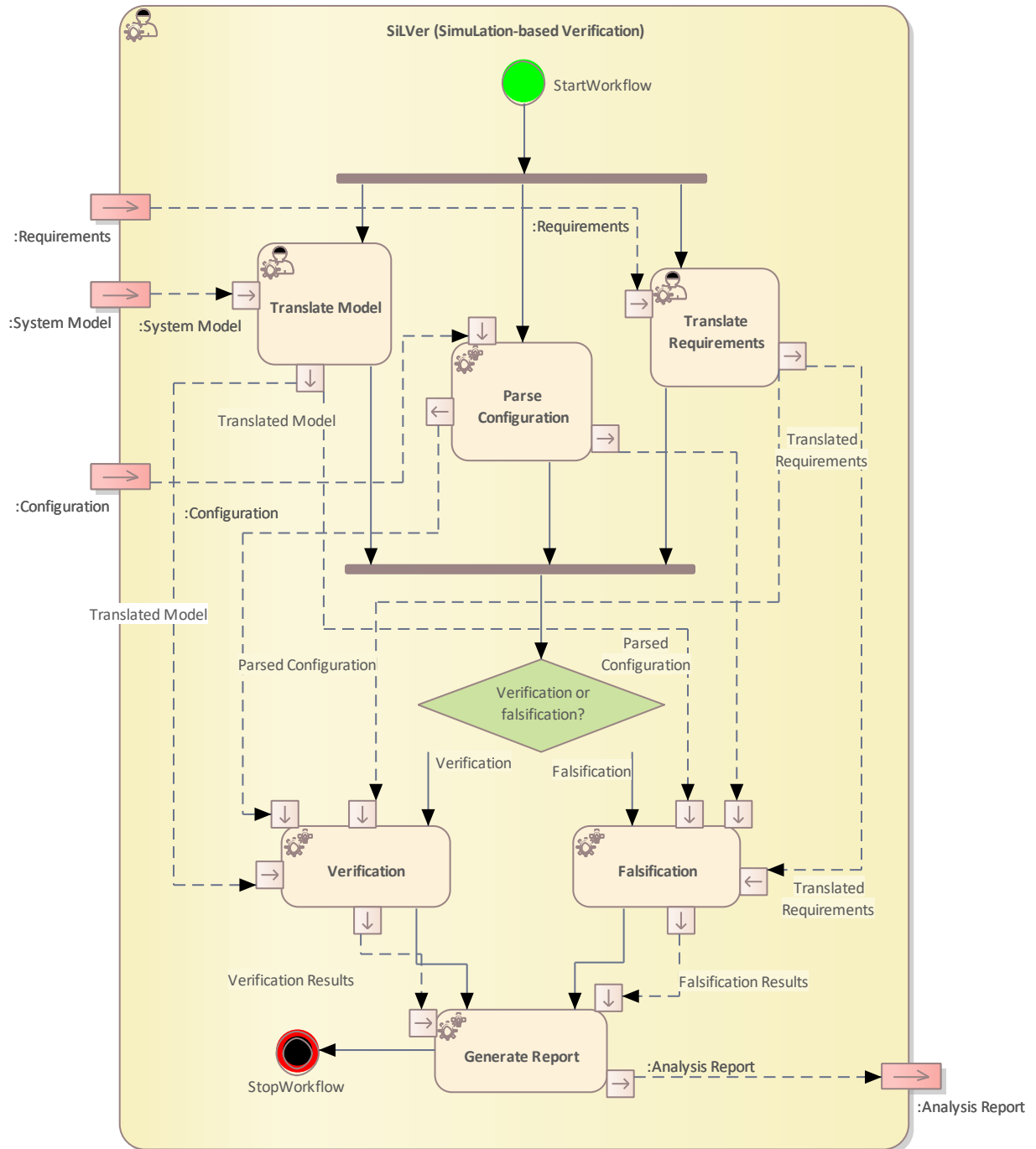


Figure 3.28 Workflow Definition diagram of SiLVer (SimuLation-based Verification) used in UC5\_UTRCI

Table 3.20 lists the activities of the workflow SiLVer (SimuLation-based Verification).

Table 3.20 List of activities performed by SiLVer (SimuLation-based Verification)

Name	Type	Description
Falsification	Automated	Performs Monte Carlo simulation runs
Generate Report	Automated	Creates textual report based on system analysis results. Optionally, includes plots.
Parse Configuration	Automated	Reads set of YAML configuration files containing information such as whether to perform verification or falsification, whether to plot the results, etc.
Translate Model	Semi-automated	Generates C++ code representing the system model based on a parametrized template and a YAML file containing specific values for parameters. (Note: system template has to be created manually)
Translate Requirements	Semi-automated	Generates C++ code representing the requirements based on parametrized templates and a YAML file containing specific values for parameters. (Note: requirement templates have to be created manually)
Verification	Automated	Performs reachability-analysis-based verification for safety-critical hybrid systems (through Affine Arithmetic)

### 3.6 V&V Workflow of Use Case 6 ESTE

UC6\_ESTE package contains the following workflow:

- Model-based safety analysis FLA

Figure 3.29 shows the UC6\_ESTE Method Definition diagram type of the V&V workflow UC6\_ESTE.

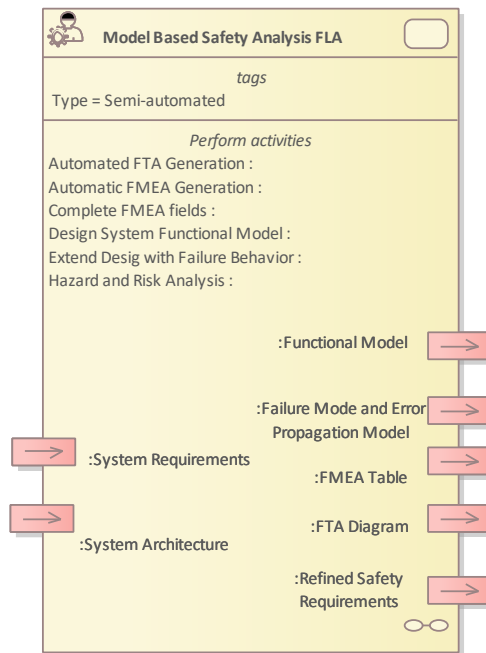


Figure 3.29 Method Definition of UC6\_ESTE defined for UC6\_ESTE

Details on the workflow are given in the Section 3.8.3. UC6 and UC8 apply the workflow Model Based Safety Analysis FLA.

### 3.7 V&V Workflow of Use Case 7 ALDAKIN

UC7\_ALDAKIN package contains the following workflow:

- MGEP V&V Workflow

Figure 3.30 shows the MGEP-2 UC7 Method Definition diagram type of the V&V workflow UC7\_ALDAKIN.

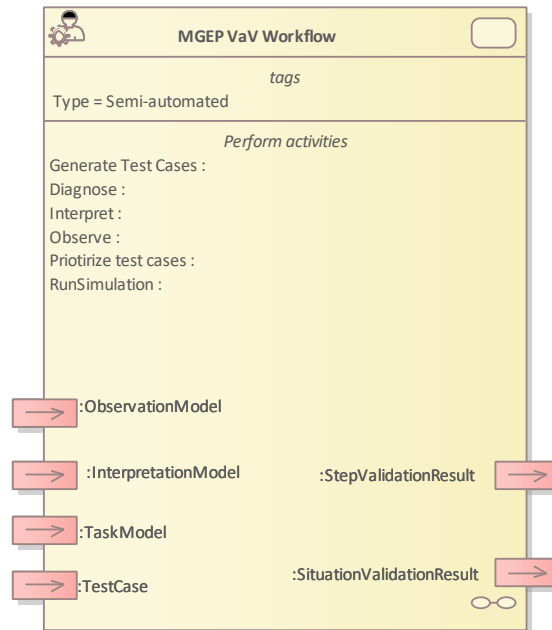


Figure 3.30 Method Definition of MGEP-2 UC7 defined for UC7\_ALDAKIN

Details on the workflow are given in the following subsections.

#### 3.7.1 Artifacts used in UC7\_ALDAKIN

Table 3.21 lists the artifacts used for the workflow defined for UC7\_ALDAKIN.

Table 3.21 List of artifact types used in UC7\_ALDAKIN

Name	Description
InterpretationModel (Information)	<p>Interpretation model: It needs to define how Steps and Situations are interpreted using constraint-based-modelling. These level does not define whether a step or situations are being carried out correctly, it describes the model to know if they are being executed or not in a specific time interval.</p> <p><b>Steps:</b> intervals that have a meaning inside a situation. They are defined using constraint base modelling.</p>

Name	Description
	<b>Situation:</b> they represent the context where the actions are taking place and are composed of observations and steps. They are defined using constraint-based-modelling.
ObservationModel (Information)	It is a XML file where the observations or facts that are need to be observed from the interactive system (simulation in this case) are defined. Those observations are the base level to define steps and situations via a constraint-based-modelling technique.
SimulationStream (Active Unit)	A stream of information that is coming from the simulation. In this particular case, different variables' information is acquired via MQTT subscription.
Situations (Active Unit)	element note
SituationValidationResult (Information)	This artifact is the result of diagnosing a situation. It will return True if the steps that have been carried out were executed correctly.
StepValidationResult (Information)	This artifact is the result of diagnosing a step. It will return True if the steps have been carried out correctly. Otherwise, it will return a False result.
TaskModel (Information)	The diagnosis level specifies how humans or robots in the simulation must be evaluated when they are carrying out a task, which involves detecting their errors. To achieve this aim, the diagnosis subsystem receives from the interpreter the steps that are being carried out and also the situations that are associated with them. The steps and situations that must be diagnosed are specified in the Task Model.
Test case (Information)	The test case will include the id of the simulation to be run and the id of the file where operator's movements are described

### 3.7.2 V&V Workflows of MGEP V&V Workflow

The ULISES framework transforms data streams generated by a Virtual Reality Interactive System into data suitable to diagnose a test case in real time. This diagnosis will generate sufficient information to validate a test case specification.

ULISES is a three-layered framework that explicitly models the unconscious process that a real human use when they supervise real activities: they first perceive the environment through their senses, then they interpret what is happening and lastly, they make a diagnosis about what happened. In order to ensure that the runtime kernel is able to observe, interpret and diagnose students' activity, the ULISES framework defines the ULISES metamodel, which is divided into three abstraction levels and each of them generically describes a set of elements that have to be particularized into the Task, Interpretation and Observation Models. In other words, the metamodel defines elements to specify how to observe the actions that are being carried out in the Interactive System, how to interpret the steps taken by user



or robots and the context in which they are taken, and how to diagnose them. Hence, each level of the metamodel represents a different phase in the task creation process.

The core unit of this level is *observation*, which represents an event or a fact *taking place during an interval of time* in the simulation. Therefore, the Observation level contains generic elements that specify how the data streams coming from the IS must be transformed into single meaningful entities (observations) that describe something perceived during an interval of time. Taking a driving simulator as an example, the driver perceives certain elements during different periods of time: the intersection he is approaching, the vehicle on his left, the solid line on the road, and so on. These observations will be used as primitives for describing students' activity on the other two levels: interpretation and diagnosis. For example, if overtaking (which is a durative action) needs to be described in the interpretation level, first the movements to other lanes and the movement of the preceding vehicle need to be observed. Additionally, observation *properties* can be defined. For example, the "Preceding vehicle" observation would have a "Distance" property that would register the change in the distance to the preceding vehicle during the interval of time when it is observed.

Allowing both discrete and continuous input is a crucial advantage at this level. Thus, both continuous observations, e.g., the driver is approaching the left lane, and instantaneous observations, e.g., the driver is pushing a button, can be defined. The observation model defines for every observation its properties, the input data (from any source) needed to generate the observation and its properties, and the ULISES Observer plugin, which processes the input data. With this model, the ULISES runtime kernel fuses the inputs from the simulation and updates in real time the set of synchronized observations that are perceived. This means that after all update cycle new observations are perceived, other observations are completed and still other observations continue with updated properties. Then, the interpretation subsystem is notified in order to perform its own interpretation cycle.

The interpretation level generically describes how to recognize human or robots' activity in the simulation, that is, it expresses digitally what is happening in the virtual environment. Just as instructors make subjective interpretations based on what they perceive, ULISES does the same when interpreting the observations from the IS. The interpretations shall be valid and complete enough so that the ULISES diagnosis subsystem can determine whether actions in the test case are valid or not. Therefore, the interpretation subsystem must recognize the actions performed by the different simulation elements and the context in which the actions are being carried out. For this reason, the interpretation level specifies how the interpretation subsystem must interpret two core elements from observations: steps and situations:

- Step: This represents an action that takes place over an interval of time and that will be diagnosed. The step model contains the necessary attributes for the ULISES runtime kernel to interpret when a step is being performed from observations. It is important to emphasize that the observations contained in the step model cannot depend on the correctness or incorrectness of the step. For example, in a driving context, if an "overtaking" step is modelled, the observation where the vehicle is in the left lane should not be included. The driver could be overtaking from the right, incorrectly, but still overtaking. Determining correctness happens at the diagnostic level rather than at the interpretation level.

- Situation: The term *context* has many definitions, but all of them underline the importance of the context when interpreting or diagnosing an activity. A situation represents a context that is relevant for diagnosing where some steps will be performed. We identify specific situations when there is a set of factors that determines the steps that students must perform. For example, driving on the highway at 100 km/h is not the same as driving at the same speed on a city street. The student is performing the same step, but it has a different meaning in each situation.

The diagnosis level contains the elements that must be particularized in order to generate the task model. This model describes the tasks that simulation elements are to perform so they can be diagnosed automatically. In order to define the composition of a task and its validation, the diagnosis level defines the following elements:

- Step: This represents the minimum diagnosable unit.
- Situation: This represents the diagnosis context. It includes the specification of possible solutions to the situation and the information (observations) that will be necessary to diagnose the steps executed.
- Solution: A solution defines if a specific task is valid within a situation. Each solution can be linked to a different diagnosis module, so multiple diagnosis techniques can be used at the same time for different solutions. Nevertheless, the solution must always be composed of steps, although the specific structure depends on the diagnosis technique that is used.

Due to the domains where the system will be validated, we chose to implement a constraint satisfaction-based diagnosis technique. Within this technique, constraints are used as an element to restrict whether a step is correct or incorrect in the solution specification. Rather than defining a way to solve a problem, constraints allow for the definition of how certain actions should be solved in order to detect mistakes. Thus, if a constraint rule is not satisfied, a mistake is detected. The advantage of constraint-based modelling is that it is possible to group actions that violate the same domain principle. In our case, we go further and define solutions over situations and steps. We group actions that violate the same domain principles, but at the same time, we are able to distinguish the context in which the mistake was committed. This distinction is very important, because there are many cases where the same error can have a different meaning.

Figure 3.31 shows the workflow specification diagram of MGEP V&V Workflow.

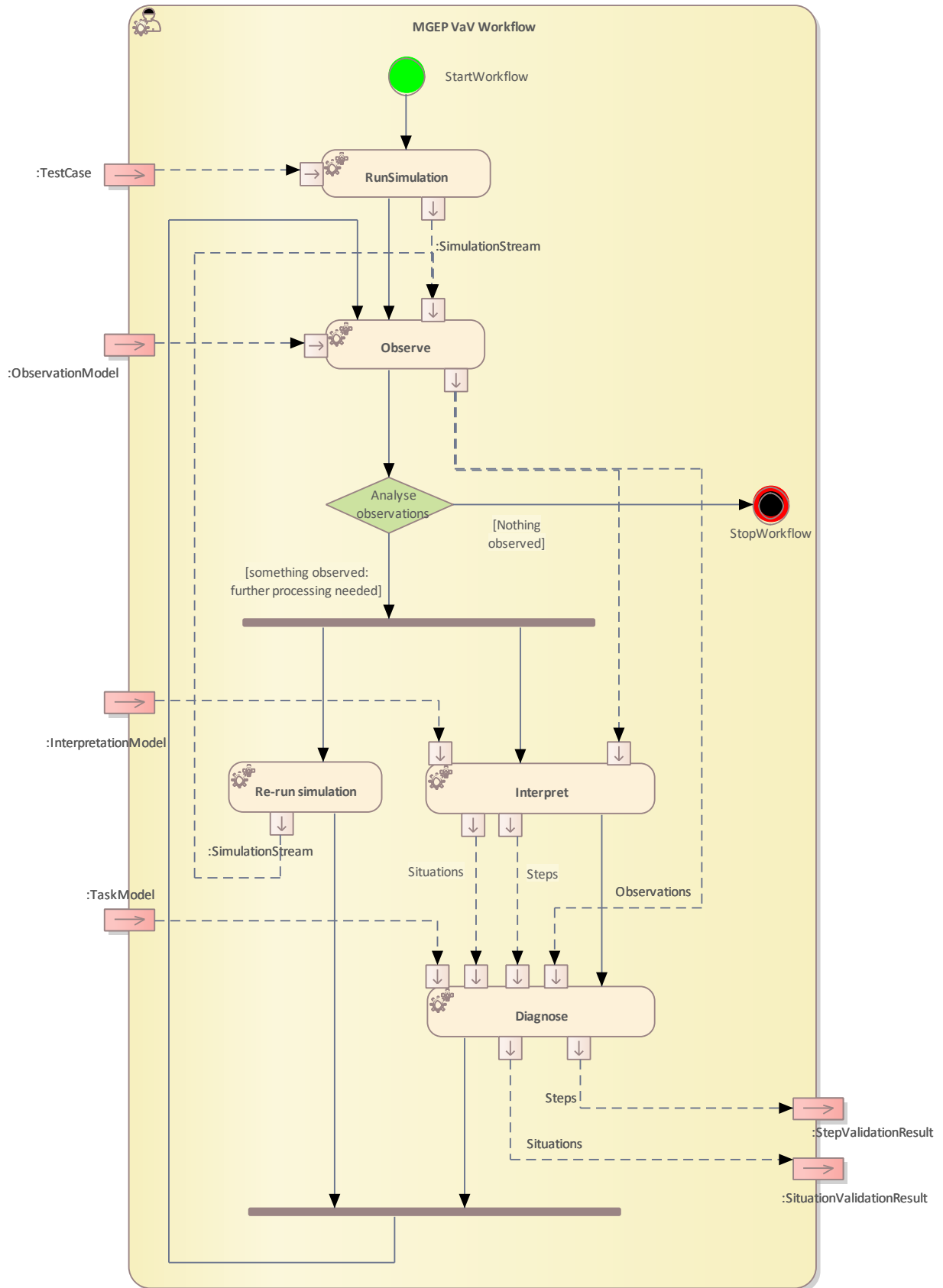


Figure 3.31 Method Definition of MGEV V&V Workflow used in UC7\_ALDAKIN

Table 3.22 lists the activities of the workflow MGEP V&V Workflow.

*Table 3.22 List of activities performed by MGEP V&V Workflow*

Name	Type	Description
Diagnose	Automated	Diagnosing means evaluating the correctness of each of the steps and situations that have been defined in the Task Model. This activity receives the observations, steps and situations that are being carried out in each simulations cycle and evaluates the correctness of steps and situations. The solution to a test is defined in the Task Model based on a Constraint-based modelling language.
Interpret	Automated	During this activity, steps and situations defined in the Interpretation Model will be interpreted. In this context, interpreting means detecting which steps are being carried out in the simulation, and in which context (situation) are being executed. These interpretations are defined on top of observations and/or other steps, whose relationships are described via a constraint-based modelling language.
Observe	Automated	The aim of the observation subsystem is gathering data streams and converting this data into information that is suitable for the other two subsystems: the interpretation and the Diagnosis subsystem. Each cycle, it will generate observations that have been specified in the Observation Model and publish them for other agents.
RunSimulation / Re-run simulation	Automated	It will run the simulation that is specified in the TestCase. Each simulation is related to one test. During the execution, every time cycle simulation streams will be transmitted to the ULISES runtime kernel so its agents gather these streams and convert them to observations.

### 3.8 V&V Workflow of Use Case 8 RGB

UC8\_RGB package contains the following workflows:

- Model Based Safety Analysis FLA
- Tailored Model-Based Assurance and Certification
- Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis
- Extended Knowledge-Centric System Traceability Management
- Single Experiment
- TC Automated Experimenting
- TC Management – TCM

Figure 3.32 shows the TC Management Method Definition diagram type of the V&V workflow UC8\_RGB.

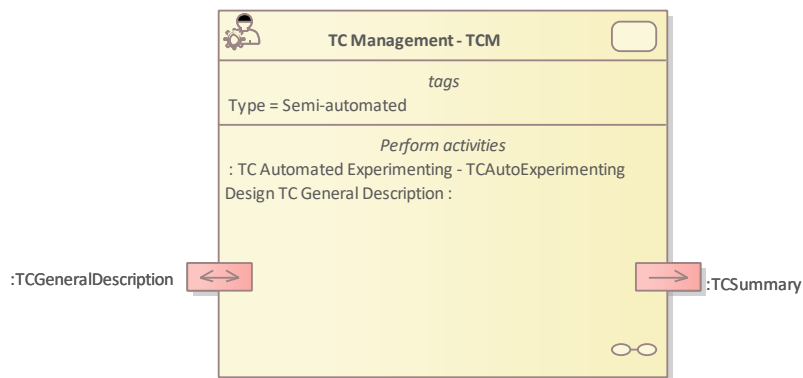


Figure 3.32 Method Definition of TC Management defined for UC8\_RGB

Figure 3.33 shows the TC Automated Experimenting Method Definition diagram type of the V&V workflow UC8\_RGB.

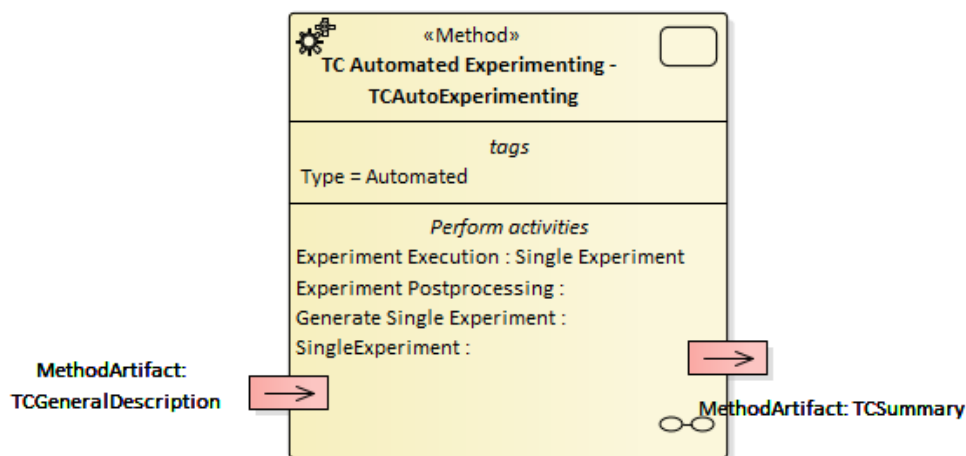


Figure 3.33 Method Definition of TC Automated experimenting for UC8\_RGB

Figure 3.34 shows the Single Experiment Method Definition diagram type of the V&V workflow UC8\_RGB.

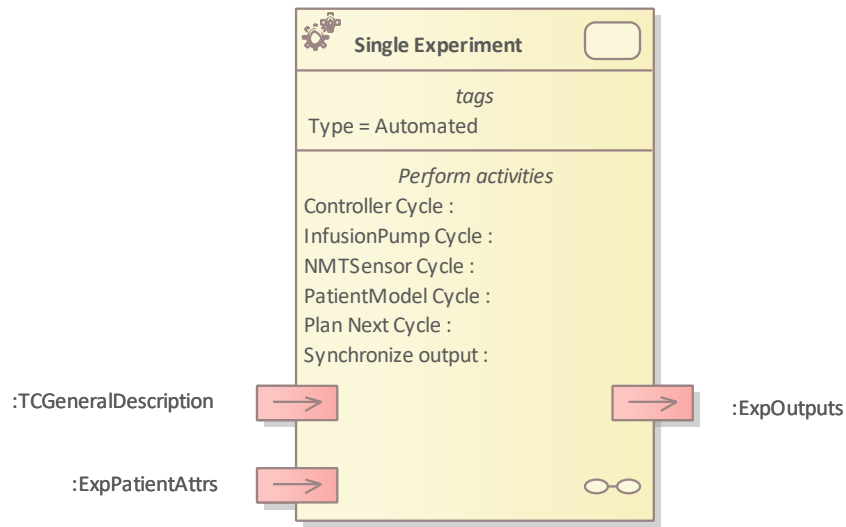


Figure 3.34 Method Definition of Single Experiment for UC8\_RGB

Figure 3.35 shows the Model based Safety Analysis FLA Method Definition diagram type of the V&V workflow UC8\_RGB.

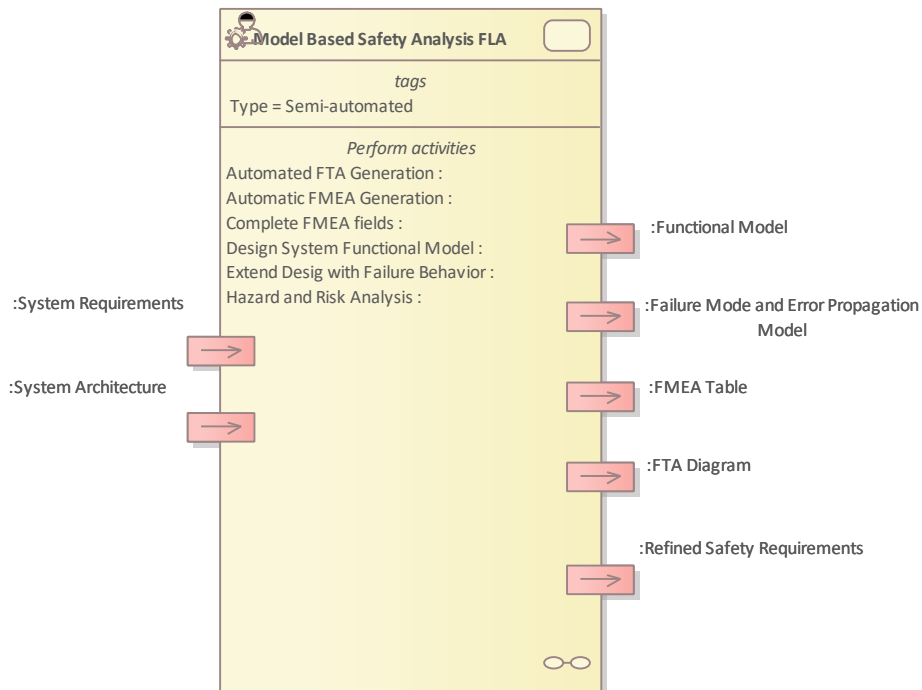


Figure 3.35 Method Definition of Model based Safety Analysis FLA defined for UC8\_RGB

Figure 3.36 shows the Tailored Model-based Assurance and Certification Method Definition diagram type of the V&V workflow UC8\_RGB.

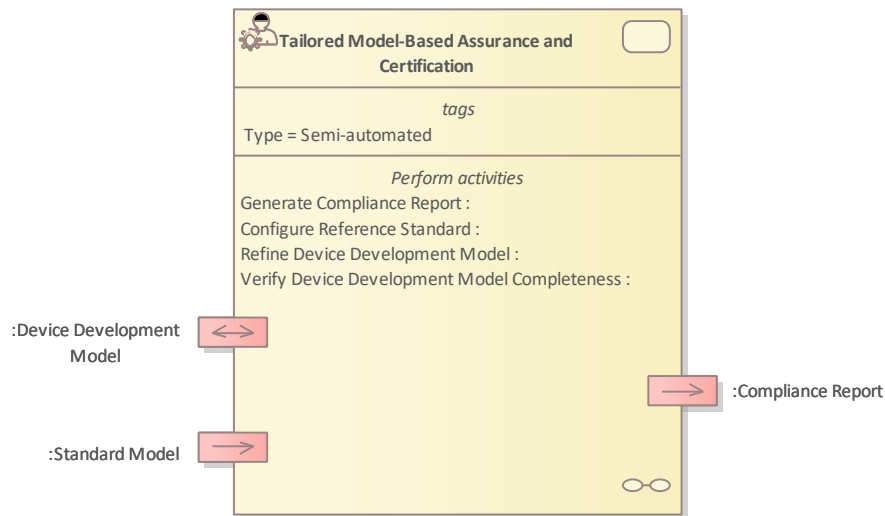


Figure 3.36 Method Definition of Tailored Model-based Assurance and Certification defined for UC8\_RGB

Figure 3.37 shows the Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis Method Definition diagram type of the V&V workflow UC8\_RGB.

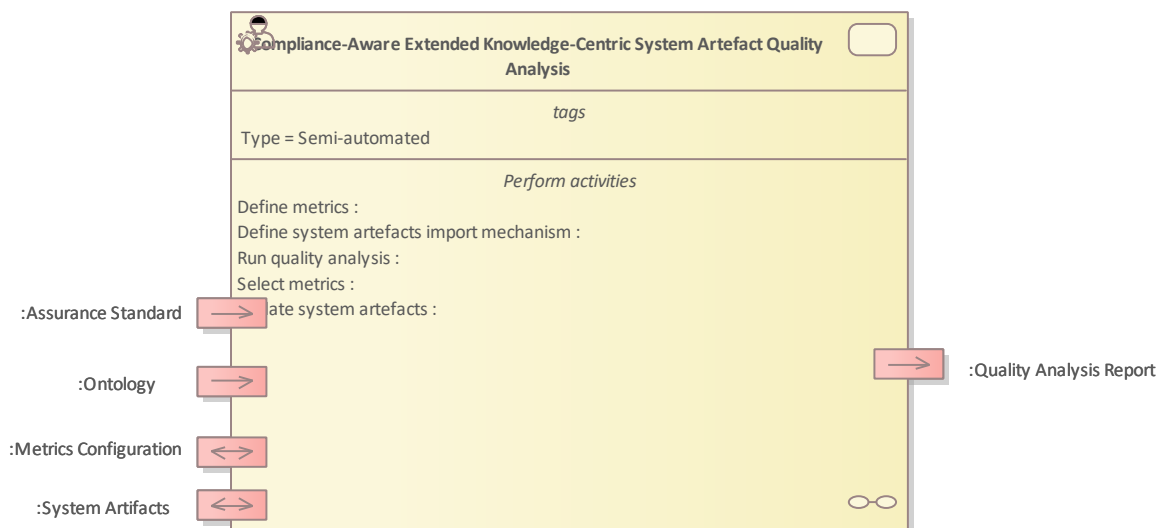


Figure 3.37 Method Definition of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis defined for UC8\_RGB

Figure 3.38 shows the Extended Knowledge-Centric System Traceability Management Method Definition diagram type of the V&V workflow UC8\_RGB.

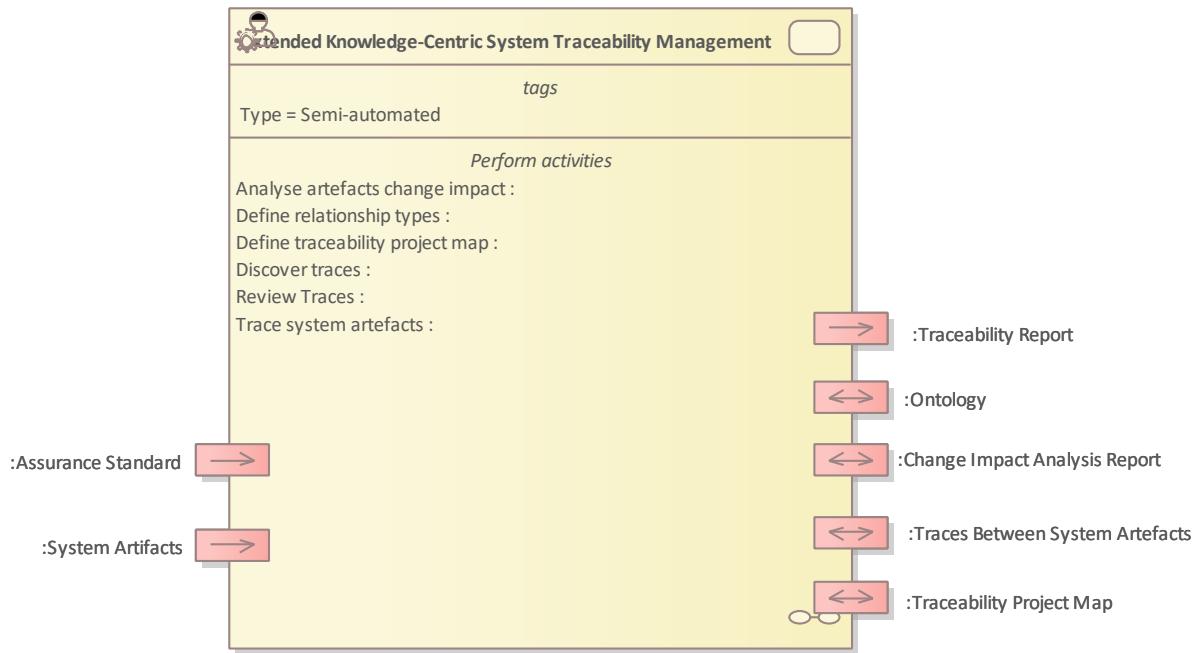


Figure 3.38 Method Definition of Extended Knowledge-Centric System Traceability Management defined for UC8\_RGB

Details on the workflows are given in the following subsections.

### 3.8.1 Artifacts used in UC8\_RGB

Table 3.23 lists the artifacts used for the workflows defined for UC8\_RGB.

Table 3.23 List of artifact types used in UC8\_RGB

Name	Description
Assurance Standard (Information)	Document with best practices to follow and criteria to meet for system assurance
Change Impact Analysis Report (Information)	Report with information about the effect that the change of a given system artifact has had or could have on other system artifacts, and how such an effect could propagate
Compliance Report (Information)	Report generated with the information of fulfilment of the system under analysis and any gap to be bridged according to the reference standard.
Device Development Model (Active Unit)	Model that describes the development activities related to the device under analysis
ExpOutputs (Information)	Time sequence of Infusion settings (determined by Anaesthesia Controller) Time sequence of Infusion outputs (by Infusion pump, with eventual injected errors)



Name	Description
	Time sequence of Patient's body responses (by Patient Model) - concentration of drug, estimated NMB in TOF/PTC, estimated time for total recovery) Time sequence of NMB Sensor measurements Logs from Anaesthesia Controller
ExpPatientAttrs (Information)	particular weight [kg] particular PK/PD parameters (generated from PatientAttributesRange)
Failure Mode and Error Propagation Model (Information)	System model extended with the annotations and decorations required to describe the fault behaviours of the functional blocks.
FMEA Table (Information)	FMEA (Failure Mode and Effect Analysis) tables record failure modes, causes, and resulting effects on the system, for each system's component.
FTA Diagram (Information)	Fault Trees are logic block diagrams that display the state of a system (top event) in terms of the states of its components (basic events). It uses a graphic "model" of the pathways within a system that can lead to a foreseeable, undesirable loss event (or a failure). The pathways connect contributory events and conditions, using standard logic symbols (AND, OR, etc.).
Functional Model (Information)	The system functional model describes all the system functional blocks, relevant in the context of the fault generation and propagation analysis, and their relationships.
Metrics Configuration (Information)	Selection of the metrics to use for a specific system quality analysis, including measurement procedures and quality levels
Ontology (Information)	Knowledge base in the form of an ontology with terms, semantic information and system specification patterns, among other elements, which can be exploited for quality analysis and traceability management of system artifacts
Quality Analysis Report (Information)	Report generated with the information about how good a system artifact is according to a metrics configuration and considering a given ontology
Refined Safety Requirements (Information)	System safety requirements refined after the Hazard and Risk Analysis.
Standard Model (Active Unit)	Model related to the reference standard that will be used to perform the compliance and gap analysis
System Architecture (Information)	Original system architecture defined by system engineers.
System Artifacts (Information)	Artifacts management during a system's lifecycle, such as requirements specifications and design diagrams

Name	Description
System Requirements (Information)	Original system requirements definition available from requirement engineering.
TCGeneralDescription (Information)	AnaesthesiaPlan - time sequence of target NMT (Neuromuscular Transmission) in TOF/PTC units planned for a particular surgery (target NMT for certain period, generally: [(NMT target level, timePeriod), ...] Number of experiments (randomly generated out of the specified ranges, then statistically processed) ExperimentConfiguration: Name of Neuromuscular blocking agent (drug: Rocuronium, CisAtracurium, ...) Allowed deviation from the target NMT (%) Flag (Enable/Disable) - Error injections into infusion pumps Flag - Error injection into NMB sensor (cuff) Anaesthesia strategy (selection of Control algorithm) PatientAttributesRange: weight range <from, to> [kg] Pharmacokinetic/Pharmacodynamic parameters (range): Muscles-to-weight ratio <from, to> Unit Volume of Distribution <from, to> EC50 (sensitivity) <from, to>
TCSummary (Information)	number of executed experiments number of experiments where NMB was out of target range number of minutes when NMB was out of target range ... other
Traceability Project Map (Information)	Definition of the system artifact sources and the parameters to consider for a given traceability management effort
Traceability Report (Information)	Report generated with information about the relationships between artifacts managed during a system's lifecycle
Traces Between System Artefacts (Information)	Set of relationship between system artifacts specified for a given traceability management effort

### 3.8.2 V&V Workflows of Tailored Model-Based Assurance and Certification

TMAC method involves the specification of meta-models for representing NMT devices and related safety standards to define specific models for devices, development processes and quality criteria, which can be automatically analysed to obtain compliance levels to reference quality models or certification standards.

Figure 3.39 shows the workflow specification diagram of Tailored Model-Based Assurance and Certification.

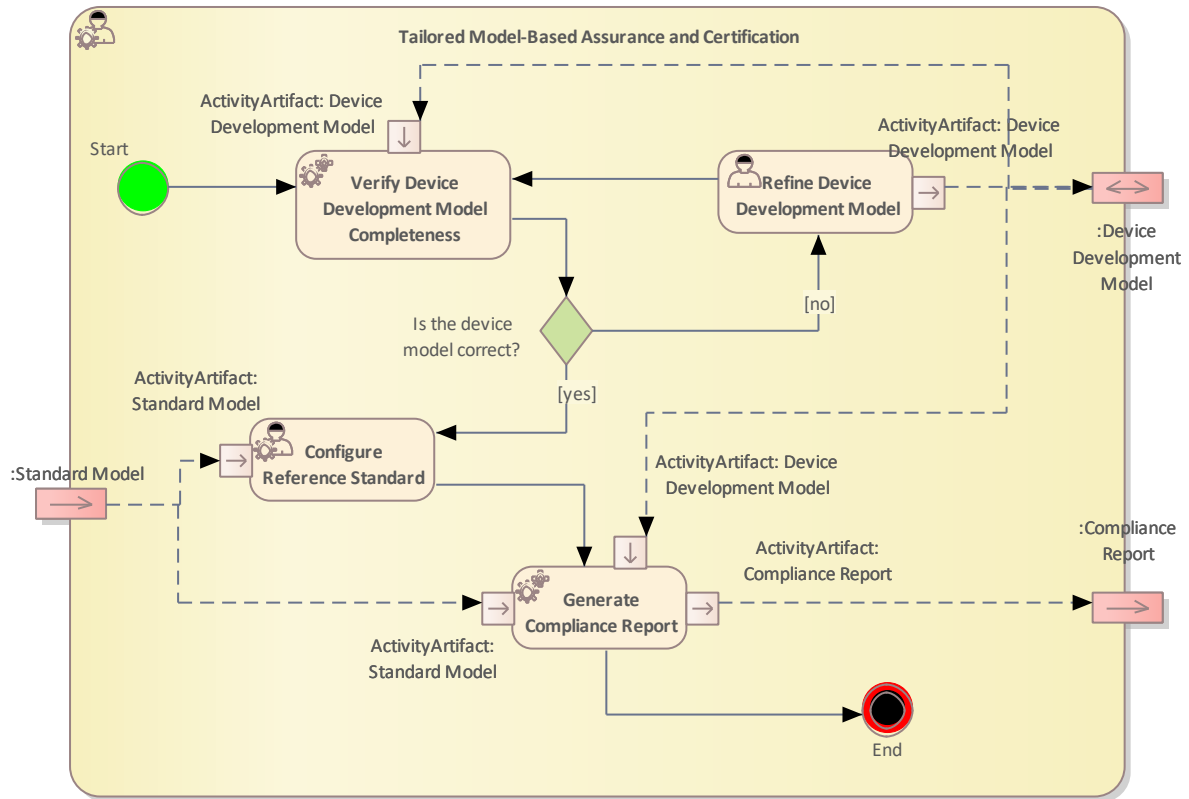


Figure 3.39 Model-Based Assurance and Certification used in UC8\_RGB

Table 3.24 lists the activities of the workflow Tailored Model-Based Assurance and Certification.

Table 3.24 List of activities performed by Tailored Model-Based Assurance and Certification

Name	Type	Description
Configure Reference Standard	Semi-automated	This task is oriented to indicate the reference standard and provide the necessary configuration according to the system to be verified.
Generate Compliance Report	Automated	The tailored method and the certification models defined are automatically analyzed to evaluate the level of compliance to the reference standards.
Refine Device Development Model	Manual	This task is manually performed to refine or fix the input device model when the verification task indicates some error.
Verify Device Development Model Completeness	Automated	This task verifies the device models defined in terms of their completeness.

### 3.8.3 V&V Workflows of Model Based Safety Analysis FLA

The workflow describes the application of the V&V method MSA - FLA (Model-based Safety Analysis with Failure Logical Analysis) to the Use Case with the aim of analysing the failure propagation phenomena and evaluating their consequences in terms of safety and reliability, based on a formal model of the system of interest, automatically generating Fault Trees and FMEA (Failure Mode and Effect Analysis) tables.

The workflow has just two inputs: System Requirements and System Architecture. Starting from these inputs, the main activities included in the workflow of the method are:

- **Design of the System Functional Model:** the system model should describe the system functional blocks, relevant in the context of the fault generation and propagation analysis, and their relationships;
- **Extended Design with Failure Behaviour:** the system model should be extended with the annotations and decorations required to describe the fault behaviours of the functional blocks. This model is called Failure Mode and Error Propagation Model;
- **Automatic FMEA Generation + Complete FMEA fields:** FMEA tables are automatically generated from the Failure Mode and Error Propagation Model. Furthermore, they can be manually completed by the safety experts
- **Automated FTA Generation:** Fault Trees are automatically generated from the Failure Mode and Error Propagation Model.
- **Hazard and Risk Analysis:** A Hazard and Risk Analysis is performed, starting from the FMEA table and the FTs. As a result, refined safety requirements may be provided.

These activities are iteratively performed until the Hazard and Risk Analysis results are acceptable, according to the related standards.

The workflow outputs are:

- Failure Mode and Error Propagation Model;
- FTA Diagram;
- FMEA Table;
- Refined Safety Requirements.

Figure 3.40 shows the workflow specification diagram of Model Based Safety Analysis FLA.

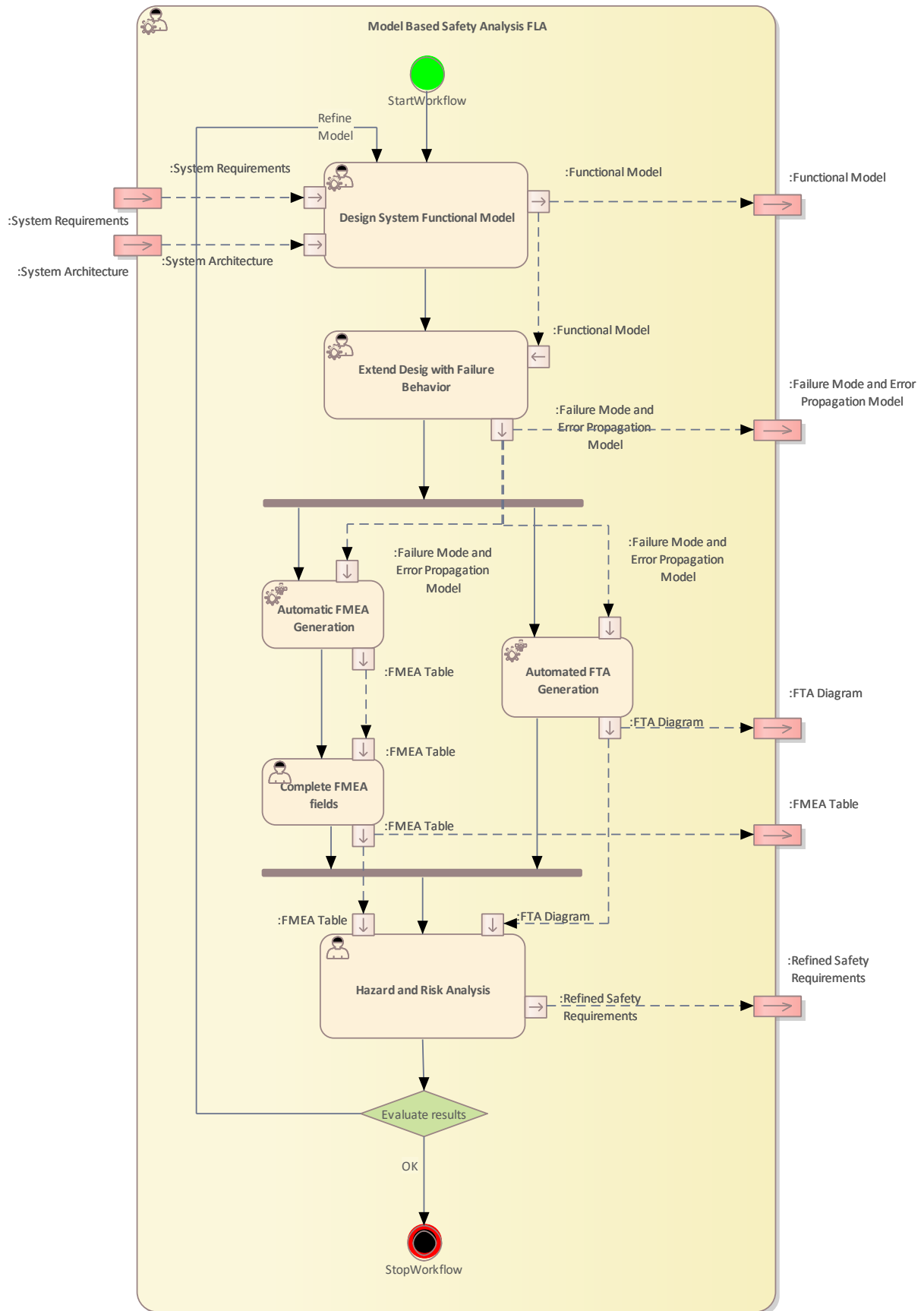


Figure 3.40 Workflow Definition diagram of Model Based Safety Analysis FLA used in UC8\_RGB

Table 3.25 lists the activities of the workflow Model Based Safety Analysis FLA.

*Table 3.25 List of activities performed by Model Based Safety Analysis FLA*

Name	Type	Description
Automated FTA Generation	Automated	The CHES-FLA plugin automatically generates the Fault Trees from the Failure Mode and Error Propagation Model.
Automatic FMEA Generation	Automated	The CHES-FLA plugin automatically generates the FMEA table from the Failure Mode and Error Propagation Model.
Complete FMEA fields	Manual	The FMEA table can be manually completed by the safety experts who want to add additional information.
Design System Functional Model	Semi-automated	Design of the system functional model. This functional model should describe the system functional blocks, relevant in the context of the fault generation and propagation analysis, and their relationships.
Extend Design with Failure Behaviour	Semi-automated	The system functional model is extended with the annotations and decorations required to describe the fault behaviours of the functional blocks. In particular, the FLA (Failure Logical Analysis) rules are used. The extended model is called Failure Mode and Error Propagation Model.
Hazard and Risk Analysis	Manual	Starting from the FMEA table and the FTs, an Hazard and Risk Analysis is performed. As a result of this action, refined safety requirements may be provided. If the results of this results of the Hazard and Risk Analysis are not acceptable, all the actions are repeated again.

### 3.8.4 V&V Workflows of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis

Method to quantitatively determine the suitability of system artifacts in different formats by exploiting ontologies and semantic information, according to selected criteria, e.g., correctness, consistency, and completeness, and considering specific compliance needs from assurance standards.

Figure 3.41 shows the workflow specification diagram of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis.

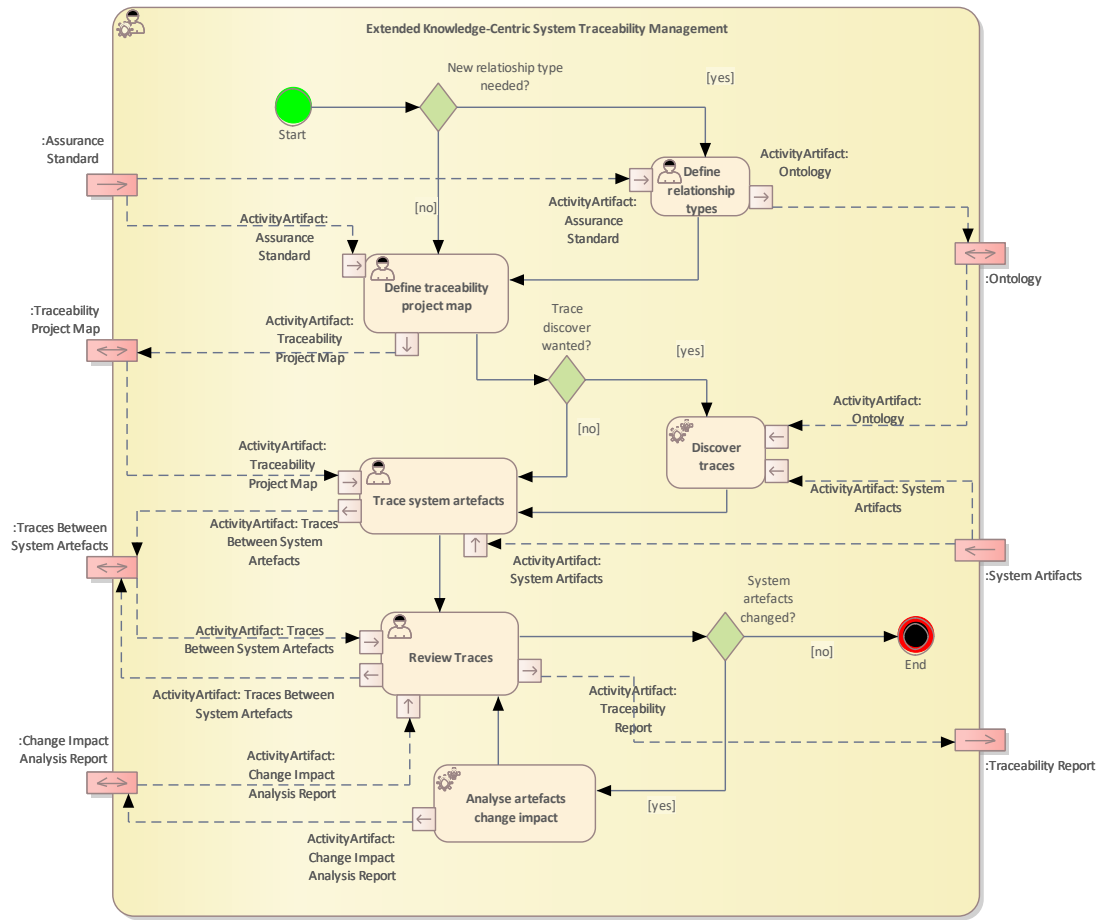


Figure 3.41 Workflow Definition diagram of Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis used in UC8\_RGB

Table 3.26 lists the activities of the workflow Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis.

Table 3.26 List of activities performed by Compliance-Aware Extended Knowledge-Centric System Artefact Quality Analysis

Name	Type	Description
Define metrics	Manual	Specification of the characteristics (measurement procedure, quality levels...) of new metrics to use for system artifact quality analysis
Define system artefacts import mechanism	Manual	Determination of the means to import system artifact data for quality analysis, and of the means will be used
Run quality analysis	Automated	Execution of a system artifact quality analysis according to a given metrics configuration and to the content of a given ontology
Select metrics	Manual	Determination of the set of metrics to use for a given system artifact quality analysis, considering the requirements from applicable assurance standards

Name	Type	Description
Update system artefacts	Manual	Revision of system artifacts in case quality issues are identified, to address the issues

### 3.8.5 V&V Workflows of Extended Knowledge-Centric System Traceability Management

Method to manage the relationships between system artifact by taking advantage of ontologies and semantic information, further supporting advanced traceability project configuration and automating trace discovery and verification.

Figure 3.42 shows the workflow specification diagram of Extended Knowledge-Centric System Traceability Management.

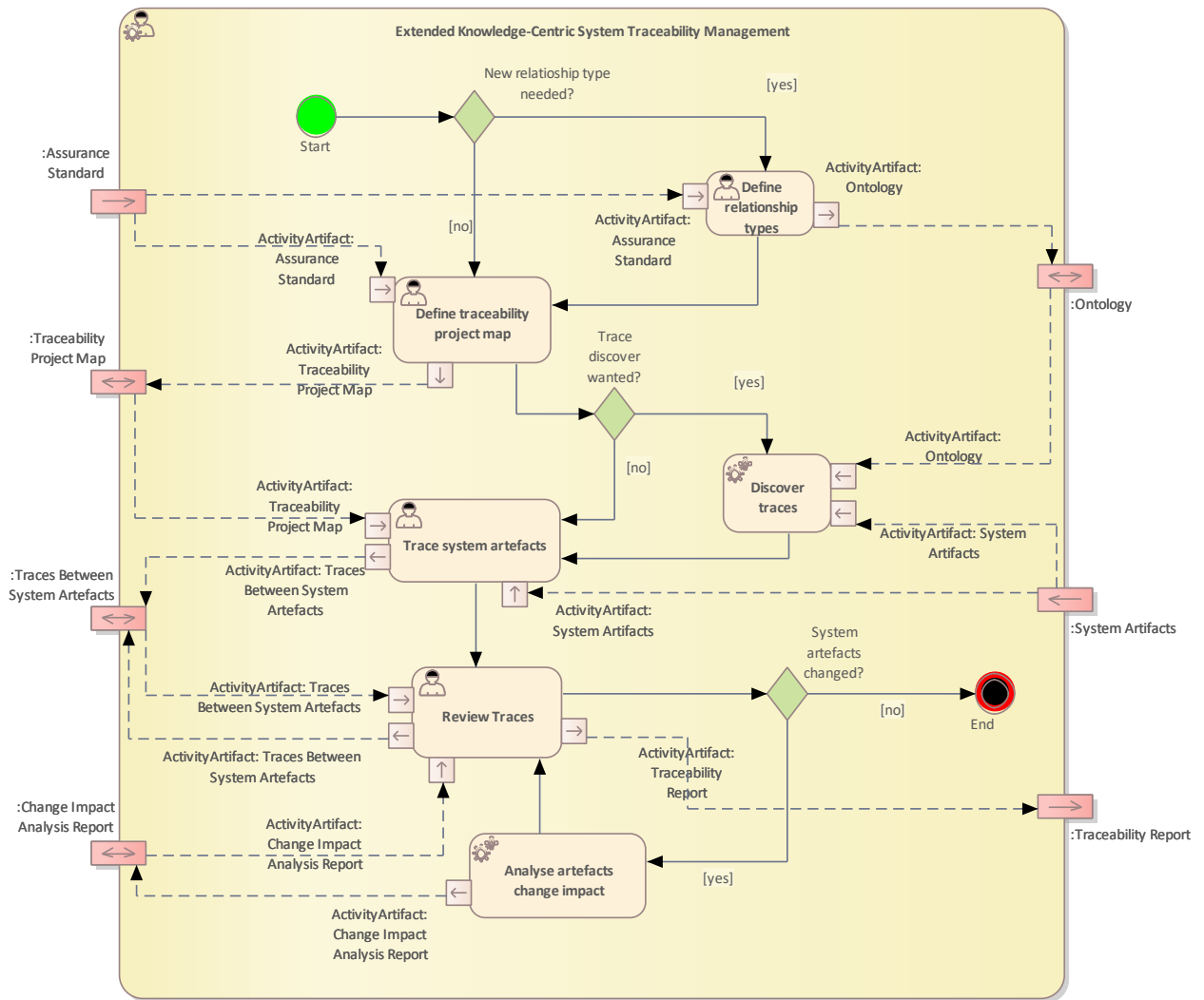


Figure 3.42 Workflow Definition diagram of Extended Knowledge-Centric System Traceability Management used in UC8\_RGB



Table 3.27 lists the activities of the workflow Extended Knowledge-Centric System Traceability Management.

*Table 3.27 List of activities performed by Extended Knowledge-Centric System Traceability Management*

Name	Type	Description
Analyse artefacts change impact	Automated	Determination of the effect that the changes of system artifacts have or might have
Define relationship types	Manual	Characterization of new types of relationships between artifacts to consider for a given traceability management effort
Define traceability project map	Manual	Specification of how a traceability project will be, considering aspects such as the system artifact sources to consider and the parameters to use in traceability tasks (trace discovery, specification, verification...)
Discover traces	Automated	Automatic determination of possible relationships between
Review Traces	Manual	Trace check to confirm that the traces specified are valid
Trace system artefacts	Manual	Specification of relationships between system artifacts

### 3.8.6 V&V Workflows of Single Experiment

This workflow is fully automatic and manages the simulation with the Patient Model and Anaesthesia Controller (either simulated or physical). Regarding the experiment definition, the method injects random errors into infusion and measurement entities.

Figure 3.43 shows the workflow specification diagram of Single Experiment.

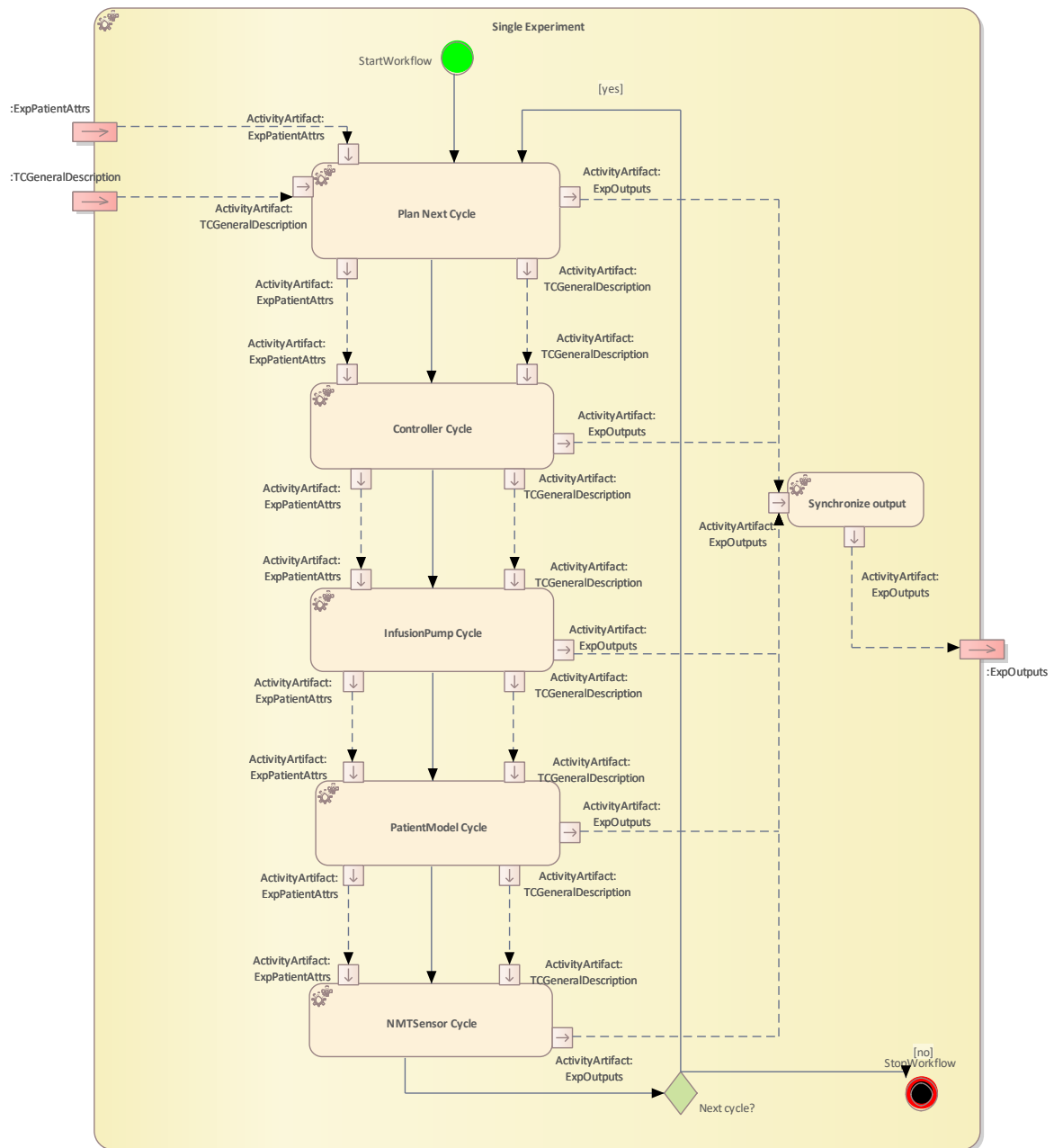


Figure 3.43 Workflow Definition diagram of Single Experiment used in UC8\_RGB

Table 3.28 lists the activities of the workflow Single Experiment.

Table 3.28 List of activities performed by Single Experiment

Name	Type	Description
Synchronize output	Automated	This activity synchronizes the outputs of the different task in the workflow and generates the final result.
Controller Cycle	Automated	This activity analyses the NMT value from the NMT sensor and calculates the next dose for the infusion pump.

Name	Type	Description
InfusionPump Cycle	Automated	This activity delivers the indicated amount of drug to the patient.
NMTSensor Cycle	Automated	This activity calculates the NMT value of the patient.
PatientModel Cycle	Automated	This activity computes the relaxation status depending on the drug received by the infusion pump.
Plan Next Cycle	Automated	This activity selects and starts next plan when actual plan finishes.

### 3.8.7 V&V Workflows of TC Automated Experimenting

Generates simulation experiments, executes them, and statistically processes them. The experiments are generated randomly from predefined ranges of input attributes. Random generation of experiments must cover the whole space of patient’s weight and sensitivity to the drug, anaesthesia strategies, etc. within scenarios of eventual application of anaesthesia.

Figure 3.44 shows the workflow specification diagram of TC Automated Experimenting.

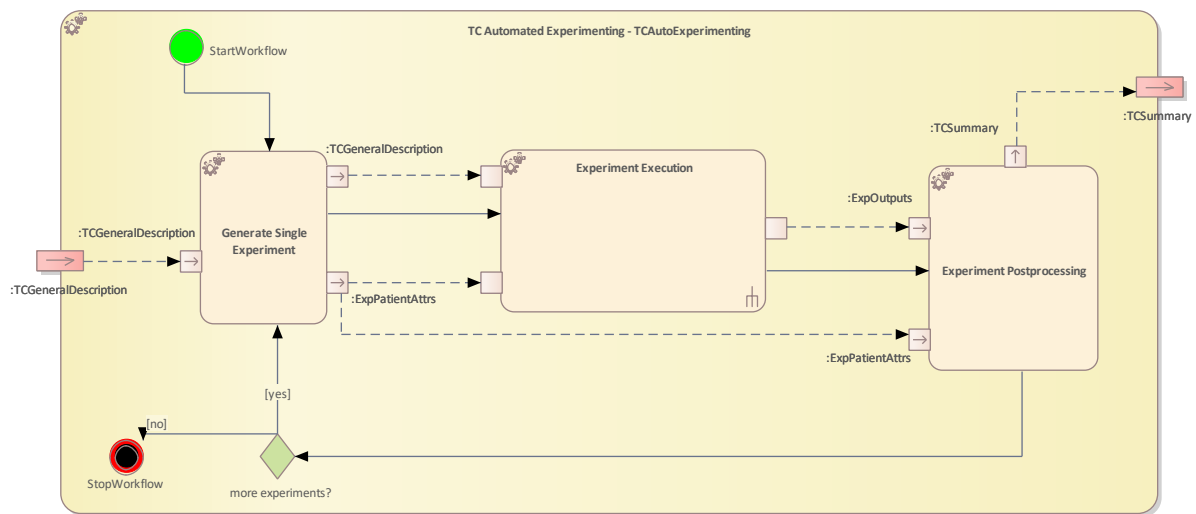


Figure 3.44 Workflow Definition diagram of TC Auto Experiment used in UC8\_RGB

Table 3.29 lists the activities of the workflow TC Automated Experimenting.

Table 3.29 List of activities performed by TC Automated Experimenting

Name	Type	Description
Experiment (CallBehavior)	Execution: Semi-automated	Execute experiments group

Name	Type	Description
Experiment Postprocessing	Automated	This activity accumulates the generated outputs (ExpOutputs) and statistically process them into an aggregated output (TCSummary)
Generate Single Experiment	Automated	Generate outputs (ExpOutputs)

### 3.8.8 V&V Workflows of TC Management

Test Case Management Methods need manual preparation of the test case general description, which is mainly defined by the intended relaxant drug and ranges of experimenting. TC Management generates simulation experiments, executes them, and statistically processes them. The experiments are generated randomly from predefined ranges of input attributes. Random generation of experiments must cover the whole space of patient’s weight and sensitivity to the drug, anaesthesia strategies, etc. within scenarios of eventual application of anaesthesia.

Figure 3.45 shows the workflow specification diagram of TC Management.

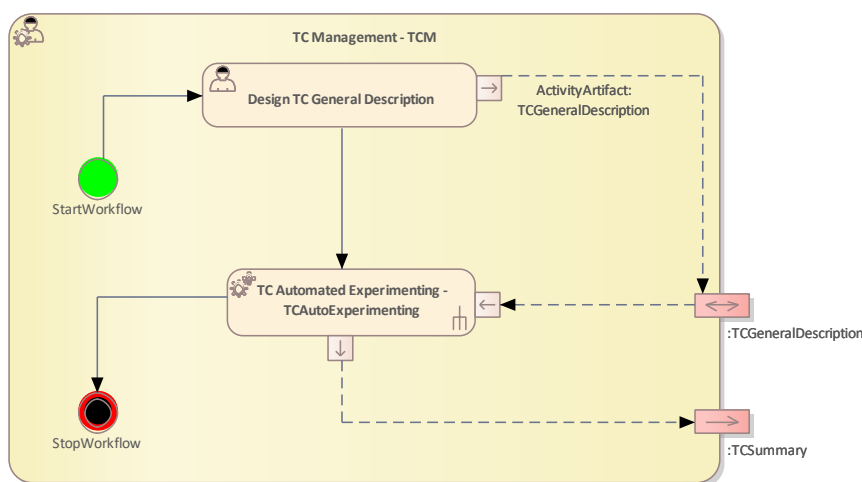


Figure 3.45 Workflow Definition diagram of TC Management used in UC8\_RGB

Table 3.30 lists the activities of the workflow TC Management.

Table 3.30 List of activities performed by TC Management

Name	Type	Description
TC Experimenting (CallBehavior)	Automated	Automated execution of test cases that include the intended relaxant drug and ranges of experimenting

Name			Type	Description
Design Description	TC	General	Manual	manual preparation of the test case, which is mainly defined by the intended relaxant drug and ranges of experimenting

### 3.9 V&V Workflow of Use Case 9 CAF

UC9\_CAF\_Submethods package contains the following workflows:

- Overall UC9 Method
- Simulation based V&V of Computer Vision System

Figure 3.46 shows the Overall UC9 workflow Method Definition diagram type of the V&V workflow UC9\_CAF\_Submethods.

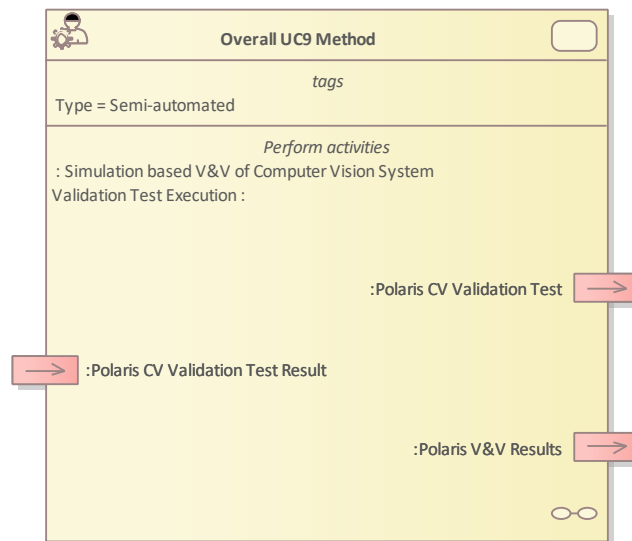


Figure 3.46 Method Definition of Overall UC9 workflow defined for UC9\_CAF

Figure 3.47 shows the Simulation based V&V of Computer Vision System Method Definition diagram type of the V&V workflow UC9\_CAF\_Submethods.

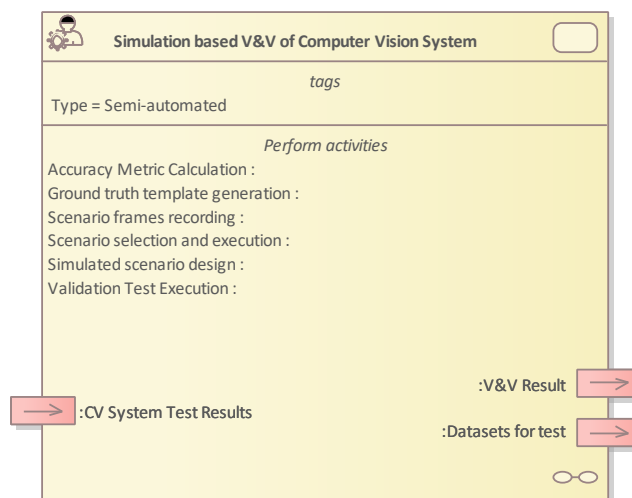


Figure 3.47 Method Definition of Simulation based V&V of Computer Vision System defined for UC9\_CAFs

Details on the workflows are given in the following subsections.

### 3.9.1 Artifacts used in UC9\_CAF

Table 3.31 lists the artifacts used for the workflow(s) defined for UC9\_CAF.

*Table 3.31 List of artifact types used in UC9\_CAF*

Name	Description
CV System Test Results (Information)	This will be the results obtained by the Polaris model, "Polaris CV Validation Test".
Datasets for test (Information)	Dataset generated with the tools DaGe4v and Train Simulator for the computer vision for testing.
Polaris CV Validation Test (Information)	Results generated by the model Polaris.
Polaris CV Validation Test Result (Information)	This will be the input for the computer vision model which is made by "Simulation based V&V of Computer Vision System" using DaGe4v and Train Simulator. The input will be the "Datasets for test" of "Simulation based V&V of Computer Vision System".
Polaris V&V Results (Information)	The results obtained from the "Simulation based V&V of Computer Vision System" method.
V&V Result (Information)	The results obtained by analysing the dataset created with the results obtained by Polaris model which are the input of the method, "CV System Test Results".

### 3.9.2 V&V Workflows of Overall UC9 Method

The overall use case 9 method defines the workflow for semi-automatically validate a computer vision system using simulated validation datasets. In particular, the use case is focused on validating CAF's CV system (Polaris), which is trained to detect traffic lights and speed signs in the railway domain. For this purpose, the workflow automates the use of the combined method defined by IKER, "Simulation based V&V of Computer Vision System". It contains a single activity that is responsible for obtaining a list of validation datasets from the combined method process and it executes automatically one after each other on the Polaris CV System. After obtaining the execution results from the system under test, it provides to the combined method to finally get the V&V results that will be evaluated by an engineer.

Figure 3.48 shows the workflow specification diagram of Overall UC9 Method.

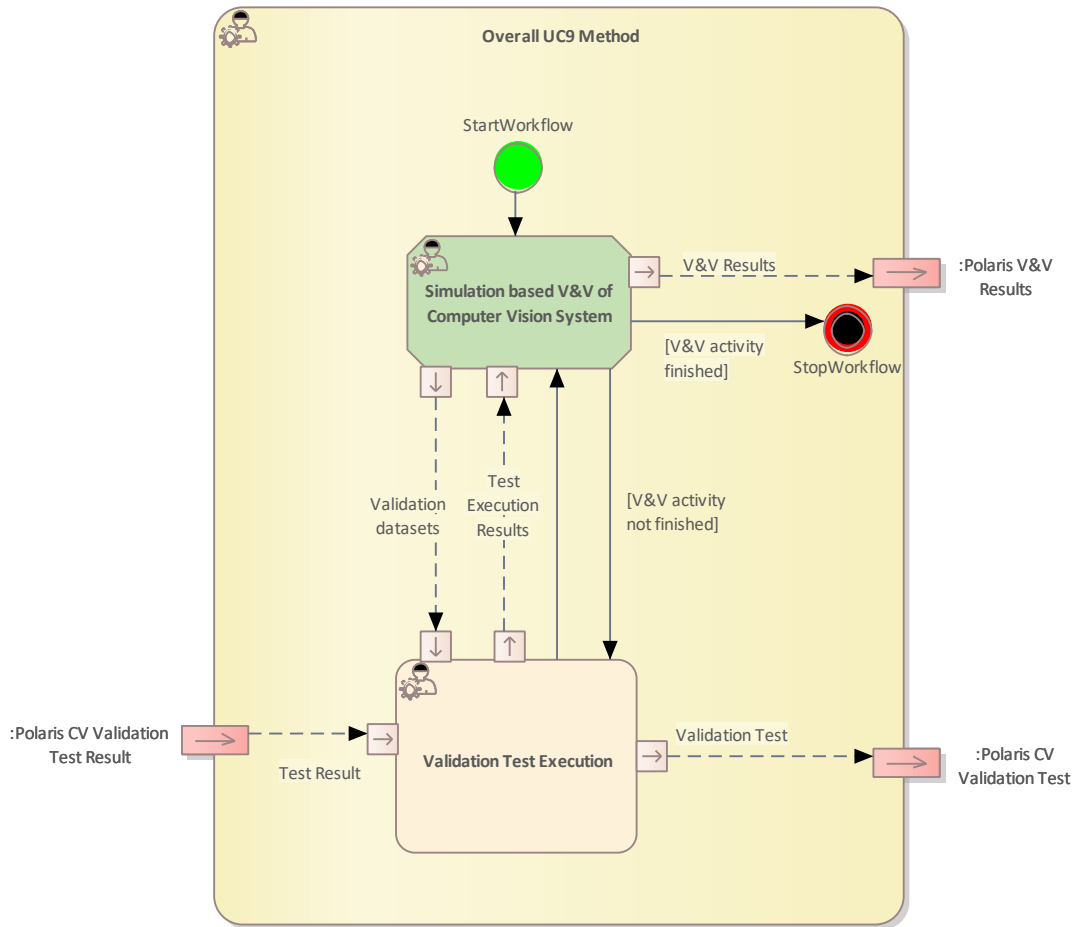


Figure 3.48 Workflow Definition diagram of UC9\_VV\_Method used in UC9\_CA

Table 3.32 lists the activities of the workflow Overall UC9 Method.

Table 3.32 List of activities performed by Overall UC9 Method

Name	Type	Description
Activity: Simulation based V&V of Computer Vision System (CallBehavior)	Semi-automated	The “Simulation based V&V of Computer Vision System” method is a combined method that enables the validation of a CV system focused on objects detection in a semi-automated way. It is based on synthetic images generated using a simulator
Validation Test Execution	Semi-automated	Validation Test Execution receives the synthetic images generated using a simulator and makes the predictions to pass it to the Simulation based V&V of Computer Vision System.



### 3.9.3 V&V Workflows of Simulation-Based V&V of Computer Vision System

The “Simulation based V&V of Computer Vision System” method is a combined method that enables the validation of a CV system focused on objects detection in a semi-automated way. It is based on synthetic images generated using a simulator. The activities that compose the workflow of the method are described below:

- **Simulated scenario design:** The first activity comprises the design of the scenarios using the simulator. At this point, by means of a simulation environment the objects that CV system should detect are placed in different scenarios.
- **Scenario selection and execution:** This activity will comprise the selection of a previously designed scenario that will be executed in the simulator. Setting the configuration parameters will enable to carry out a simulation in different conditions.
- **Scenario frames recording:** During a simulation execution, the Dataset Generator for Validation (DaGe4V) tool records the frames and related metadata from the simulator. As a result, the validation datasets for testing the system under test are generated.
- **Ground truth template generation:** This activity comprises semi-automatically labelling the validation datasets, using external tools, such as DarkLabel, to get the ground truth information with accurate information on objects location in each frame.
- **Accuracy Metric Calculation:** In this activity, Validation Test Result Analyser (VaTRA) tool will analyse the results obtained in several tests carried out for the CV system. It will compare the results get on each test with the ground truth template, providing several metrics and identifying potential safety violations due to incorrect object detection during the tests, generating the V&V Results artifact.

Figure 3.49 shows the workflow specification diagram of Simulation based V&V of Computer Vision System.

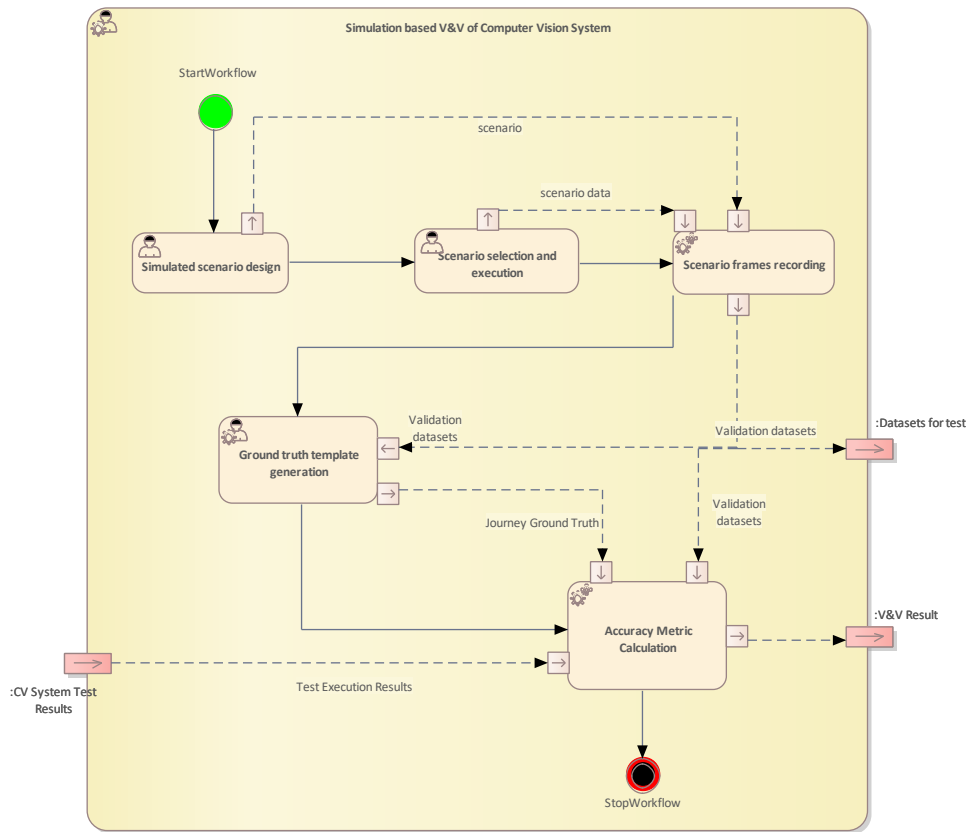


Figure 3.49 Workflow Definition diagram of Simulation based V&V of Computer Vision used in UC9\_CAF\_Submethods

Table 3.33 lists the activities of the workflow Simulation based V&V of Computer Vision System.

Table 3.33 List of activities performed by Simulation based V&V of Computer Vision System

Name	Type	Description
Simulated scenario design	Manual	The first activity comprises the design of the scenarios using the simulator. At this point, by means of a simulation environment the objects that CV system should detect are placed in different scenarios.
Scenario selection and execution	Manual	This activity will comprise the selection of a previously designed scenario that will be executed in the simulator. Setting the configuration parameters will enable to carry out a simulation in different conditions.
Scenario frames recording	Automated	During a simulation execution, the Dataset Generator for Validation (DaGe4V) tool records the frames and related metadata from the simulator. As a result, the validation datasets for testing the system under test are generated.
Ground truth template generation	Semi-automated	This activity comprises semi-automatically labelling the validation datasets, using external tools, such as DarkLabel,

Name	Type	Description
		to get the ground truth information with accurate information on objects location in each frame.
Accuracy Metric Calculation	Automated	In this activity, Validation Test Result Analyser (VaTRA) tool will analyse the results obtained in several tests carried out for the CV system. It will compare the results get on each test with the ground truth template, providing several metrics and identifying potential safety violations due to incorrect object detection during the tests, generating the V&V Results artifact.

### 3.10 V&V Workflow of Use Case 10 BT

UC10\_BT package contains the following workflows:

- UC10 Overall Method
- Model Checking Families of Real-Time Specifications
- Optimize Fault Injection Experiments Using Model-Based Mutation Testing
- Behaviour-driven model development and test-driven model review

Figure 3.50 shows the UC10 Overall Method Definition diagram type of the V&V workflow UC10\_BT.

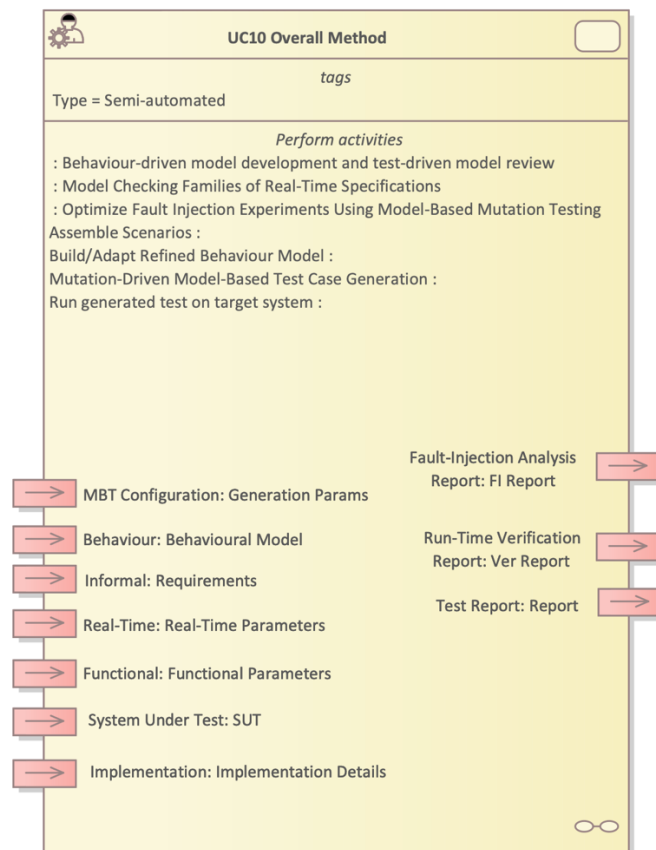


Figure 3.50 Method Definition of UC10 Overall Method defined for UC10\_BT

Figure 3.51 shows the Model Checking Families of Real Time Systems Method Definition diagram type of the V&V workflow UC10\_BT.

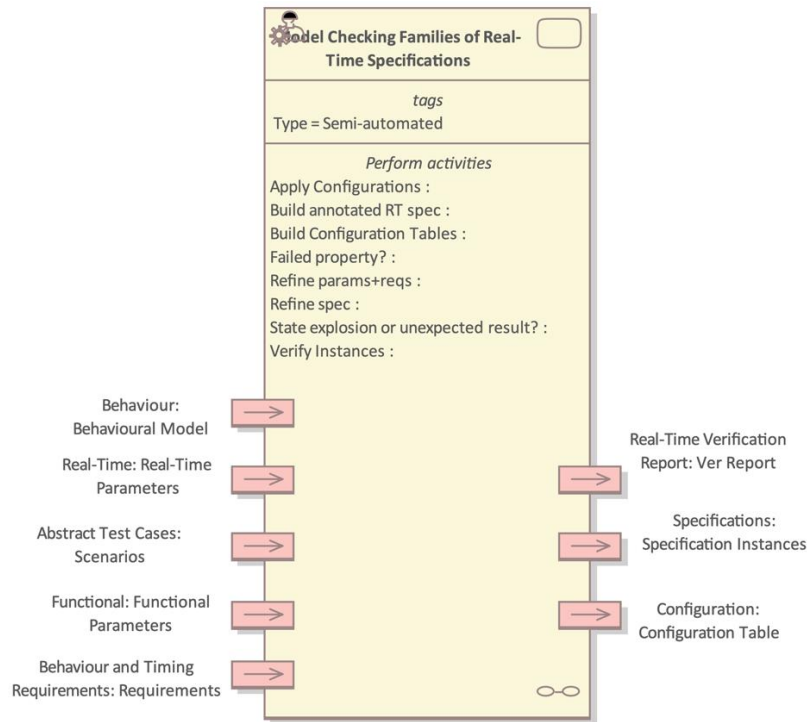


Figure 3.51 Method Definition of Model Checking Families of Real Time Systems - Method defined for UC10\_BT

Figure 3.52 shows the Optimize Fault Injection Experiments Using Model-Based Mutation Testing Method Definition diagram type of the V&V workflow UC10\_BT.

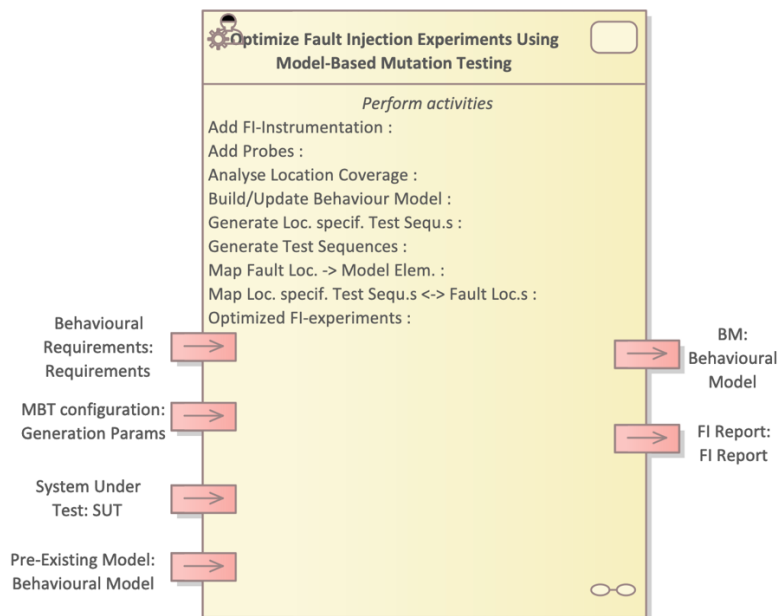


Figure 3.52 Method Definition of Optimize Fault Injection Experiments Using Model-Based Mutation Testing defined for UC10\_BT

Figure 3.53 shows the Behaviour-driven model development and test-driven model review Method Definition diagram type of the V&V workflow UC10\_BT.

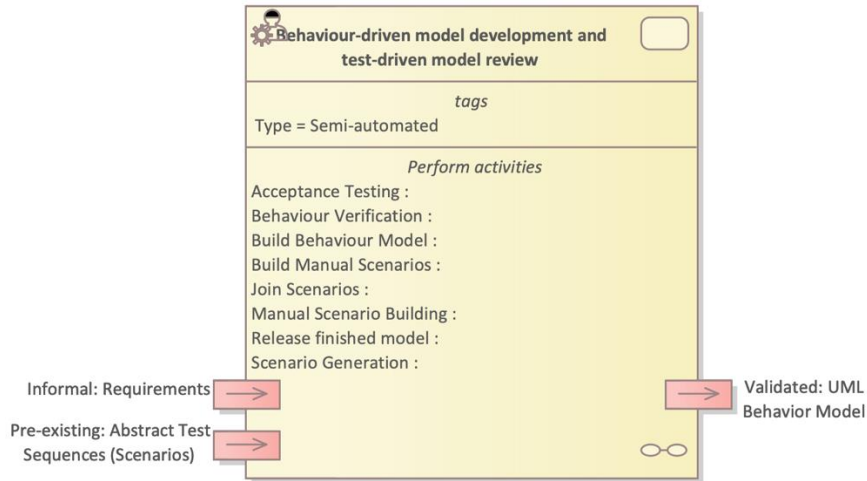


Figure 3.53 Behaviour-driven model development and test-driven model review

Details on the workflows are given in the following subsections.

### 3.10.1 Artifacts used in UC10\_BT

Table 3.34 lists the artifacts used for the workflows defined for UC10\_BT.

Table 3.34 List of artifact types used in UC10\_BT

Name	Description
Abstract Test Sequences (Scenarios) (Information)	Set of possible sequences of events that describe tests over the behaviour of a system.
Annotated RT Spec (Active Unit)	Formal specification of a real-time specification, annotated with placeholders that can be modified to produce variations of the specification.
Behavioural Model (Information)	Informal description of the behaviour of a specific case-study.
Configuration Table (Information)	Compilation of configuration values, including functional parameters, real-time parameters, requirements, and scenarios.
FI Report (Information)	Report resulting from running the analysis of the system-under-test after injecting faults.
Functional Parameters (Information)	Concrete values that will influence the generation of concrete instances, focused on functional behaviour.
Generation Params (Information)	Generation parameters are the concrete set of values that will be used to configure an automated activity that produces test sequences for a given behaviour model.
Real-Time Parameters (Information)	Combinations of temporal restrictions of the behaviour of the use-case.
Requirements (Information)	Requirements is an informal description (e.g., text, spreadsheets) of desirable properties that should be considered

Name	Description
	during the design and verification of a system. These can be safety or liveness properties, depending on the particular system being analysed.
Scenarios (Information)	Description of context information, including a sequence of expected input messages.
Specification Instances (Active Unit)	Set of concrete instances of specifications of a system built from a more generic and parameterized specification.
SUT (Active Unit)	System Under Test (SUT) is the concrete implementation that will be analysed and verified.
UML Behaviour Model (Information)	A UML behavioural model of the system. This allows us to generate the tests.
Ver Report (Information)	Report regarding the formal verification analysis.

### 3.10.2 V&V Workflows of UC10 Overall Method

UC10 is a DC motor controller in the railway domain targeted for a Tolerable Hazard Rate (THR) of  $5 \cdot 10^{-12}$  according to the railway safety standard EN50129 [14]. In this use case, the platforms are different hardware configurations of COTS functional safety SoC. In VALU3S, we investigate and validate the safety functions implemented on different platforms and try to move the safety functions between environments and assess the compliance with the railway standards. With this approach, we explore the possibility to reduce the time and cost of functional safety product development in the railway system. Moreover, the possibility to increase the system availability is explored.

In UC10, three V&V methods will be applied/used:

1. For functional testing, Model-Based Mutation Testing is applied.
2. The combined method Scenario Generation and Validation of Real-Time Systems is specialised here, using Model-Based Mutation Testing as a specialisation of its part method Model-Based Testing to provide scenarios for the validation of real-time properties of the system.
3. Fault injection experiments are done. For optimizing the experiments, the combined method Optimize Fault Injection Experiments Using Model-Based Mutation Testing is used.

The UML behavior models used in all three methods will be developed using Behaviour-driven model development and test-driven model review – one possibility to optimize building behaviour models for analysis and testing. This diagram depicts how the methods interact and how the workflow for Scenario Generation and Validation of Real-Time Systems is adapted to use Model-Based Mutation Testing.

Tools that will be used are:

- MoMuT's integration into Enterprise Architect for Behaviour-driven model development and test-driven model review MoMuT::UML: for Model-Based Mutation Testing [15]

- UPPAAL [16] for Model Checking Families of Real-Time Systems
- Xilinx Vivado Design Suite [17] and the Healing Core Feature for Fault Injection into FPGAs.

The tools are not integrated at the moment – it is not clear yet how far integration on the tool side will go, apart from exchanging V&V artefacts.

The actual fault injection experiments can be done in software simulation or on the actual hardware prototype, deployed into the FPGA. Both approaches derive an intermediate input from a system description that can be shared/reused.

Figure 3.54 shows the workflow specification diagram of UC10 Overall Method.

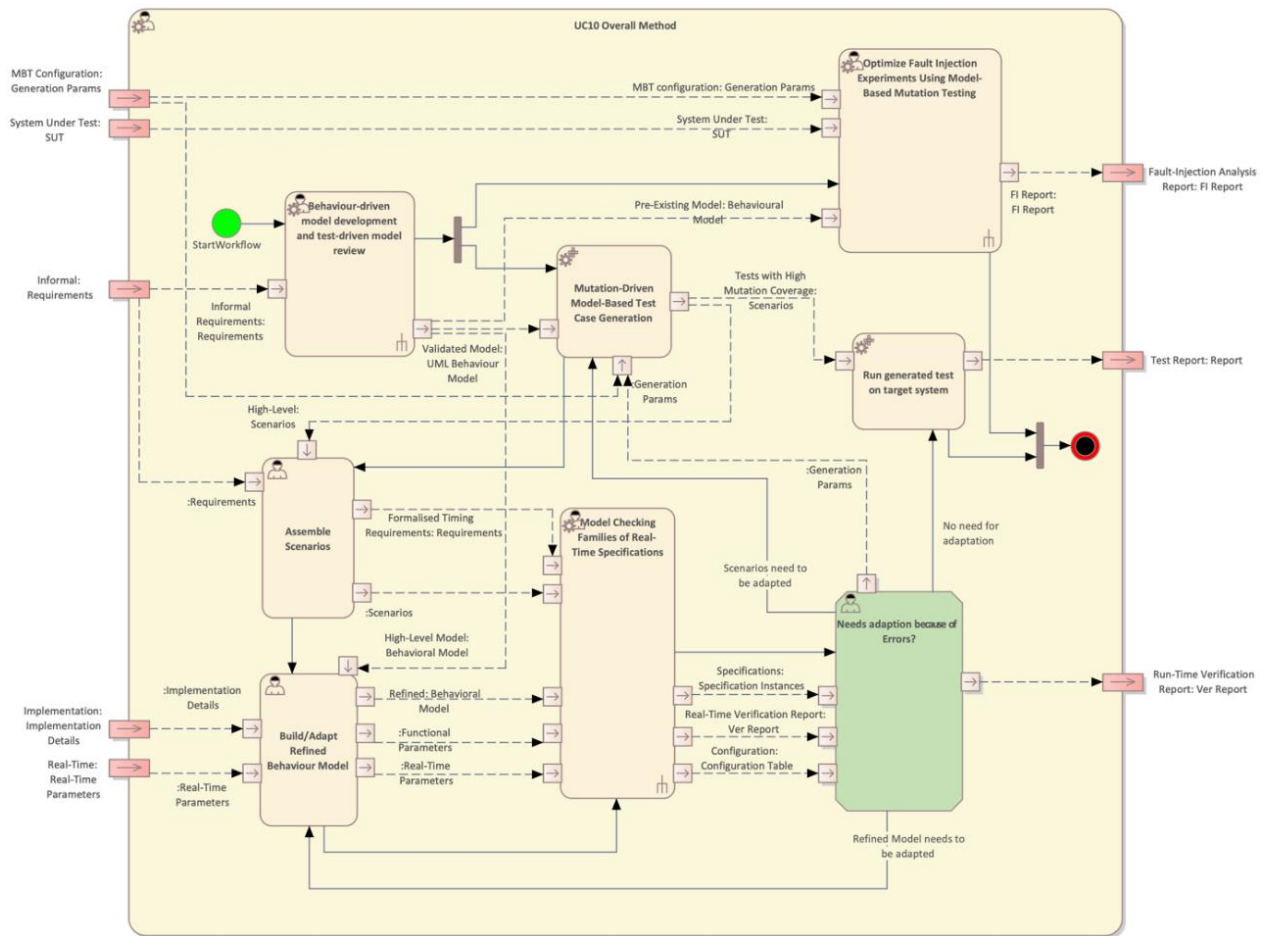


Figure 3.54 Workflow Definition diagram of UC10 Overall Method Workflow used in UC10\_BT

Table 3.35 lists the activities of the workflow UC10 Overall Method.

Table 3.35 List of activities performed by UC10 Overall Method

Name	Type	Description
Activity: Behaviour-driven model development and test-	Semi-automated	Build scenarios from informal requirements. The scenarios can be for example: process control by of sensor data, server



Name	Type	Description
driven model review (CallBehavior)		and PLC communication, anomaly detection at component and system level. The output are abstract test sequences.
Activity: Model Checking Families of Real-Time Specifications (CallBehavior)	Semi-automated	This method uses a model checker to statically verify a set of requirements with real-time aspects. It uses an annotated description of the formal model and a table of possible parameters, and effectively checks each requirement against a set of sensible variations of the system model. Some of these parameters capture the possible scenarios produced by <generation-activity>. This exploration of variants is realized by a new in-house tool named Uppex.
Activity: Optimize Fault Injection Experiments Using Model-Based Mutation Testing (CallBehavior)	Semi-automated	Fault-based testing plays an important role in the verification and validation of systems, as it is able to demonstrate the absence of certain faults. Model-based mutation testing (MBMT) is a particular instance of fault-based testing. MBMT is working in a black box setting, which means that it does not look at the implementation during test case generation. Instead, it is working off a model of the system under test (SUT) that is directly derived from the requirements and usually is more abstract than the implementation. Given a behavioural model of the SUT and a set of generic fault models, i.e., so-called mutation-operators, MBMT strives to automatically generate test cases that can reveal whether any modelled fault has been implemented. To this end, MBMT will take the original model, apply one mutation operator at one particular location a time, deriving a so-called mutant, compare the behaviour of the original model with the one of the mutant, and - once a difference is found - write out a test case that steers the SUT towards this difference.
Assemble Scenarios	Manual	Prepare a given set of high-level scenarios to be provided as input to the model-checking method, using its dedicated input format.
Build/Adapt Refined Behaviour Model	Manual	Prepare or adapt the behaviour model using the specification language used by the model-checking method.
Mutation-Driven Model-Based Test Case Generation	Automated	Generate test cases using a mutation-based technique over the behaviour model.
Needs adaption because of Errors?	Manual	Evaluating all results of the "Model Checking Families ..." activity, it is decided whether the scenarios need further adaption or not.
Run generated test on target system	Automated	Execute the test cases on the concrete implementation of the target system.

### 3.10.3 V&V Workflows of Model Checking Families of Real-Time Specifications

The goal of this method is to guide the model checking process of many variations of similar specifications of Real Time behaviour, using an intermediate set of configuration tables to guide the set of variants.

Model checking is a method to verify if a model of the system under verification satisfies its requirements. In complex models verification becomes infeasible due to the large state space, requiring many small variations of the model, one for each set of related requirements. Typically, these variations are built independently, intersecting efforts, and not guaranteeing that they are kept consistent.

This method explores how to define this of variations and respective requirements based on a single configurable model, leveraging on principles from software product line engineering (SPLE), which are here applied to formal specifications rather than software specifications. It focuses on the UPPAAL real-time model checker, and proceeds in two phases: (1) annotation of the specifications, and (2) automatic configuration of these annotated blocks via a product line of specifications.

#### *Annotated Specifications*

In many cases the relation between the abstract model and the implementation is maintained via personal meetings and reports using natural languages. Automating this synchronisation involves a large effort and is in many cases impractical. Our approach involves using a set of tables in Microsoft Excel to maintain the key parameters of the formal models, including scenarios and requirements of the system. This set of tables is easy to be read and modified by both developers of the system and by developers of the formal models and is automatically entangled with the models used by the formal analysis tools. On one hand, the system developers can update this table and check which requirements can be verified; on the other hand, the designers of the formal models can adapt the model to either include new details, or to relax aspects that introduce state explosions. By using an intermediate representation of the core parameters of the formal specifications, we reduce the expected knowledge of the system developers over formal models. This includes ranges of estimated time executions of individual components, the number of times certain actions may occur, and sequences of input scenarios. Experts in formal modelling are kept in the loop to refine the models and property specifications as needed.

#### *Product Line of Specifications*

A common approach to avoid exploring too many states while trying to verify a property is to bound the state space. Some tools support bounded model checking, limiting the depth of search in the state space. Our approach supports the specification of variants, where the state-space can be reduced by modifying different parameters of the specification. For example, 2 variants could remove 2 independent parts of the specification, allowing the verification of properties for these two variants instead of the full model. Ultimately, the goal is to verify a large-enough set of variants to cover the relevant cases without incurring in state-space explosions.

The method workflow is as follows. The informal behavioural description, different parameters, input scenarios, and requirements are provided as input. These are used to manually build: (1) an initial global and annotated real-time specification, and (2) a collection of Excel tables with information on how to populate the annotated blocks of the specifications and with the set of requirements. Our own tool (Uppex) is then used to automatically apply the configurations in the Excel tables to the annotated specifications, producing a set of specification instances. These are also automatically analysed via UPPAAL, with a provided time-out, and a report is produced over which requirements hold in which configuration. The instances can also be manually inspected within UPPAAL, leading to a manual refinement of the annotated specifications and the Excel tables, until all desired requirements are met for a rich enough set of configurations.

Figure 3.55 shows the workflow specification diagram of Model Checking Families of Real Time Systems.

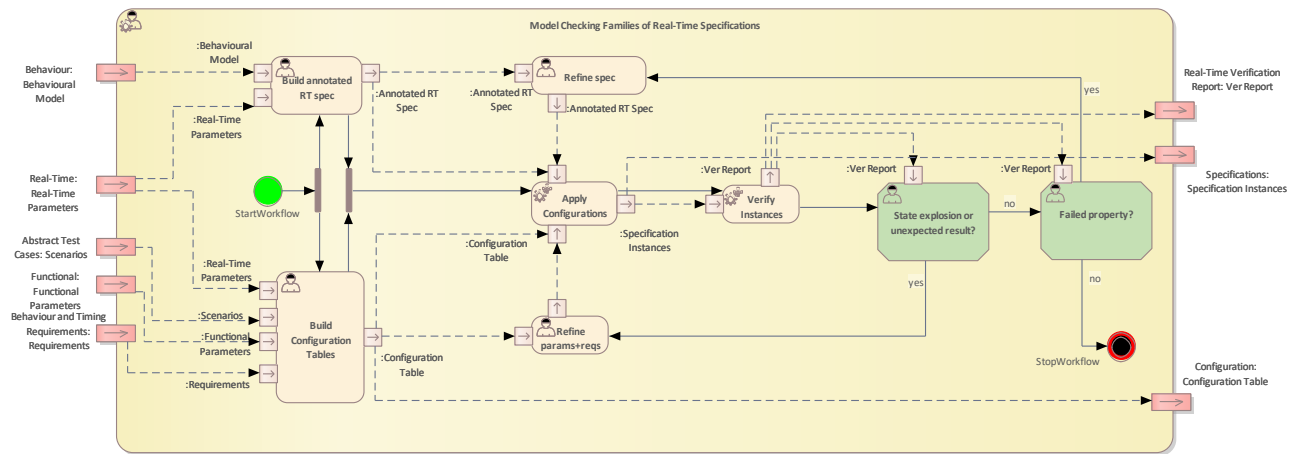


Figure 3.55 Workflow Definition diagram of Model Checking Families of Real Time Systems used in UC10\_BT

Table 3.36 lists the activities of the workflow Model Checking Families of Real-Time Specifications.

Table 3.36 List of activities performed by Model Checking Families of Real-Time Specifications

Name	Type	Description
Apply Configurations	Automated	Given an annotated specification (with customizable parts) and a set of configuration tables (describing alternative parameters to customize the specification), automatically create many concrete instances of the annotated specification with the information from the tables.
Build Configuration Tables	Manual	Construct a set of tables describing different sets of valid parameters, scenarios, and requirements. Identify a set of "products", i.e., groups of parameters, scenarios, and requirements that can be considered together.
Build annotated RT spec	Manual	Construct a specification for a Real-Time system, e.g., an input model for the UPPAAL model checker, and annotate

Name	Type	Description
		it using special commented blocks indicating variability areas. I.e., marking parts of the specification as customizable.
Failed property?	Manual	Verify if any of the properties failed the model-checking phase.
Refine params+reqs	Manual	Update the parameter's values and the requirements based on what properties failed and why.
Refine spec	Manual	Update the specification of the behaviour based on what properties failed and why.
State explosion or unexpected result?	Manual	Check if the verification phase timed-out or if some output seems different from the expected one.
Verify Instances	Automated	Run the model checker over a set of specifications and their requirements.

### 3.10.4 V&V Workflows of Optimize Fault Injection Experiments Using Model-Based Mutation Testing

For fault injection experiments, system stimuli are needed to operate the system under test while faults are injected. This is needed to verify that a fault tolerant system is gracefully handling a fault in all (relevant) situations. While the number of possible injected faults grows linearly with the size of the system, the number of possible input stimuli combinations, system states, and paths over the system states (i.e., test sequences) tends to grow exponentially with the size of the system.

By naively combining all injected faults with all tests, the time to run fault injection experiments would also grow exponentially with the size of the system. Therefore, ways to limit the size of the test suite, ensure the quality of the test suite, and select relevant test sequences per injected fault are crucial.

The combination with model-based mutation testing can support this by generating tests that:

- cover the system behaviour as complete as possible,
- do so with a low number of test sequences,
- ensure to propagate a problem caused by the injected fault long enough to become observable at the test interface.

If the system under evaluation can be instrumented to show which test sequences exercise the part of the system where the fault is injected, this can generally be used to limit the tests that need to be run per injected fault.

In this case, if model-based testing is used to create the test sequences, the tests can be used to establish correlations between model elements and parts of the system where faults are injected. By only mutating these model elements during model-based mutation testing, a specific, small test suite can be generated to be run for all fault injections into the related system part. Since the tests are optimized for problems

in the specific part of the system, the overall execution time of the fault injection experiments can be substantially lower than when using only selected tests of a standard test suite. Tests for the complete system with regression testing as test goal would try to achieve as much coverage as possible with a test, independently if the exercised system parts are of interest for the fault injection, or not. Tests for the complete system with debug testing as test goal would try to reach each coverage goal with as little steps as possible but running all relevant out of such a test suite would re-run shared prefixes of the test sequences numerous times.

Figure 3.56 shows the workflow specification diagram of Optimize Fault Injection Experiments Using Model-Based Mutation Testing.

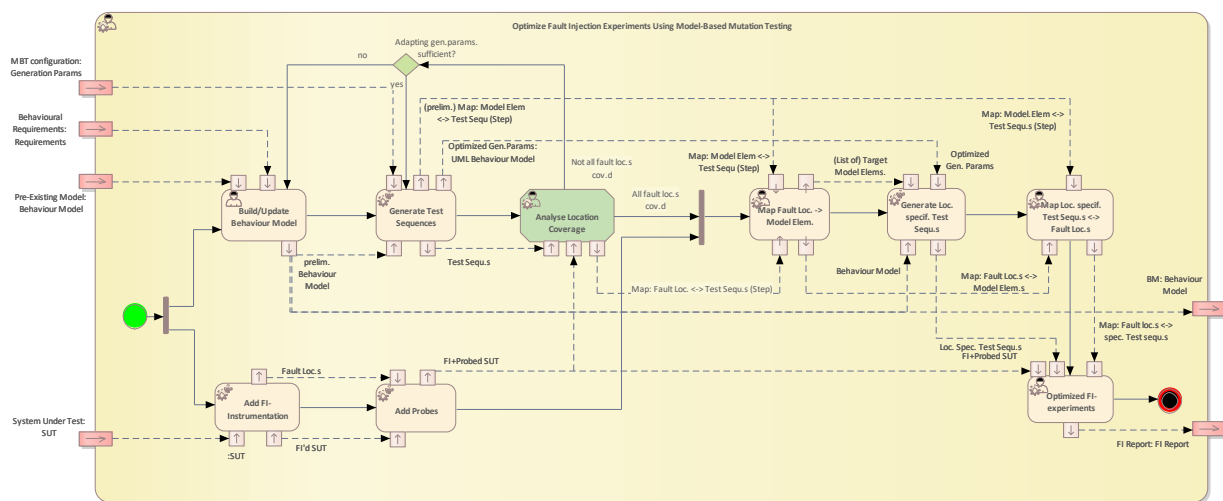


Figure 3.56 Workflow Definition diagram of Optimize Fault Injection Experiments Using Model-Based Mutation Testing used in UC10\_BT

Table 3.37 lists the activities of the workflow Optimize Fault Injection Experiments Using Model-Based Mutation Testing.

Table 3.37 List of activities performed by Optimize Fault Injection Experiments Using Model-Based Mutation Testing

Name	Type	Description
Add FI-Instrumentation	Automated	The SUT is modified in a way that allows later to inject faults.
Add Probes	Automated	Into the modified SUT means are inserted that provide information to the outside (of the SUT) about inner parameters.
Analyse Location Coverage	Semi-automated	Check whether the generated test sequences cover all fault locations added to the SUT in the "Add FI-instrumentation" activity.
Build/Update Model	Behaviour Manual	The behaviour of the SUT is modelled to a level of detail that covers all relevant behavioural aspects; for instance, as UML state machine.

Name	Type	Description
		If the model is too coarse (i.e. the control flow arrives activity from inner loop), extend model according to findings from location coverage analysis.
Generate Loc. specif. Test Sequ.s	Automated	The test sequences are extended in order to touch the model elements corresponding to fault locations.
Generate Test Sequences	Automated	Using a set of predefined mutations (e.g. modifying transition conditions or follow-up states), test sequences are generated that enforce the mutated model to behave differently than the original.
Map Fault Loc. -> Model Elem.	Semi-automated	The fault locations (see activity "Add FI-instrumentation") are mapped to the corresponding items in the behaviour model.
Map Loc. specif. Test Sequ.s <-> Fault Loc.s	Semi-automated	Convert the test sequences into form that can be applied to SUT.
Optimized FI-experiments	Semi-automated	Apply optimized test sequences to FI+Probed SUT.

### 3.10.5 V&V Workflows of Behaviour-Driven Model Development and Test-Driven Model Review

Here we aim to build scenarios from informal requirements. The scenarios can be for example: process control by of sensor data, server and PLC communication, anomaly detection at component and system level. The output are abstract test sequences. Figure 3.57 shows the workflow specification diagram of Optimize Fault Injection Experiments Using Model-Based Mutation Testing.

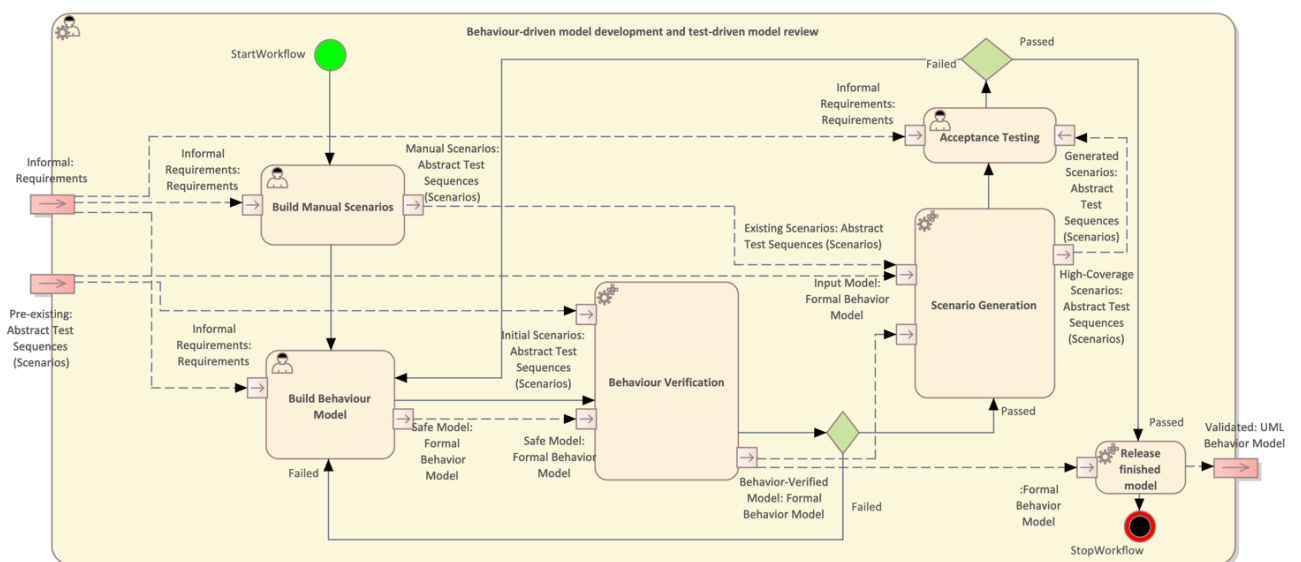


Figure 3.57 Workflow Definition diagram of Behaviour-driven model development and test-driven model review used in UC10\_BT

Table 3.38 lists the activities of the workflow Optimize Fault Injection Experiments Using Model-Based Mutation Testing.

*Table 3.38 List of activities performed by Behaviour-driven model development and test-driven model review*

Name	Type	Description
Build Manual Scenarios	Manual	Build scenarios from informal requirements. The scenarios can be for example: process control by of sensor data, server and PLC communication, anomaly detection at component and system level. The output are abstract test sequences.
Build Behaviour Model	Manual	The behaviour model of the system is built around informal requirement, use case scenarios and high-level descriptions of the behaviour to produce test cases, i.e., traces of interactions with external components that maximize coverage and a more refined descriptions of the behaviour to model check requirements, enriched with the scenarios derived from requirements. The output is a formal behaviour model for the model verification process.
Behaviour Verification	Manual	Behaviour verification of the formal Behaviour Model against manual defined test scenarios and abstract test sequences. The loop-back to the Behaviour Model activity is performed as long as the Behaviour Model passes all verification criteria.
Acceptance Testing	Manual	It performs an acceptance testing to validate the behaviour model of the system.
Scenario Generation	Automated	Both manual defined test scenarios and scenarios derived from behaviour models verify the transfer of systems behaviour from the real environment to the model environment. Given specific inputs, it allows the execution of meaningful scenarios designed to check that the system complies with the informal requirements.
Release finished model	Automated	After passing the acceptance test the Formal Behaviour Model is released as Validated UML Behaviour Model.

### 3.11 V&V Workflow of Use Case 11 OTOKAR

UC11\_OTOKAR package contains the following workflows:

- Penetration Testing
- Model-Based Formal Specification and Verification of Robotic Systems
- Simulation-based Verification
- Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks

Figure 3.58 shows the Penetration testing Method Definition diagram type of the V&V workflow UC11\_OTOKAR.

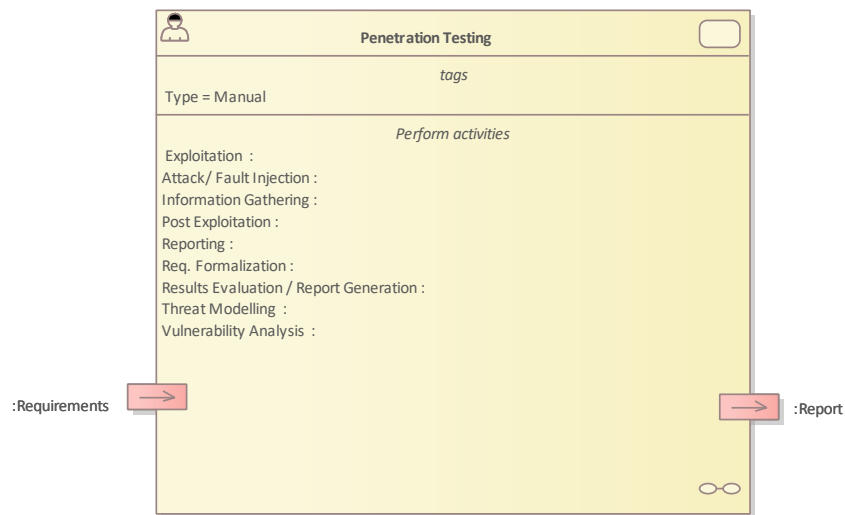


Figure 3.58 Method Definition of UC11\_OTOKAR\_2\_Penetration\_testing defined for UC11\_OTOKAR

Figure 3.59 shows the Model-Based Formal Specification and Verification of Robotic Systems Method Definition diagram type of the V&V workflow UC11\_OTOKAR.



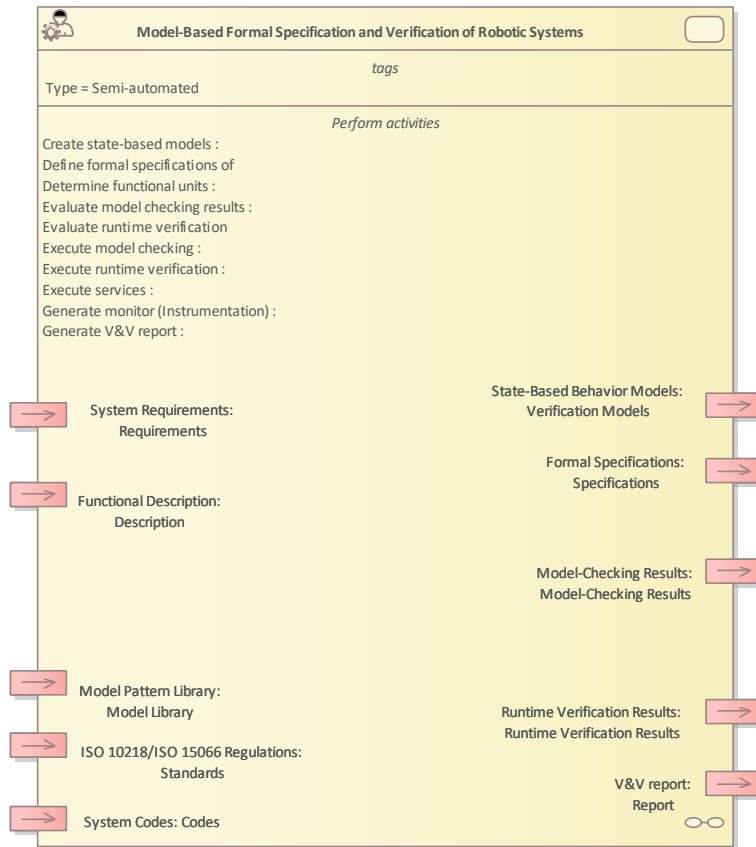


Figure 3.59 Method Definition of Model-Based Formal Specification and Verification of Robotic Systems defined for UC11\_OTOKAR

Figure 3.60 shows the Simulation-based Verification Method Definition diagram type of the V&V workflow UC11\_OTOKAR.

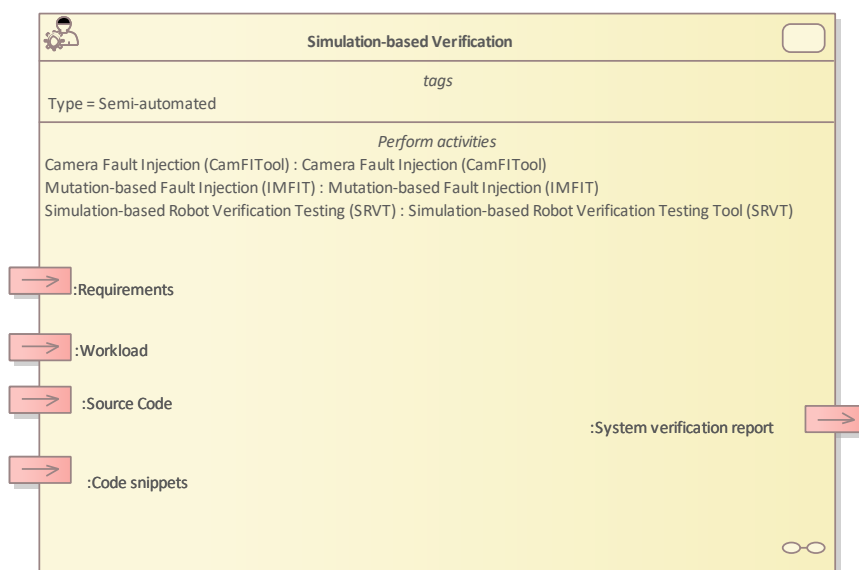


Figure 3.60 Method Definition of Simulation-based Verification defined for UC11\_OTOKAR

Figure 3.61 shows the Vulnerability Analysis of Cryptographic Modules Against Hardware-Based Attacks Method Definition diagram type of the V&V workflow UC11\_OTOKAR.

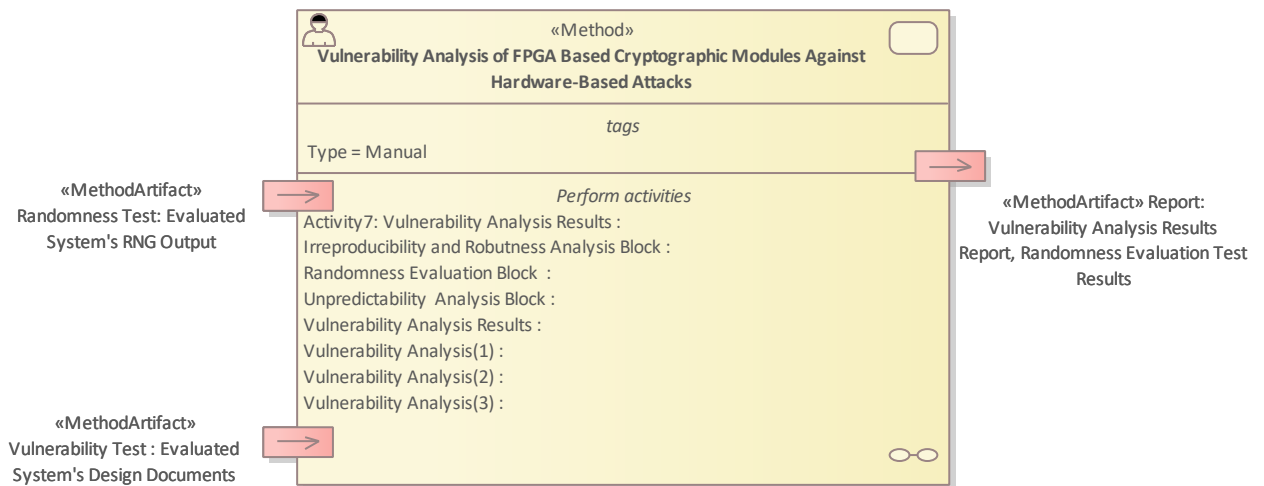


Figure 3.61 Method Definition of Vulnerability Analysis of Cryptographic Modules Against Hardware-Based Attacks defined for UC11\_OTOKAR

Details on the workflows are given in the following subsections.

### 3.11.1 Artifacts used in UC11\_OTOKAR

Table 3.39 lists the artifacts used for the workflow(s) defined for UC11\_OTOKAR.

Table 3.39 List of artifact types used in UC11\_OTOKAR

Name	Description
Code snippets (Information)	Code snippets are a programming term for a small region of reusable Python source codes for the robotic system.
Codes (Information)	They are system software codes created in accordance with the verified model.
Data /Log Files (Information)	Collecting data to prepare a security attack.
Description (Information)	It is a textual description of what the system will do and how it will behave.
Fault Injection Plan (Information)	In the images taken from the cameras on the arms of the robots will be injected.
Fault Library (Information)	Salt&Pepper, Gaussian, Poisson, Open, Close, Dilation, Erosion, Gradient, Motionblur will be injected to the images.
Model-Checking Results (Information)	These are the verification results that are created after the state-based behaviour models are checked whether they meet the specifications.

Name	Description
Model Library (Information)	It is a model library that contains formal models for the functional modules of robotic systems, which are created as patterns.
Report (Information)	Providing a detailed report of strategies to improve your security
Report (Information)	This reports the overall V&V results.
Requirements (Information)	The simulation models of the robotic system to be verified, the physical dimensions of the real environment, the task location lists applied by the robotic system and the software codes/scripts from the system, if any.
Requirements (Information)	It is a list of statements that identify the required functionality and safety of the system.
Requirements (Information)	Monitoring and inspection of unexpected network data activity. In case of an unexpected network data activity system will be shut down with safety protocols and will start working with back-up server.  WAN connection must prevent unauthorized access from WAN. (bypass firewall, DLP etc systems)
Runtime Verification Results (Information)	These are the verification results that are created after checking the fulfilment of the specifications during the execution of system programs.
Source Code (Information)	Robotic system codes to be verified.
Specifications (Information)	They are the formal specifications of the requirements.
Standards (Information)	They are the formal specifications of the standards to be used in the verification of robotic systems.
System verification report (Information)	Task completion times, safety trajectory plans for robots and fault reports (verification report) of the robotic system.
Verification Models (Information)	They are the models of the robotic system which are verified by model-checking, and then they are used to construct system software.
Workload (Information)	The workload is the structure that includes the source codes written for the task that robotic systems must perform. Within the source codes, the V&V operation is used as a reference for the applicable parts.

### 3.11.2 V&V Workflows of Model-Based Formal Specification and Verification of Robotic Systems

The workflow includes six input and four output artifacts. The input artifacts are required to implement workflow. These are functional descriptions, analysis models, model pattern library, ISO 10218 [18] /

ISO 15066 [19] regulations, and the system codes. The functional descriptions are the textual description of the designed system provided by the customer or systems owner. The analysis models are the models which were constructed during the analysis stage of the system, like use-case diagrams, activity diagrams, etc. Model pattern library consists of state-based models which resemble the common behaviours of the robotic systems. ISO 10218/ISO 15066 regulations are the robotic standards used to determine the system's compliance with the standards. The last input artifact is the systems codes. The codes are used to verify the system in the execution stage.

A state-based model, which is one of the output artifacts, reflects the system's behaviour. The specifications are constructed formally for the properties of the system based on the requirements. The method utilizes two techniques in two stages. Model-checking results and runtime verification results are the verification results obtained in each stage.

Figure 3.62 shows the workflow specification diagram of Model-Based Formal Specification and Verification of Robotic Systems.

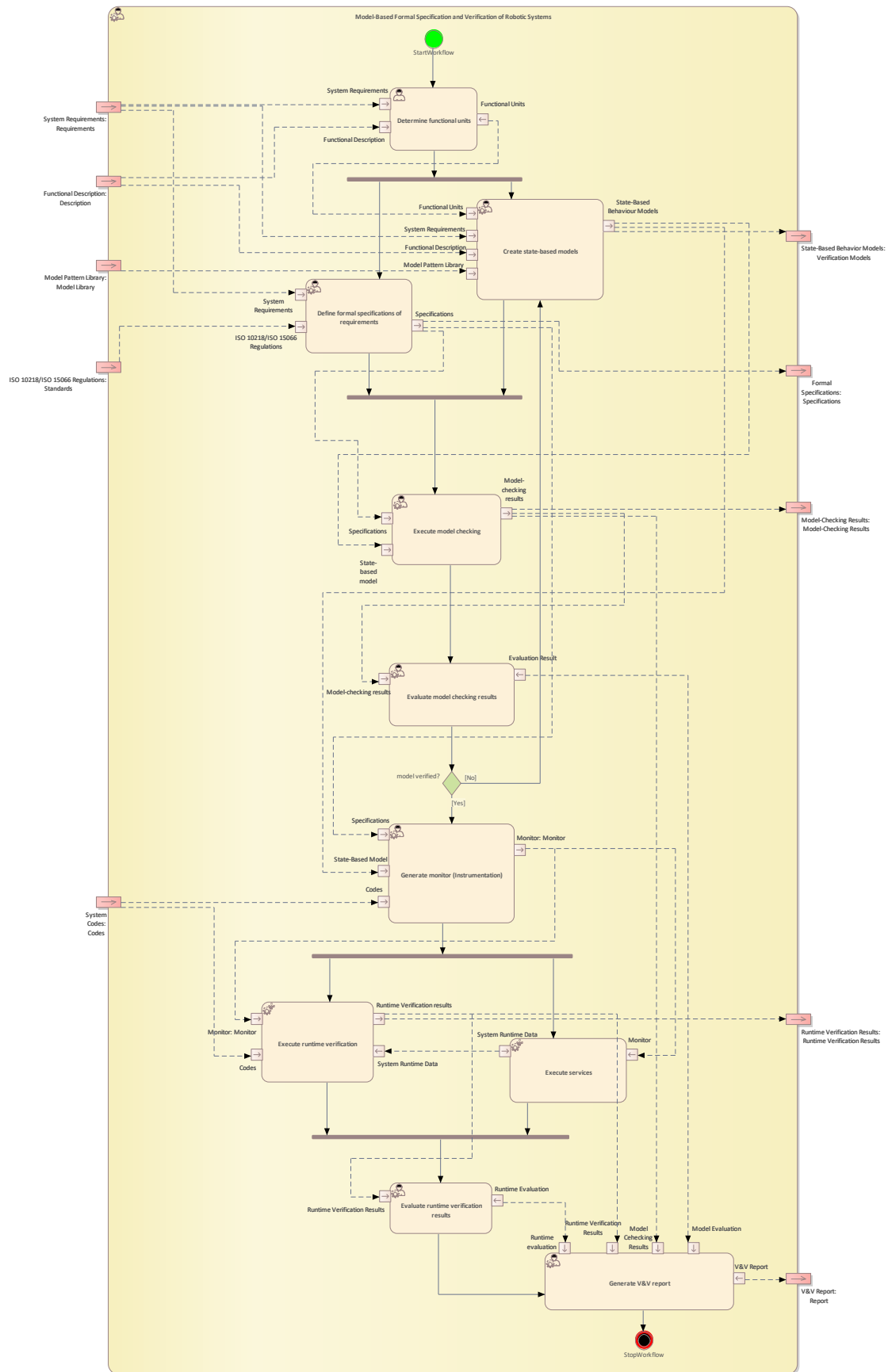


Figure 3.62 Workflow Definition diagram of MBF used in UC11\_OTOKAR

Table 3.40 lists the activities of the workflow Model-Based Formal Specification and Verification of Robotic Systems.

*Table 3.40 List of activities performed by Model-Based Formal Specification and Verification of Robotic Systems*

Name	Type	Description
Create state-based models	Semi-automated	A state-based model of the system is created. This model describes the behaviour of the system. If it is created in a way that fully represents the system, the fulfilment of the requirements related to the system can be checked.
Define formal specifications of requirements	Semi-automated	The safety and functional requirements are formally specified. Thus, the specifications are utilized to check the system model whether it meets.
Determine functional units	Manual	The software of the systems usually consists of multiple functional units. If these functional units are identified, it can facilitate system verification.  System requirements and description are utilized to determine the functional units.
Evaluate model checking results	Semi-automated	The model-checking results are analysed. If any properties are not met by the model, then the model needs to be revised.
Evaluate runtime verification results	Semi-automated	Runtime verification results are analysed. If there are cases where the requirements are not met by the running system, then the codes need to be revised.
Execute model checking	Semi-automated	The model is checked whether it fulfil the formally specified requirements. There are many tools for the purpose of checking model.
Execute runtime verification	Automated	Runtime verification is executed. The system codes and monitor is run.
Execute services	Automated	In addition to monitor, services that provides processed data about the system being verified (e.g. online distance tracker) are run.
Generate monitor (Instrumentation)	Semi-automated	During the runtime verification, the states and required values of the software to be verified need to be monitored. A monitor is generated in order to track these data. The model, properties, and system codes are used to configure the monitor.
Generate V&V report	Semi-automated	This activity compiles the verification results and creates a detailed report.

### 3.11.3 V&V Workflows of Penetration Testing

There are plenty of different techniques in data manipulation where MITM, DoS and ARP poisoning are emerging and commonly exploited. MITM (Men-in-the-Middle, also called person-in-the-middle) is a cyber-attack technique. Basically, in this technique, the attacker is positioning himself between two sides of communication for listening and resolving any information in communication [20]. DoS (a Denial-of-Service) attack is a cyber-attack in which the perpetrator aims to make a machine or network resource unavailable to its intended users by temporarily or permanently disrupting services of a host connected to the Internet [21] [22]. ARP is a communication protocol for link layer in ISO reference model at RFC 826 [20]. ARP Poisoning is also called ARP spoofing, ARP cache poisoning, or ARP poison routing. It is a technique by which an attacker sends (spoofed) ARP messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead [20] [23]. Industrial systems can be tested to detect these issues.

Figure 3.63 shows the workflow specification diagram of Penetration Testing.

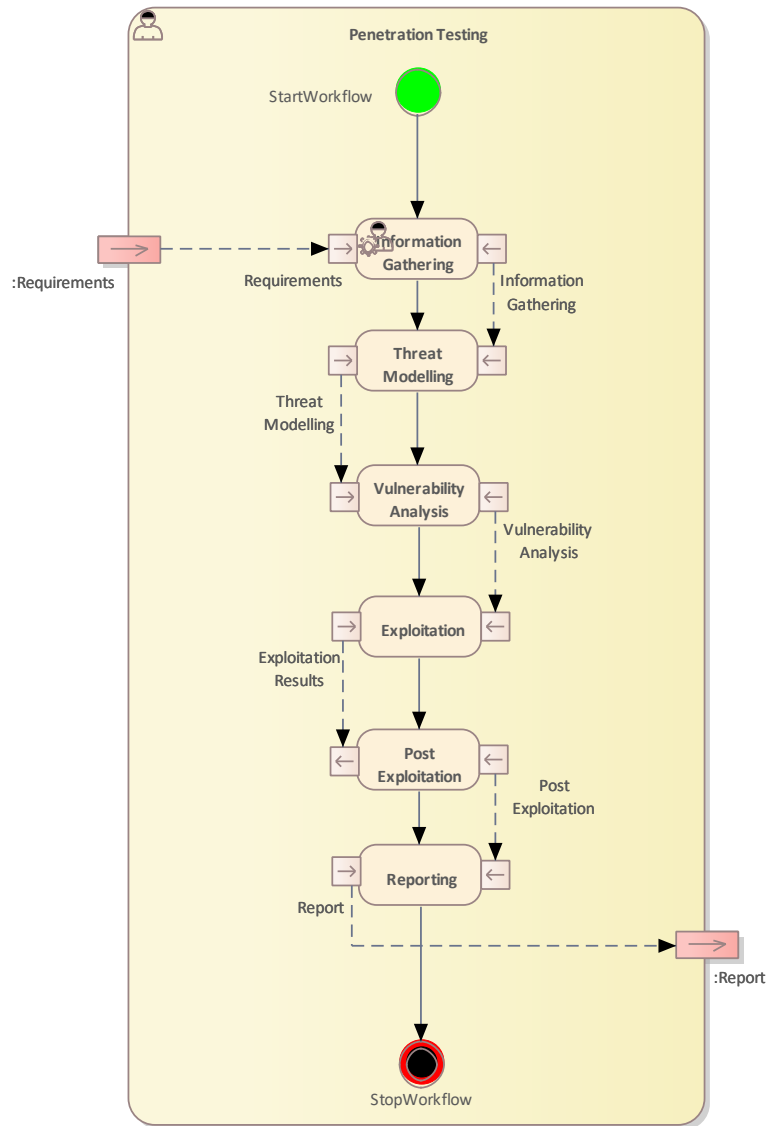


Figure 3.63 Workflow Definition diagram of Penetration Testing used in UC11\_OTOKAR

Table 3.41 lists the activities of the workflow Penetration Testing.

Table 3.41 List of activities performed by Penetration Testing

Name	Type	Description
Exploitation	Semi-automated	Attempting to gain sensitive data
Information Gathering	Semi-automated	Collecting data to prepare a security attack
Post Exploitation	Semi-automated	Evaluating the level of risk to your business known weaknesses
Reporting	Semi-automated	Providing a detailed report of strategies to improve your security
Threat Modelling	Semi-automated	Designing ways to test the weaknesses



Name	Type	Description
Vulnerability Analysis	Semi-automated	Defining the possible points of entry

### 3.11.4 V&V Workflows of Simulation-Based Verification

Description: In industrial operations, failure of an autonomous robot system can cause a significant hazard to that system. The safety of autonomous systems must be verified to prevent such accidents and to prevent possible loss of life and property. Currently, most systems are tested through field tests, which are costly, time consuming, limited in repeatable scenarios, and risky in case of unacceptable behaviour. To mitigate these issues, their software can be pre-validated using simulation-based testing.

Based on the simulation-based robot verification tests method, simulations of autonomous system operation were created in a virtual environment similar to the real environment in which the system will operate. In this system created, it has become possible to verify the robots. With the safe trajectory planned for the robots, the safety of the robots has been tried to be ensured. In addition, various fault injection mechanisms have been put into use on these systems. These fault injection mechanisms have their own elements for system testing and verification. These mechanisms provide different system corruption functions at compilation and runtime of a system.

In this method, a test system has been developed with studies on the Simulation Based Verification method, which focuses on the simulation and observation of robot behaviour and safety trajectory optimization in the automotive body inspection system. The safety of the robotic system is ensured by these tests made with the developed system.

Figure 3.64 above shows the workflow specification diagram of Simulation-based Verification.

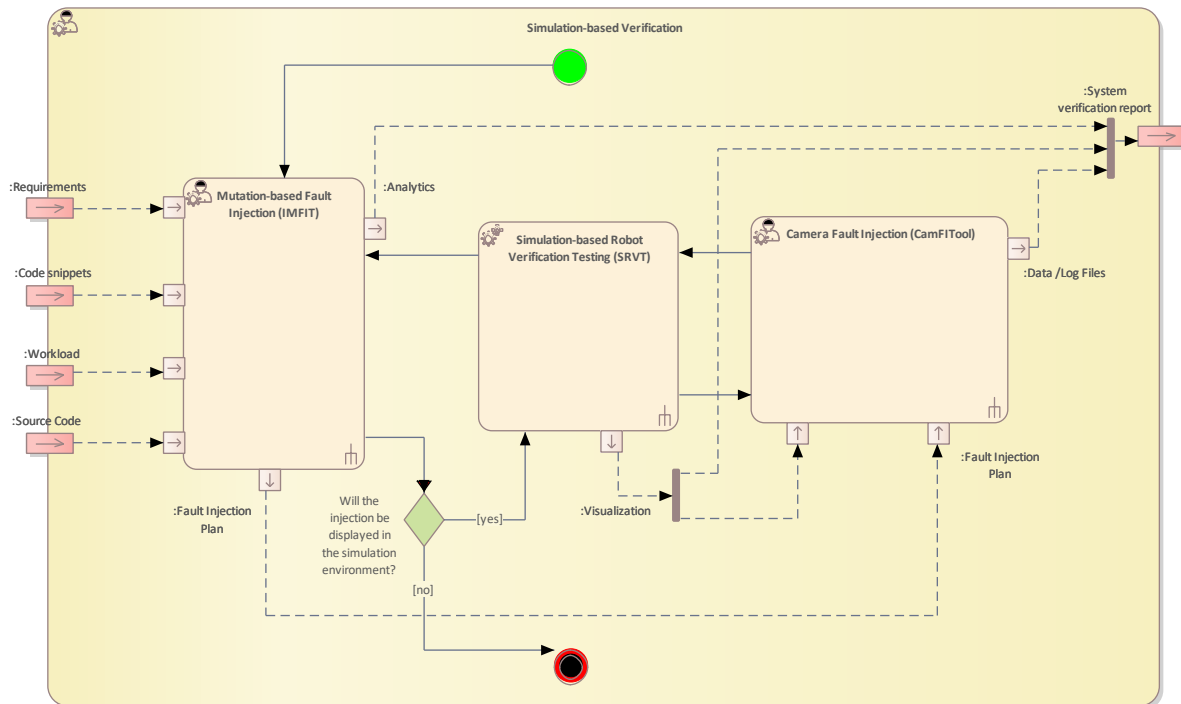


Figure 3.64 Workflow Definition diagram of Simulation-based Verification - Workflow used in UC11\_OTOKAR

Table 3.42 lists the activities of the workflow Simulation-based Verification.

Table 3.42 List of activities performed by Simulation-based Verification

Name	Type	Description
Camera Fault Injection (CamFITool) (CallBehavior)	Semi-automated	<p>Camera Fault Injection Tool (CamFITool) is a simple interface that allows injection of image faults/distortion into robot cameras. Thanks to this interface, you can create new image libraries by injecting the fault types you have determined, both real-time to ROS cameras, and to the image libraries previously recorded by these cameras.</p> <ul style="list-style-type: none"> <li>• Applicability of 6 fault types regulated for cameras.</li> <li>• Having both offline and realtime fault injection features.</li> <li>• User-friendly, easy to develop and open source.</li> </ul>
Mutation-based Fault Injection (IMFIT) (CallBehavior)	Semi-automated	<p>IMFIT is a simulation-based fault injection tool. Intended purpose of this tool is to identify and understand potential failures in the simulation environment. With this tool, the user injects some faults, create failures or errors and monitor their effects in a simulation environment. This tool is designed to inject faults to the robots which are used in industry.</p>

Name	Type	Description
		<p>With the popular use of ROS (Robot Operating System) in industrial robotics, this tool is developed as compatible with ROS and Gazebo.</p>
<p>Simulation-based Robot Verification Testing Tool (SRVT) (CallBehavior)</p>	<p>Semi-automated</p>	<p>Description: SRVT is a testing tool system that enables industrial robots to be verified &amp; validated in simulation environments. Using the Robot Operating System (ROS) as its infrastructure, SRVT uses Gazebo to create the simulation environment, MoveIt for robot motion trajectory planning, and the ROS Smach library as the finite state machine system. SRVT is formed by bringing these components into a system.</p> <p>These are components for SRVT:</p> <ul style="list-style-type: none"> <li>• <u>ROS</u>: The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications (<a href="https://www.ros.org">https://www.ros.org</a>).</li> <li>• <u>Gazebo</u>: Gazebo, which is the most widely used simulation engine in ROS (<a href="http://gazebosim.org">http://gazebosim.org</a>).</li> <li>• <u>MoveIt</u>: MoveIt is a planning framework where we transfer the planned trajectory into the SRVT system and use it (<a href="https://moveit.ros.org">https://moveit.ros.org</a>).</li> </ul> <p><b>Improvement in Valu3s:</b> The robotic system (UC11) used before VALU3S used a C#-based manual control system and had to be constantly observed by an operator. In addition, the task completion times and working processes of the robotic system were not optimized.</p> <p>With SRVT, optimization of this robotic system and improvements in task completion times are realized. With the tests performed at SRVT, the most ideal trajectory planning algorithms for this robotic system were determined and it was observed that the tasks were completed in 30-40% shorter time with this algorithm. In addition, the safe implementation of the trajectories obtained with the help of Moveit by the robot has been observed in Gazebo.</p> <p>Link: <a href="https://github.com/inomuh/srvt-ros">https://github.com/inomuh/srvt-ros</a></p>

### 3.11.5 V&V Workflows of Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks

ERARGE automates the verification and validation processes of chaotic oscillator and ring oscillator based RNGs designed in FPGA. In this context, it is aimed that the RNGs in the design phase will work without any security vulnerabilities throughout their lifetimes.

Artifact and Method Interfaces:

- Evaluated System's Design Documents. These documents must be in a format that can be imported into Anadigm Designer or Vivado Design Suite.
- Evaluated System's RNG Output. This is a string of bits. It can be uploaded as an input in .bit or .hex format. This bit string is tested by the Randomness Test Suite programs. (MATLAB, Octave etc.)
- Vulnerability Analysis Report, Randomness Evaluation Test Results. This is a report. It is created in .pdf or .doc format. It is prepared as design and performance feedback about the RNG to the RNG designer or the authority in any specific organization. The main output of the method is this report of vulnerability analysis assessment and test results.

Activities in the workflow with their control flows and data flows: There is a workflow without hierarchy here. Each of the workflow steps contributes to the resulting report from different perspectives. If there is no obvious error in the RNG design documents after preliminary inspection, the RNG outputs can be tested as a bit string (must be at least 1 million bits).

Randomness Evaluation Block: NIST 800-22 [24], BigCrush [25] and DieHard [26] tests are standard randomness tests. In this block, RNG outputs are taken as input and randomness tests are performed. Results are reported.

Unpredictability Analysis Block: Design documents are reviewed by ERARGE. Whether the evaluated RNG is unpredictable is examined with special techniques. These techniques are very diverse, some versions can be seen in scientific publications published in [27].

Irreproducibility and Robustness Analysis Block: Design documents are reviewed by ERARGE. If the output bit sequences of the evaluated RNG have a pattern and are immune to external interference, the RNG cannot pass this test (in such a case the test results indicate that the targeted design is failed, and the designer should correct/improve the previous design). This process is examined with special techniques. These techniques are very diverse, some versions can be seen in relevant scientific publications [28] [29]. According to these publications of ERARGE (mainly on cryptanalysis), if the RNG examined is a ring oscillator-based RNG, it is statistically checked whether the interim outputs of the RNG's inner blocks are correlated with each other. These techniques are very diverse, and alternative designs can be adapted to various needs of cyber-physical systems.

Figure 3.65 shows the workflow specification diagram of Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks.

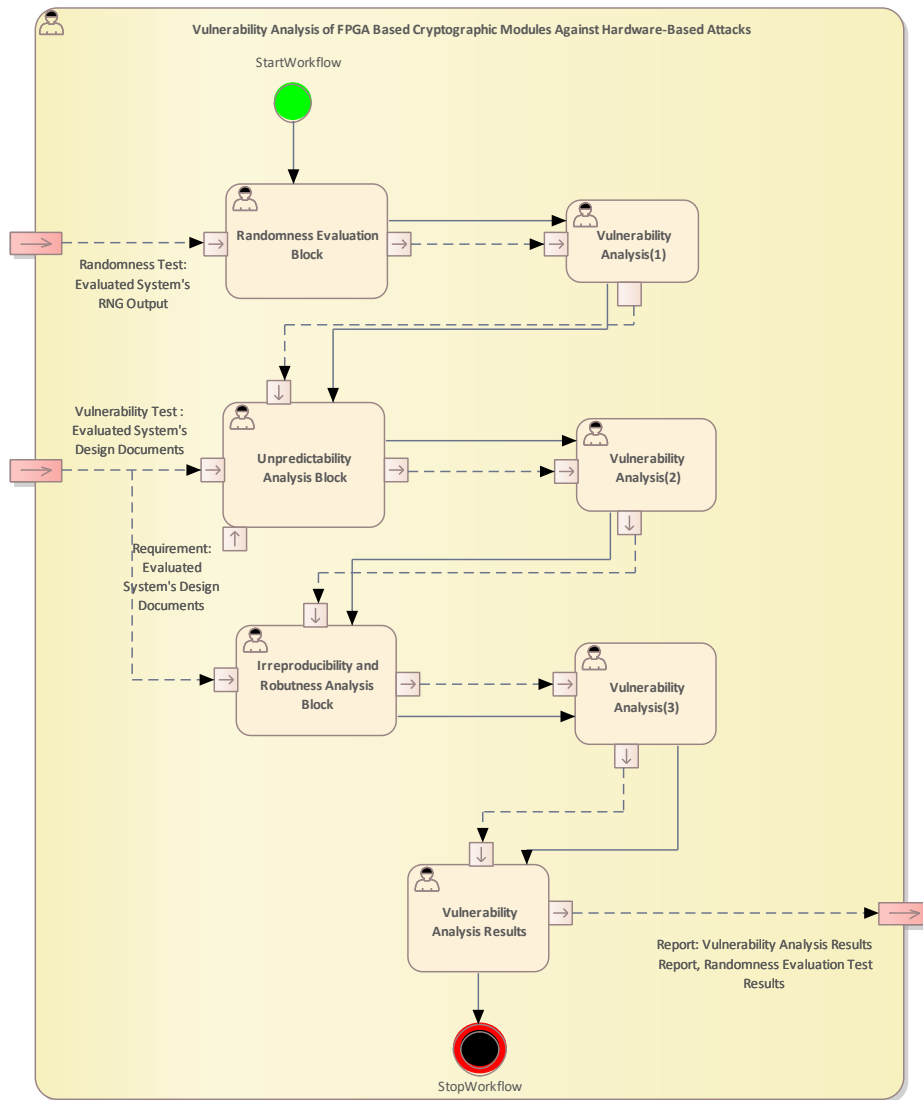


Figure 3.65 Workflow Definition diagram of Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks used in UC11\_OTOKAR

Table 3.43 lists the activities of the workflow Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks.

Table 3.43 List of activities performed by Vulnerability Analysis of FPGA Based Cryptographic Modules Against Hardware-Based Attacks

Name	Type	Description
Irreproducibility and Robustness Analysis Block	Manual	The position of the ring oscillator based RNGs in the FPGA hardware and the sampling technique used increase the correlation between the ring oscillators, this block is used to analyse it.

Name	Type	Description
Randomness Evaluation Block	Manual	At the end of all design applications, RNG outputs are subjected to randomness tests. Thus, it contributes to verifying the unpredictability and non-reproducibility of cryptographic keys.
Unpredictability Analysis Block	Manual	Circuits with chaotic ring oscillators are based on some basic chaotic mathematical equations (example: jerk equation). This block is used to prevent the risk of obtaining the variable coefficients/voltage values of these structures by scalar time series analysis and the estimation of all the data produced by this RNG.
Vulnerability Analysis Results	Manual	Contains the result report of all applied analyses.
Vulnerability Analysis(1)	Manual	Contains the result report of randomness analyses.
Vulnerability Analysis(2)	Manual	Contains the result report of Unpredictability analyses.
Vulnerability Analysis(3)	Manual	Contains the result report of Irreproducibility and Robustness analyses.

### 3.12 V&V Workflow of Use Case 13 SIEMENS

UC13\_SIEMENS package contains the following workflows:

- UC13 - SIEMENS
- Model-Based Mutation Testing
- Monitoring Enriched Test Execution
- Mutation-Driven Model-Based Test Case Generation

Figure 3.66 shows the UC13 - SIEMENS Method Definition diagram type of the V&V workflow UC13\_SIEMENS.

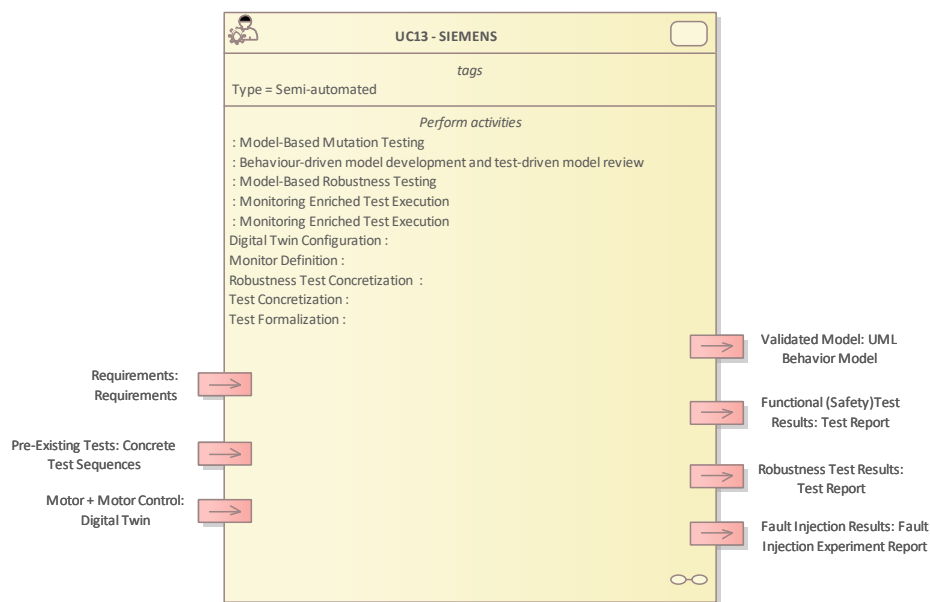


Figure 3.66 Method Definition of UC13 - SIEMENS defined for UC13\_SIEMENS

Figure 3.67 shows the Model-Based Mutation Testing Method Definition diagram type of the V&V workflow UC13\_SIEMENS.

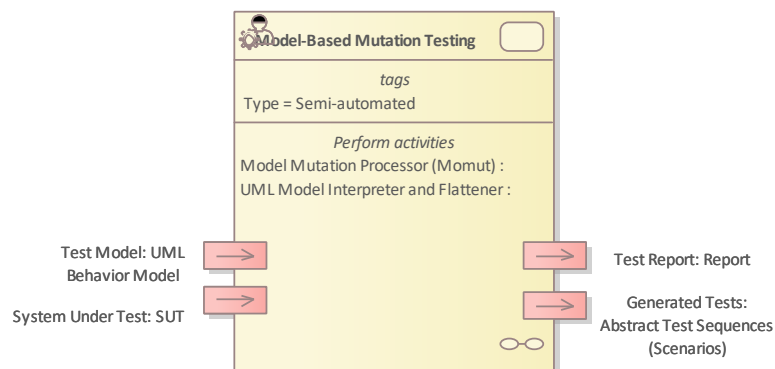


Figure 3.67 Method Definition of Model-Based Mutation Testing defined for UC13\_SIEMENS

Figure 3.68 shows the Monitoring Enriched Test Execution Method Definition diagram type of the V&V workflow UC13\_SIEMENS.

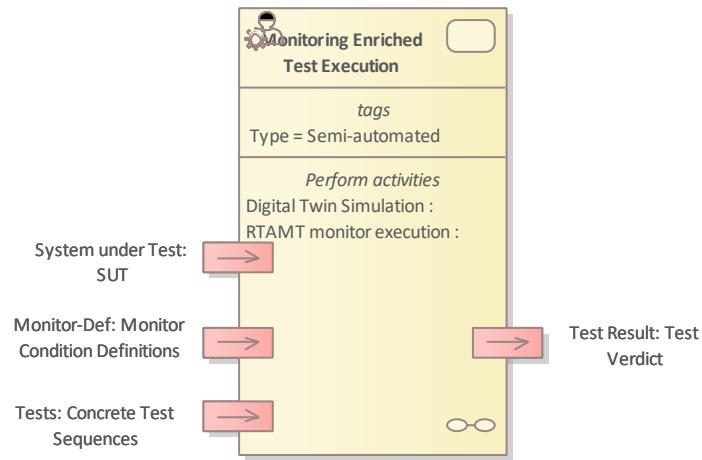


Figure 3.68 Method Definition of Monitoring Enriched Test Execution defined for UC13\_SIEMENS

Figure 3.69 shows the Mutation-Driven Model-Based Test Case Generation Method Definition diagram type of the V&V workflow UC13\_SIEMENS.

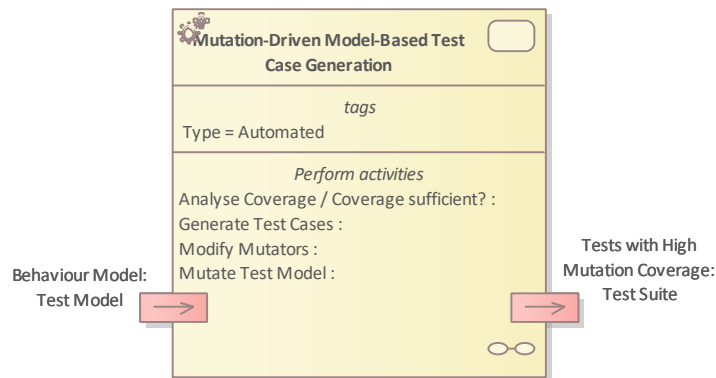


Figure 3.69 Method Definition of Mutation-Driven Model-Based Test Case Generation defined for UC13\_SIEMENS

Details on the workflows are given in the following subsections.

### 3.12.1 Artifacts used in UC13\_SIEMENS

Table 3.44 lists the artifacts used for the workflow(s) defined for UC13\_SIEMENS.



Table 3.44 List of artifact types used in UC13\_SIEMENS

Name	Description
Artifact (Information)	Report on passed/failed tests of the system regarding safety.
Artifact (Information)	A behavioural model of the use case system in UML.
Behaviour Model (Active Unit)	A test model that describes the required behaviour of the SUT (and can be executed).
Concrete Test Sequences (Information)	A series of tests cases for checking the correct implementation of the motion control application – based on pre-existing tests.
Digital Twin (Active Unit)	A digital twin implementation of the system including hardware models such as motor control platform and physical motor model, and relevant software such as motor control algorithms.
Fault Injection Experiment Report (Information)	Report on the passed/failed behaviour of the system regarding fault-injection tests.
Monitor Condition Definitions (Information)	The monitor is the correctness specification. It can be described in Linear Temporal Logic or Signal Temporal Logic (for cyber-physical systems)
monitor formal analysis test result (Information)	Test report on successful/failed tests.
Requirements (Active Unit)	Requirements for the system.
Scenarios (Abstract Test Sequences)	Test sequences that detect faults (modelled as mutations of the correct model) in the model, hence 'abstract'. They represent test scenarios.
Simulation traces (Information)	A series of tests cases for checking the correct implementation of the motion control application – based on pre-existing tests.
SUT (Active Unit)	The system under test containing the digital twin.
Test Report (Information)	Test report on successful/failed tests.
Test Report for Functional Safety (Information)	Test report on successful/failed tests regarding functional safety.
Test Suite ( <i>from Mutation-Driven Model-Based TCG</i> )	Set of tests that together cover a certain aspect or criterion (of the SUT), e.g., its requirements.
Test Verdict (Information)	Test report on successful/failed test execution
UML Behavioural Model (Information)	The system UML behavioural model.
Updated and Flattened Model (Information)	The tests can be used to establish correlations between model elements and parts of the system where faults are injected
Updated Model with FI Instrumentation (Information)	Updated model with Fault-injection hooks

### 3.12.2 V&V Workflows of UC13 - SIEMENS

The group of V&V methods applied to the motor control for industrial drives use case (UC13), which includes the migration of a digital twin to another processor platform, are:

- Behaviour-Driven Model Development and Test-Driven Model Review
- Model-Based Mutation Testing
- Model-Based Robustness Testing
- Monitoring Enriched Test Execution (Test Oracle Observation at Runtime)

As such, the method takes the digital twin, requirements and potentially pre-existing test cases as input and delivers a validated model and reports on functional and robustness tests as well as fault injection.

This overall workflow depicts how the methods interact in the context of the use case. *Behaviour-Driven Model Development and Test-Driven Model Review* provides a behaviour model for the two model-based test case generation approaches. The resulting abstract tests are concretized and run in a monitoring-enriched test environment. This environment makes use of formally defined monitors to ensure that properties in the discrete or continuous domain, such as voltage and current, of the system under test hold. This is an application of the *Test Oracle Observation at Runtime* method (as part of Monitoring Enriched Test Execution).

Figure 3.70 shows the workflow specification diagram of UC13 - SIEMENS.

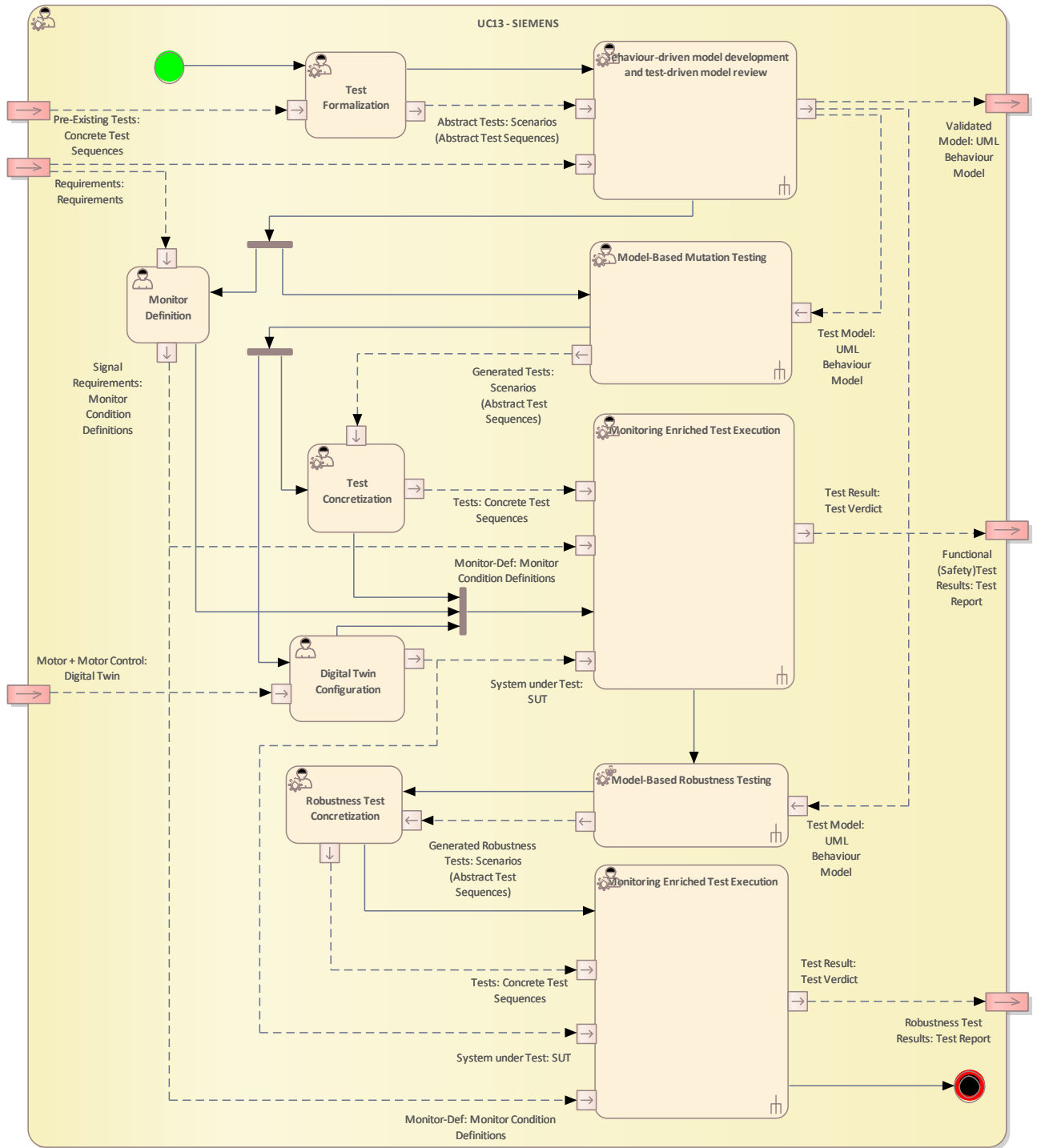


Figure 3.70 Workflow Definition diagram of UC13 - SIEMENS used in UC13\_SIEMENS

Table 3.45 lists the activities of the workflow UC13 - SIEMENS.

Table 3.45 List of activities performed by UC13 - SIEMENS

Name	Type	Description
Activity: Behaviour-driven model development and test-driven model review (CallBehavior)	Semi-automated	The behaviour of a system-under-test (e.g., cyber-physical system) is modelled by the application of modelling languages. In this use case, UML-based models are designed in Enterprise Architect. Taking the information from informal requirements and formalized pre-existing test scenarios, the (executable) system model is reviewed and validated. Outputs are a validated system model and test models for Model-Based Mutation Testing and Model-Based Robustness Testing.
Activity: Model-Based Mutation Testing (CallBehavior)	Semi-automated	Test mutation has the goal of covering the system behaviour as completely as possible with as few test sequences as necessary. New tests are generated based on the test model (input) by applying test mutations.
Activity: Model-Based Robustness Testing (CallBehavior)	Automated	Model-Based Robustness Testing takes the validated test model as input and creates test cases for running robustness tests (e.g., the generation of fault-injection tests). Interface fault injection (or robustness testing) requires that the system/component under test faces erroneous input conditions, which are usually defined based on typical developer mistakes or wrong assumptions. Erroneous input conditions can be also generated at random in some robustness testing scenarios. In a more general fault injection context, erroneous inputs injected at the interface of a given component can represent failures in preceding components that forward their erroneous outputs to the target component.
Activity: Monitoring Enriched Test Execution (CallBehavior)	Semi-automated	The concretized test cases, monitor definitions and the system under test (configured digital twin) are inputs for this activity, where the system is executed in a monitoring-enriched environment. This environment makes use of formally defined monitors to ensure that properties of the continuous outputs (e.g., motor speed, current, voltage) of the system under test hold. This is an application of the <i>Test Oracle Observation at Runtime</i> method (as part of Monitoring Enriched Test Execution). A test result report is written as output of this activity.
Activity: Monitoring Enriched Test Execution (CallBehavior)	Semi-automated	The concretized test cases, monitor definitions and the system under test (configured digital twin) are inputs for this activity, where the system is executed in a monitoring-enriched environment. This environment makes use of formally defined monitors to ensure that properties of the continuous outputs (e.g., motor speed, current, voltage) of the system under test hold. This is an application of the <i>Test</i>

Name	Type	Description
		<i>Oracle Observation at Runtime</i> method (as part of Monitoring Enriched Test Execution). A robustness test result report is written as output of this activity.
Digital Twin Configuration	Manual	Digital twin configurations (configuration of test runtimes, set-ups of motors and control algorithms, batch simulation scenarios, etc.) are created for simulation with concrete tests in Monitoring Enriched Test Execution.
Monitor Definition	Manual	Based on the system requirements, monitors are defined with the capability of analysing signals for the digital twin. Monitor specification is supported by formal languages (signal temporal logic), e.g., the motor shaft rotation speed requirements are formalized, which serve as input for creating the signal monitor definitions.
Test Concretization	Semi-automated	Abstract test cases are concretized (data format, test structure) for their application in a test environment connected to the system-under-test realized as a digital twin.
Robustness Concretization	Test Semi-automated	Abstract robustness test cases are concretized (data format, test structure) for their application in a test environment connected to the system-under-test realized as a digital twin.
Test Formalization	Semi-automated	Pre-existing tests (concrete test sequences) are formalized using notations suitable for the method <i>Behaviour-Driven Model Development and Test Driven Model Review</i> , which require abstract test sequences and scenarios.

### 3.12.3 V&V Workflows of Model-Based Mutation Testing

Model-Based Testing is an approach for testcase generation for black-box testing. Model-based mutation testing was pioneered by Lipton in 1971. Since there have been several approaches described in literature, using several modelling formalisms and several commercial tools are available as well. Many applications of the approach are in the safety critical systems domain, probably because there the additional effort of creating a sufficiently complete model for testing is easier to argue.

The model-based mutation testing technique uses the input model to create a number of mutants, which differ from the original model in tiny details. The goal is then to find tests that differentiate the mutant from the original. These tests can then be used to test the implementation of the model.

An example would be a UML state machine that represents the behaviour of a car alarm system. The model would arm the alarm when the doors are locked and raise an alarm when a door is open before the car is unlocked. This model could be used to derive tests over the input/output behaviour of the alarm system. These tests can be used to test a real-world implementation of the alarm system.

Figure 3.71 shows the workflow specification diagram of Model-Based Mutation Testing.

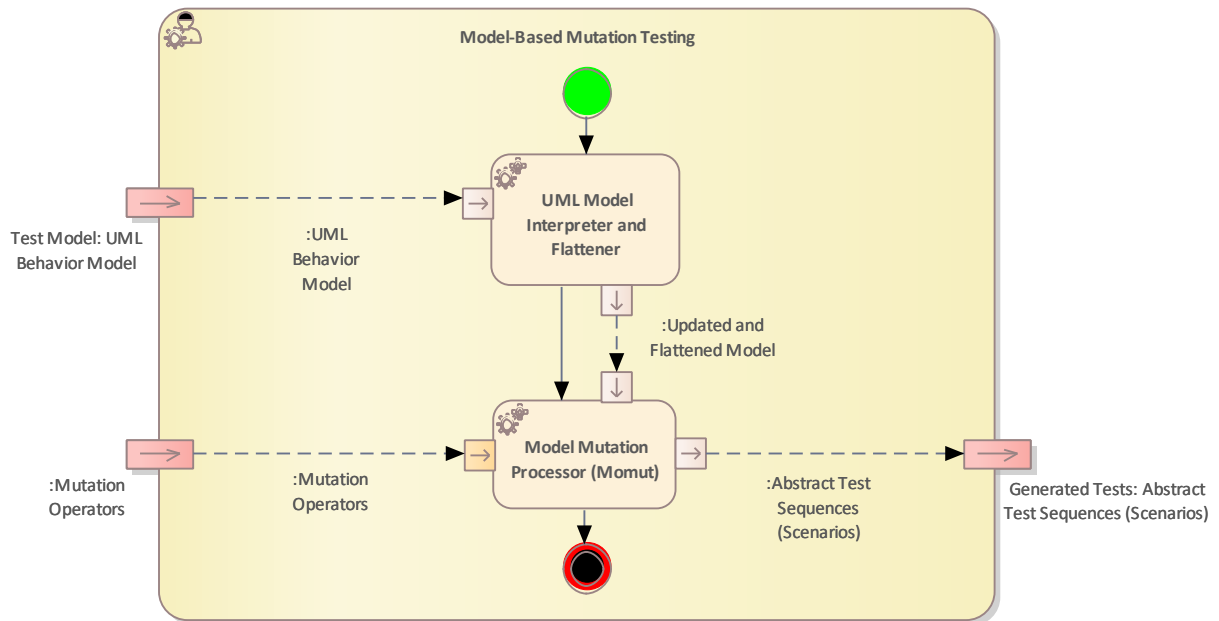


Figure 3.71 Workflow Definition diagram of Model-Based Mutation Testing - Workflow used in UC13\_SIEMENS

Table 3.46 lists the activities of the workflow Model-Based Mutation Testing.

Table 3.46 List of activities performed by Model-Based Mutation Testing

Name	Type	Description
Model Mutation Processor (Momut)	Automated	In a first step, the model mutation processor syntactically alter the original test model and produces a set of mutated models. In the next step automatically generate test cases try to kill the model mutants, i.e., reveal their non-conforming behaviour. This is accomplished by a conformance check between the original and the mutated models. As the test model is an abstraction of the SUT, also the derived test sequences are abstract. Hence, they have to be concretised, i.e., mapped to the level of detail of the SUT
UML Model Interpreter and Flattener	Automated	The main parts of the UML behaviour model are class diagrams, state machines and interfaces for the input and output signals. First, the interpreter checks the UML model for syntactical correctness. In a further step the hierarchical UML model will be converted to a flat model representation.

### 3.12.4 V&V Workflows of Monitoring Enriched Test Execution

In this use case, Monitoring Enriched Test Execution enables the analysis of signals based on formal requirement definitions. The system is tested by the simulation with a digital twin and chosen system traces, such as discrete/continuous domain signals, are exported to the Real-Time Analog Monitoring Tool (RTAMT [30]) which analyses signals and checks them against the formally specified requirements. As results, the digital twin simulation outputs a test report, while the RTAMT tool generates an analysis test result potentially showing requirement violations and indicators for bug fixes and optimizations.

Figure 3.72 shows the workflow specification diagram of Monitoring Enriched Test Execution.

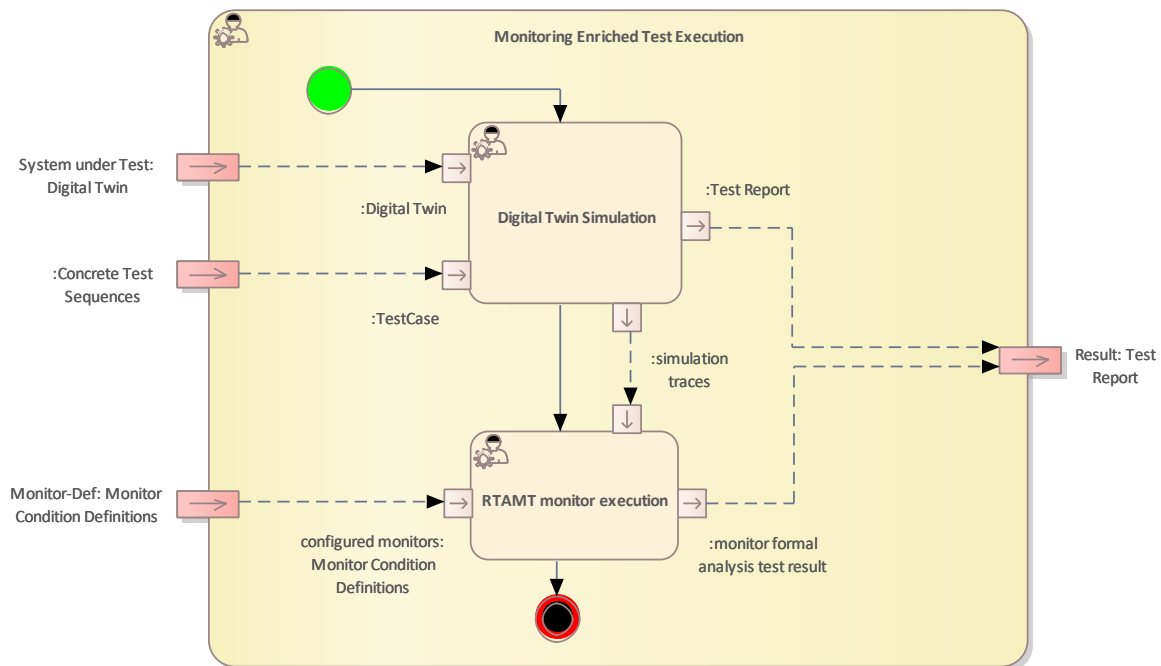


Figure 3.72 Workflow Definition diagram of «Method» Monitoring Enriched Test Execution used in UC13\_SIEMENS

Table 3.47 lists the activities of the workflow Monitoring Enriched Test Execution.

Table 3.47 List of activities performed by Monitoring Enriched Test Execution

Name	Type	Description
Digital Twin Simulation	Semi-automated	The digital twin consists of a motor control platform, a motor model and a motor control software. The digital twin is started based on the provided test case sequences enabling combined analysis of safety, security and performance. After successful simulation, a test report is output and simulation traces are exported to the RTAMT monitor for further signal analysis.

Name	Type	Description
RTAMT monitor execution	Semi-automated	RTAMT is a Python (2- and 3-compatible) library for monitoring of Signal Temporal Logic (STL). The library implements algorithms offline and online monitoring of discrete-time and dense-time STL. The online monitors support the bounded future fragment of STL. The online discrete-time part of the library has an optimized C++ back-end.

### 3.12.5 Mutation-Driven Model-Based Test Case Generation

Model-based testing (MBT) is used to automatically create test cases for diverse Systems Under Test (SUT), described in form of a formal system test model. This system test model incorporates the specification of the SUT. In this case MBT provide the opportunity to verify that the implemented system conforms to its specification.

A variant of MBT is the mutation-driven model-based testing, a fault-based variant of MBT. The generated test cases detect faulty implementation versions of the specification. The method illustrates that during the system implementation the specified requirements were correctly understood and that the SUT is free of the faults which are injected in the specification - in this case in the test model.

Mutation-driven model-based testing is a semantically very rich test case generation technique. Due to the high overhead of test sequence processing that detects the differences between the original specification and the mutations, this test technology is often considered impractical and not applicable.

Figure 3.73 shows the workflow specification diagram of Monitoring Enriched Test Execution.

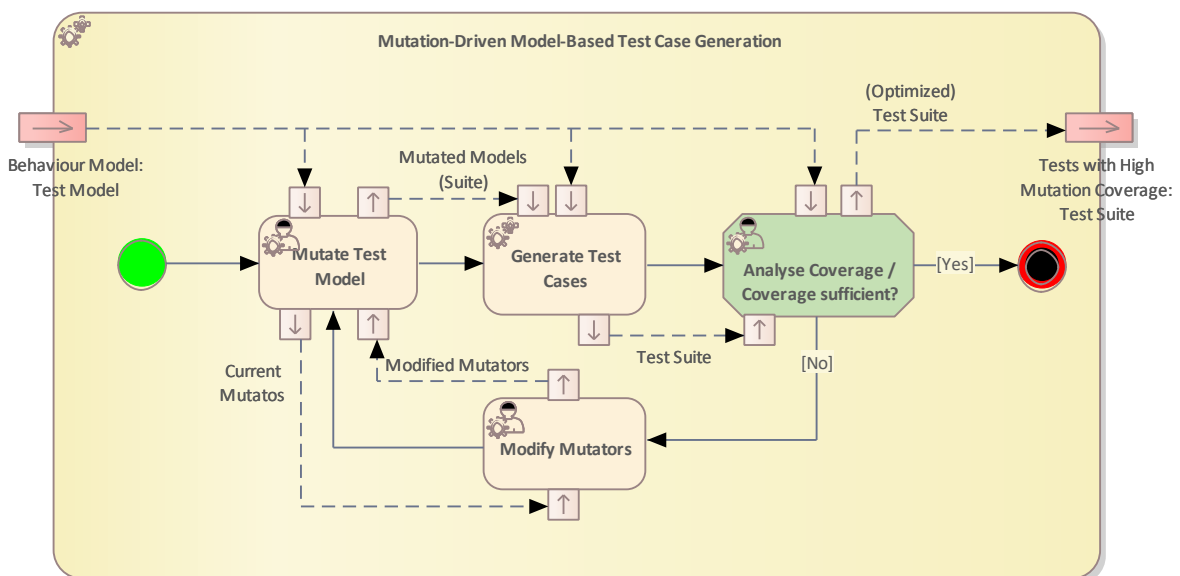


Figure 3.73 Workflow Definition diagram of Mutation-Driven Model-Based Test Case Generation used in UC13



Table 3.48 lists the activities of the workflow Mutation-Driven Model-Based Test Case Generation.

*Table 3.48 List of activities performed by Mutation-Driven Model-Based Test Case Generation*

Name	Type	Description
Analyse Coverage / Coverage sufficient?	Semi-automated	Assess how many elements of the behavior model are accessed by the found test suite.
Generate Test Cases	Automated	I/O sequences are searched, that force the mutated model ('mutant') to behave observably different than the original test model. That mutant is then considered to be "covered". This is done for each mutant. It can be tried to find I/O-sequences that cover several mutants". The resulting I/O-sequences build the resulting test suite.
Modify Mutators	Semi-automated	The set of mutators is modified such that model elements currently not tangled can be modified. (Usually, further mutators are added.)
Mutate Test Model	Semi-automated	Generate variants of the Test Model by applying so-called 'mutators' to it. For state machines, a mutator can e.g. change the targets of transitions, or modify the transition conditions. Initially, a predefined set of mutators is used.

### 3.13 V&V Workflow of Use Case 14 CARDIOID

UC14\_CARDIOID package contains the following workflows:

- Biometric Model Performance and Privacy Validation
- Hardware in the Loop Validation & Verification
- Verification of Driver Monitoring Models
- Safe Generation and Instrumentation of Runtime Verification Architectures
- Software-Implemented Fault Injection

Figure 3.74 shows the Biometric Model Performance and Privacy Validation method definition diagram type of the V&V workflow UC14\_CARDIOID.

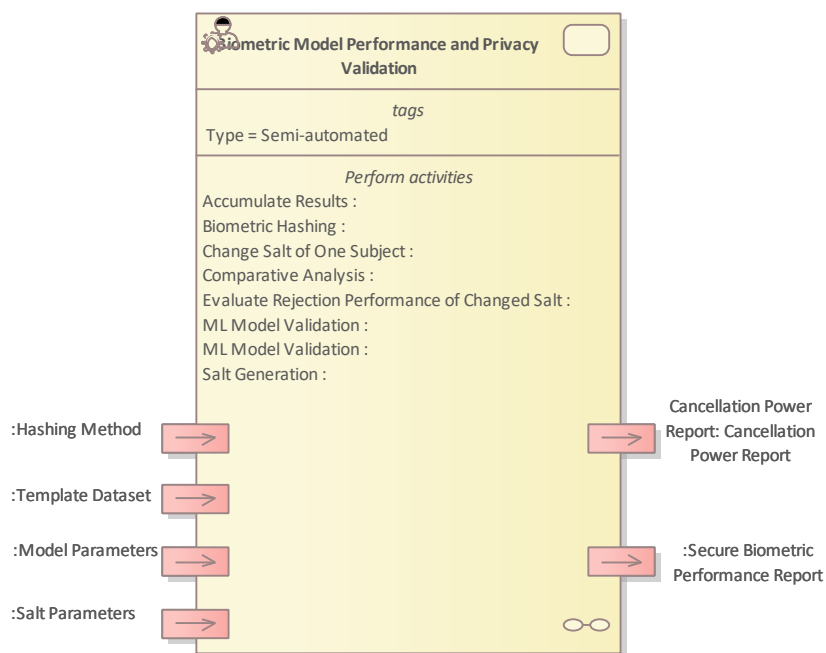


Figure 3.74 Method Definition of Biometric Model Performance and Privacy Validation defined for UC14\_CARDIOID

Figure 3.75 shows the Hardware in the Loop Validation & Verification method definition diagram type of the V&V workflow UC14\_CARDIOID.

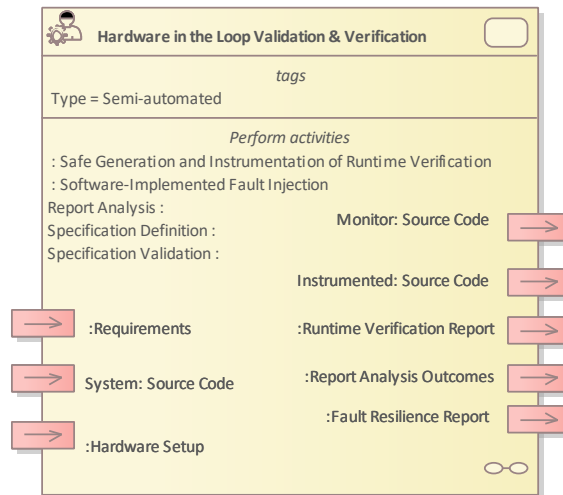


Figure 3.75 Method Definition of Hardware in the Loop Validation & Verification defined for UC14\_CARDIOID

Figure 3.76 above shows the Verification of Driver Monitoring Models Method Definition diagram type of the V&V workflow UC14\_CARDIOID.

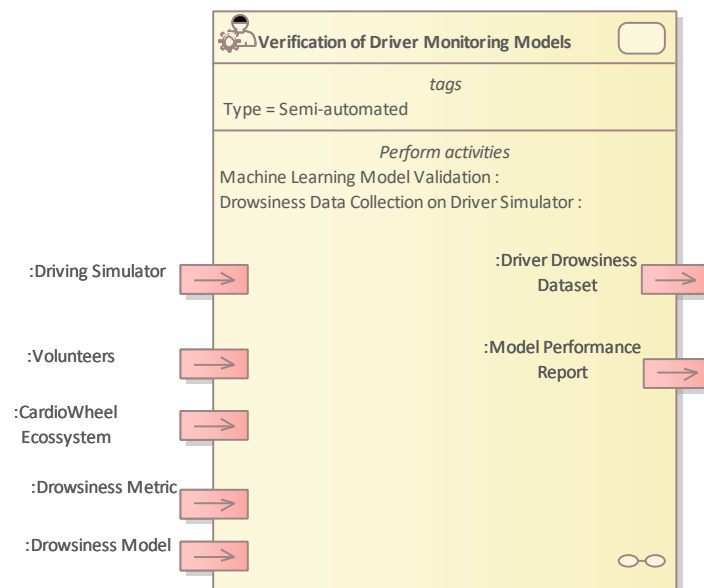


Figure 3.76 Method Definition of Verification of Driver Monitoring Models defined for UC14\_CARDIOID

Figure 3.77 shows the Safe Generation and Instrumentation of Runtime Verification Architectures Method Definition diagram type of the V&V workflow UC14\_CARDIOID.

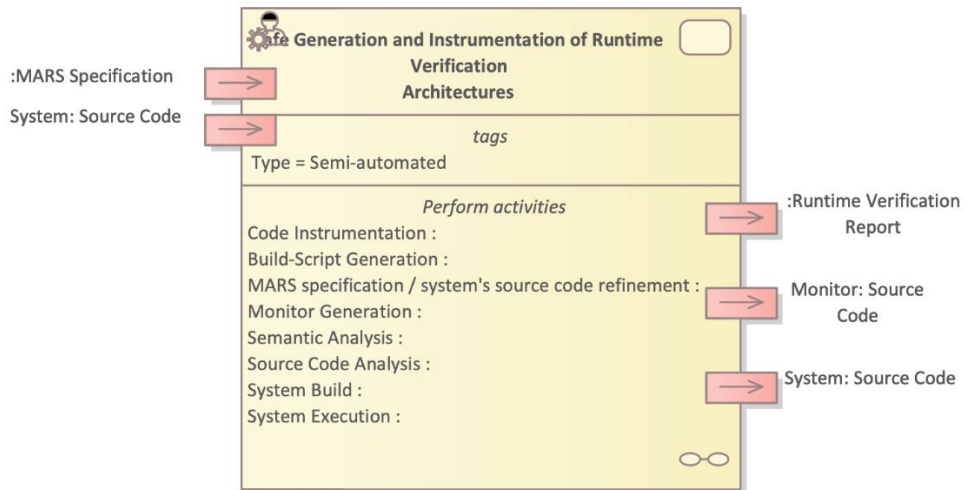


Figure 3.77 Method Definition of Safe Generation and Instrumentation of Runtime Verification Architectures defined for UC14\_CARDIOID

Figure 3.78 shows the Software-Implemented Fault Injection Method Definition diagram type of the V&V workflow UC14\_CARDIOID.

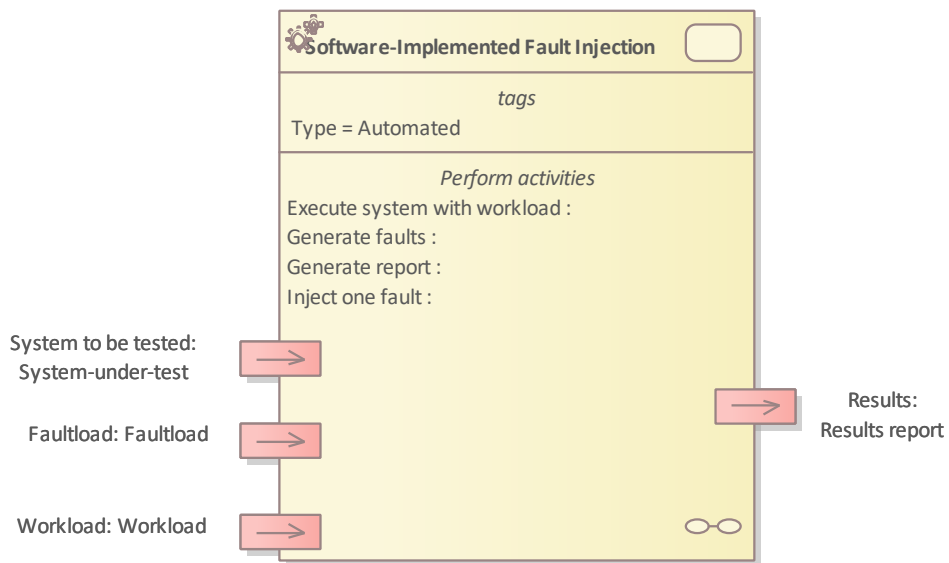


Figure 3.78 Method Definition of Software-Implemented Fault Injection defined for UC14\_CARDIOID

Details on the workflows are given in the following subsections.

### 3.13.1 Artifacts used in UC14\_CARDIOID

UC14\_Artifacts package contains the artifacts defined and used in the workflows for UC14.

Table 3.49 lists the artifacts used for the workflow(s) defined for UC14\_CARDIOID.

Table 3.49 List of artifact types used in UC14\_CARDIOID

Name	Description
Cancellation Power Report (Information)	Report on the success of the system in blocking access from cancelled templates.
CardioWheel Ecosystem (Active Unit)	System used to acquire volunteer's biosignals.
Driver Drowsiness Dataset (Information)	Dataset containing both ECG recordings during driving sessions, as well as independent drowsiness related metrics.
Driving Simulator (Active Unit)	Driving Simulator where a volunteer can be immersed in a realistic driving environment while being instrumented to have biosignals acquired during the driving sessions.
Drowsiness Metric (Information)	Auxiliary signal that allows independent inference of drowsiness levels.
Drowsiness Model (Active Unit)	Machine Learning model that estimates the drowsiness level from acquired ECG signal.
Failure Classification (Information)	Information characterizing the failure that was detected during the experiment run (or information that no failure was detected). This information can include the failure mode classification and further details regarding the failure.
Fault details (Information)	Information about the fault that was injected, e.g., the affected register, affected bit, pre-injection value of the register, injection timestamp, etc.
Fault Resilience Report (Information)	Report describing the effects that each fault injection had on the system.
Faultload (Information)	Faultload defines how to emulate the type of faults that we want to study, Usually these emulate either hardware (e.g., soft errors in the CPU) or software faults.
Hardware Setup (Active Unit)	Hardware setup where runtime verification should run on.
Hashing Method (Active Unit)	Hashing method that obfuscates original biometric features while allowing comparisons in the hashed space.
MARS Specification (Information)	This artifact represents a formal specification of what the monitor will verify in the target system.
Model Parameters (Information)	Parameters for the biometric models.
Model Performance Report (Information)	Report containing performance metrics of the model, including Matthews Correlation Coefficient and Safety Cage systems related results.
Monitor's Source Code (Active Unit)	Code definition of produced runtime monitors.
Report Analysis Outcomes (Information)	Outcomes from the analysis of Monitor's report, returning either a validation certificate, a list of found failures, or a list of found abnormal failures that demand a re-iteration of specification definition.

Name	Description
Requirements (Information)	Set of system requirements related with runtime timing properties.
Results report (Information)	The results report contains information about the failure modes and probabilities, and overall dependability, of the system-under-test.
Runtime Verification Report (Information)	Compilation of results from running a system instrumented with a set of runtime monitors.
Salt Parameters (Information)	Parameters for salt generation.
Secure Biometric Performance Report (Information)	Report on the performance of the biometric models using obfuscated templates, compared with models using the original templates.
Source Code (Active Unit)	Source code of some software artifact. This is used in the context of full systems, monitors, and instrumented systems.
System-under-test (Active Unit)	The system being tested. Can be a prototype or the final version.
Template Dataset (Information)	Dataset containing ECG templates associated with an identity.
Volunteers (Active Unit)	Subjects that volunteer to have their data acquired.
Workload (Active Unit)	The workload that will be executed in the System-under-test while fault injection is taking place. It exercises the system to foment faults to propagate into failures. The workload should be the same, or at least similar, to the workloads that will be executed in the system when fault injection is not being performed.

### 3.13.2 V&V Workflows of Biometric Model Performance and Privacy Validation

This Workflow is designed to provide a platform to test and validate both the performance and the privacy/security properties of biometric models and biometric template hashing methodologies.

With an annotated database of ECG (ElectroCardioGraphic) templates, a biometric model parametric definition and a hashing function, the model performance using raw (unprotected) and hashed features are compared to measure the impact that hashing has on the system's capability to correctly identify enrolled subjects.

Furthermore, hashed template cancellation is performed, and the cancelled template is provided as an input to ensure that it is rejected, validating the system's template cancellation feature.

Figure 3.79 shows the workflow specification diagram of Biometric Model Performance and Privacy Validation.

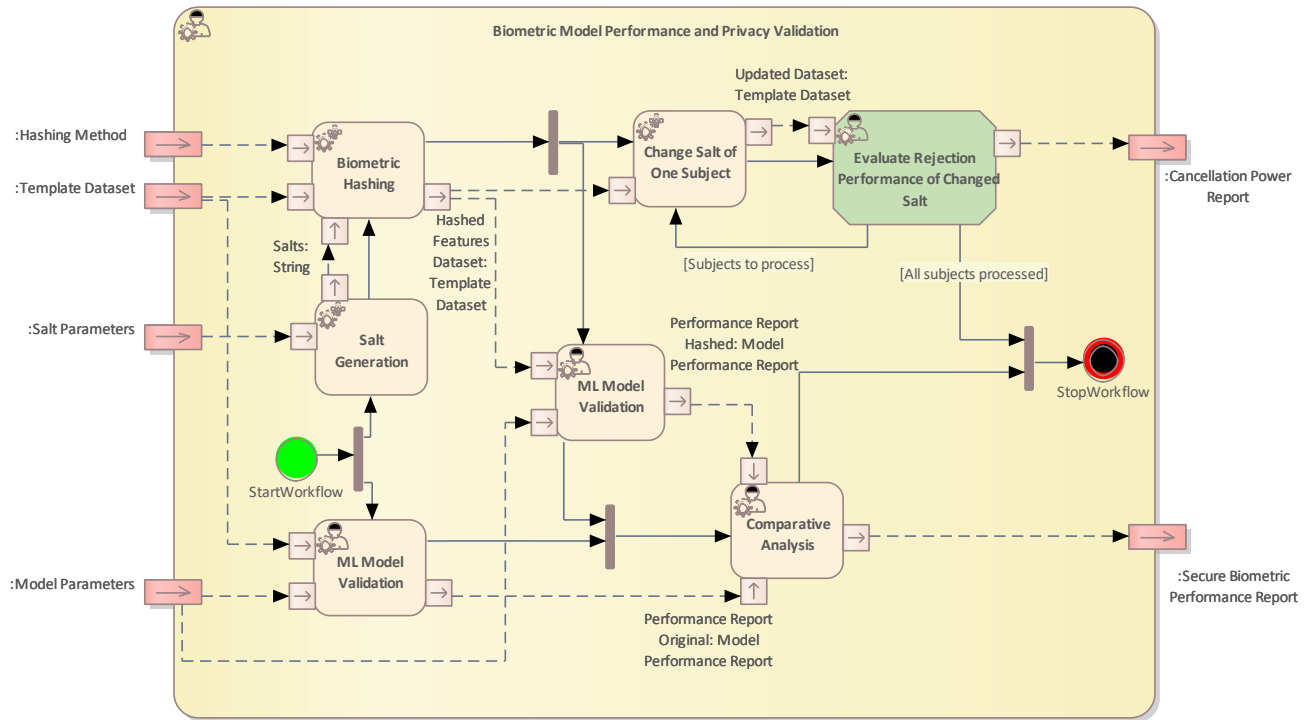


Figure 3.79 Workflow Definition diagram of Biometric Model Performance and Privacy Validation used in UC14\_CARDIROID

Table 3.50 lists the activities of the workflow Biometric Model Performance and Privacy Validation.

Table 3.50 List of activities performed by Biometric Model Performance and Privacy Validation

Name	Type	Description
Biometric Hashing	Automated	Hash the biometric templates using the generated salts.
Change Salt of One Subject	Automated	Re-hash the biometric templates with a different salt. This is done iteratively for each of the subjects present in the database.
Comparative Analysis	Semi-automated	Compare the results of ML Model Validation to characterize the impact of hashing in biometric performance.
Evaluate Rejection Performance of Changed Salt	Semi-automated	Assert that the changed salt results in denied access, even when using the same biometric templates.
ML Model Validation	Semi-automated	Evaluate ML performance metrics for biometry, namely EER and FAR.
Salt Generation	Automated	Generate a random salt to append to the biometric features in the hashing process.

### 3.13.3 V&V Workflows of Hardware in the Loop Validation & Verification

This V&V workflow intends to streamline the validity of real-time properties of the embedded systems of CardioWheel that are hard or even impossible to verify through model checking or static code analysis. Starting from the list of requirements related with runtime and timing properties, and the system's source code, a set of formal specifications written in MARS will be defined. A verification step of these specifications prompts the iteration of requirements, assuring that no conflicts arise. After refining the requirements and producing a final set of specification, those specifications and the system's source code are used to run the method "Runtime Verification Based on Formal specifications" on an hardware setup that emulates the embedded systems where the final validated product is deployed. This method returns the code definitions of the different monitors, an instrumented version of the system's code, and a report that details the findings of such monitors, either validating or finding errors/fragilities in the real time properties of the system. With this report, a rapid step of analysis is performed, either deciding that the system is fully validated, that it needs reworking of the system's source code, or, if abnormal errors are encountered, that further requirement refinement is needed. In parallel, a fault injection method is used to test the system's resilience to faults.

Figure 3.80 shows the workflow specification diagram of Hardware in the Loop Validation & Verification.

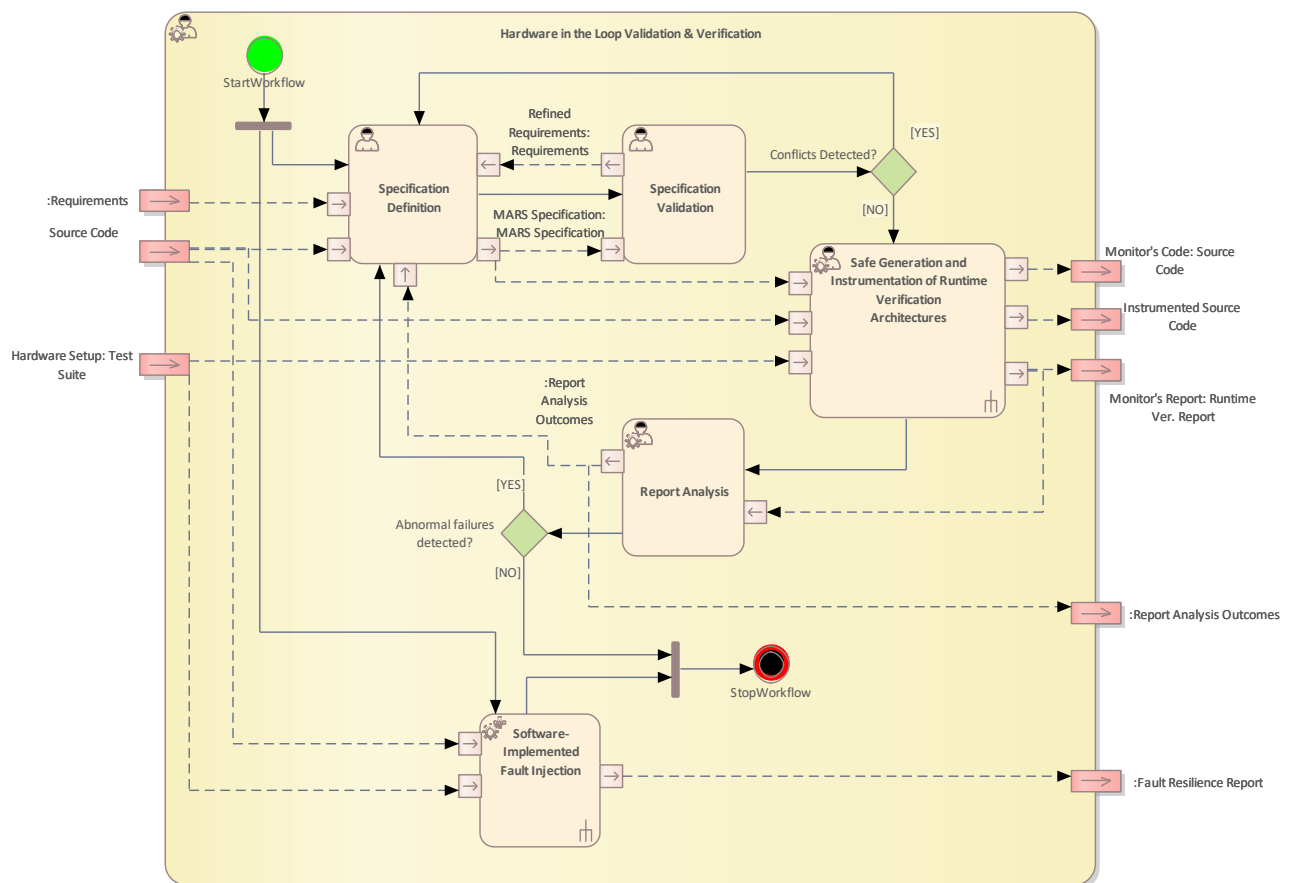


Figure 3.80 Workflow Definition diagram of Hardware in the Loop Validation & Verification used in UC14\_CARDIOID



Table 3.51 lists the activities of the workflow Hardware in the Loop Validation & Verification.

Table 3.51 List of activities performed by Hardware in the Loop Validation & Verification

Name	Type	Description
Activity: Safe Generation and Instrumentation of Runtime Verification Architectures (CallBehavior)	Semi-Automated	Execution of the system under test together with the runtime monitors generated by MARS.
Activity: Software-Implemented Fault Injection (CallBehavior)	Automated	<p>Purpose: The purpose of software implemented fault injection is the deliberate insertion of upsets (faults or errors) in computer systems and/or components to evaluate its behaviour in the presence of faults or validate specific fault tolerance mechanisms in the target system</p> <p>Description: Software-implemented fault injection, abbreviated as SWIFI, uses a variety of software-based techniques for inserting faults or errors in a system-under-study.</p>
Report Analysis	Semi-automated	This activity quickly evaluates the full report produced by the method "Runtime verification based on formal specifications" to identify abnormal failure points that might indicate incorrect requirement definition, prompting a return to the specification definition step.
Specification Definition	Manual	Re-writing of system requirements in MARS domain language to be integrated as monitor specifications within the instrumented code produced in the method Runtime Verification Based on Formal Specifications.
Specification Validation	Manual	Validation step that checks the produced specification set for conflicts or lack of coverage.

### 3.13.4 V&V Workflows of Safe Generation and Instrumentation of Runtime Verification Architectures

Development of a toolchain to (1) formally specify monitors and their deployment environment, and to (2) generate monitors that comply with safety properties of a target system, according to the corresponding VVML diagram.

When making use of monitors in critical systems, it must be ensured that they neither negatively influence the security aspects of the original system nor affect the functional and the safety non-functional requirements of the system (e.g., task scheduling). Guaranteeing that the deployment of such solutions does not negatively influence the dependability properties of systems can be overly complicated and time-consuming when no proper integration methods are used.

To abstract the formalities of correctly integrating monitoring architectures in the target system and reduce the steep learning curve associated with the usage of formal specification languages, which is common with Runtime Verification (RV) based on formal specifications, we propose a new domain specific language (and associated tools) named MARS. MARS will let developers focus what needs to be monitored instead of worrying about how to safely integrate such monitoring solutions to their target systems. To achieve that, MARS will allow users to associate RV specifications with the components of a target system, providing support for a timing analysis over the combined system coupled with the instrumented monitors. MARS will ensure compliance with timeliness requirements and will support the generation of monitors from the formal specifications following a correct-by-construction approach. The generated monitors will be coupled with the target system via a runtime monitoring architecture that will link the interfaces of the system with those of the generated monitors.

Figure 3.81 shows the workflow specification diagram of Safe Generation and Instrumentation of Runtime Verification Architectures.

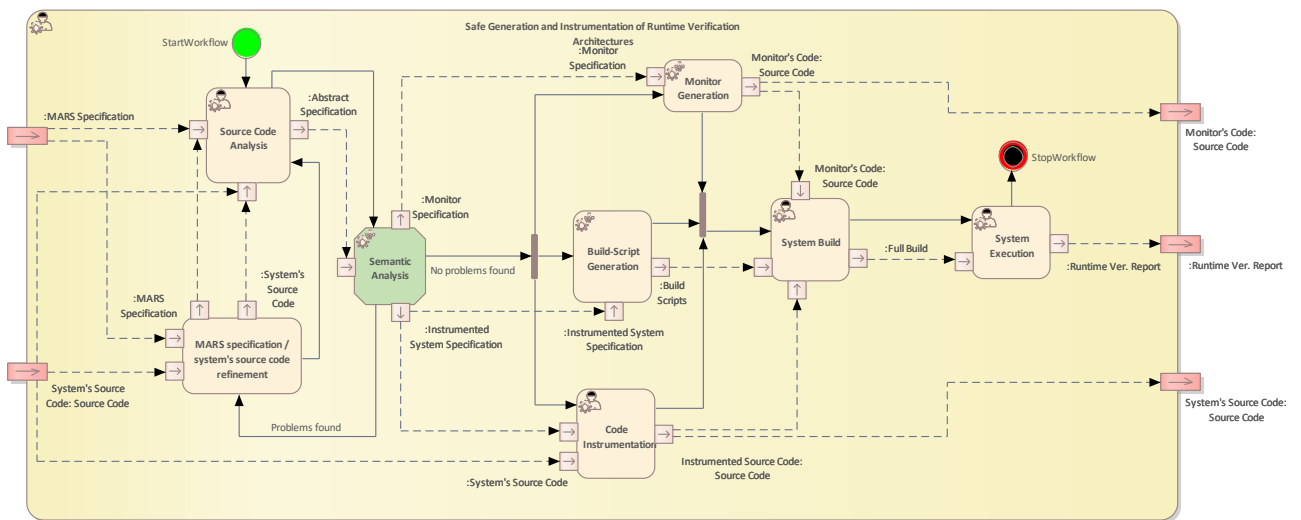


Figure 3.81 Workflow Definition diagram of Safe Generation and Instrumentation of Runtime Verification Architectures used in UC14\_CARDIOID

Table 3.52 lists the activities of the workflow Safe Generation and Instrumentation of Runtime Verification Architectures.

Table 3.52 List of activities performed by Safe Generation and Instrumentation of Runtime Verification Architectures

Name	Type	Description
Build-Script Generation	Automated	This step generates a set of scripts instructing the compiler to instrument the generated monitors in the target system's source code for a given user-selected platform.
Code Instrumentation	Semi-automated	This step performs modifications on the source code, adapting the system to incorporate the generated monitors in the next steps.

Name	Type	Description
MARS specification / system's source code refinement	Semi-Automated	Whenever the semantic analysis identifies safety problems (e.g., the monitors would turn the system not schedulable), the developers must perform refinements in the MARS specification and/or the system's source code. The refined inputs are then fed back to the toolchain for another round of analysis.
Monitor Generation	Automated	Given a formal specification of what a target system should do, this step generates a correct-by-construction standalone monitor in a target language.
Semantic Analysis	Automated	In this step, a set of static formal verification is performed to verify the compliance of the integration of the specified monitors into the target system and the target system's safety requirements. Examples of possible safety verifications include, but are not limited to, schedulability analysis of real-time systems and memory checks.
Source Code Analysis	Semi-automated	This initial step is responsible for analysing the monitor specifications written in MARS and the system's source code and abstracting them for the analysis performed in the next step.
System Build	Semi-automated	This step builds the final system by binding the generated monitors, the original system's source code, the instructions on how these monitors should be integrated into the system, and the target platform where the system will run.
System Execution	Semi-automated	This step describes the actual execution of a system instrumented with monitors. These monitors will verify (following a user-defined periodicity/trigger policy) if the system is performing as expected and issue verdicts about it.

### 3.13.5 V&V Workflows of Software-Implemented Fault Injection

This workflow details the process needed to continuously build a robust dataset against which driver monitoring mod

The method takes the system-under-test, the faultload and the workload as input and produces a results report describing the failure modes and probabilities and other dependability-related metrics of the system.

Figure 3.82 shows the workflow specification diagram of Software-Implemented Fault Injection.

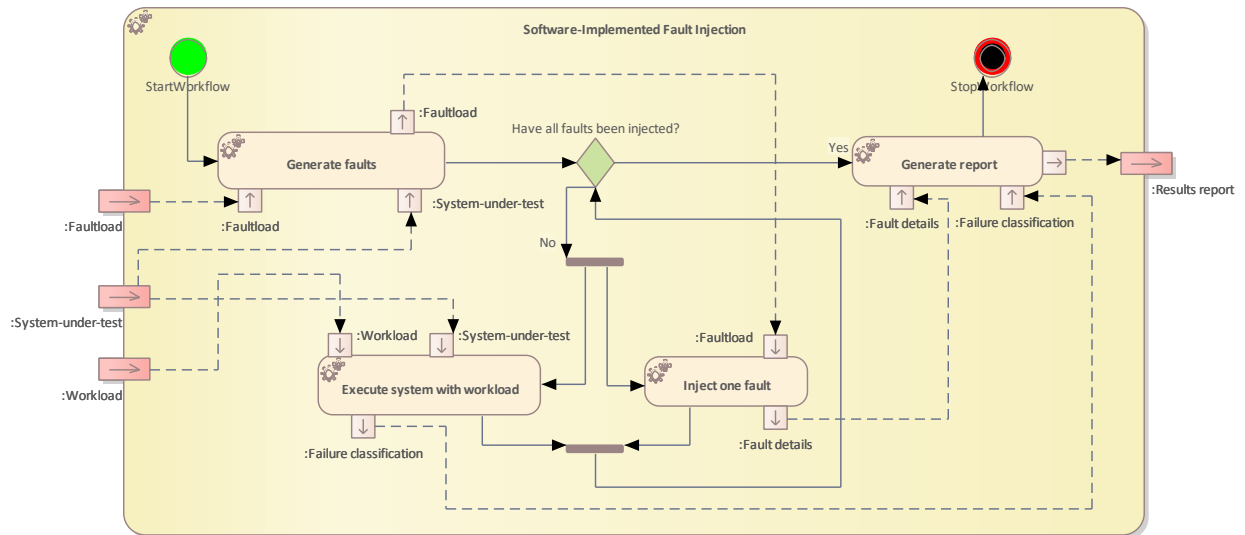


Figure 3.82 Workflow Definition diagram of Software Implemented Fault Injection used in UC14\_CARDIOID

Table 3.53 lists the activities of the workflow Software-Implemented Fault Injection.

Table 3.53 List of activities performed by Software-Implemented Fault Injection

Name	Type	Description
Execute system with workload	Automated	Execute the system-under-test with the chosen workload.
Generate faults	Automated	Generate the faults that will be injected in the system-under-test according to the faultload and the workload.
Generate report	Automated	After all the faults have been injected, compile the results into a report that should contain, at least, information about the experienced failure modes and their probabilities. This report can include more detailed information, such as which faults were more likely to cause failures, what was the failure latency and others.
Inject one fault	Automated	While the workload is executing, choose one of the faults from the set of faults that are yet to be injected and inject it in the system.

### 3.13.6 V&V Workflows of Verification of Driver Monitoring Models

This workflow details the process needed to continuously build a robust dataset against which driver monitoring models, such as drowsiness detection, can be tested.

The continuous update of the database serves to facilitate constant development of the driver monitoring models, making them more robust against different physiological signal measurement conditions, as well as to mitigate inter-subject variability related uncertainty within these models.

Figure 3.83 shows the workflow specification diagram of Verification of Driver Monitoring Models.

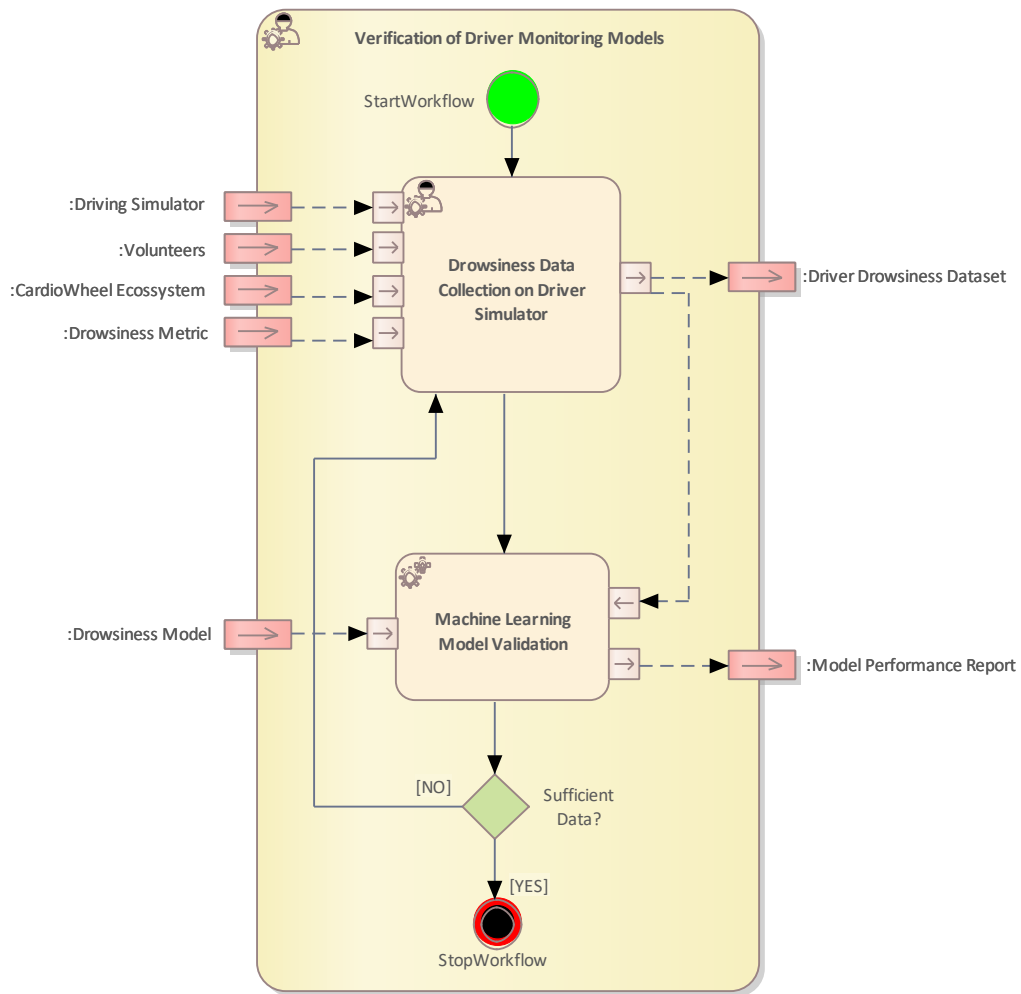


Figure 3.83 Workflow Definition diagram of Verification of Driver Monitoring Models used in UC14\_CARDIOID

Table 3.54 lists the activities of the workflow Verification of Driver Monitoring Models.

Table 3.54 List of activities performed by Verification of Driver Monitoring Models

Name	Type	Description
Drowsiness Data Collection on Driver Simulator	Semi-automated	Driver Drowsiness data collection protocol, using a driving simulator where real drivers have their ECG collected, as well as other biosignals to infer drowsiness levels.
Machine Learning Model Validation	Automated	Measurement of ML model performance when using an annotated dataset, as well as behaviour observation against corrupted and unobserved data when implementing Safety Cage systems.



## 4 Conclusion

This document shows the final results of Task 4.2: the approach for the workflow modelling of V&V activities (VVML) and the final state of the 42 V&V workflows that have been developed for the project use cases using VVML.

The modelling language VVML has been developed as a domain-specific language for describing validation and verification activities. It enables the design of re-usable workflow assets such as V&V activities and artifacts that are exchanged between workflows. VVML is applicable and tailorable to different industrial domains and their specific constraints, V&V methods, and toolchains. The outcome of the V&V workflow modelling activity for all use cases is shown and described in the document.

In order to facilitate the modelling with VVML, appropriate tool support has been provided. A dedicated plug-in for the state-of-the-art modelling tool Enterprise Architect (EA) has been developed and adapted to the needs of the test experts in the different use cases. The wide use of VVML models and results from EA is supported by the export function to standardized XML/XMI formats, which can be processed by other modelling tools.

The main benefit of the workflow modelling approach is to enable system and domain experts as well as V&V experts to design, discuss, and improve V&V activities in the development processes of complex technical systems. The notation is kept simple with few modelling elements and the ability to reuse existing workflows and gradually compose them into more complex V&V processes. The applicability of the approach to industrial use cases with different quality properties, V&V methods, tools, and levels of granularity have been shown in the final phase of Task 4.2.

Greater effort has been devoted to ongoing trainings and reviews to assist users in starting, improving, and optimizing their V&V modelling activities. The rules, guidelines, and lessons learnt will be compiled in the VVML handbook, a document that will be made available to the community at the end of the VALU3S project.

The V&V Workflows will be implemented and supported by V&V tools and tool chains, which have been developed and extended in Task 4.3 of the project. An important prerequisite for the application of tools and tool chain in the development process for complex safety-related systems is the so-called tool qualification, which is a formal process for demonstrating the quality of tools and tool chains with regard to defined properties. Challenges and possible strategies for tool qualification of VALU3S V&V tools and tool chains are being discussed between the partners. The results will be prepared and published as part of the final activities and deliverables in WP5, which deals with project demonstrators and evaluation.



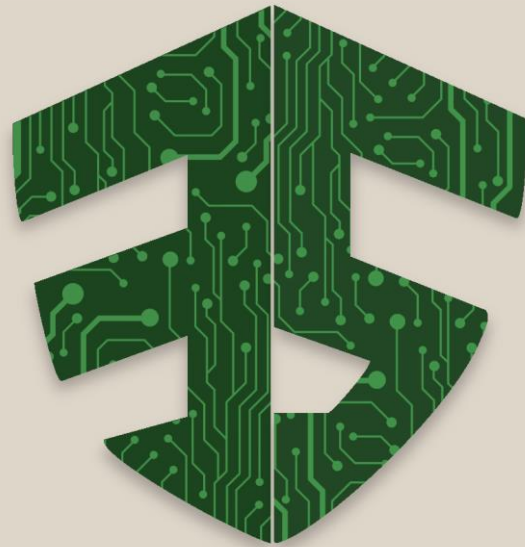


## References

- [1] Fraunhofer IESE et al., “Deliverable D4.6 - Interim detailed description of improved process workflows,” VALU3S Consortium, 2022.
- [2] Fraunhofer IESE et al., “Deliverable D4.3 - Concept for using virtual prototypes (e.g., digital twins) in the context of the simulated verification,” VALU3S Consortium, 2021.
- [3] Siemens, “Active Safety System Simulation | Siemens Software,” [Online]. Available: <https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html>.
- [4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, “CARLA: An Open Urban Driving Simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.
- [5] LG, “An autonomous vehicle simulator based on Unity,” [Online]. Available: <https://www.lgsvlsimulator.com/>.
- [6] GmbH, BeamNG, “A dynamic soft-body physics vehicle simulator capable of doing just about anything,” [Online]. Available: <https://www.beamng.gmbh/research>.
- [7] ISO (International Standard Organization), *ISO 21448:2022 - Road vehicles — Safety of the intended functionality*, ISO, 2022.
- [8] AMLAS, “Guidance on the Assurance of Machine Learning in Autonomous Systems (AMLAS)”.
- [9] UNECE, “UNECE World Forum for Harmonization of Vehicle Regulations (WP.29),” [Online]. Available: <https://unece.org/wp29-introduction>. [Accessed 2022].
- [10] S. Christoph, S. Bernhard and K. Sandra, “Asset Driven ISO/SAE 21434 Compliant Automotive Cybersecurity Analysis with ThreatGet,” in *CCIS, volume 1442*, 2021.
- [11] T. Kuhn, T. Forster, T. Braun and R. Gotzhein, “Feral - framework for simulator coupling on requirements and architecture level,” *ACM/IEEE MEMOCODE*, p. 11–22, 2013.
- [12] D. Giannakopoulou et al., “Generation of Formal Requirements from Structured Natural Language,” in *Requirements Engineering: Foundation for Software Quality. REFSQ 2020*, 2020.

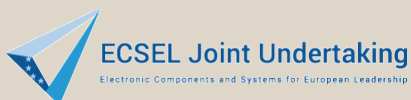
- [13] A. Champion et al., “CoCoSpec: A Mode-Aware Contract Language for Reactive Systems,” in *Software Engineering and Formal Methods. SEFM 2016*, 2016.
- [14] *EN 50129 Railway applications. Communication, signalling and processing systems. Safety related electronic systems for signalling*, CENELEC, 2018.
- [15] AIT and TU Graz, “MoMut,” 2022. [Online]. Available: <https://momut.org/>.
- [16] A. David, K. G. Larsen, A. Legay, M. Mikušionis and D. B. Poulsen, “Uppaal SMC tutorial,” *STTT*, vol. 17, no. 4, pp. 397-415, 2015.
- [17] “Vivado Design,” 2022. [Online]. Available: <https://www.xilinx.com/products/design-tools/vivado.html>.
- [18] ISO (International Standard Organization), *ISO 10218-2:2011 - Robots and robotic devices — Safety requirements for industrial robots*, ISO, 2011.
- [19] ISO (International Standard Organization), *ISO/TS 15066:2016 - Robots and robotic devices — Collaborative robots*, ISO, 2016.
- [20] Y. Yang et al., “Man-in-the-middle attack test-bed investigating cyber-security vulnerabilities in smart grid SCADA systems.,” in *International Conference on Sustainable Power Generation and Supply (SUPERGEN 2012)*, 2012.
- [21] A. Bechtsoudis and N. Sklavos, “Aiming at higher network security through extensive penetration tests,” *IEEE Latin america transactions*, vol. 10, no. 3, pp. 1752-1756, 2012.
- [22] “Denial-of-service attack,” [Online]. Available: [https://en.wikipedia.org/wiki/Denial-of-service\\_attack](https://en.wikipedia.org/wiki/Denial-of-service_attack).
- [23] M. e. a. Denis, “Penetration testing: Concepts, attack methods, and defense strategies,” in *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016.
- [24] L. E. Bassham et al., “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications - SP 800-22 Rev. 1a.,” National Institute of Standards and Technology, 2010..
- [25] P. L'Ecuyer and R. Simard, “TestU01: A C library for empirical testing of random number generators,” *ACM Trans. Math. Softw.*, vol. 33, no. 4, 2007.
- [26] “Die Harder Web-site,” [Online]. Available: <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>.

- [27] H. Kaysici and S. Ergün, "Random Number Generator Based on Metastabilities of Ring Oscillators and Irregular Sampling," in *27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020.
- [28] B. Acar and S. Ergün, "A Robust Digital Random Number Generator Based on Transient Effect of Ring Oscillator," in *IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, 2020.
- [29] S. Ergün, "Attack on a Microcomputer-Based Random Number Generator Using Auto-synchronization," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2019.
- [30] D. Nickovic, "RTAMT Web-site," [Online]. Available: <https://github.com/nickovic/rtamt>.



**VALU3S**

[www.valu3s.eu](http://www.valu3s.eu)



This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.