

PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE CRYSTAL CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE CESAR CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT. THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.



CRritical **SY**STem Engineering **Acce**Leration

**Specification, Development and Assessment for Safety
Engineering - V1
D604.011**

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Specification, Development and Assessment for Safety Engineering - V1
Deliverable No.	D604.011
Dissemination Level	CO
Nature	P
Document Version	V1.00
Date	2014-02-07
Contact	Rupert Schlick
Organization	AIT
Phone	+43 50 550 – 4124
E-Mail	rupert.schlick@ait.ac.at

AUTHORS TABLE

Name	Company	E-Mail
Rupert Schlick	AIT	rupert.schlick@ait.ac.at
Thomas Gruber	AIT	thomas.gruber@ait.ac.at
Egbert Althammer	AIT	egbert.althammer@ait.ac.at
Frédérique Vallée	All4Tec	frederique.vallee@all4tec.net
Elie Soubiran	ALSTOM	Elie.Soubiran@transport.alstom.com
Vidal-delmas Tchapet-Nya	ALSTOM	vidal-delmas.tchapet-nya-ext@transport.alstom.com
Andreas Mitschke	EADS IW G	Andreas.Mitschke@eads.net
Marco Bozzano	FBK	bozzano@fbk.eu
Stefano Tonetta	FBK	tonettas@fbk.eu
Soeren Kemmann	Fraunhofer IESE	Soeren.Kemmann@iese.fraunhofer.de
Santiago Velasco	Fraunhofer IESE	santiago.velasco@iese.fraunhofer.de
Maria del Carmen Lomba Sorrondegui	GMV	mclomba@gmv.com
Elena Alaña	GMV	ealana@gmv.com
Serrie Chapman	Infineon UK	serrie.chapman@infineon.com
Aleksander Lodwich	ITK Engineering	aleksander.lodwich@itk-engineering.de
Daniel Sauter	ITK Engineering	daniel.sauter@itk-engineering.de
Jiri Barnat	MU	barnat@fi.muni.cz
Susana Pérez Sanchez	Tecnalia	susana.perezsanchez@tecnalia.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	05.11.2013	Initial document structure	All
0.02	10.12.2013	Initial inputs from AIT, All4Tec, Alstom, FBK, GMV, ITKE, MU, IESE, Tecnalía; integration/consolidation	All
0.03	02.01.2014	Integrated updates, consolidated layout, ready for WP internal review	All
0.04	14.01.2014	ITKE input merged, review remarks for the whole document	All
0.05	20.01.2014	Integrated EADS and IESE input and responses to review remarks from WP internal review	All
0.06	05.02.2014	Integrated responses to review remarks from WP external review	All
1.00	06.02.2014	Finalization (Layout etc.)	All

CONTENT

1	INTRODUCTION.....	12
1.1	ROLE OF DELIVERABLE	12
1.2	OVERVIEW OF THE WORK PACKAGE.....	12
1.3	RELATIONSHIP TO OTHER CRYSTAL DOCUMENTS	14
1.4	STRUCTURE OF THIS DOCUMENT	14
2	WEFACT	16
2.1	OVERVIEW	16
2.1.1	<i>General Description.....</i>	<i>16</i>
2.1.2	<i>Related Use Cases.....</i>	<i>17</i>
2.2	SPECIFICATION	18
2.2.1	<i>Requirements from the UCs.....</i>	<i>18</i>
2.2.2	<i>How will this brick be integrated in the UC.....</i>	<i>19</i>
2.2.3	<i>Requirements fulfilled by initial tool/method version.....</i>	<i>20</i>
2.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	<i>20</i>
2.3	IMPLEMENTATION/ELABORATION.....	21
2.4	EVALUATION.....	21
3	MB RAMS	22
3.1	OVERVIEW	22
3.1.1	<i>General Description.....</i>	<i>22</i>
3.1.2	<i>Related Use Cases.....</i>	<i>23</i>
3.2	SPECIFICATION	23
3.2.1	<i>Requirements from the UCs.....</i>	<i>23</i>
3.2.2	<i>How will this brick be integrated in the UC.....</i>	<i>24</i>
3.2.3	<i>Requirements fulfilled by initial tool/method version.....</i>	<i>25</i>
3.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	<i>25</i>
3.3	IMPLEMENTATION/ELABORATION.....	28
3.4	EVALUATION.....	28
4	MOMUT::UML.....	29
4.1	OVERVIEW	29
4.1.1	<i>General Description.....</i>	<i>29</i>
4.1.2	<i>Related Use Cases.....</i>	<i>32</i>
4.2	SPECIFICATION	33
4.2.1	<i>Requirements from the UCs.....</i>	<i>33</i>
4.2.2	<i>How will this brick be integrated in the UC.....</i>	<i>35</i>
4.2.3	<i>Requirements fulfilled by initial tool/method version.....</i>	<i>35</i>
4.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	<i>35</i>
4.3	IMPLEMENTATION/ELABORATION.....	37
4.4	EVALUATION.....	38
5	MOMUT::SCADE.....	39
5.1	OVERVIEW	39
5.1.1	<i>General Description.....</i>	<i>39</i>
5.1.2	<i>Related Use cases.....</i>	<i>39</i>
5.2	SPECIFICATION	39
5.2.1	<i>Requirements from the UCs.....</i>	<i>39</i>
5.2.2	<i>How will this brick be integrated in the UC.....</i>	<i>39</i>
5.2.3	<i>Requirements fulfilled by initial tool/method version.....</i>	<i>40</i>



5.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	40
5.3	IMPLEMENTATION/ELABORATION	41
5.4	EVALUATION	41
6	FT+ BRICK INTEGRATION INTO RATIONAL RHAPSODY	42
6.1	OVERVIEW	42
6.1.1	<i>General Description</i>	42
6.1.2	<i>Related Use cases</i>	42
6.2	SPECIFICATION	42
6.2.1	<i>Requirements from the UCs</i>	42
6.2.2	<i>How will this brick be integrated in the UC</i>	44
6.2.3	<i>Requirements fulfilled by initial tool/method version</i>	50
6.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	50
6.3	IMPLEMENTATION/ELABORATION	50
6.4	EVALUATION	51
7	NUSMV	52
7.1	OVERVIEW	52
7.1.1	<i>General Description</i>	52
7.1.2	<i>Related Use cases</i>	52
7.2	SPECIFICATION	53
7.2.1	<i>Requirements from the UCs</i>	53
7.2.2	<i>How will this brick be integrated in the UC</i>	54
7.2.3	<i>Requirements fulfilled by initial tool/method version</i>	56
7.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	56
7.3	IMPLEMENTATION/ELABORATION	57
7.4	EVALUATION	57
8	C²FT	58
8.1	OVERVIEW	58
8.1.1	<i>General Description</i>	58
8.1.2	<i>Related Use cases</i>	58
8.2	SPECIFICATION	59
8.2.1	<i>Requirements from the UCs</i>	59
8.2.2	<i>How will this brick be integrated in the UC</i>	59
8.2.3	<i>Requirements fulfilled by initial tool/method version</i>	59
8.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	60
8.3	IMPLEMENTATION/ELABORATION	61
8.4	EVALUATION	61
9	SAFETY FOR AVIONIC DESIGN AND ANALYSIS FRAMEWORK	62
9.1	OVERVIEW	62
9.1.1	<i>General Description</i>	62
9.1.2	<i>Related Use cases</i>	62
9.2	SPECIFICATION	63
9.2.1	<i>Requirements from the UCs</i>	63
9.2.2	<i>How will this brick be integrated in the UC</i>	67
9.2.3	<i>Requirements fulfilled by initial tool/method version</i>	67
9.2.4	<i>What will be implemented/provided in the CRYSTAL project</i>	67
9.3	IMPLEMENTATION/ELABORATION	67
9.4	EVALUATION	67
10	CLAIMS LANGUAGE BOILERPLATE	68
10.1	OVERVIEW	68



10.1.1	General Description.....	68
10.1.2	Related Use cases.....	68
10.2	SPECIFICATION	69
10.2.1	Requirements from the UCs.....	69
10.2.2	How will this brick be integrated in the UC.....	70
10.2.3	Requirements fulfilled by initial tool/method version.....	71
10.2.4	What will be implemented/provided in the CRYSTAL project	71
10.3	IMPLEMENTATION/ELABORATION.....	71
10.4	EVALUATION.....	71
11	DAD DATA ANALYSER DASHBOARD.....	72
11.1	OVERVIEW	72
11.1.1	General Description.....	72
11.1.2	Related Use cases.....	72
11.2	SPECIFICATION	72
11.2.1	Requirements from the UCs.....	72
11.2.2	How will this brick be integrated in the UC.....	73
11.2.3	Requirements fulfilled by initial tool/method version.....	74
11.2.4	What will be implemented/provided in the CRYSTAL project	74
11.3	IMPLEMENTATION.....	75
11.4	EVALUATION.....	75
12	RECOMMENDED METHODOLOGY ACCORDING ISO 26262	76
12.1	OVERVIEW	76
12.1.1	General Description.....	76
12.1.2	Related Use Cases.....	78
12.2	SPECIFICATION	78
12.3	IMPLEMENTATION.....	78
12.4	EVALUATION.....	78
13	RISK ASSESSMENT AND HAZARD ANALYSIS.....	79
13.1	OVERVIEW	79
13.1.1	General Description.....	79
13.1.2	Related Use Cases.....	80
13.2	SPECIFICATION	80
13.3	IMPLEMENTATION.....	80
13.4	EVALUATION.....	81
14	FTA, FMEA AND FMEDA (ITKE)	82
14.1	OVERVIEW	82
14.1.1	General Description.....	82
14.1.2	Related Use Cases.....	82
14.2	SPECIFICATION	83
14.2.1	Requirements from the UC.....	83
14.3	IMPLEMENTATION.....	86
14.4	EVALUATION.....	86
15	FEATURE DOCUMENTATION IN MODEL BASED SOFTWARE.....	87
15.1	OVERVIEW	87
15.2	SPECIFICATION	88
15.3	IMPLEMENTATION.....	88
15.4	EVALUATION.....	88
16	ISOGRAPH FAULTTREE+	89



16.1	OVERVIEW	89
16.2	SPECIFICATION	89
16.3	IMPLEMENTATION.....	89
16.4	EVALUATION.....	89
17	SAFETY ARCHITECT (ALL4TEC).....	90
17.1	OVERVIEW	90
17.1.1	General Description.....	90
17.1.2	Related Use cases.....	91
17.2	SPECIFICATION	91
17.2.1	Requirements from the UCs.....	91
17.2.2	How will this brick be integrated in the UC.....	93
17.2.3	Global process.....	94
17.2.4	Requirements fulfilled by initial tool/method version.....	98
17.2.5	What will be implemented/provided in the CRYSTAL project	98
17.3	IMPLEMENTATION/ELABORATION.....	102
17.4	EVALUATION.....	102
18	MU SAFETY ANALYSIS TOOL (MUSAT)	103
18.1	OVERVIEW	103
18.1.1	General Description.....	103
18.1.2	Related Use cases.....	103
18.2	SPECIFICATION	104
18.2.1	Requirements from the UCs.....	104
18.2.2	How will this brick be integrated in the UC.....	104
18.2.3	Requirements fulfilled by initial tool/method version.....	104
18.2.4	What will be implemented/provided in the CRYSTAL project	104
18.3	IMPLEMENTATION/ELABORATION.....	105
18.4	EVALUATION.....	105
19	SAFETY-ANALYSIS FOR AEROSPACE (ESA STANDARDS)	106
19.1	OVERVIEW	106
19.1.1	General Description.....	106
19.1.2	Related Use cases.....	111
19.2	SPECIFICATION	112
19.2.1	Requirements from the UCs.....	112
19.2.2	How will this brick be integrated in the UC.....	113
19.2.3	Requirements fulfilled by initial tool/method version.....	113
19.2.4	What will be implemented/provided in the CRYSTAL project	114
19.3	IMPLEMENTATION/ELABORATION.....	114
19.4	EVALUATION.....	114
20	AUTONOMOUS FAULT TOLERANT SYSTEM DESIGN METHODOLOGY (AFTS DM).....	115
20.1	OVERVIEW	115
20.1.1	General Description.....	115
20.1.2	Related Use cases.....	118
20.2	SPECIFICATION	118
20.2.1	Requirements from the UCs.....	118
20.2.2	How will this brick be integrated in the UC.....	119
20.2.3	Requirements fulfilled by initial tool/method version.....	120
20.2.4	What will be implemented/provided in the CRYSTAL project	120
20.3	IMPLEMENTATION/ELABORATION.....	121
20.4	EVALUATION.....	121
21	SUMMARY	122



22	TERMS, ABBREVIATIONS AND DEFINITIONS	123
23	REFERENCES	125

List of Figures

Figure 2-1: Overall organization of the WEFACT framework	16
Figure 2-2: Dependencies of V&V management on other artefacts	17
Figure 2-3: Processing of a V&V activity in WEFACT	19
Figure 3-1: Data Flow for MB-RAMS activities	25
Figure 4-1: Mutation-based test case generation is at the heart of MoMuT::UML	30
Figure 4-2: Usage variants of mutation-based TCG tools	30
Figure 4-3: Screenshot of the MoMuT::UML tool	31
Figure 4-4: Automatic test case generation and related data flows	32
Figure 7-1: Failure Conditions retrieval.	54
Figure 7-2: NuSMV invocation.	55
Figure 7-3: Failure Components are sent back.	55
Figure 7-4: Generation of Fault Trees.	56
Figure 8-1: C ² FT	59
Figure 8-2: Open Safety Model (OSM)	60
Figure 9-1: Avionic Design and Analysis Framework	63
Figure 9-2: Core of Avionic Design and Analysis Framework	64
Figure 9-3: Possible extensions of the Safety for Avionic Design and Analysis Framework	65
Figure 9-4: Extension of the framework to cover UC2.1b	66
Figure 10-1: Quality gateway	69
Figure 10-2: DODT	69
Figure 10-3: Manual Quality flow to be replaced	70
Figure 14-1. Example of an X-ray system developed in use-case 4.2	83
Figure 14-2: Overview of interrelations between parts of the safety risk management process	84
Figure 17-1 Relations between the requirements management process and design modelling process	93
Figure 17-2 Requirements analysis process	95
Figure 19-1: ECSS standards hierarchy	107
Figure 20-1: Partial Reconfiguration concept	116
Figure 20-2: Software engineering process	120
Figure 20-3: Dual-FPGA board potential lay-out	121

List of Tables

Table 1-1: Bricks Overview	14
Table 1-2: Related Deliverables	14
Table 2-1: UC5.2 requirements addressed by WEFACT	19
Table 2-2: UC5.2 requirements already fulfilled by current WEFACT version	20
Table 2-3: General improvement requirements for WEFACT	20
Table 2-4: Interoperability and integration requirements for WEFACT	21
Table 3-1: UC5.2 requirements addressed by MB RAMS.....	24
Table 3-2: General improvement requirements for MB RAMS.....	27
Table 3-3: Interoperability and integration requirements for MB RAMS.....	28
Table 4-1: UC5.2 requirements addressed by MoMuT::UML.....	34
Table 4-2: Requirements already fulfilled by current MoMuT::UML version	35
Table 4-3: General improvement requirements for MoMuT::UML.....	36
Table 4-4: Interoperability/Integration requirements for MoMuT::UML	37
Table 5-1: UC5.2 requirements fulfilled by current MoMuT::SCADE version	40
Table 5-2: General improvement requirements for MoMuT::SCADE.....	40
Table 5-3: Interoperability and integration requirements for MoMuT::SCADE	41
Table 17-1 System analysis process.....	96
Table 17-2 Safety analysis process.....	97
Table 17-3 Hazard log	98
Table 17-4 SHA format in Safety Architect.....	101
Table 19-1: Software criticality categories.....	109
Table 19-2: Criticality classification (extracted from ECSS-Q-ST-40C)	109
Table 19-3: SFMEA template	113
Table 22-1: Terms, Abbreviations and Definitions	124

1 Introduction

1.1 Role of deliverable

In this deliverable, all bricks are documented that are developed in the work package WP6.4 “Tools for Safety Engineering”. There will be three iterations of the deliverable corresponding to the milestones of the CRYSTAL project. Therefore the brick documentations in this document represent an evolutionary process of continuous development and enhancement of the CRYSTAL solutions.

The final iteration of the deliverable will contain the specification, development, and use case independent assessment of all bricks of the work package. This first iteration only contains the initial specifications for the bricks. In the next iteration, there will also be work related to commonalities of and synergies between the bricks, especially related to the interoperability aspects.

1.2 Overview of the Work Package

As all the CRYSTAL work packages 6.3-6.13, the purpose of this work package is to develop general improvements and interoperability extensions for a group of CRYSTAL bricks. The bricks collected in this work package are either directly targeting steps in the safety process (methods and tools) or support the verification and validation of systems with a focus on safety critical systems (e.g. because the increased effort can only be argued for mission or safety critical systems).

The respective CRYSTAL main objective is to provide tools and methods for safety analysis and early safety validation of systems and their components. These methods and tools are prepared for the integration into the RTP and for the use in the CRYSTAL use cases.

The work package supports cost-effective and standards-compliant cross-domain re-use of components for safety-relevant systems. The innovations from the bricks shall support incremental cross-domain safety certification for whole systems and their components satisfying the system dependability requirements common to safety standards like DO 178B, DO 178C, DO 254, ARP4754, ARP4761, EN 50126-1, EN 50128, EN 50129, IEC 61508, ISO 26262, or the ESA standard ECSS-Q-ST-30C.

The final result of WP6.4 is to provide innovative and efficient methods and tools integrated in the RTP, which are ready for demonstration in the respective use cases and subsequent industrial application in production environments.

Table 1-1 on the next page lists the bricks of the work package and the respective use cases they are applied in. The bricks can be grouped into categories as follows:

- Safety methodology (6 Bricks): general and safety standard specific procedures and methods for managing safety
- Safety analysis tools (6 Bricks): tools explicitly addressing support for manual safety analysis
- Safety requirements engineering (1 Brick): expressing/formalizing safety requirements
- Verification management (2 Bricks): planning, execution and result analysis of verification tasks
- Test case generators (2 Bricks): model based test case generation, automatically deriving tests from test or implementation models
- Safety analysis automation and verification (2 Bricks): tools for automated derivation and automated verification of safety requirements

				Use Cases (names shortened)											
				UC2.1 Environmental Control Systems	UC2.3 Mission Support Equipment	UC2.5 Space Toolset applied to Avionics ECU	UC2.6 Multi-Mode Navigation System	UC2.8 Public Use Case AEROSPACE	UC3.1 Function development for heavy vehicles	UC3.2 Safety related assistance system	UC3.3 Functional power train development	UC4.2 Safety layer of interventional X-ray system	UC5.1 ERTMS/ETCS Interoperable testing	UC 5.2 Integrated modelling of TAS Platform	UC5.3 Traction Systems
Task ¹	Brick ID ²	Brick Name	Provider												
Safety methodology															
6.4.20	B2.53	Safety-analysis for Aerospace (ESA Standards)	GMV			X									
6.4.21	B2.54	Autonomous Fault Tolerant System Design Methodology (AFTS DM)	Tecnalia			X									
6.4.12	B3.04	Recommended methodology according ISO26262	ITKE							X	X				
6.4.13	B3.05	Risk assessment and hazard analysis	ITKE							X		X			
6.4.14	B3.06	FTA, FMEA and FMEDA	ITKE									X			
6.4.16	B3.10	Feature documentation in model based software	ITKE							X					
Safety analysis tools															
6.4.5	B2.22	FT+ brick integration into Rational Rhapsody	EADS IW-G		X										
6.4.17	B3.55	Isograph FaultTree+	ITKE									X			
6.4.2	B5.6	MB RAMS	AIT											X	
6.4.18	B5.10	SAFETY ARCHITECT	ALL4TEC												X
6.4.7	B3.97	C²FT	FhG IESE						X						
6.4.9	B2.41	Safety for Avionic Design and Analysis Framework	EADS IW-G	X											
Safety requirements engineering															
6.4.10	(B3.99)	Claims Language Boilerplate	IFX-UK								X				
Verification management															
6.4.11	B3.91a	DaD - Data Analyser Dashboard	IFX-UK								X				
6.4.1	B5.9	WEFACT	AIT								X			X	
Test case generation															
6.4.4	B5.7	MoMuT::SCADE³	AIT												
6.4.3	B5.8	MoMuT::UML	AIT						X				X	X	
Safety analysis automation and verification															
6.4.6	B2.43	NuSMV	FBK				X	X							
6.4.19	B2.52	MU Safety Analysis Tool	MU				X								

¹ Bricks are sorted by task number in this document, starting with task 6.4.1 – WEFACT in section 2

² Brick IDs as used in the CRYSTAL project proposal and description of work documents

³ MoMuT::SCADE lost its use case and will either be dropped or assigned to a yet to find other use case.

Table 1-1: Bricks Overview

1.3 Relationship to other CRYSTAL Documents

The specifications in this deliverable build on the use case specifications for the use cases where the bricks are applied. These are documented in the following deliverables:

Use Case	Deliverable #	Deliverable Name
UC2.1 Airbus Environmental Control Systems	D201.011	ECS Use Case description and derived requirements - V1
UC2.3 Mission Support Equipment	D203.011	MSE Report - V1
UC2.5 CRYSTAL Space Toolset applied to Avionics Control Unit	D205.010	Space Use Case Requirements
UC2.6 Multi-Mode Navigation System	D206.010	Multi-Mode Navigation System Analysis, Development Needs, and the Proposed Tool-Chain Functionality
UC2.8 Public Use Case AEROSPACE	D208.010	CRYSTAL aerospace use case description
UC3.1 Function development for heavy vehicles	D301.010	Use case definition
UC3.2 Development of a safety related assistance system	D302.011	Milestone Report - V1
UC3.3 Functional power train architecture & control development	D303.011	Milestone Report - V1
UC4.2 Safety layer of an interventional X-ray system	D402.010	Use Case definition
UC5.1ERTMS/ETCS Interoperable testing	D501.020	Use Case Requirements Specifications
UC 5.2 Integrated modelling of core algorithms in TAS Control Platform	D502.010	Use case definition
UC5.3 Traction Systems	D503.010	Use case definition

Table 1-2: Related Deliverables

The deliverables documenting the building and assessment of the SEE (System Engineering Environment) for use cases UC2.1, UC2.3, UC2.5, UC2.6, UC2.8, UC3.1, UC3.2, UC3.3, UC4.2, UC5.1, UC5.2, UC5.3 will probably refer back to the iterations of this document.

1.4 Structure of this document

In the following, each brick is presented in a separate chapter. In later iterations, the safety related concepts and aspects shared across bricks will be further harmonized and discussed in an own chapter.

The brick chapters are structured as follows:

- X.1 Overview
 - X.1.1 General Description
 - Use and features of the Brick
 - X.1.2 Related Use Cases
 - Use cases where the Brick is applied and (short) role of the brick
- X.2 Specification

- X.2.1 Requirements from the Use Cases
 - User requirements per originating use case
- X.2.2 How will this brick be integrated in the UC
 - Detailed interaction of the brick in the use case
- X.2.3 Requirements already fulfilled by pre-existing tool/method version
 - In case that the brick is not new, a list of requirements from the use cases already fulfilled by the pre-existing brick.
- X.2.4 What will be implemented/provided in the CRYSTAL project
 - X.2.4.1 General Improvements/Features
 - Requirements regarding new and improved general functionality/methodology properties
 - X.2.4.2 Integration/Interoperability
 - Requirements regarding integration and interoperation of the brick with other bricks and existing development environments and processes.
- X.3 Implementation/Elaboration
 - To be filled and sub-structured in future iterations
- X.4 Evaluation
 - To be filled and sub-structured in future iterations

The document is concluded by a short summary section.

2 WEFACT

Provider: AIT

Task #: T6.4.1

Brick #: B5.9

Category: Verification management

2.1 Overview

2.1.1 General Description

WEFACT is a workflow-oriented tool controlling validation, verification and certification processes of critical systems and components as a basis for automated safety cases. Using IBM Rational Doors requirements management, WEFACT connects V&V tools and artifacts to be validated or certified with flexible instantiations of generic validation plans for the individual standards and safety integrity levels (in CRYSTAL for standards EN 50128, EN 50129 and ISO26262). The “Workflow Engine for Analysis, Certification and Test” (WEFACT) has the goal to facilitate validation, verification and certification of safety-critical systems in a modular manner.

WEFACT consists of the WEFACT framework which provides a flexible infrastructure for defining and executing the V&V process and the external resources – external processes, tools and standards – which are integrated into the WEFACT framework by well-defined interfaces. Additionally, an extensive on-line user guide (“help file”) including a v-plan cook book (“How to develop a v-plan”) is available.

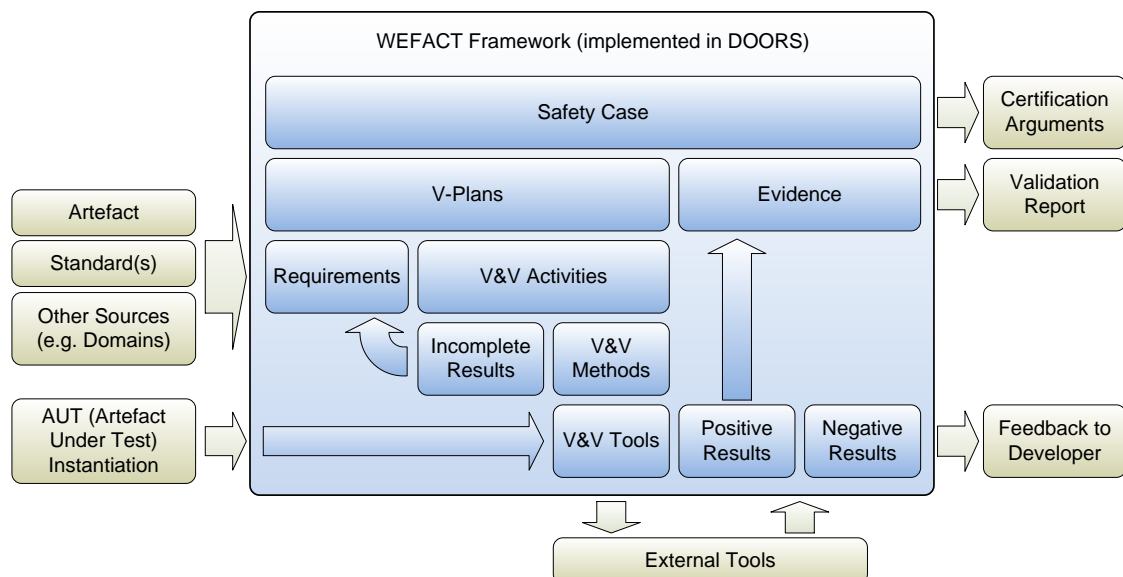


Figure 2-1: Overall organization of the WEFACT framework

The overall organization of the WEFACT framework is shown in Figure 2-1. The blue boxes show the elements of the WEFACT framework, the grey boxes show the rest of the elements of the WEFACT (belonging to the external systems), vertical alignments indicate ‘uses’ or ‘consists of’ relationships whereas the arrows indicate major information flows.

The safety case is an argumentation to convince a regulation authority that a product is “sufficiently safe”. Typically a safety case comprises the necessary safety arguments which correspond to the validation plans (v-plans) for each artefact under test (AUT) and the related evidence. Please note that depending on the safety standard and the context, there might be a more specific understanding of the term, e.g. giving a document structure and expected content for the safety case.

The WEFACT framework is implemented with IBM Rational DOORS which is based on distributed client/server architecture. The data such as v-plans, V&V activities, and requirements is stored in the central DOORS database whereas the documents such as evidence and reports are stored in a separate document repository. Several access modes to these repositories are supported, one of them is OSLC Asset Management, implemented when integrating WEFACT into the RTP of the Artemis project MBAT. In order to setup the WEFACT framework for a user, he or she installs the DOORS client in order to have access to the data and sets up the access to the document repository.

WEFACT supports intermediate results, which are observed for changes. If an input for a V&V activity changes, its former output is treated as out of date. Chains of such dependencies make up the workflows in WEFACT. There is also support for completeness analysis and report generation.

Data Flow

The WEFACT tool does not exchange data with other tools but rather manages links and references to elements of other documents, data base records, model elements, etc.

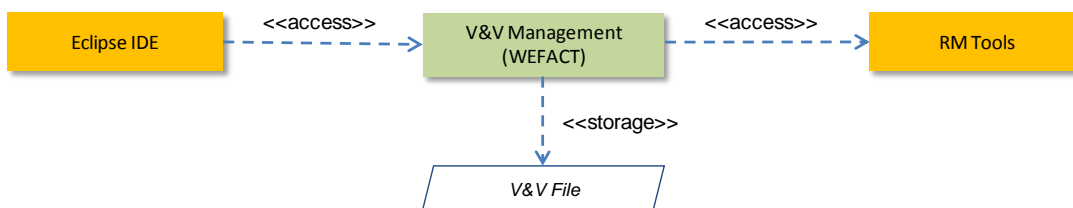


Figure 2-2: Dependencies of V&V management on other artefacts

Based on the CRYSTAL IOS, the tool will interface with any other requirements management tool (not only DOORS). It is planned to also make the V&V activities accessible via CRYSTAL IOS. This will be used in an Eclipse plugin to integrate WEFACT into the development environment, but would also make integration into IBM’s JAZZ platform possible – putting V&V activities a user is responsible for in the same list of tasks as the tasks from issue management.

2.1.2 Related Use Cases

WEFACT is applied to the use cases UC3.3 (AVL) and 5.2 (TRAIL). See the respective deliverables: M9 report for WP 303 for UC3.3 and the deliverable D502.010 for UC5.2.

2.2 Specification

2.2.1 Requirements from the UCs

The following requirements will be addressed by this brick, grouped by use case:

2.2.1.1 UC3.3 - Functional power train architecture & control development with respect to integrated system, safety and requirements engineering (AVL)

Requirements from this use case are still under discussion, with the use case requirements only due M9. Some brick requirements have been anticipated nonetheless. The finalized requirements from the use case will be addressed here when they are available.

2.2.1.2 UC 5.2 - Integrated Modelling of Core Algorithms in TAS Control Platform (TRAIL)

ID	Title	Description	Priority
UC5.2-GEN-01	Eclipse Integration	Thales Austria aims for integration of as many development related task types as possible into their Eclipse based development environment. The need for switches to other tools/work environments for the tasks addressed in the SEE shall be minimized (within reasonable effort).	HIGH
UC5.2-GEN-02	RM agnostic integration	Integration with requirement management shall be transparent with respect to the used Requirements Management Tool and the real location of the requirement for other tools in the SEE.	HIGH
UC5.2-GEN-03	Traceability between MDE artefacts and code	Traceability shall be granted for all levels of requirements and associated artefacts down to the source code level	HIGH
UC5.2-VVM-01	V&V success tracing	Trace successful V&V activities to fulfilled (safety) requirements	HIGH
UC5.2-VVM-02	V&V fail tracing	Trace failed V&V activities to requirements	HIGH
UC5.2-VVM-03	TCG fail tracing	Trace failed test cases to model elements in the test model	HIGH
UC5.2-VVM-04	Standards guidance	Systematic guidance through requirements of applicable CENELEC standards	HIGH
UC5.2-VVM-05	Hierarchical V&V activities list	Creation and maintenance of a hierarchical list of verification activities	HIGH
UC5.2-VVM-06	Verification status input	Edit verification status of each defined activity	HIGH

ID	Title	Description	Priority
UC5.2-VVM-07	Creation of traceability links	Create traceability links to elements of other artefacts (paragraph in document, test reports, etc.)	HIGH
UC5.2-VVM-08	Detection of re-validation need	Detect need for re-validation of documents / test results	HIGH
UC5.2-VVM-09	Verification status statistics	Generate overall statistics of verification status	MEDIUM
UC5.2-VVM-10	Artefact versioning	Support versioning of associated artefacts	HIGH
UC5.2-VVM-11	V-plan version control	Support version control of V-plan	HIGH
UC5.2-VVM-12	Traceability for updated artefacts	Transfer link to new version of target artefact	MEDIUM

Table 2-1: UC5.2 requirements addressed by WEFACT

2.2.2 How will this brick be integrated in the UC

WEFACT is initiating tasks fulfilled by/with other tools and collecting traceability information generated by these tools. The supported integration levels for initiating tasks range from manual execution (where a user is informed per email or in his task list, e.g. to perform a not automated task like measurement of electromagnetic emissions of a device) to full automation (facilitating OSLC Automation Specification and MBAT IOS Traceability Tracking). Figure 2-3 illustrates the processing of a V&V activity in WEFACT. The status of the V&V activity changes from “Not ready” to “Ready” when all input data is available. In the status “Processing” it interacts with the V&V tool linked to it. If the processing is successful and the result is positive (passed) the status changes to “Completed”. The traceability links are visible through the red and orange triangles for out-links and in-link, respectively.

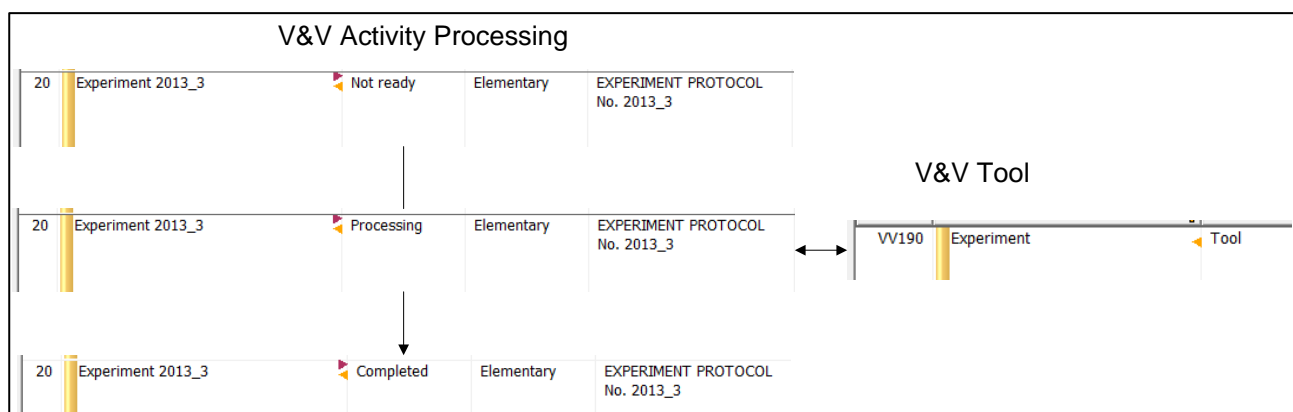


Figure 2-3: Processing of a V&V activity in WEFACT

2.2.3 Requirements fulfilled by initial tool/method version

ID	Title	Description	Priority
UC5.2-VVM-01	V&V success tracing	Trace successful V&V activities to fulfilled (safety) requirements	HIGH
UC5.2-VVM-02	V&V fail tracing	Trace failed V&V activities to requirements	HIGH
UC5.2-VVM-05	Hierarchical V&V activities list	Creation and maintenance of a hierarchical list of verification activities	HIGH
UC5.2-VVM-06	Verification status input	Edit verification status of each defined activity	HIGH
UC5.2-VVM-08	Detection of re-validation need	Detect need for re-validation of documents / test results	HIGH
UC5.2-VVM-09	Verification status statistics	Generate overall statistics of verification status	MEDIUM

Table 2-2: UC5.2 requirements already fulfilled by current WEFACT version

2.2.4 What will be implemented/provided in the CRYSTAL project

Below, refined requirements towards the tool/method/integration are given, grouped by general improvements and interoperability/integration features.

2.2.4.1 General Improvements

ID	Title	Description	Priority	UC requirements
B5.9-F1	Import traceability data	Support import of traceability information from V&V tools	HIGH	UC5.2-GEN-03, UC5.2-VVM-03
B5.9-F2	Query traceability data	Provide query mechanism for traceability information	HIGH	UC5.2-GEN-03, UC5.2-VVM-03
B5.9-F3	Standards guidance	Systematic guidance through requirements of applicable CENELEC standards	HIGH	UC5.2-VVM-04
B5.9-F4	Creation of traceability links	Create traceability links to elements of other artefacts (paragraph in document, test reports, etc.)	HIGH	UC5.2-VVM-07
B5.9-F5	Artefact versioning	Support versioning of associated artefacts	HIGH	UC5.2-VVM-10
B5.9-F6	V-plan version control	Support version control of V-plan	HIGH	UC5.2-VVM-11
B5.9-F7	Traceability for new artefacts	Transfer link to new version of target artefact	MEDIUM	UC5.2-VVM-12

Table 2-3: General improvement requirements for WEFACT

New and existing features shall be evaluated regarding their need for tool qualification in order to be compliant with the respective safety standards applying to the use cases (CENELEC 50128 tool classes T1 to T3, ISO26262)

2.2.4.2 Interoperability/Integration

ID	Title	Description	Priority	UC requirements
B5.9-I1	WEFACT Web Access	Make WEFACT available via a web interface, to be used in a browser pane in ECLIPSE	HIGH	UC5.2-GEN-01
B5.9-I1	IOS requirements references	Support references to requirements in another IOS connected RM manager	HIGH	UC5.2-GEN-02

Table 2-4: Interoperability and integration requirements for WEFACT

2.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

2.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

3 MB RAMS

Provider: AIT

Task #: T6.4.2

Brick #: B5.6

Group: Safety Analysis Tools

3.1 Overview

3.1.1 General Description

The MB RAMS brick contains both a partial safety process and a tool to support this process. It focuses on model-based safety and risk analysis based on early system models (especially FMEA). The goal of the brick is to integrate MB RAMS' safety process with tools used for requirements engineering, design, implementation and test/verification.

The partially known process will be further elaborated and existing tools mainly from project partners will be evaluated for supporting it. A set of selected pre-existing or newly developed tools will be integrated using the CRYSTAL IOS. Potential candidates are Eclipse Modelling Framework, SafetyArchitect® of All4Tec, the shareware tool Papyrus Eclipse, the Event-B method based on the Rodin platform, the Eclipse plugin ProR, OBEO Designer® of OBEO, WEFAC of AIT, DOORS® and Rhapsody® of IBM Rational, or RAM Commander® of RELIASS.

The starting point of MB-RAMS is an architectural and functional system model expressed in UML/SYSML. The FMEA shall be realized as a structured list, for which an appropriate tool will be selected. The *Failure Cause* objects contained in this list will be associated with the respective attributes (according to the FMEA process) and linked to requirements and SACs (safety application conditions). Together with fault propagation properties, this represents the safety model. For safety modelling, adequate tool support shall be provided, preferably by using and - if necessary - adapting existing tools. All relevant artefacts in this model will be interlinked consistently by traceability references.

A minimum requirement to be implemented in the brick is a check function verifying completeness and consistency between structural and functional system model and safety model. The ideal goal is to have one united model composed of a system and a safety model, which contains the system specification and detailed requirements about it, a functional system model, the corresponding design of the architecture and safety properties like fault propagation or fault models. Different views on this comprehensive model (e.g. safety viewpoint or architecture viewpoint) satisfy the needs of people responsible for the various activities in the safety life cycle while the consistent model guarantees full traceability at any stage of system development and operation. As far as possible, FMEA and FTA shall be supported by (semi-)automatic functions using the consistent overall model for deriving the respective analyses. A possible outcome is a UML/SysML safety profile.

General goals are transparent tool interconnections via OSLC and the IOS-compliant, transparent exposure of architectural, functional as well as safety model elements. This shall also enable independence of the concrete requirements repository. Another highly desirable feature is tool-based support for analysis of the impact of changes in requirements, functional / architectural model or components used on the safety model.

3.1.2 Related Use Cases

This brick is used in the Thales railway use case described in deliverable D502.010. It aims at a significant efficiency improvement by enabling a transition from mainly manual safety processes to computer supported processes allowing safety modelling based on system and fault models. The goal is a process based on a consistent and stable database containing system and fault models as well as fault propagation properties, which allows different perspectives on the data like a safety viewpoint or a logical architecture viewpoint. The model furthermore allows deriving respective contributions to the safety case automatically.

During the entire process, traceability across all artefacts is maintained covering the system model, the fault propagation model, SACs, single rows in the FMEA as well as the nodes and edges in the fault tree. Changes are recognized by impact analysis so that respective revalidation activities can be triggered and consistency can be re-established.

3.2 Specification

3.2.1 Requirements from the UCs

The following requirements will be addressed by this brick

3.2.1.1 UC 5.2 - Integrated Modelling of Core Algorithms in TAS Control Platform

Below are the requirements applying to MB-RAMS taken from D502.010.

ID	Title	Description	Priority
GEN-01	Eclipse Integration	Thales Austria aims for the integration of as many development related task types as possible into their Eclipse based development environment. The need for switches to other tools/work environments for the tasks addressed in the SEE shall be minimized (within reasonable effort).	HIGH
GEN-02	RM agnostic integration	Integration with the requirements management shall be transparent with respect to the used Requirements Management Tool and the real location of the requirements for other tools in the SEE.	HIGH
GEN-03	Traceability between MDE artefacts and code	Traceability shall be granted for all levels of requirements and associated artefacts down to the source code level	HIGH
MBR-01	Safety model tool support	Tool support for creating the safety model	HIGH
MBR-02	Safety model check	Consistency and completeness check between architectural/functional model and safety model	HIGH
MBR-03	Safety model visualisation	Visualization of the safety model (Safety Viewpoint)	HIGH
MBR-04	FMEA structures	Deriving structures for a qualitative FMEA from system model and fault models.	HIGH
MBR-05	Safety model traceability	Realize full traceability between (1) all items in the safety model including safety	HIGH

ID	Title	Description	Priority
		properties and relations (2) functions and components or elements in the system model, (3) the safety requirements and (4) objects of the FMEA and FTA.	
MBR-06	FMEA generation	Semi-automatic generation of qualitative FMEA including effects, based on system model, fault models and safety model	HIGH
MBR-07	FTA generation	Semi-automatic generation of FTA for treating effects of multiple faults based on system model and fault models	MEDIUM
MBR-08	Impact analysis	Tool-based support for analysing the impact of changes in requirements, functional/architectural model or components used on the safety model	MEDIUM
MBR-09	Safety model generation	Automatic creation of the safety model from system model, FMEA, FTA and SACs.	LOW
MBR-10	Safety mechanisms tool	Provide tool support for (a) checking implemented safety mechanisms for consistency with safety model and safety requirements, or (b) proposing or creating respective safety functions (barriers) to cope with the safety requirements	LOW

Table 3-1: UC5.2 requirements addressed by MB RAMS

3.2.2 How will this brick be integrated in the UC

The safety analysis is interacting with the following other activities:

- Architecture/System modelling (input artefact, might need changes in feedback loop, introducing additional model elements; FMEA artefacts link to model elements)
- Requirements Engineering (storage of derived safety requirements, together with relation to FMEA artefacts)

Figure 3-1 shows the data flow for MB-RAMS activities and the tool types used in the MB-RAMS activities.

Wide arrows represent the data flows; dashed arrows show for which data flows or verification activities tools are used.

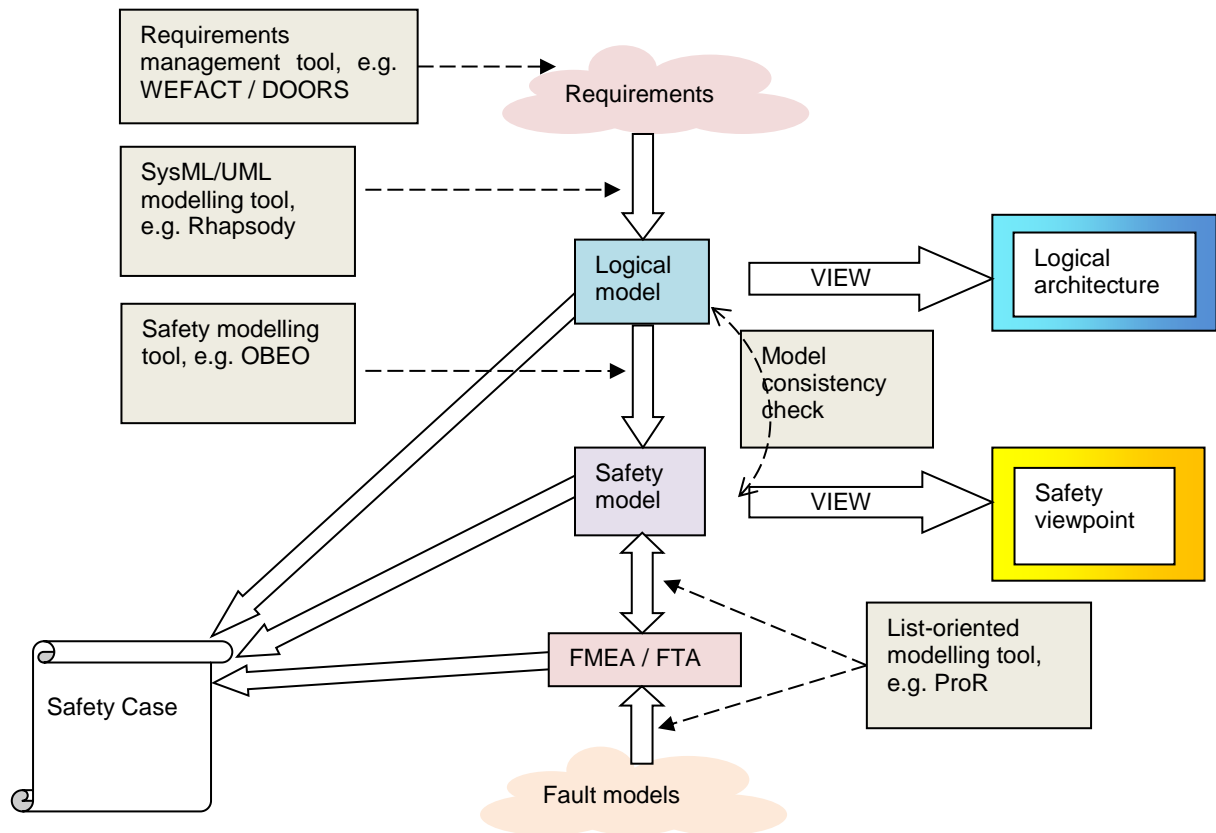


Figure 3-1: Data Flow for MB-RAMS activities

3.2.3 Requirements fulfilled by initial tool/method version

The building blocks for this brick (model editor, safety analysis tool) are still to be selected, so details about which requirements are already fulfilled will be known after this selection process.

3.2.4 What will be implemented/provided in the CRYSTAL project

Currently, all required features are treated as new - this might change due to the selection of existing components for building the overall solution, mentioned above in section 3.2.3.

The requirements lists below are also not final yet and might be updated in the next iteration of the deliverable.

3.2.4.1 General Improvements

The following table lists the planned new features of the brick.

ID	Title	Description	Priority	UC requirements
B5.6-F1	Safety meta-model	A dedicated safety meta-model (or profile) for the safety viewpoint will be designed, covering all relevant properties like fault probabilities, possible fault propagation, fault detection, etc	HIGH	MBR-01
B5.6-F2	Safety model editor	Model editor to build a model according to the safety meta model	HIGH	MBR-01
B5.6-F3	Fault model editor	Model editor to write down a fault model	HIGH	MBR-01
B5.6-F4	Safety model seeding	Pre-fill a safety model / safety view with artefacts from the architecture model	HIGH	MBR-01
B5.6-F5	Safety-Model consistency check	Check a given safety model for compliance with the meta model and some basic internal consistency checks	HIGH	MBR-01
B5.6-F6	Model synchronisation	Synchronize safety model/safety view with architecture model	HIGH	MBR-02
B5.6-F7	Inter-Model consistency check	Check safety view and architecture view for inconsistencies	HIGH	MBR-02
B5.6-F8	Safety model visualisation	Provide a visualization of the safety model (Safety Viewpoint) in a way that supports the tasks of the safety engineer	HIGH	MBR-03
B5.6-F9	FMEA editor	Provide or select a way to edit FMEA	HIGH	MBR-04
B5.6-F10	FMEA structures pre-fill	Deriving structures for a qualitative FMEA from system model and fault models	HIGH	MBR-04
B5.6-F11	Traceability – requirement -> architecture model element	Provide and maintain link from requirements to architecture model	HIGH	MBR-05
B5.6-F12	Traceability –model element -> FMEA/FTA artefact	Provide and maintain link from architecture or safety model to FMEA/FTA artefact	HIGH	MBR-05
B5.6-F13	Traceability – architecture model element -> safety model elements	Provide and maintain links between architecture and safety model elements	HIGH	MBR-05
B5.6-F14	Traceability – FMEA/FTA artefact -> derived safety requirement	Provide and maintain link from safety analysis artefacts/entries to derived safety requirements	HIGH	MBR-05
B5.6-F15	FMEA generation	Semi-automatically fill pre-seeded qualitative FMEA with modes and effects, based on system model, fault	HIGH	MBR-06

ID	Title	Description	Priority	UC requirements
		models and safety model		
B5.6-F16	FTA generation	Semi-automatic generation of FTA for treating effects of multiple faults based on system model and fault models	MEDIUM	MBR-07
B5.6-F17	Impact analysis of requirement changes	Tool support for analysing the impact of changes in requirements	MEDIUM	MBR-08
B5.6-F18	Impact analysis of functional / architectural model	Tool support for analysing the impact of changes in functional/architectural model	MEDIUM	MBR-08
B5.6-F19	Impact analysis of safety model changes	Tool support for analysing the impact of changes in safety model	MEDIUM	MBR-08
B5.6-F20	Safety model generation	Automatic creation of the safety model from system model, FMEA, FTA and SACs.	LOW	MBR-09
B5.6-F21	Safety mechanisms tool	Provide tool support for (a) checking implemented safety mechanisms for consistency with safety model and safety requirements, or (b) proposing or creating respective safety functions (barriers) to cope with the safety requirements	LOW	MBR-10

Table 3-2: General improvement requirements for MB RAMS

Generally, the model-based (MB) RAMS process shall be achieved mainly by re-using existing MB tools and where necessary adapting them to the specific needs of the use case.

3.2.4.2 Interoperability/Integration

ID	Title	Description	Priority	UC requirements
B5.6-I1	Connect to RM tool (in particular also WEFACT)	Store and link to derived safety requirements, independent of the concrete tool and the location of the requirements repository.	HIGH	GEN-02
B5.6-I2	Connect to safety modelling tool	Store and link to derived safety features in model	HIGH	MBR-01
B5.6-I3	Integration with modelling environment	For this purpose using the shareware tool Papyrus Eclipse is foreseen. However, also UML/SYSML models created and maintained with IBM Rational Rhapsody shall be integrated.	HIGH	GEN-01
B5.6-I4	Connect to V&V management (WEFACT)	The automated steps of the improved safety process shall be integrated with WEFACT by implementing them as V-	HIGH	MBR-01 and MBR-08

		plans.		
B5.6-I5	Safety model traceability interoperability	Expose safety model elements for generic linking via IOS	HIGH	GEN.03 and MBR-05
B5.6-I6	Architecture model traceability interoperability	Expose/consume functional / architecture model elements for generic linking via IOS	HIGH	GEN.03 and MBR-05
B5.6-I7	Requirements traceability interoperability	Consume requirements for generic linking via IOS	HIGH	GEN.03 and MBR-05
B5.6-I8	Requirements link generation interoperability	For derived safety requirements, use IOS means to store them to a requirements management system	HIGH	GEN.03 and MBR-05

Table 3-3: Interoperability and integration requirements for MB RAMS

3.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

3.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

4 MoMuT::UML

Provider: AIT

Task #: T6.4.3

Brick #: B5.8

Category: Test case generation

(former brickTool name: UMMU)

4.1 Overview

4.1.1 General Description

MoMuT is the name of a family of test case generators from AIT which were developed in cooperation with TUG. *MoMuT::UML* uniquely combines a powerful fault-based test case generation strategy with standard techniques for assembling test cases. It delivers high quality test suites with an excellent cost/benefit ratio. The heart of this technology is the concept of fault seeding or mutation. Figure 4-1 depicts our underlying model-based mutation testing approach: *MoMuT::UML* uses customizable mutation operators to derive mutated models from the original test model. A mutated model is an exact copy of the original except for a small change introduced by the mutation operator. Given a mutant and the original specification, the backend searches for a sequence of inputs and outputs that show that the mutant's behaviour is in conflict with the behaviour specified by the original model. It is in the nature of mutation-based test case generation that one such sequence, i.e. test, finds ("kills") multiple mutated models and, hence, has the ability to find faults that are not directly modelled by a mutation operator.

Mutation-based test case generation is the most fine-grained and versatile test generation technique available today. In principle, mutation-based test case generation can not only be used to test functional properties of designs but also to generate tests that detect certain non-functional defects. It also allows *MoMuT::UML* to know exactly which faults are caught by a particular test case, analyse or extend existing test sets, and help localizing faults by

- (a) automatically selecting a set of mutants that can explain faulty behaviour
- (b) creating a test case providing a short path to a fault found during regression testing to help with debugging (the original test case might test a lot of other things before exposing the fault).

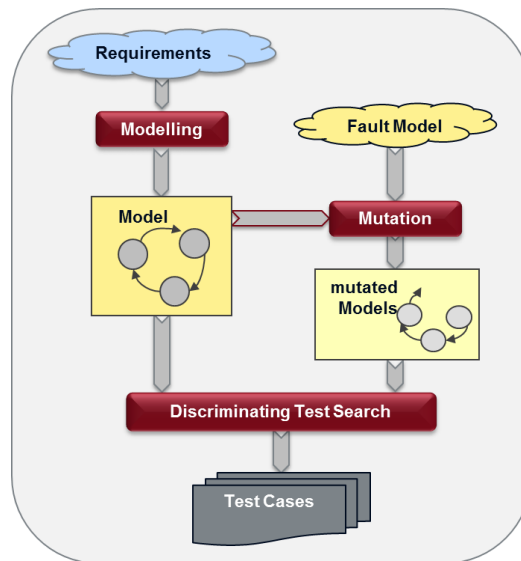


Figure 4-1: Mutation-based test case generation is at the heart of MoMuT::UML

MoMuT::UML either uses the *ioco* (input-output conformance) relation or the refinement relation to verify if a mutant complies with the specified behaviour in the original model. It translates UML to an internal representation with clearly defined semantics (action systems). The tool set can connect to state-of-the-art model checkers for further design verification and provides two test case generation methods (one working enumerative and one working symbolic). The tool set also supports model animation features.

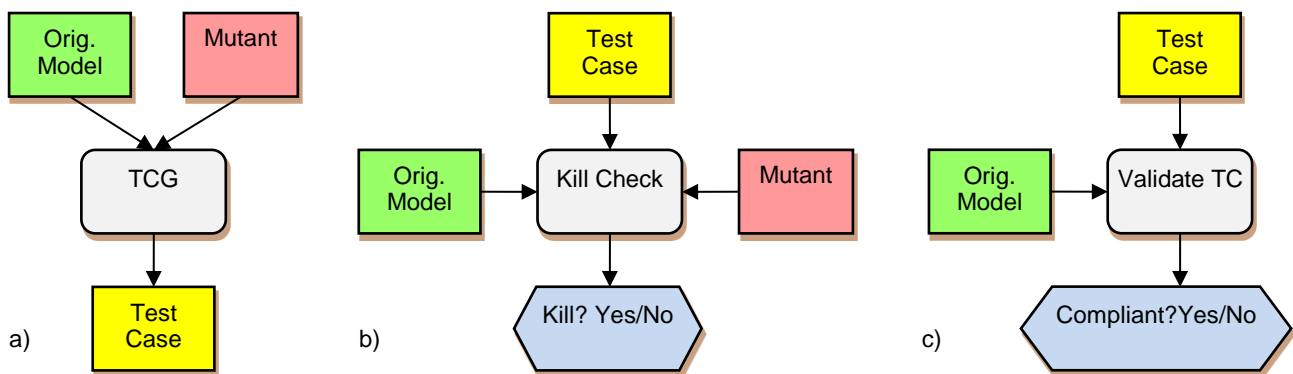


Figure 4-2: Usage variants of mutation-based TCG tools

As it is shown in Figure 4-2, there are multiple ways to use the tool:

- a) generating the test cases for a mutant
- b) checking the quality of test cases by checking which/how many mutants they kill
- c) using the model as an oracle to decide which behaviours are compliant with the model.

By combining variants a) and b), pre-existing test cases can be evaluated for their mutation coverage and only test cases for the missing mutants are created then. The pre-existing test cases can be legacy test cases, can come from other tools (including white-box test case generators) or can come from prior system iterations.

Figure 4-3 shows an example screenshot of the graphical user interface of MoMuT::UML, which was produced with the symbolic TCG backend.

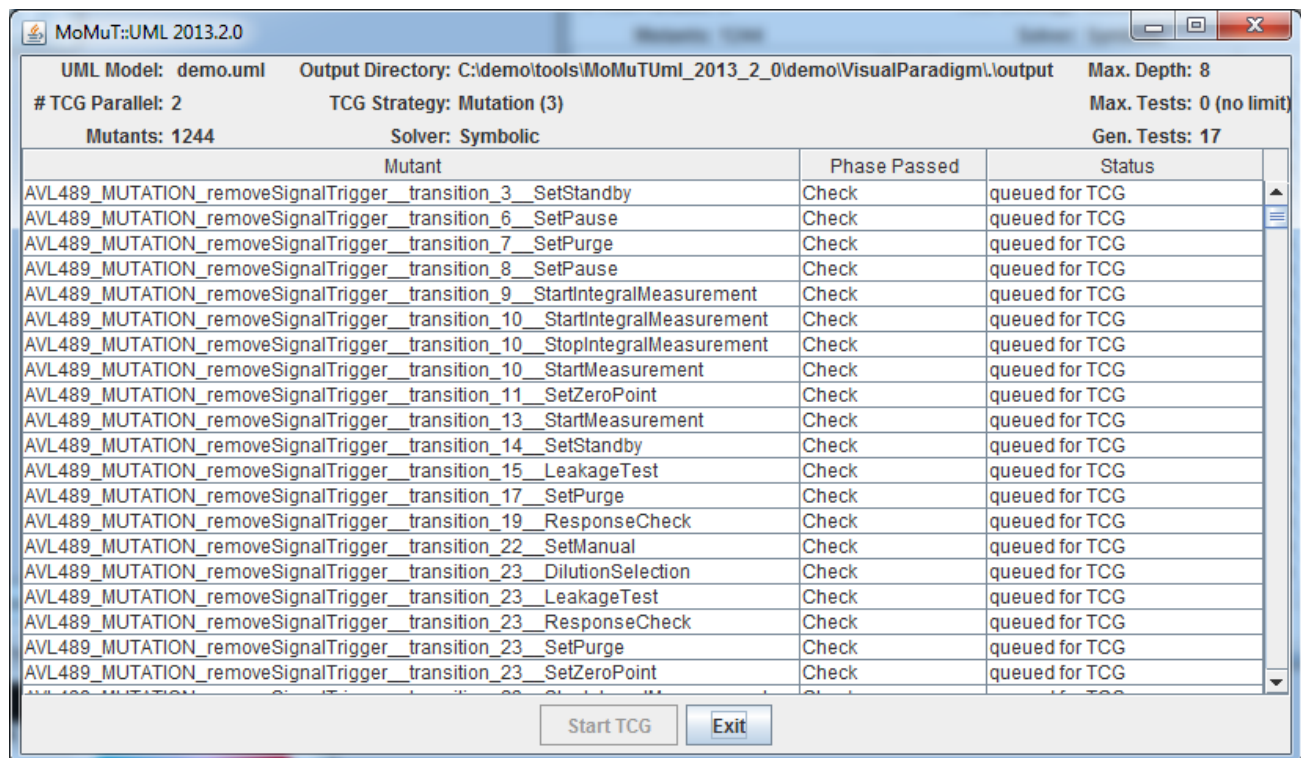


Figure 4-3: Screenshot of the MoMuT::UML tool

Data Flow

The following artefacts are involved in the envisaged data flow in model-based testing:

Input to TCG:

- (1) Test model (System model including traceability from requirements to model objects)
- (2) Test configuration (definition of model variables, instantiation, etc.)
- (3) Pre-existing test cases (optional)

Output of TCG:

- (4) Abstract test cases (sequences of input / output events)
- (5) Traceability matrix test case to requirements

The role of these artefacts is depicted in the following data flow diagram:

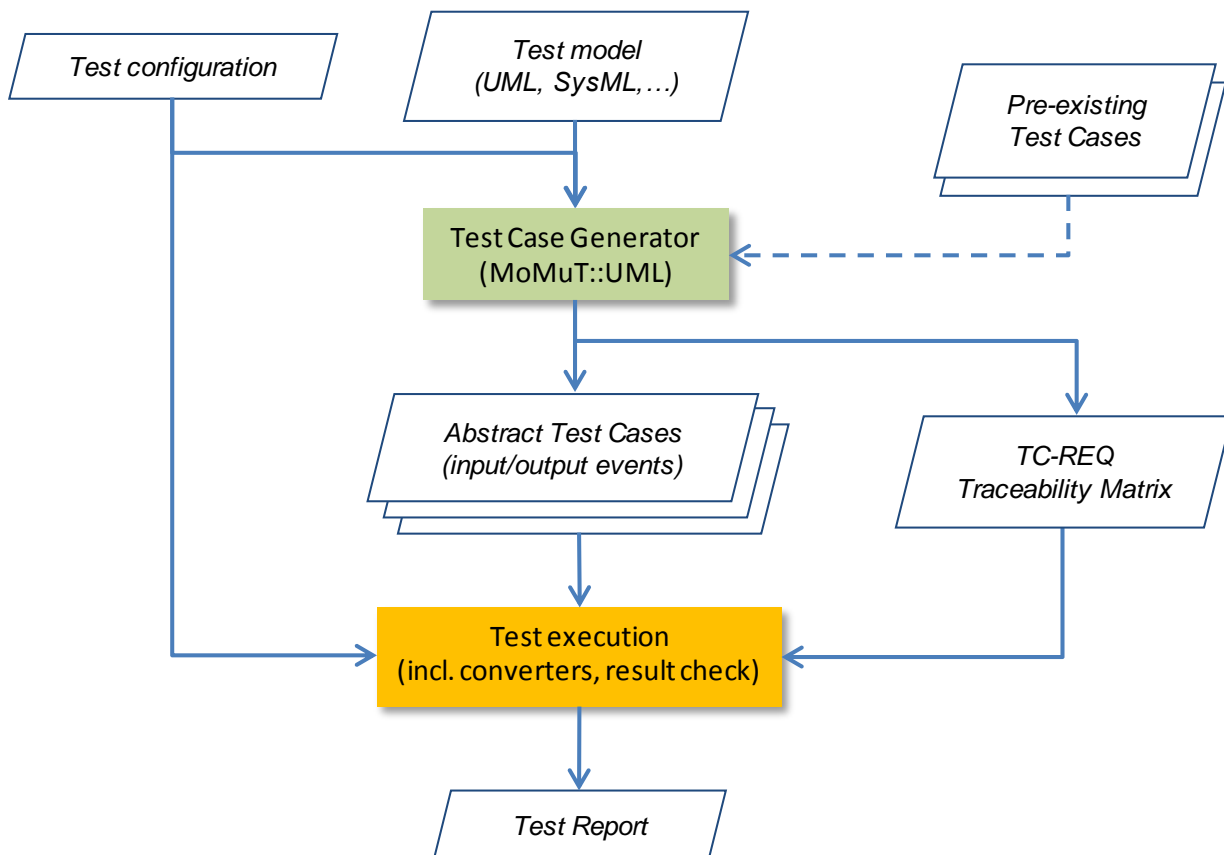


Figure 4-4: Automatic test case generation and related data flows

The test execution may contain several stages, including

- test case translation (to specific test scenario and golden trace files)
- actual test run
- test result check (verdict)
- test report compilation

MoMuT::UML can be configured by means of a configuration file that controls the test case generation. This file defines e.g. the mutation operators and the mutated elements.

Information about failing test cases can be fed back into the test case generator for generation of short test cases to isolate the problem.

4.1.2 Related Use Cases

MoMuT::UML is used in:

- UC3.1
- UC5.1
- UC5.2

For Volvo, in UC3.1, MoMuT::UML will generate test cases from a design model, delivering test cases which will not only be used for testing the simulated system as well as the real implementation, but will also be used by brick B3.3 DTFsim for evaluating the communication bus and overall timing behavior.

At the time of writing, the exact setup of the Ansaldo use-case (UC5.1) was still under discussion. A tentative idea for UC5.1 is to evaluate test cases from other sources for their mutation coverage and to generate all missing mutants. The overall test model will be split into multiple sub-models, related to groups of requirements. The test case generator shall be improved to take advantage of this decomposition information.

In Thales Austria's use case 5.2, MoMuT::UML interacts with WEFACT (B5.9) and possibly with another test case generator yet to decide on. However, a clear division of responsibilities is possible. WEFACT will drive the tool execution and collect the necessary traceability information. MoMuT::UML will either generate test cases on its own or cross check the mutation coverage for tests generated by the alternative test case generator and provide test cases for the missing mutants.

4.2 Specification

4.2.1 Requirements from the UCs

The following requirements will be addressed by this brick, grouped by use case:

4.2.1.1 UC3.1 - Function development for heavy vehicles (Volvo)

There are no individualized, numbered requirements in in D301.010 which were suitable for tabular presentation but it contains descriptions of involved requirements which are cited verbatim hereafter. Requirements for new features and integration derived from this further below are noted to be related to this use case.

Test case generation in this use case involves the following steps:

1. Behavioural models, requirements, MSCs, MSDs, and functional components have been defined as it is described in section 'system behavioural modelling' of deliverable D301.010
2. The models are made available to the test case generation tool (MoMuT::UML).
3. The system models are augmented with information needed for test case generation – possibly using data from the sub-use case "AUTOSAR ECU integration and generation"; Corresponding model elements in the test case tool are linked to the elements in the system modelling tool (e.g. DCs).
4. Test cases are generated on DC and possibly integration level. Existing test cases from prior iterations are reused as far as possible.
5. Test cases are linked to requirements, MSCs, MSDs, and/or behavioural models. Furthermore, the full test cases are made available to SystemWeaver. Note that performing the actual tests is not in the scope of this use case.

The interoperability challenges in the Volvo use case come in two forms:

- i) establishing and maintaining data links across tools, and
- ii) exchanging whole models or parts of models across tools.

The first form is necessary to enable traceability and consistency across tools. This means that it should be possible to denote e.g. that two modelling entities in different tools in fact represent the same entity or that they are related somehow. In a central information model these kinds of links are typically already present,

Version	Nature	Date	Page
V1.0	P	2014-02-07	33 of 125

e.g. the SystemWeaver meta-model used in the Volvo use case contains many links already. The challenge is to extend these links to also include other tools not using the SystemWeaver meta-model. However, this form of interoperability assumes that the models in the different tools have been made independently from each other. That is, before the links can be added, individual models must exist in the tools of interest. Typically these models are manually constructed. In many cases it would be more efficient to be able to generate a model skeleton from an already existing model. This would not only speed up the model construction, it would also enforce consistency of the generated model with the existing. Moreover, the generated model could be formed according to well-defined guidelines, which simplifies understanding of the model and automated analysis. Therefore, the Volvo use case considers also the second form of interoperability: model exchange. Since a large portion of the system data is already available in the SystemWeaver meta-model, and this meta-model is based on an early version of EAST-ADL, our intention is to use EAST-ADL as the exchange format between tools. For lower abstraction levels, AUTOSAR formats will be used.

ID	Title	Description	Priority
UC5.2-GEN-01	Eclipse Integration	Thales Austria aims for integration of as many development related task types as possible into their Eclipse based development environment. The need for switches to other tools/work environments for the tasks addressed in the SEE shall be minimized (within reasonable effort).	HIGH
UC5.2-GEN-02	RM agnostic integration	Integration with requirement management shall be transparent with respect to the used Requirements Management Tool and the real location of the requirement for other tools in the SEE.	HIGH
UC5.2-GEN-03	Traceability between MDE artefacts and code	Traceability shall be granted for all levels of requirements and associated artefacts down to the source code level	HIGH
UC5.2-TCG-01	Generate test cases from a UML test model	Generate test cases from a system test model in UML (Black Box Testing)	HIGH
UC5.2-TCG-02	Coverage selection	Select coverage for test cases by mutation operators, related requirements, and model elements.	HIGH
UC5.2-TCG-03	Test model elements traceability	Relate test model elements to requirements (safety or not)	HIGH
UC5.2-TCG-04	Test case - model traceability	Relate test cases to model elements	HIGH
UC5.2-TCG-05	V&V activity - test case traceability	Relate V&V activities to test cases	HIGH
UC5.2-TCG-06	Generate component test cases (Black Box Testing)	Generate test cases from a component test model	HIGH
UC5.2-TCG-08	Incremental test case generation	Extending component test cases to integration or system test cases.	MEDIUM

Table 4-1: UC5.2 requirements addressed by MoMuT::UML

4.2.1.2 UC 5.1 - ERTMS/ETCS Interoperable testing New Way (ASTS)

Requirements from this use case are still under discussion at the time of writing. Some brick requirements have been anticipated nonetheless and are noted to be related to this use case.

4.2.1.3 UC 5.2 - Integrated Modelling of Core Algorithms in TAS Control Platform (TRAIL)

Requirements from UC5.2 addressed by MoMuT::UML are listed in Table 4-1

4.2.2 How will this brick be integrated in the UC

MoMuT::UML will be integrated in the use cases for test case evaluation (provide coverage information from test model, existing test cases and mutation information i.e. mutation operators and mutation location selection), for test case generation (provide test cases for a given (sub-) model with given mutation information) and as a test oracle (evaluate consistency of given test cases with a given test model), as depicted in Figure 4-2, parts a) to c).

4.2.3 Requirements fulfilled by initial tool/method version

ID	Title	Description	Priority	Remarks
UC5.2-TCG-01	Generate test cases from a UML test model	Generate test cases from a system test model in UML (Black Box Testing)	HIGH	Available, IOS integration under development in MBAT. Performance improvements open.
UC5.2-TCG-02	Coverage selection	Select coverage for test cases by mutation operators, related requirements, and model elements.	HIGH	Available, IOS integration under development in MBAT
UC5.2-TCG-04	Test case - model traceability	Relate test cases to model elements	HIGH	Available, IOS integration under development in MBAT
UC5.2-TCG-06	Generate component test cases (Black Box Testing)	Generate test cases from a component test model	HIGH	Available, IOS integration under development in MBAT. Performance improvements open.

Table 4-2: Requirements already fulfilled by current MoMuT::UML version

4.2.4 What will be implemented/provided in the CRYSTAL project

Below, refined requirements towards the tool/integration are given, grouped by general improvements and interoperability/integration features

4.2.4.1 General improvements

ID	Title	Description	Priority	UC requirements
B5.8-F1	Improved UML TCG performance (system level)	Further reduce test case generation times	HIGH	UC5.2-TCG-01, UC5.1 related, UC3.1 related
B5.8-F2	Incremental test case generation	Extending component test cases to integration or system test cases.	MEDIUM	UC5.2-TCG-08
B5.8-F3	Test model animation	Provide animation features to evaluate/plausibility check the test model.	MEDIUM	(usability, all use cases)
B5.8-F4	Test model consistency check	Improve test model validation features to spot problems in the model before trying to generate test cases from it.	MEDIUM	(usability, all use cases)
B5.8-F5	Test coverage visualisation	Provide features to visualise various coverage criteria in relation to model elements and test cases.	HIGH	(usability, all use cases)
B5.8-F6	Use composition information	Take advantage of model substructure for improving test case generation times	HIGH	UC5.1 related
B5.8-F7	Improve usability	Usability shall be improved based on user feedback	HIGH	(usability, all use cases)

Table 4-3: General improvement requirements for MoMuT::UML

4.2.4.2 Interoperability/Integration requirements

ID	Title	Description	Priority	UC requirements
B5.8-I1	Connect UML elements to RM tool	Provide traceability from UML elements to requirements management system via OSLC (depends on used UML editors, might be split into one requirement per UML editor)	HIGH	UC5.2-GEN-02, UC5.2-GEN-03
B5.8-I2	Integration with ECLIPSE environment	Allow the following activities from within the ECLIPSE development environment: <ul style="list-style-type: none"> - Model editing - Annotating models elements with requirements - Browse requirements - Create requirements - Select model elements for TCG - Initiate TCG - Browse test cases 	HIGH	UC5.2-GEN-01, UC5.2-GEN-02, UC5.2-GEN-03, UC5.2-TCG-03
B5.8-I3	Expose test cases via IOS	Make the generated test cases available to WEFAC for referencing them in V&V activities	HIGH	UC5.2-TCG-05
B5.8-I4	IOS based test target selection	Use IOS based references to model elements for test target coverage selection	HIGH	UC5.2-TCG-02
B5.8-I5	Test model seeding	Provide import from EAST-ADL model for initial test models.	HIGH	UC3.1 related
B5.8-I6	Sync test model with EAST-ADL model	Provide automated sync or consistency check between test model and architecture model (EAST-ADL /AUTOSAR)	MEDIUM	UC3.1 related
B5.8-I7	New test model format support	Support use of UML models built with the model editor and modelling style used for compositional modelling. (still to be defined)	HIGH	UC5.1 related

Table 4-4: Interoperability/Integration requirements for MoMuT::UML

4.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]



4.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

5 MoMuT::SCADE

Provider: AIT

Task #: T6.4.4

Brick #: B5.7

Category: Test case generators
(former brick name: SCAMU)

5.1 Overview

5.1.1 General Description

MoMuT is the name of a family of test case generators from AIT developed in cooperation with TUG. MoMuT family members generate test suites from behaviour models achieving a given fault coverage with faults represented by mutants of the original models.

MoMuT::SCADE is a model-based mutation testing application for the Esterel Technologies' SCADE Suite. Since the SCADE data flow models usually are implementation models, MoMuT::SCADE in contrast to the other MoMuT variants is used for white box testing.

The resulting test cases are near-optimal for the targeted coverage. The tool set can be also used to evaluate and extend existing test cases w.r.t. fault coverage, independent from their origin (manual design, prior product iterations, other members of a product line, other test case generators).

The tool provides traceability between test model elements and test cases.

5.1.2 Related Use cases

MoMuT::SCADE was planned to be applied to UC5.2 - Integrated Modelling of Core Algorithms in TAS Control Platform, which is described in D502.010. The brick has been dropped from the use case, because SCADE will not be used in the use case anymore. At the time of writing, there is no decision taken yet, if another use case can be adopted or if an additional, different test model formalism for MoMuT will be integrated into UC 5.2.

5.2 Specification

5.2.1 Requirements from the UCs

Currently no use case defined – see section 5.1.2.

5.2.2 How will this brick be integrated in the UC

MoMuT::SCADE generates test cases from given models and provides traceability information from model elements to test cases. It can also be used to analyse mutation coverage on a given test suite.

It can be integrated with WEFACT (see Section 2) by providing an interface to initiate automated test case generation runs and by importing the traceability information into WEFACT. Test case inspection would be done in SCADE itself.

5.2.3 Requirements fulfilled by initial tool/method version

Although MoMuT::SCADE is no longer part of UC5.2, the requirements already fulfilled have been left here, until a decision regarding the further approach for the brick has been taken.

ID	Title	Description	Priority	Remarks
UC5.2-TCG-02	Coverage selection	Select coverage for test cases by mutation operators, related requirements, and model elements.	HIGH	
UC5.2-TCG-04	Test case - model traceability	Relate test cases to model elements	HIGH	Available, IOS integration under development in MBAT
UC5.2-TCG-05	V&V activity - test case traceability	Relate V&V activities to test cases	HIGH	Available, IOS integration under development in MBAT
UC5.2-TCG-07	Generate test cases from an implementation model	Generate test cases from an implementation model in SCADE	MEDIUM	Available, IOS integration under development in MBAT. Might need improvements.

Table 5-1: UC5.2 requirements fulfilled by current MoMuT::SCADE version

5.2.4 What will be implemented/provided in the CRYSTAL project

Again, although the brick dropped from use case UC5.2, the requirements derived for it are retained here for now.

5.2.4.1 General Improvements

ID	Title	Description	Priority	UC requirements
B5.7-F1	Improve SCADE TCG performance	If needed for the respective models, improve TCG performance e.g. by ordering mutants.	MEDIUM	UC5.2-TCG-07

Table 5-2: General improvement requirements for MoMuT::SCADE

5.2.4.2 Interoperability/Integration

ID	Title	Description	Priority	UC requirements
B5.7-I1	Connect SCADE elements to RM tool	Provide traceability from SCADE elements to requirements management system via OSLC (possibly leveraging on the SCADE-Doorslink-Integration)	HIGH	UC5.2-GEN-02, UC5.2-GEN-03
B5.7-I2	TCG initiation from ECLIPSE environment	Allow starting test case generation from within the ECLIPSE development environment (SCADE System)	HIGH	UC5.2-GEN-01
B5.7-I3	TCG initiation from SCADE Suite	Allow starting test case generation from within SCADE Suite	HIGH	UC5.2-GEN-01,

Table 5-3: Interoperability and integration requirements for MoMuT::SCADE

5.3 Implementation/Elaboration

This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

5.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

6 FT+ brick integration into Rational Rhapsody

Provider: EADS IW-G

Task #: T6.4.5

Brick #: B2.22

Category: Safety analysis tools

6.1 Overview

6.1.1 General Description

The main objective of this brick is to enable the integration of the safety tool “Isograph FT+” with the design tool “IBM Rational Rhapsody”. The deliverable D203.011 from EADS Cassidian provides a very detailed dedicated chapter on the needs for this integration. Briefly summarizing these needs, the FT+ brick integration into Rational Rhapsody shall fulfil the following needs:

- Export Functions with related data from a Design Model managed by Rhapsody into a Fault-tree model managed by Isograph FT+ as libraries
 - o Note: An automatic creation of Fault-trees from the Rhapsody model is NOT envisaged
- Define traceability links between functions (in the Rhapsody models) and libraries in FT+
- Enable consistency checks between Rhapsody model elements and Isograph FT+ elements (e.g. check if the functions and related data in Rhapsody are consistent with the respective libraries in FT+)
- As an option, a Suspect Link feature should be implemented. It means that a link between a Rhapsody element and an FT+ element should be marked as “suspect” if one of the linked elements has changed.
- All features should be implemented such that their execution does not take so long that they negatively impact the engineering work flow.

6.1.2 Related Use cases

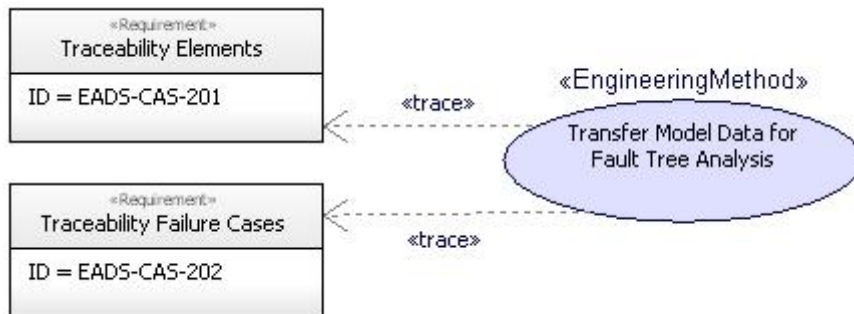
This brick is related to the use case 2.3 from EADS Cassidian.

6.2 Specification

6.2.1 Requirements from the UCs

The Requirements for the FT+ brick integration into Rational Rhapsody have been taken from the Use Case Description D203.011, Chapter 8.2 “Perform Functional Safety Analysis”.

EM202_01_01 - Transfer Model Data for Fault Tree Analysis



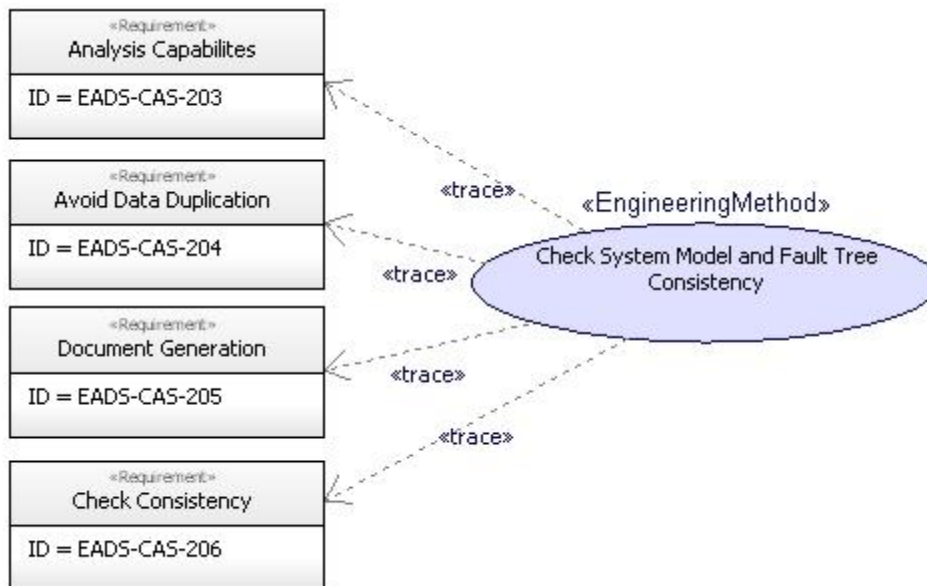
EADS-CAS-201

The SEE shall allow to trace from basic elements in a Fault Tree Analysis tool to use cases, blocks and primitive operations of a SysML model.

EADS-CAS-202

The SEE shall allow to trace from failure cases in a Fault Tree Analysis tool to tagged values of use cases, blocks and primitive operations of a SysML model.

EM202_01_02 - Check System Model and Fault Tree Consistency



EADS-CAS-203

The SEE shall provide safety analysis capabilities related to system design. The system design model shall be the input and baseline for the analysis.

EADS-CAS-204

The SEE shall ensure consistency of the data between System and Fault Tree Analysis model. Duplication of data shall be avoided.

EADS-CAS-206

The SEE shall allow to compare baselines of system (SysML) model and fault tree models. The SEE shall highlight inconsistencies using different colours. Per user setting, matching parts shall be suppressed.

EADS-CAS-205

The SEE shall provide cross system (SysML) model and fault tree model report and document generation capabilities on the basis of a selected baselines.

6.2.2 How will this brick be integrated in the UC

Integration of the brick into the UC will be managed by EADS Innovation Works and EADS Cassidian. The integration has been described in the deliverable D203.011, PA2012_01 Perform Functional Safety Analysis, via Engineering Method description for the Engineering Methods “US202_Check System Model and Fault Tree Consistency_EM202_01_02” and US202_Transfer Model Data for Fault Tree Analysis_EM202_01_01.

PA202_01 - Perform Functional Safety Analysis (from EADS Cassidian Deliverable D_203.011):

Engineering Method: US202_Check System Model and Fault Tree Consistency_EM202_01_02					
Purpose: Check that the functions, related failure cases and severity classification is consistent to the fault tree elements and there related reliability figures.					
Comments: none					
Pre-Condition		Engineering Activity		Post-Condition	
FTA performed		01. Set System Model Scope 02. Retrieve FTA Data 03. Correlate System Model Elements With FTA Elements 04. Compare Reliability Figures 05. Generate Safety Coherence Report		Traceability between model and fault tree elements established	
Notes: -		Notes: -		Notes: -	
Artefacts Required as inputs of the Activities		Artefacts used internally within the Activities (optional)		Artefacts Provided as outputs of the Activities	
Name:	SystemArchitecture	Name:	-	Name:	-
Generic Type:	SysMLModel	Type:	-	Generic Type:	-
Required Properties:	Content MethodCompliance	Properties:	-	Provided Properties:	-

Description & Interoperability Additional Constraints: MethodCompliance = MBSE guide / IBM Harmony SE Content = Logical (opt. Physical) Architecture		Description: -		Description & Interoperability Additional Constraints: -	
Name:	SystemFTA	Name:	-	Name:	-
Generic Type:	FaultTree	Type:	-	Generic Type:	-
Required Properties:	Content	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: Content = Fault Tree (Validated)		Description: -		Description & Interoperability Additional Constraints: -	
Name:	-	Name:	-	Name:	ConsistencyReport
Generic Type:	-	Type:	-	Generic Type:	Report
Required Properties:	-	Properties:	-	Provided Properties:	
Description & Interoperability Additional Constraints: -		Description: -		Description & Interoperability Additional Constraints:	
Name:	SysUseCase	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = UseCase		Description: -		Description & Interoperability Additional Constraints: -	
Name:	PhysicalBlock	Name:	-	Name:	-

Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = Block		Description: -		Description & Interoperability Additional Constraints: -	
Name:	LogicalBlock	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = Block		Description: -		Description & Interoperability Additional Constraints: -	
Name:	SystemFunction	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-

Description & Interoperability Additional Constraints: SystemModelElementType = PrimitiveOperation		Description: -		Description & Interoperability Additional Constraints: -	
Name:	SystemSafetyTrace	Name:	-	Name:	-
Generic Type:	Link	Type:	-	Generic Type:	-
Required Properties:	ArtefactStatus LinkSource LinkTarget	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: LinkSource = SystemModelElement LinkTarget = FaultTreeElement ArtefactStatus = Validated		Description: -		Description & Interoperability Additional Constraints: -	

Engineering Method: US202_Transfer Model Data for Fault Tree Analysis_EM202_01_01		
Purpose: Providing functions and architecture elements with associated reliability figures to the safety analysis tool for FTA.		
Comments: Reliability figures are stored within the system model. Rationale: The system model is part of the specification as such. The fault tree as an analysis model is not (it is the justification for reliability figures required).		
Pre-Condition	Engineering Activity	Post-Condition
Functional analysis with logical and/or physical architecture performed. Data provided in SysML model with dedicated structure.	01.Extract System Use Cases from model 02. Select Use Case and type for FTA 03. Extract physical architecture data (physical FTA only) 04. Extract logical architecture data 05. Generate basic elements library 05. Generate basic elements library 06. Establish traces to originating elements	Model elements transformed into a set of artefacts (basic elements) as input for fault tree analysis.
Notes: -	Notes: -	Notes: -

Artefacts Required as inputs of the Activities		Artefacts used internally within the Activities (optional)		Artefacts Provided as outputs of the Activities	
Name:	SystemArchitecture	Name:	-	Name:	-
Generic Type:	SysMLModel	Type:	-	Generic Type:	-
Required Properties:	Content MethodCompliance	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: MethodCompliance = MBSE guide / IBM Harmony SE Content = Logical (opt. Physical) Architecture		Description: -		Description & Interoperability Additional Constraints: -	
Name:	SysUseCase	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = UseCase		Description: -		Description & Interoperability Additional Constraints: -	
Name:	LogicalBlock	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElementType SystemModelElementType SystemModelElementType	Properties:	-	Provided Properties:	-

	SystemModelElement				
Description & Interoperability Additional Constraints: SystemModelElementType = Block		Description: -		Description & Interoperability Additional Constraints: -	
Name:	PhysicalBlock	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElement SystemModelElement SystemModelElement SystemModelElement	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = Block		Description: -		Description & Interoperability Additional Constraints: -	
Name:	SystemFunction	Name:	-	Name:	-
Generic Type:	SysMLModelElement	Type:	-	Generic Type:	-
Required Properties:	SystemModelElement SystemModelElement SystemModelElement SystemModelElement	Properties:	-	Provided Properties:	-
Description & Interoperability Additional Constraints: SystemModelElementType = PrimitiveOperation		Description: -		Description & Interoperability Additional Constraints: -	
Name:	-	Name:	-	Name:	SystemFTA
Generic Type:	-	Type:	-	Generic Type:	FaultTree
Required Properties:	-	Properties:	-	Provided Properties:	Content

Description & Interoperability Additional Constraints: -		Description: -		Description & Interoperability Additional Constraints: Content = FT Elements (library)	
Name:	-	Name:	-	Name:	SystemSafetyTrace
Generic Type:	-	Type:	-	Generic Type:	Link
Required Properties:	-	Properties:	-	Provided Properties:	ArtefactStatus LinkSource LinkTarget
Description & Interoperability Additional Constraints: -		Description: -		Description & Interoperability Additional Constraints: LinkSource = SystemModelElement LinkTarget = FaultTreeElement ArtefactStatus = Validated	

6.2.3 Requirements fulfilled by initial tool/method version

The envisaged functionality as described in 6.2.1 is not yet available and will be developed in the frame of CRYSTAL.

6.2.4 What will be implemented/provided in the CRYSTAL project

The envisaged functionality as described in 6.2.1 is not yet available and will be developed in the frame of CRYSTAL.

6.2.4.1 New and improved features

See Chapter 6.2.1

6.2.4.2 Interoperability requirements

See Chapter 6.2.1

6.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]



6.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

7 NuSMV

Provider: FBK

Task #: T6.4.6

Brick #: B2.43

Category: Safety Analysis automation and verification

7.1 Overview

7.1.1 General Description

The extended version of the NuSMV model checker is a tool suite for model-based development that covers several phases of system engineering, including requirement analysis and validation, verification of functional and non-functional requirements, safety assessment, and contract-based architectural design. For this purpose, the extended version of the NuSMV model checker includes several tools, namely nuXmv (for requirements analysis and functional verification), xSAP (for safety assessment) and OCRA (for contract-based design). More in detail, the extended version of NuSMV supports the following activities:

- Requirements validation:
to check the quality of a set of requirements by formalizing them into a formal language, e.g. Linear Temporal Logic (LTL), and using verification techniques to discover errors such inconsistencies, missing requirements, wrong conditions, over-specifications.
- Verification of functional correctness:
to check the compliance of a system model with respect to a set of properties using model checking for infinite-state systems.
- Safety assessment:
to check the compliance of a system model with respect to safety properties, and analyse its robustness in presence of faults; it includes techniques such as Fault Tree Analysis and Failure Modes and Effects Analysis.
- Contract-based architectural design:
to analyse the architectural decomposition of a system using contract-based design, and use contracts for compositional reasoning and a proper reuse of components.

7.1.2 Related Use cases

The NuSMV brick is a cross-domain toolset for model-based engineering. In Crystal, it will be evaluated in the aerospace domain. In particular, the brick will be integrated in Use Case 2.8 ("Public Use Case Aerospace"). The brick will implement functionalities for the following engineering methods identified in UC2.8:

- Verify Design against requirements (functionality: verification of functional correctness)
- Fault Tree generation (functionality: safety assessment)

It is also under consideration the integration of the brick into Use Case 2.6 ("Multi-Mode Navigation System"), in particular for the following engineering method:

- Requirements validation

Version	Nature	Date	Page
V1.0	P	2014-02-07	52 of 125

7.2 Specification

7.2.1 Requirements from the UCs

The NuSMV brick will address the following requirements coming from the UCs.

7.2.1.1 Use Case 2.6

We have identified the following requirements:

- The tool shall be able to retrieve, using IOS primitives, the list of requirements to be validated
- The tool shall be able to check consistency and completeness of a set of requirements
- The tool shall be able to generate verification results, such as traces, in a format to be made available through the IOS

7.2.1.2 Use Case 2.8

We have identified the following requirements:

- The tool shall be able to retrieve, using IOS primitives, the nominal and dysfunctional model(s), written in Altarica language
- The tool shall be able to retrieve, using IOS primitives, the set of properties to be used for functional verification
- The tool shall be able to retrieve, using IOS primitives, the list of Failure Conditions to perform Fault Tree Analysis
- The tool shall be able to perform functional verification, given a set of nominal (and dysfunctional) model(s) and a set of properties
- The tool shall be able to generate a Fault Tree, given the nominal and dysfunctional model(s), written in Altarica language, and the list of Failure Conditions
- The tool shall be able to generate traces in a format to be made available through the IOS
- The tool shall be able to generate Fault Trees in a format to be made available through the IOS
-

7.2.2 How will this brick be integrated in the UC

We exemplify the way NuSMV will be integrated with UC 2.8, to perform Fault Tree Analysis. We consider the following scenario:

- The safety designer would like to generate fault trees corresponding to a list of failure conditions
- The safety data are stored in a safety in-house tool
- Dysfunctional models are available

In this scenario, we assume that NuSMV provides the machinery for generating Fault Trees, and a user-level application (e.g., FT+) is used to generate requests to NuSMV, and also as a graphical display for the generated artefacts.

The interaction between the tools can be described as follows.

First, failure conditions are retrieved (points 1, 2 and 3 in Figure 7-1); the request generated from the application (FT+ in this example) is forwarded to a safety repository, and the result is sent back.

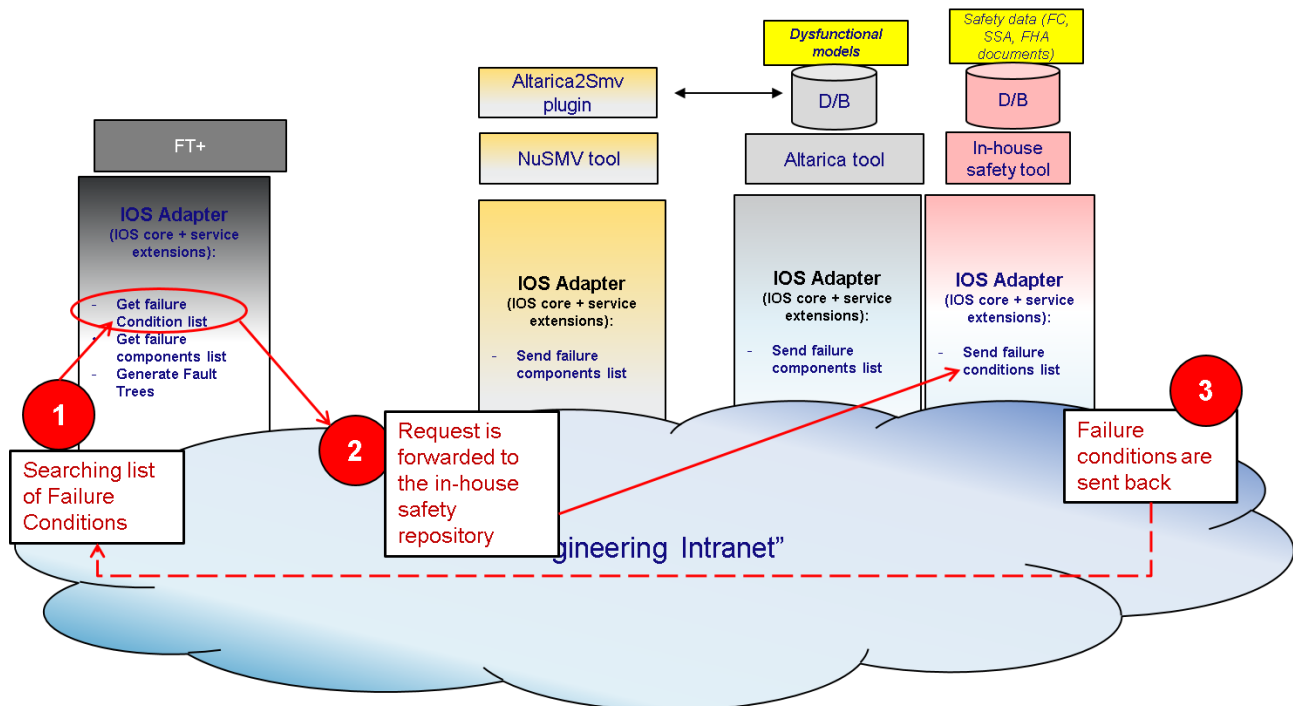


Figure 7-1: Failure Conditions retrieval.

Second, the NuSMV tool is invoked in order to generate the list of failure components (point 4 in Figure 7-2). NuSMV uses as input the nominal and dysfunctional models, written in Altarica, and the list of failure conditions. The Altarica models are converted into SMV language, the input language of the NuSMV model checker, using the Altarica2Smv plugin. Internally, NuSMV implements routines, based on model checking techniques, to generate Fault Trees.

In this example, NuSMV generates the list of failure components corresponding to the set of Minimal Cut Sets (MCSs) – i.e. a flat (two-level) FT.

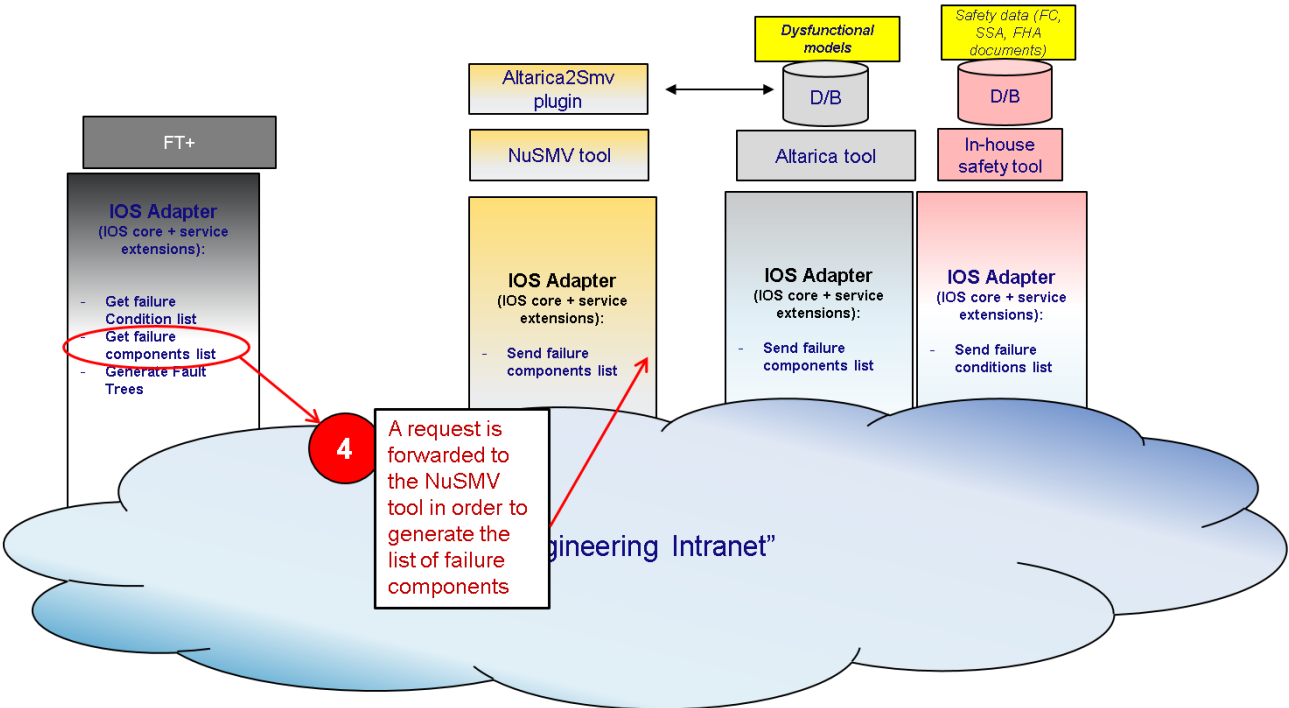


Figure 7-2: NuSMV invocation.

Then, the failure components are sent to FT+ (point 5 in Figure 7-3).

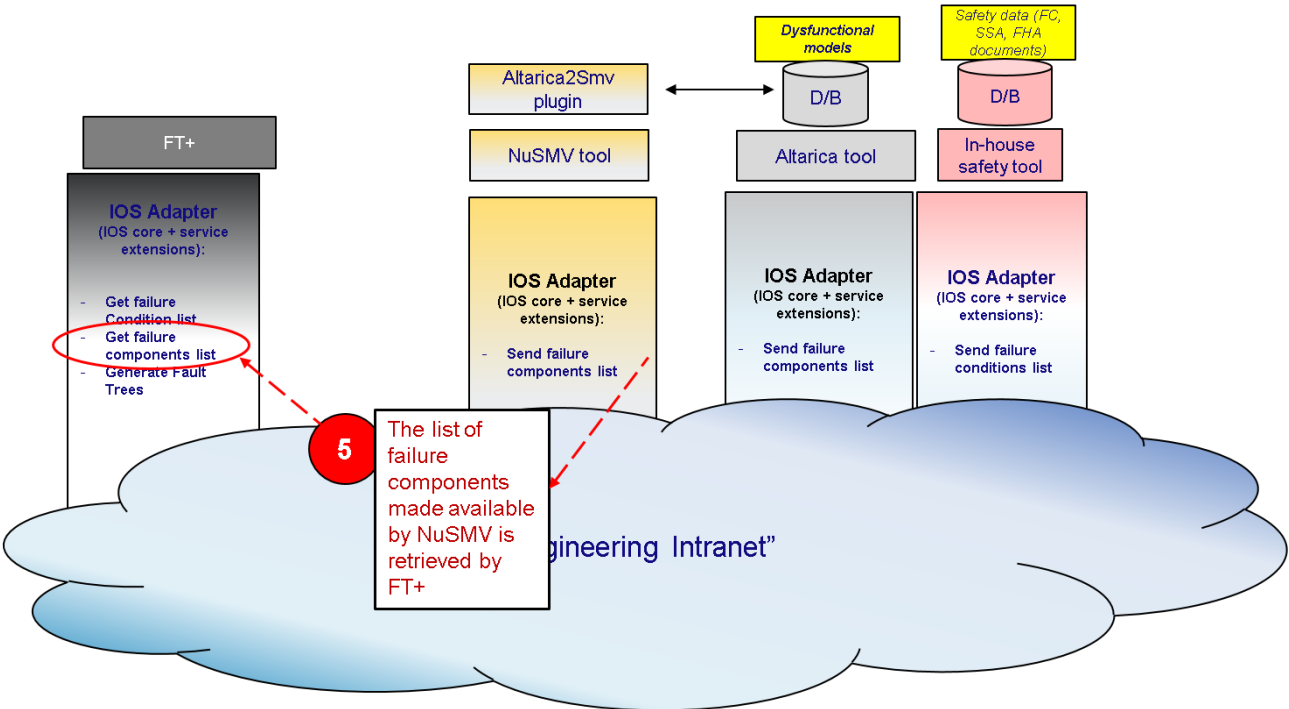


Figure 7-3: Failure Components are sent back.

Finally, the list of failure components produced by NuSMV is assembled and visualized in FT+ using standard Fault Tree notation (point 6 in Figure 7-4).

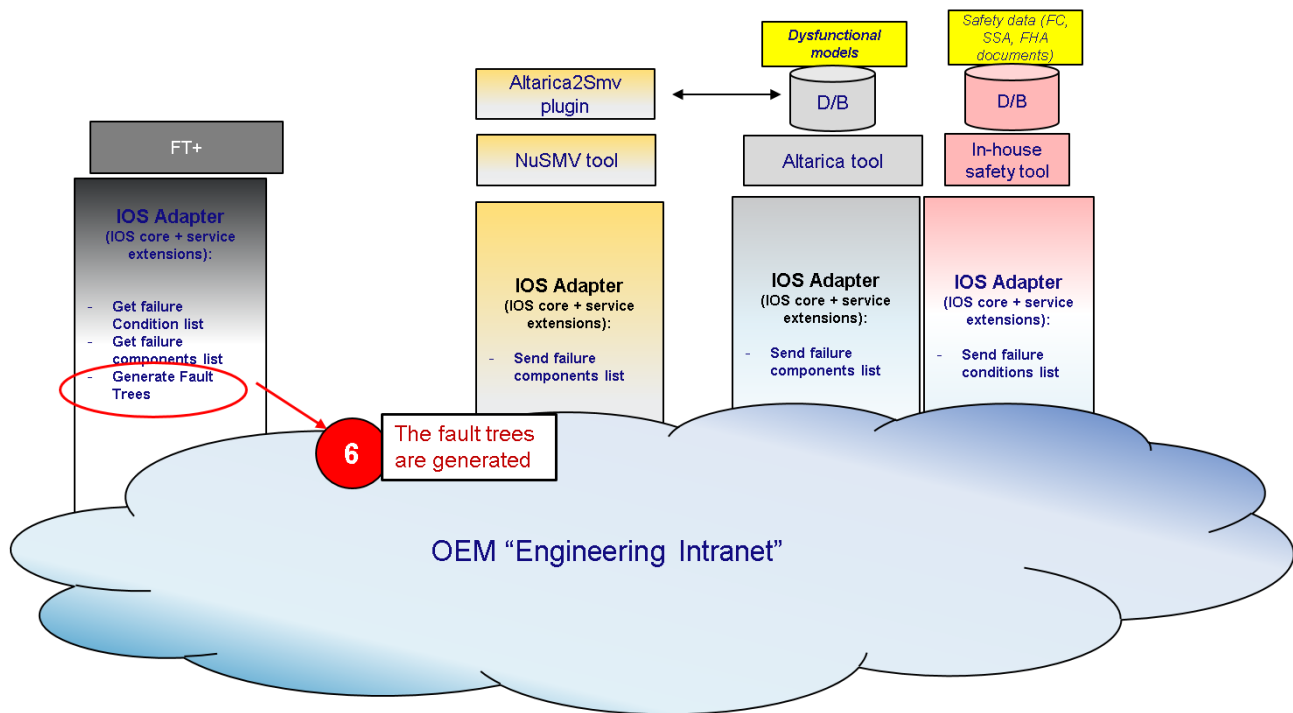


Figure 7-4: Generation of Fault Trees.

7.2.3 Requirements fulfilled by initial tool/method version

The NuSMV brick already implements routines for requirements analysis, functional verification, safety assessment and contract-based architectural design. Such functionalities will be extended and adapted to the Crystal RTP. The new features are explained in detail in Section 7.2.4.1.

Additionally, the tool will be extended to allow interoperability with the Crystal IOS and RTP. The corresponding requirements are discussed in more detail in Section 7.2.4.2.

7.2.4 What will be implemented/provided in the CRYSTAL project

The extended version of the NuSMV model checker will be developed by FBK, and adapted to Crystal needs, in order to support the IOS specification and to be integrated in the Crystal RTP.

7.2.4.1 New and improved features

We envisage the following directions of development:

- The nuXmv tool will be extended in order to incorporate the requirements analysis functionality implemented in the RAT tool (Requirements Analysis tool – also developed by FBK) so as to create a seamless environment for requirements validation and functional verification

- The NuSMV/OCAS plugin will be extended and possibly adapted to match the dialect of Altarica used in UC2.8. Some syntactical and semantical limitations of the current version will be removed. Moreover, we have identified some implementation-level improvements that may affect the performance of the plugin.
- Concerning the safety assessment engine, we will investigate the possibility to improve the readability of the generated Fault Trees. Currently, the tool is capable to generate flat (two-level) FTs. Possibility of generating some form of hierarchical layout – using contract-based techniques - is currently under investigation. Such improvement may also be beneficial in terms of performance.

7.2.4.2 Interoperability requirements

NuSMV will be extended with new interfaces, according to users' needs and as a consequence of requirements coming from the UCs. In particular, we envisage the following extensions:

- New formats will be defined in order to exchange verification data, such as traces and Fault Trees.
- Requirements will be linked with models.
- Traceability of artefacts will be supported, in particular it will be possible to trace verification and safety artefacts to models.

NuSMV will be extended in order to be integrated into the Crystal IOS. Moreover, based on the ongoing integration into SafeCer CTF (Certification Tool Framework), NuSMV will be updated to be inserted into the CRYSTAL RTP. More specifically:

- An adapter will be implemented to interface NuSMV and make it compliant with the CRYSTAL IOS.
- The existing NuSMV/OCAS interface will be extended and adapted to match CRYSTAL RTP.
- Results formats, such as formats for execution traces and for safety artefacts (FTs and FMEA tables), will be created in order to exchange results with CRYSTAL tools.

7.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

7.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

8 C²FT

Provider: FhG IESE

Task #: T6.4.7

Brick #: B3.97

Category: Safety Analysis Tools

8.1 Overview

8.1.1 General Description

C²FT is the evolution of Fault Tree Analysis (FTA) and Component Fault Trees (CFT). This technique has been created with the aim of facilitating fault tree analysis during the design process. This is achieved by the C²FT approach by defining a formal relation between a CFT and a component in a component model. This relation is not only established between the two models but also between their interfaces, so that failure modes of the CFT are associated with the incoming and outgoing interfaces of components. See Figure 8-1.

The formalization of the relation between the safety model and the component model, represents and enhancement with respect to the predecessor techniques regarding:

- Consistency
- Traceability
- Maintainability
- Reusability

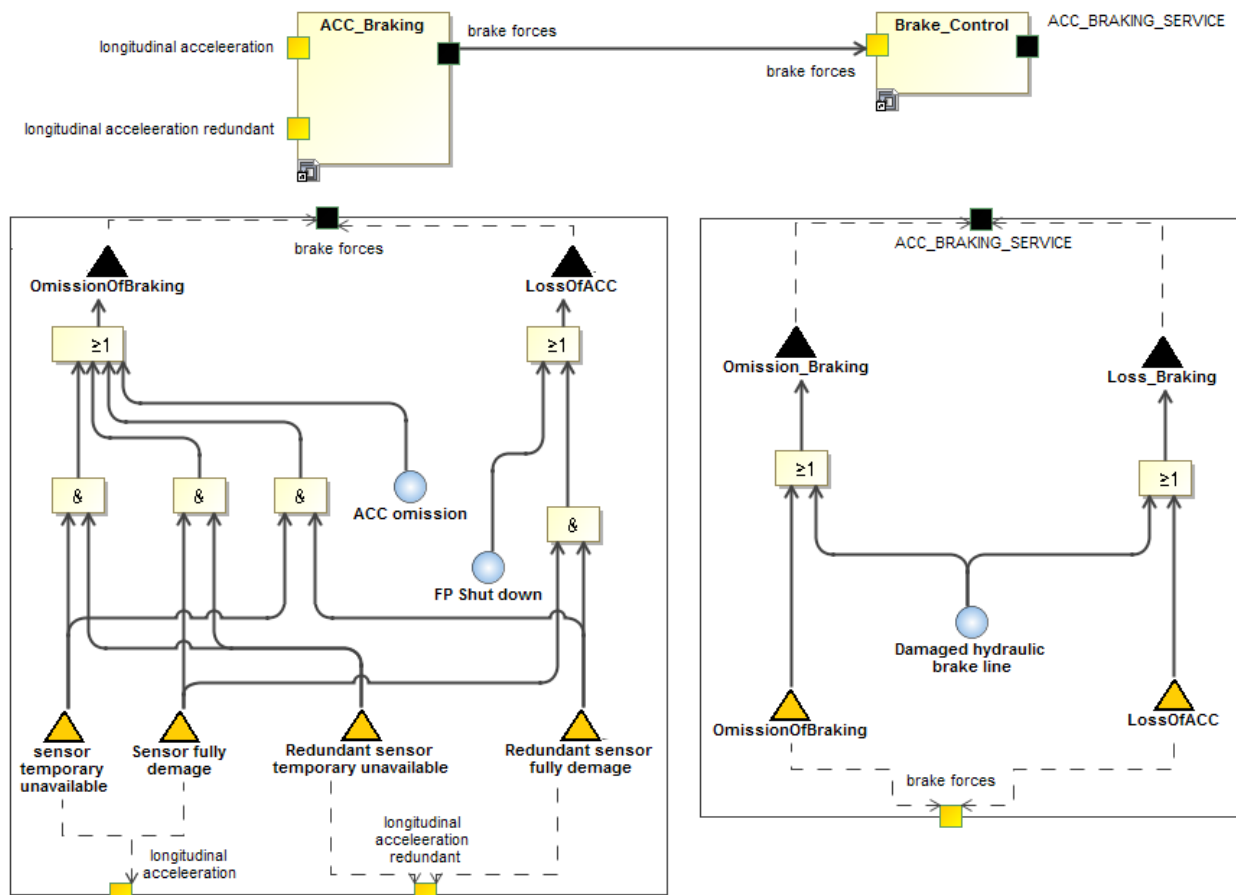
Furthermore, C²FTs also help handling the complexity of the safety analysis by keeping the same modular and hierarchical structures as can be defined in the component/architectural models.

An implementation of the technique already exist in form of plug ins provided as an extension of the commercial tool Magic Draw⁴.

8.1.2 Related Use cases

C²FTs is relevant for the use case UC3.3 “*Functional power train architecture & control development*” on charge of (AVL). See deliverable “*Milestone report – V1*” (D303.011) for detailed information on this use case.

⁴ Magic Draw is a software and system modeling tool developed and distributed by No Magic Inc.

Figure 8-1: C²FT

8.2 Specification

8.2.1 Requirements from the UCs

Requirements for C²FTs with respect to the use case UC3.3 “*Functional power train architecture & control development*” are still under discussion. These would be documented here as soon as they have reached a stable version.

8.2.2 How will this brick be integrated in the UC

The C²FT tool will provide the capabilities to model Fault Trees, Component Fault Trees (CFT) and Component Integrated Fault Trees (C²FTs). Furthermore, qualitative (e.g. minimal cutset) as well as quantitative analysis (e.g. top event probability) can be performed. C²FT would allow to perform FTA on basis of architectural models, as described in section 8.1.1.

8.2.3 Requirements fulfilled by initial tool/method version

Recall section 8.2.1.

8.2.4 What will be implemented/provided in the CRYSTAL project

In a world with growing number of distributed development approaches, the need for a safety integrated development has become very urgent. This has been evidenced during the definition and evaluation of the C²FTs method. For this reason, FHG IESE has the main objective of facilitating this integration process at several levels. On the one hand, we desire to integrated several safety analysis techniques, so that heterogeneous analysis are possible, and for which C²FT represent a reached milestone. Furthermore, we also want to reduce the gap between different tools, by enhancing their interoperability.

In order to facilitate the interaction between tools, FHG IESE want to develop an Open Safety Model (OSM) see Figure 8-2. The function of the OSM is to define the supported analysis techniques and serve as a model exchange layer by offering an API that allows to exchange model relevant data as well as analysis results independent of the tool making use of it.

Traditionally, the integration between tools have been achieved by defining interfaces and a exchange format for every pair of tools. This is however a non ending task. Therefore, the OSM pretends to avoid this situation by offering a generalized format that could be used by all tools. The advantage of it would be that compatibility between two tools does not need to be guaranteed but only between the tool and the OSM.

8.2.4.1 New and improved features

The interoperability with other tools is of high relevance for FHG IESE, therefore one of the goals to be achieved in Crystal relates to achieve higher levels of data exchange. Although the current implementation offers already the exchange of model information, this is restricted to a small set of tools. Therefore with the implementation of the OSM we expect to increase the chances of interoperability of tools and integration of techniques.

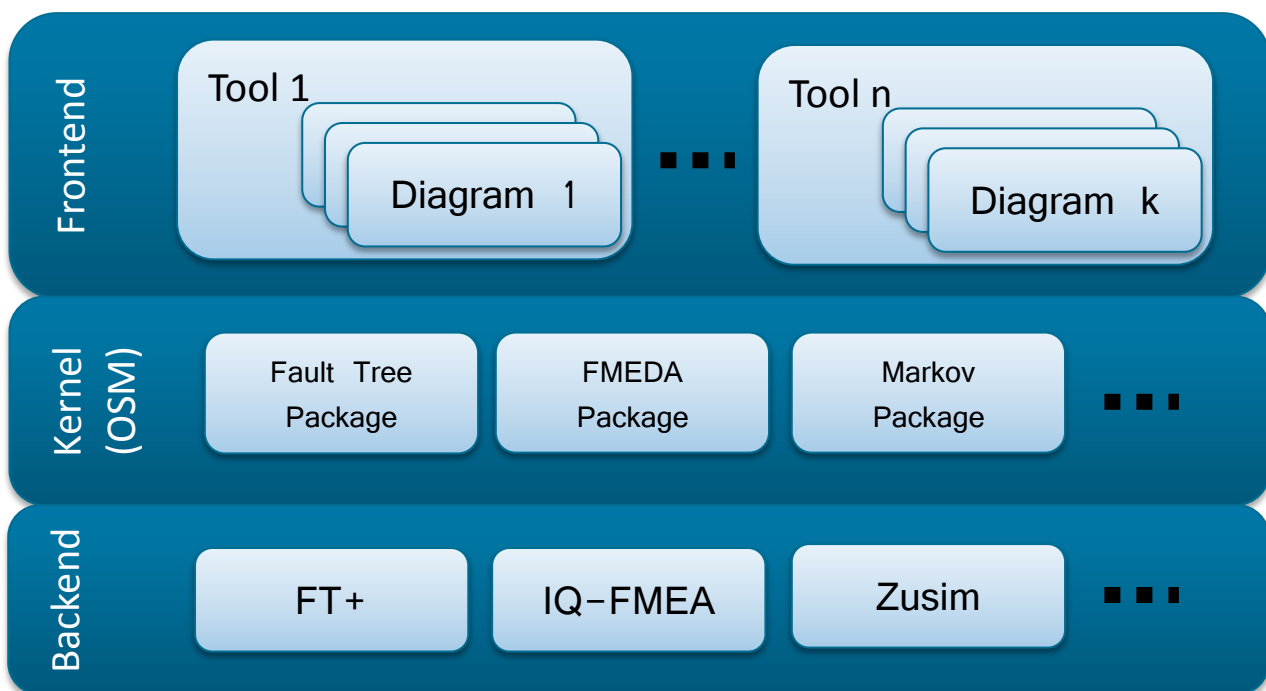


Figure 8-2: Open Safety Model (OSM)

8.2.4.2 Interoperability requirements

With the focus in interoperability the tools created at FHG IESE will be developed with the aim to comply with the Crystal IOS as well as to be integrated in the Crystal RTP.

8.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

8.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

9 Safety for Avionic Design and Analysis Framework

Provider: EADS IW-G

Task #: T6.4.9

Brick #: B2.41

Category: Safety Analysis Tools

9.1 Overview

The basic motivation for developing this Safety Brick for an Avionic Design and Analysis Framework is to provide a mean to System and Safety Engineers that enables the definition of System Architectures optimized for Safety & Reliability and following robust design principles.

9.1.1 General Description

Figure 9-1 below provides a first draft overview of the envisaged Avionic Design and Analysis Framework. It is based mainly on input provided by the Airbus Environmental Control System Use Case. The core part of the framework is dedicated to the integration of safety and design models managed by Simulink. A connection to Design Verifier is needed for analysis purposes. The framework will also include a dedicated tool that allows triggering of failure injections and visualization of safety analysis results.

It is expected to later extend the framework to other tools such as Isograph FT+, IBM Doors, or Airbus internal tools for Particular Risk Analysis.

The connections between the different tools involved in this framework shall be based on the CRYSTAL IOS.

9.1.2 Related Use cases

The Safety framework for Avionic Design and Analysis is currently driven mainly by the Use Case 2.1a – Airbus Environmental Control Systems.

It is expected to extend the framework such that it can also support the Use Case 2.1b – Airbus Simulation for Particular Risk Analysis, and Use Case 2.1c – Airbus Fuel Management Risk Analysis.

The definition of the core part of framework is tightly linked to the definition of the SEE of the Airbus Environmental Control System Use Case as described in the deliverable D2.1.1.1-1 (Use Case Definition, Airbus-Germany, Environmental Control Systems)

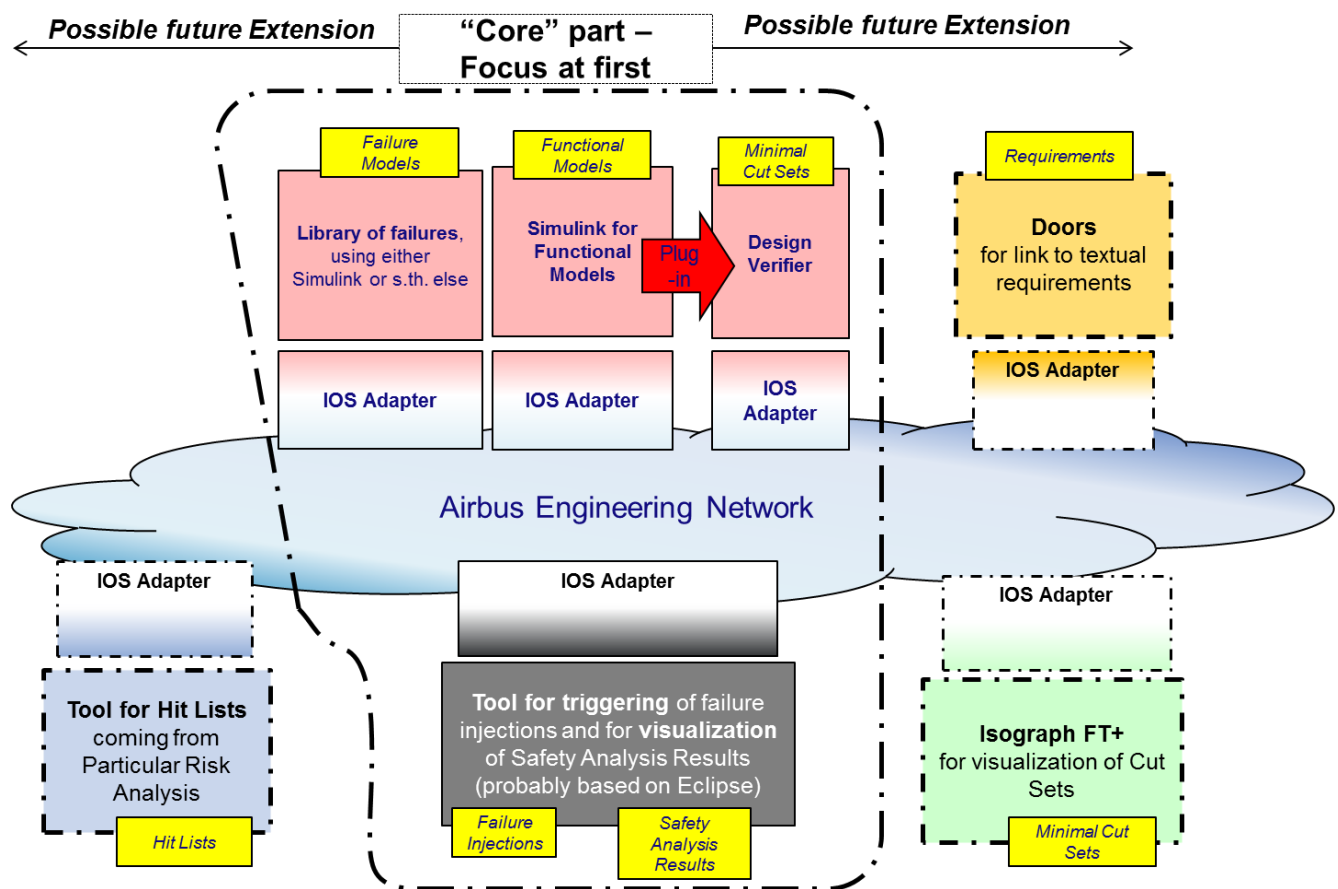


Figure 9-1: Avionic Design and Analysis Framework

9.2 Specification

9.2.1 Requirements from the UCs

The following requirements will be addressed by this brick, grouped by use case:

9.2.1.1 UC2.1a - Airbus Environmental Control Systems

Figure 9-2 provides a more detailed view on the envisaged way of working of the core part of the Safety for Avionic Design and Analysis Framework:

1. The framework shall consist of a dedicated tool that allows the safety engineer to trigger failure injections by selection of either relevant Simulink blocks of the design model or the entire design model, and by the selection of relevant Failure classes and fault representatives.
2. Based on this, functional models defined by Simulink shall be enriched by observers and failures and then sent to Design Verifier tool.
3. The Design Verifier will be used to compute counter-examples for a dedicated Safety Requirement. Design Verifier stops its calculation when it has found a first counter example. For this counter-

example it will provide the minimal cut set. After stopping, Design Verifier shall be automatically re-launched to compute other counter-examples

4. A dedicated visualization tool shall extract out of a set of counter examples the minimal cut sets.

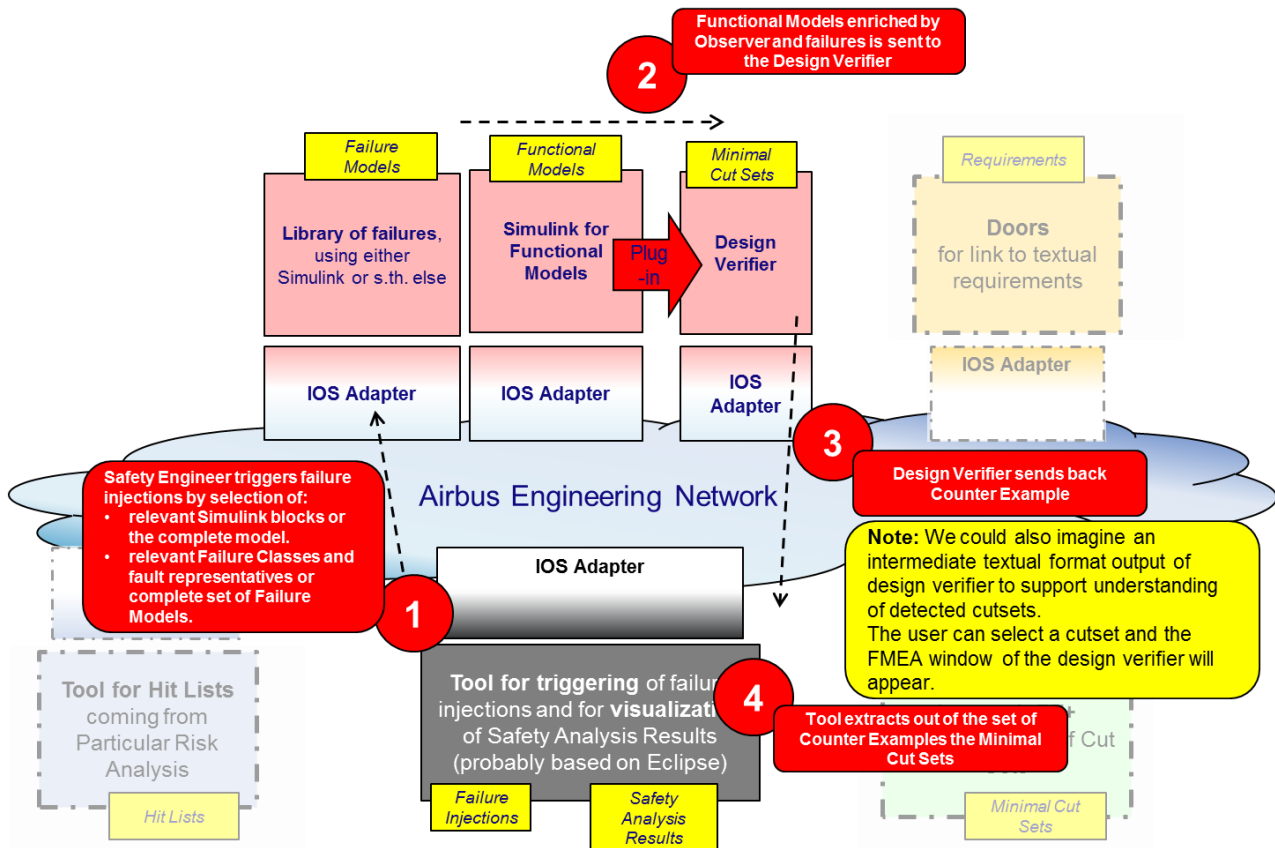


Figure 9-2: Core of Avionic Design and Analysis Framework

The next pictures show briefly possible extension of the Safety for Avionic Design and Analysis Framework. It is planned to connect the tool Isograph FT+ to the framework in order to visualize the failure model that results from the minimal cut sets and the design model.

It is also planned to add IBM Doors to this framework in order to provide traceability functionality between observers and textual requirements.

These parts will however be further analysed and specified for the next version of this deliverable.

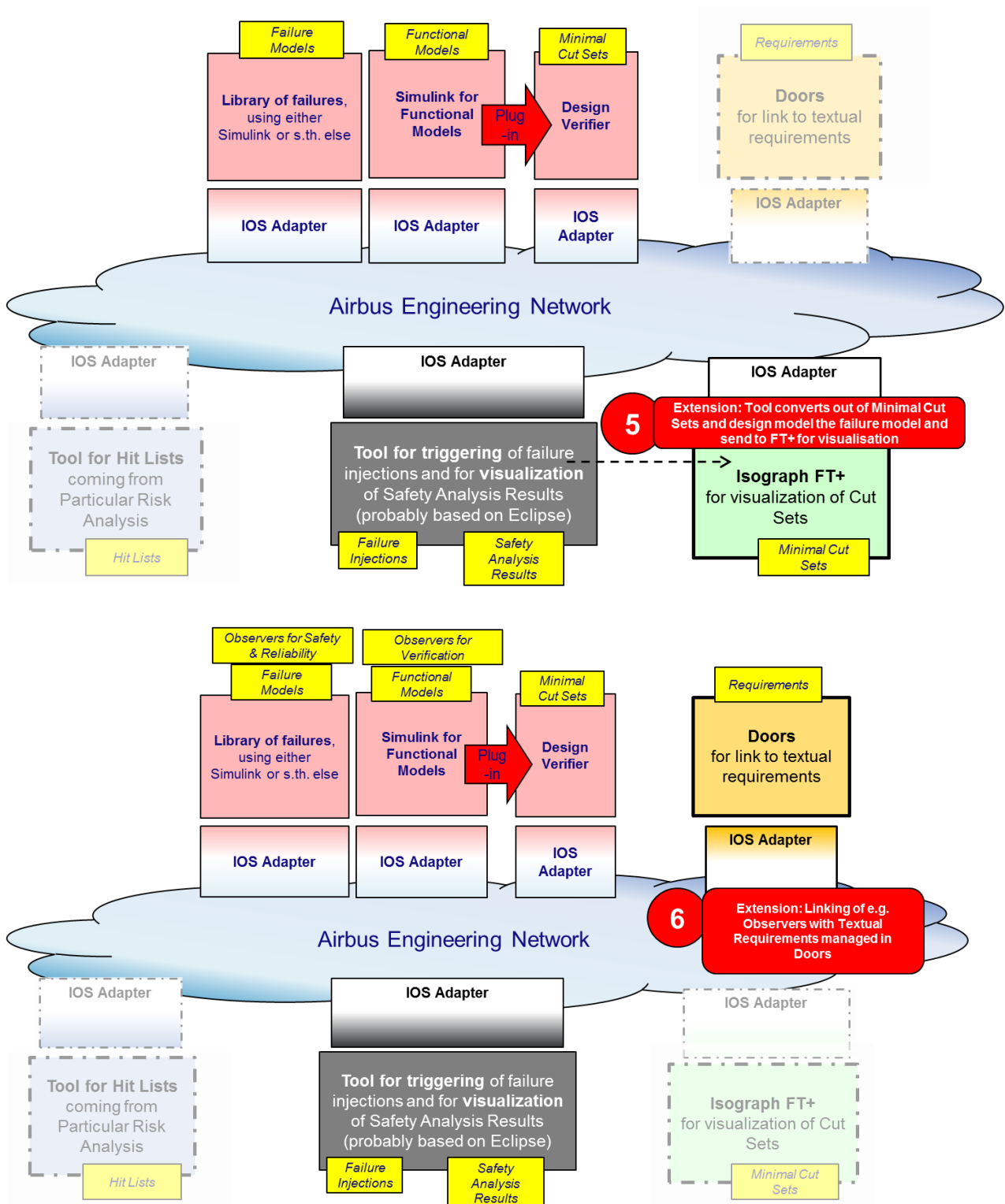


Figure 9-3: Possible extensions of the Safety for Avionic Design and Analysis Framework

An explanation of the terms “observers” “counter-examples” and “minimal cut-sets” and of acceptable levels of minimal cut sets for Avionics can be found in the deliverable D201.011.

9.2.1.2 UC2.1b - Airbus Simulation for Particular Risk Analysis

It is assumed that the core part of the framework will also support the Airbus Simulation for Particular Risk Analysis Use Case. Additional Requirements that are specific for the Airbus Simulation for Particular Risk Analysis Use Case will mostly be defined for a later iteration of this document.

A possible extension of the framework to cover UC2.1b needs is represented in the picture below. It is planned to add an Airbus internal tool to the framework that contains the Hit List that is resulting from Particular Risk Analysis. The Hit List includes all system components (e.g., sensors, computers, switches, power centers, cable routings) that are positioned within the trajectory of debris resulting from a particular risk event such as an engine rotor burst and that will be hit and destroyed by such an event. This Hit List could be used as another input to trigger failure injections.

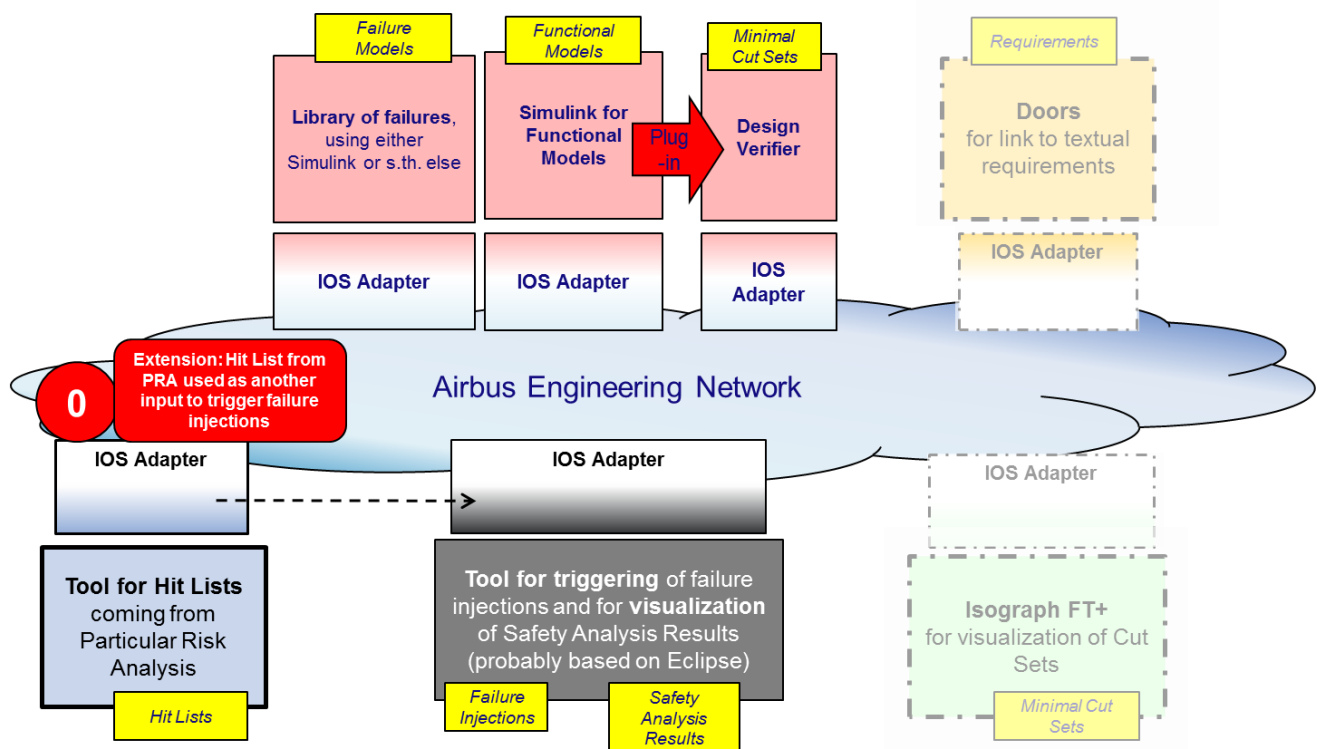


Figure 9-4: Extension of the framework to cover UC2.1b

9.2.1.3 UC2.1c - Airbus Fuel Management Risk Analysis.

It is assumed that the core part of the framework will also support the Airbus Fuel Management Risk Analysis Use Case. Additional Requirements that are specific for the Airbus Fuel Management Risk Analysis Use Case will be defined for a later iteration of this document.

9.2.2 How will this brick be integrated in the UC

This brick will be the core part of the SEE of U.C 2.1a. It will also play a key role in UC 2.1b and UC 2.1c.

9.2.3 Requirements fulfilled by initial tool/method version

TBD for the next version of this deliverable.

9.2.4 What will be implemented/provided in the CRYSTAL project

Within CRYSTAL, it is planned to focus in the first place on the definition of the dedicated tool for triggering the failure injections and for visualization of Safety Analysis Results, and on the interfaces between this tool and Matlab Simulink, Design Verifier, and other tools relevant for future framework extensions.

9.2.4.1 New and improved features

This part will be defined for the next version of the deliverable.

9.2.4.2 Interoperability requirements

This part will be defined for the next version of the deliverable.

9.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

9.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

10 Claims Language Boilerplate

Provider: IFX-UK

Task #: T6.4.10

Brick #: B3.99

Category: Safety requirements engineering

Replaced brick "URML".

10.1 Overview

Extend the Security semi-formal notation language from the 1991 ITSEC standard http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf for safety, implement a boilerplate and tooling to translate from Natural Language to the Claims extension.

10.1.1 General Description

The ISO26262 strongly recommends for ASIL C and D that the requirements are written with a semi-formal notation. Originally Infineon considered using a new UML extension called URML from Siemens as it was restricted enough to be used for Requirements. The analysis of all the options possible was undergone within the ARTEMIS VeTess project concluded that for Infineon, the best solution would be to extend the ITSEC security standard Claims language (see http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf) for Safety. Therefore we plan to extend the Claims boilerplate and write a tool to translate from Natural language to the Claims format – possibly the DODT that was developed under the ARTEMIS CESAR project http://publik.tuwien.ac.at/files/PubDat_201539.pdf

10.1.2 Related Use cases

Currently within the Requirements Engineering flow we have a very manual process during the collation and storage of the requirements at the start phase. The Requirements are all in a natural language format and are translated to a semi-formal style notation within a manual process. The review and quality gateway is also a manual and time-consuming process subject to misinterpretation and data integrity issues as shown in the excerpt from the Requirements Engineering DFD diagram of use case 3.3 and the automotive public use-case.

Within the Crystal project we plan to automate the translation process from natural language to semi-formal notation to avoid manual errors. This tool will be a brick and may need to interact with other tools, for example should the naming be variant related it may need to access the information from the Information database KiD (Knowledge and Information Database).

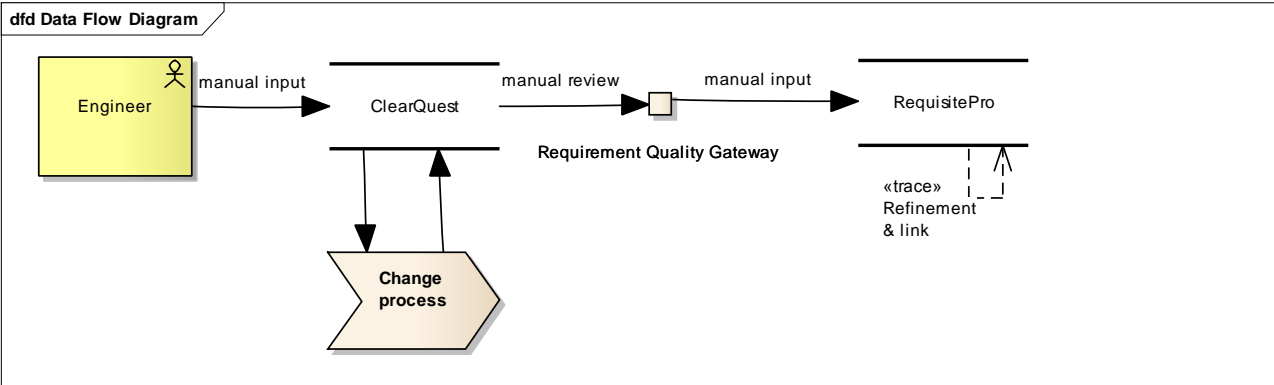


Figure 10-1: Quality gateway

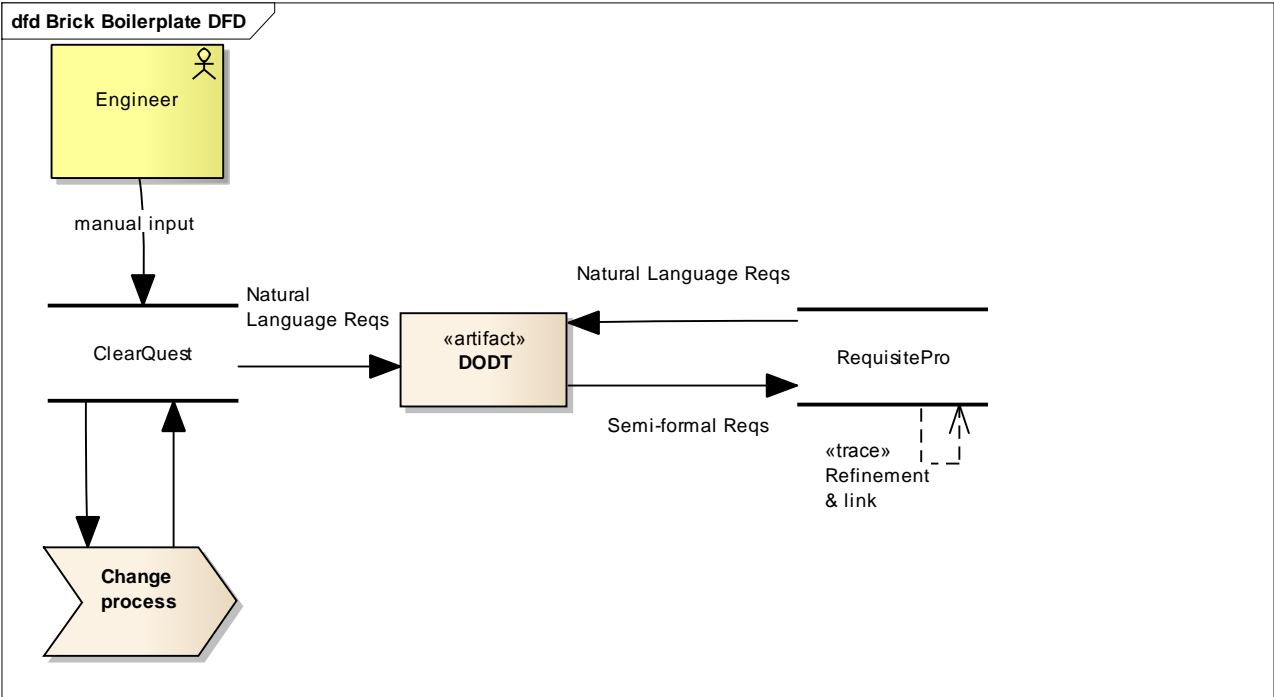


Figure 10-2: DODT

10.2 Specification

Currently the DODT is being analysed and syntax options are discussed with AVL.

10.2.1 Requirements from the UCs

3.3.3 Quality Gateway : " Implement strict rule set to ensure semi-formal notation " will be addressed by this brick.

10.2.1.1 Use Case 3.3

10.2.2 How will this brick be integrated in the UC

The current top-level of the use-case which includes the quality check and translation both with the initial requirement list and the ones that come through the change management process, will be replaced over time by the new automated quality check.

The figure below shows in the red circle the current flow – this will be replaced by the data flow in the red square,

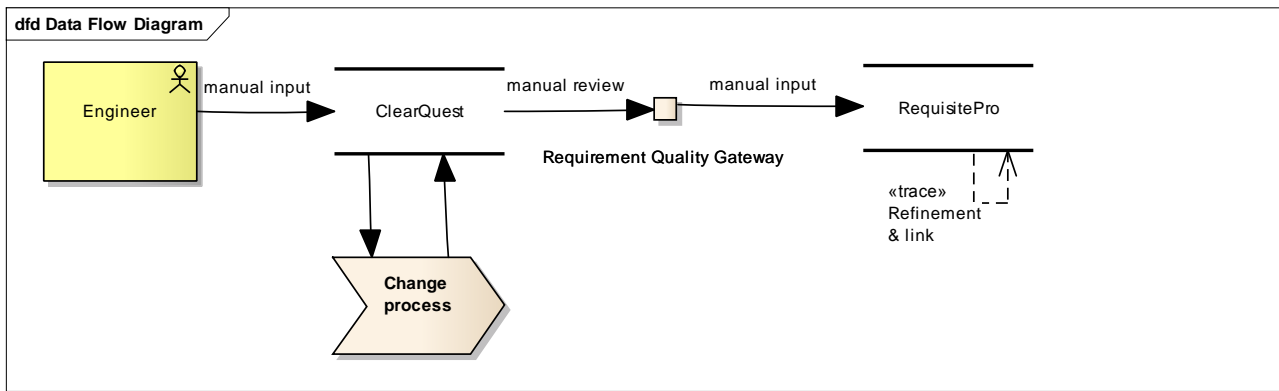


Figure 10-3: Manual Quality flow to be replaced

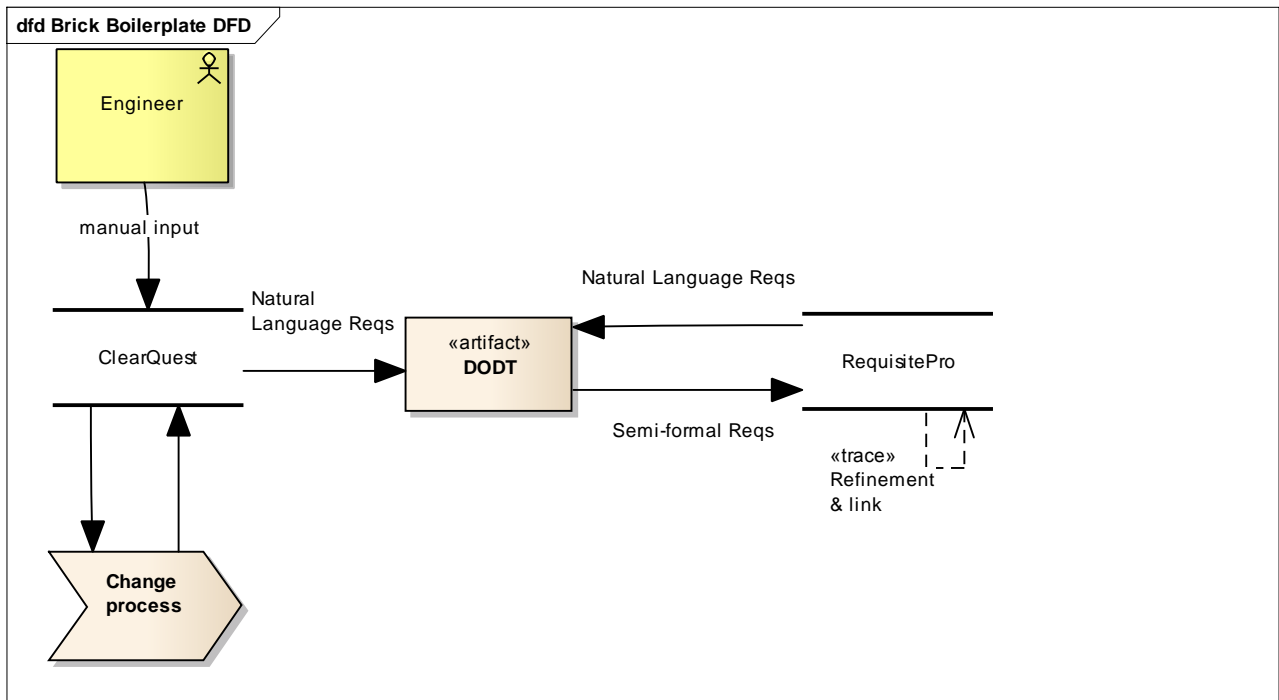


Figure 10-4: Replaced with automated Quality flow

10.2.3 Requirements fulfilled by initial tool/method version

Currently none

10.2.4 What will be implemented/provided in the CRYSTAL project

10.2.4.1 New and improved features

- An agreed set of semi-formal notations,
- A semi-automated check of requirement quality
- A translation of NL Requirements into the agreed semi-formal ones.

The semi-formal notations ensure that the requirements are at an atomic level and therefore can be reused within the flow.

10.2.4.2 Interoperability requirements

Currently the tool is standalone – however if the ontology/rule set differs between the variants then we may look at using CRYSTAL IOS to link the data from the KiD to the DODT or equivalent

10.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

10.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

11 DaD Data Analyser Dashboard

Provider: IFX-UK

Task #: T6.4.11

Brick #: B3.91a

Category: Verification management

(former brick name: Cross Domain Data Analyser)

11.1 Overview

DaD is an internally specified tool to allow for ease of managing the various data across the multiple projects and domains.

11.1.1 General Description

During implementation of the Requirements tracing flow it was identified that the communication of data between not only the domains but across the Hierarchy was problematic. It was decided that a dashboard that could analyse all of the related information necessary to 'view' the project as a whole would be of help here. DaD will analyse information from the different tools and documents released into the Configuration management tools and present it in a graphical user interface which can be configured on a per view basis.

11.1.2 Related Use cases

This is planned to be applied in UC3.3 Functional power train development.

11.2 Specification

11.2.1 Requirements from the UCs

11.2.1.1 Use Case 3.3

The requirements coverage figures will also be added to the tool to allow ease of visibility on the maturity of the project and the contact name for each set of results. All of this will allow for a greater communication between the projects, the verification/validation teams and the requirements engineering group.

11.2.2 How will this brick be integrated in the UC

The commercial tool asuresign from TVS currently allows for software and hardware test and regression management as well as linking requirements into tests and analysed hardware test results. It will be expanded to all domains such as software, firmware and validation to ensure reusability and also a single interface for all groups into the requirements tracing flow. The tool will have a single database instance for each module and per variant, this information will be gathered into a central area under configuration management and a tool will be added to view the information all centrally. This tool will be called DAD (Data Analyser Dashboard) and will assist the different domain managers in having visibility of which requirements are being tested by the other domains, this will assist with ensuring that the requirements are not over verified/tested by multiple groups and also that it is verified/tested at least once in the flow. It also will allow grading of the requirements to allow early IP release to the SoC team, IP is intellectual Property in the form of a module, SoC's are system-on-chips and are made up of multiple modules which are tested in standalone testbenches and then tied together by a system team. To do this we can define a grading system such as "Gold, silver, bronze" for example and assign a release maturity to a module. So if we define all of the tests which just test interfaces as a bronze or first release, then the SoC team will be aware that no other functionality has been tested and should they uncover a bug in an internal functionality they will not debug it as it might be currently under debug by the IP team. This will save duplication of effort and allow the safe staged release of IP's to speed up the SoC implementation and verification.

Asuresign is reliant on the passing of data and as such is reliant on data being under Configuration management and stable in its place of storage. It is at the bottom of the requirements trace flow linking through from the results and translating them into an ARQE.xml format which is readable by both asuresign and Reqtify, thus bringing the info into the requirements traceability tree.

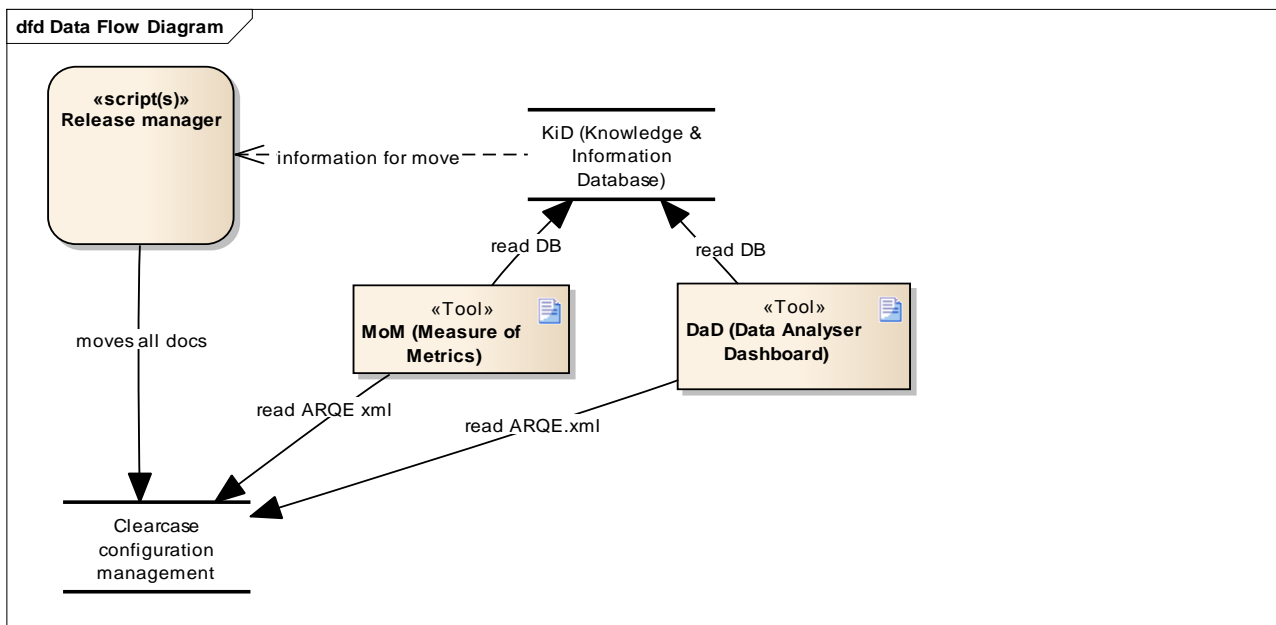


Figure 11-1: DaD within the Requirements Engineering flow

A proposed example for DAD is shown in Figure 11-2 below.

Domain		IP	SOC	IP Val	SOC Val	Test	SW	Requirements Traceability
Features/Hierarchy								
27x product	4 x ASCLIN	No	Yes	No	Yes	no	no	90% (overall)
			Pass		pass			
			bronze		Gold			
26x product	3 x ASCLIN	No	Yes	No	Yes	no	no	PRD – 100% ITS – 60%
			pass		pass			
			Gold		bronze			
24x product	1 x ASCLIN	Yes	Yes	Yes	Yes	no	no	Set of Pie charts? (SC have an example)
			pass	pass	fail			
			bronze	bronze	silver			
ASCLIN	Feature X	Yes	no	Yes	no	no	no	
			pass		pass			
			gold		bronze			
	Feature Y	No	No	No	No	No	No	

Figure 11-2: DaD information analysis

11.2.3 Requirements fulfilled by initial tool/method version

Improved communication across domains and better project management.

11.2.4 What will be implemented/provided in the CRYSTAL project

11.2.4.1 New and improved features

The Tool is new and internal to IFX, the tool will be completed and any improvement areas analysed and reworked into the tool over time within the project. As this is a completely new tool, it is likely to evolve over time as it gets used as other users may identify either problems or improvements that they wish to have fixed or introduced.

11.2.4.2 Interoperability requirements

This tool is a data analyser. It is dependent on getting the correct information from the correct area/tool. This information shall be held within the KiD database (a new internal database under construction but not part of CRYSTAL). So it will interface between the Configuration management storage in Clearcase and KiD. ARQE.xml and a database query may be used for reading the data, for the database info OSLC may also be investigated.

11.3 Implementation

Currently this is under construction, a full set of UML design diagrams has been created and is under review. A trial program was already implemented and is being analysed before being continued with tool implementation.

11.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

12 Recommended methodology according ISO 26262

Provider: ITKE

Task #: T6.4.12

Brick #: B3.04

Category: Safety Methodology

12.1 Overview

12.1.1 General Description

ISO 26262 is an established norm for the automotive industry which is not yet finished transforming respective production and development. The characteristic property of the norm is that it must be concretized before use. ITK Engineering is active in several projects of different manufacturers which are based on the ISO 26262. This peculiarity happens to make us familiar with the different concretizations. From this experience we see the need for a directive how to complete a transition from status quo to solutions based on the future Crystal RTP.

The purpose of this task/brick is to maintain applicability of CRYSTAL solutions to development environments based on ISO 26262. The definition of a cross-domain interoperability requires the division of the problem into typical technical patterns of information exchange and into a set of domain-specific symbol declarations (incl. convention over their meaning = Ontology) in order to control the technology. They are both part of the IOS but it is only the latter that is relevant to ISO 26262 applicability. The output of this brick can be threefold:

1. Derive Instructions for the automotive domain on the use of CRYSTAL RTP
2. Possibly extension of generic CRYSTAL ontologies for ISO 26262 purposes
3. Identification of remaining technological shortcomings and resulting dangers for automotive setups

The freedom to lay out the ISO 26262 has consequences for interoperability of software because software is relying on communication of objects with some known meaning. Automotive companies have three important degrees of freedom in ISO 26262 implementation which give rise to the risk of having to adapt implemented methodology before being able to use Crystal RTP. These freedoms are:

- 1) Freedom of Interpretation
- 2) Freedom of Selection
- 3) Freedom of Combination

Freedom number one relates to the OEM's right but also duty to specify what specific terms mean in the context of a given company. Ontogenesis of these terms involves the evolutionary definition of properties, establishment of taxonomical order with similar concepts and declaration of valid operations. From this refinement different kinds of process objects and artefacts can emerge which are embedded in an OEM-specific process landscape and ecosystem of tools.

Freedom number two relates to the OEM's right but also duty to select between several equivalent or at least similar approaches in order to achieve ISO 26262 defined goals. OEMs will choose between different tools, between different kinds of issue management, between different kinds of organizations of workers or between different kinds of innovation cycles. For example, a new development may very well be performed as a strategic project or as a market-triggered innovation cycle. Possibly one quality management system is based on SPICE and another on CMMI. Maybe the product has been inherited from pre-ISO times or it is a

Version	Nature	Date	Page
V1.0	P	2014-02-07	76 of 125

new product designed from scratch. Hence, manufacturers will find it difficult to implement methods and procedures in the same way as they have different resources, product complexities, production realities, legal frames and so on. Numerous technological tools are involved in this as well.

Freedom number three relates to the OEM's right but also duty to combine various steps in such a way that the output is relevant to the goal. Once the building blocks had been selected to conform to an ISO 26262 development environment, it is important to define how information will flow through it. And this is much less a freedom but more like a "take it or leave it". The large body of realities swamping in from "freedom #1" and "freedom #2" lead to a lock-in of tools designed to live in a specific niche. For example, the obstacles for switching from one set of software tools to another can be subtle but grave. What software *a* supports easily can be very difficult to achieve with software *b*. Sometimes this is just a detail like a missing API or unreliable exporter or a patent issue preventing the use of some software under local jurisdiction. Using software *c* with similar capabilities could be impossible because the terminology and concepts behind it do not exactly match domain specific conventions, etc.

The problem with the three freedoms is that they are impeding each other and CRYSTAL is visibly trying to better distribute the trade-offs to be made among them. As a rule of thumb, the Freedom of Interpretation has to be restrained a little in order to gain options during Selection and Combination. The most likely approach for achieving this will be to specify more generic terminology embedded in a more generic but modular process framework. This can fuse items which have not been considered the same or related until now. This again could open new possibilities for linking items together (this is probably desired but also a new source of error) and it could interrupt processes which will require inconsistent terminology (this is probably an improvement but at first it is a hassle).

A brief made-up example shall serve as an illustration of the problem: The idea of tests (term "test") can be either generic or special, depending whether several organizational units have to interact with each other or not. If they do not have to interact with each other then we observe the establishment of very specific understandings. In such specific cases the term "test" will mean either a single row in an Excel sheet or a single mechanical operation. Obviously, it is very difficult to establish interoperability at this level of detail. In more general cases the term "test" will describe a set of related objects and functions which are verified to satisfy certain qualities. The number and style of attributes used to describe the act of testing will most likely differ between companies, will often differ between departments and sometimes even differ from case to case. As long as the number of properties is small and their values are generic interoperability appears to be much more viable.

For example, if some ISO 26262 compliant organization tries to verify material properties then the term "test" will probably relate to visual inspections, to chemical reaction experiments or to procedures for inducing mechanical stress. Such tests can be destructive (specimen is lost) and may have a high cost which must be monitored and assigned to responsible staff. In contrast, for some software company a test will probably only relate to a piece of developed software and it will run automatically at no noticeable delay. From this difference, it might not appeal to require any reference to monetary properties but it might appeal to store paths to configuration data and to instructions for external test devices. A software company may call some kind of test "user satisfaction assessment" but it could be unaware that it can be explicitly expressed in a requirements-testing framework. Hence, a company producing materials for the automotive industry will most likely have a different understanding of what properties properly characterize tests and may prefer to use different terminology. One company will speak of "quality assessments" and the other will speak of "functional verifications". Without generalization these two terms will lead to the creation of two incompatible process landscapes.

Luckily, if stakeholders have to work together more frequently then concepts in use will approximate to the point where certain terminology condenses to be a domain specific jargon. In CRYSTAL the ontology tasks attempt to recapture what has become accepted terminology so far and to develop it further in order to allow deeper entanglement of objects from different domains. The creation of an overall ontology could help in this but it might entail the creation of an official taxonomy which is different from all existing ones. Such CRYSTAL taxonomy will reduce interpretational freedom at one hand but will increase *Freedom of Selection* and *Combination* at the other. The companies who develop embedded systems with such new tools will hopefully perceive the new practical freedoms so convincing that they will accept less domain- (and less company-) specific development practices.

Version	Nature	Date	Page
V1.0	P	2014-02-07	77 of 125

From the CRYSTAL project it is generally expected that it will create terminology at a greater level of abstraction in order to encompass concepts used in multiple domains. It cannot be guaranteed that this new terminology will nicely map to the participating companies' original terminology. Hence, the purpose of this task is to basically reverse the process, to map older terminology to new one and to describe how to handle the differences when using tool integrations based on the ARTEMIS RTP. Some realism is required in this. It is true that during the CRYSTAL project several concrete use-cases are played though and optimized using a new interoperability technology but the resulting configurations are probably nowhere close to allowing arbitrary alteration. The purpose of this task must be the formulation of effective approaches, revelation of real options and warning off pitfalls when employing ARTEMIS RTP integrations in an ISO 26262 context.

12.1.2 Related Use Cases

UC3.2, UC3.3

12.2 Specification

This is a methodological brick.

12.3 Implementation

Experience from the project will be condensed in a publicly available directive for IT architects.

12.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

13 Risk assessment and hazard analysis

Provider: ITKE

Task #: T6.4.13

Brick #: B3.05

Category: Safety Methodology

13.1 Overview

13.1.1 General Description

Risk assessment and hazard analysis is an important step in almost any norm governing the development and production of potentially harmful technical systems. In this process a variety of artefacts is generated which have certain relationships among them. An analysis will have a relationship with the analysed models, for example. In many cases these relationships are implicitly maintained by naming conventions for the files and by their placement in folders or by databases holding them. CRYSTAL IOS is attempting to extend the means of organization and interaction by providing rich explicit relationship networks and object abstractions which are driven by the developers needs and not by available features of tools. In this task it is important to identify which kinds of relationship and object categories are really required.

ITK Engineering and TNO have together performed a series of meetings discussing various terms related to risk and hazards analysis. The specific choice of a risk evaluation approaches gives rise to different kinds of procedures and/or additional analysis steps required to be performed for design changes. It also opens up or closes doors to specific classes of mathematical analysis tools. For example a system's risk could be defined as 1) immediate hazard proximity, 2) as the probability of loss or 3) a decision theoretically motivated access to an undesired state space.

1. The first concept is adequate for describing individual deterministic processes which end in an uncontrollable harmful situation. A hazard could then be understood as remote (there is no path to harm and it cannot be easily established: rock is far off the cliff), as dormant (there is no path to harm but it could be easily established: rock is at the brink of cliff), as potential (there is a path to a harm but it is under control: rock is close to falling off the cliff but its direction is controlled with bumpers and target area was cleared) or as active (there is a path to a harm and it is not under control: rock is falling off the cliff). A "path" in these categories is an uninterrupted sequence of events stemming from the system's dynamics in an individual scenario.
2. The second concept is adequate for modeling hazards which are activated at predictable probabilistic rates but which cannot be predicted at individual demands. Such hazards typically arise from physical component failures after long stress. The third is a little bit unfamiliar but makes more sense in context of software where individual situation classifications and failure probabilities do not seem to be a perfectly reasonable approach because software is not analyzed at instance level (instead we analyze the algorithm) and it is neither operated randomly nor does it operate randomly. The problem with software is that a system can be dangerous despite the software being fault-free and the underlying hardware being reliable. The question then is how an operator can decide whether his action policy will unlock a system hazard or not. The problem with bringing in an operator is that he acts intentionally and that his policies are not best modeled with probabilistic distributions (with some effort this is possible). This approach will try to answer how many bits of information must be provided by the operator system to the operated system despite the information gain communicated from the operated system to the operator in order to unlock a specific hazard in

the worst case. The number of bits communicated in both directions is posing the length of the hazard key. The longer it is the safer the system because there is a strong separation between similar operator policies and the action is absolutely intentional.

Spontaneously, it may not appear plausible to everybody why the third option is different from the first one. Therefore an example shall briefly illustrate the difference: Let's assume that we are considering the danger of deleting a desired file on disk. Let's further assume that before deletion the software will ask the user a series of questions and will warn him about his operation. For example, at first the software could ask the user if he really wants to delete the selected file. Before the user can answer "yes" the software reminds him of the file's content with a small preview of the file. If the file is somehow linked to other project files then it could ask him again if it should really delete this file and show the user how these files are linked together. If the file happens to be protected by the system the user will have to enter the administrator password before proceeding and in the end the software will ask him if he really, really wants to delete this system protected file. We finally arrive at a situation where the user will see this dialog with "yes" and "no" on it. In this situation there is this potential hazard to delete a desired file but it is not reasonable to say that the software poses a large hazard in terms of undesired deletions. Despite the system being potentially able to perform undesired deletions it is not potentially dangerous.

Such ideas can be explored for their usefulness in CRYSTAL use-cases.

For example, UC 3.2 finds itself in a situation where ISO 26262 norm conform risk classifications do not work for the use-case. An adoption of ASIL* classification has become necessary which is custom to the project. Characteristic for the product in the use-case is the controlled environment where probabilistic failures from hardware wear-out are being negligible and where probabilities for vehicle situations and exposure rates are artificially designed on a case by case basis and not real-life estimates on a fleet basis as it is assumed by the norm. A new hazard classification based on decision theoretic qualities could yield a more holistic guideline for the overall system design which is compartmentalized at the moment (department processes, on-board steering unit, series control units and software for trajectory generation are analyzed mostly separately).

UC 4.2 is also facing hazards which are difficult to quantify from field data. In this case the severity and the likelihood of specific events could be based on simulations or access models (3). This specific use-case is also interested in smart field data collection and evaluation which is difficult if systems have no own knowledge and fully rely on operator feedback. If systems can monitor their operation with rich knowledge of assumptions made during development then patterns of operation which violate these assumptions could be more easily detected.

From the perspective of the tool-chain it is very important to be able to easily identify all solutions designed to prevent certain harmful events (traceability). The challenge in doing this is to support a large variety of tools (tools for software development, tools for designing E/E platforms, tools for mechanical design and tools for procedures and documentation design.)

13.1.2 Related Use Cases

UC3.2, UC4.2

13.2 Specification

This is a methodological brick.

13.3 Implementation

Concerned use-cases implement IOS-based solutions. Implementation of software for hazard estimation based on information units, operator models and system descriptions are thinkable.

Version	Nature	Date	Page
V1.0	P	2014-02-07	80 of 125



Experience from the project will be condensed in a publicly available directive for safety analysts.

13.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

14 FTA, FMEA and FMEDA (ITKE)

Provider: ITKE

Task #: T6.4.14

Brick #: B3.06

Category: Safety Methodology

14.1 Overview

14.1.1 General Description

Fault Tree Analysis, Failure Mode Effect Analysis and the more hardware oriented Failure Mode Effects and Diagnosis Analysis is a popular technique for analysing systems for their potential to generate failures associated with harm. These techniques are part of an early analysis and help to decide among various design options. This variety will grow if we think of a company with a broad product portfolio. The problem of tracking the results and changes becomes pressing when systems are similar. In such cases humans can mistakenly refer to wrong but similar documents. A scenario of this kind can be found in Use-Case 4.2.

Sometimes such analysis is done in a semi-technical way. It must be of high concern to introduce such analysis as well-managed objects to the development environment because only then robust linking to development artefacts can be achieved.

This task must also deal with practical aspects of technique execution. For example, FTAs can be a very powerful method for detecting weak points in a design. The means to achieve this is to compute minimum cutting sets (cross-sections). However, inadequate modelling can lead to false results. ITK Engineering will help to detect or to define best-practice for techniques found to be relevant to use-cases. Living up to best-practices is preliminary to further technological improvements like the introduction of an IOS-networked tool-chain. For example, if a naming strategy for tree objects poses an obstacle to efficient computation of cutting sets then an improved technological representation is not automatically going to make the computation of those sets more efficient or correct.

There are several ways how CRYSTAL could yield improvements for engineering of critical embedded systems. If use-case-providers decide to update their tool chains in order to include best in class tools for faults and failure analysis, which allows for stably referentiable objects then producing IOS interoperability will lead to development of respective adapters.

ITK Engineering and TNO have investigated for options to establish modern analysis techniques going beyond FTA, FMEA and FMEDA. Such techniques, like bow-tie/event-tree, LOPAs or GOMS-based operator interaction analysis, but also variants like the FMECA, RBDs and various statistical analysis methods can be beneficial cross-domain and should be considered as part of this task.

Especially important are dynamical aspects of the analysis where diagnostics and active counter-measures could lead to undesired dangers (recurrent analysis). Such analysis could be realized by conventional analysis models if they could relate to each other. Establishment of such relationships could be achieved with IOS-links.

The methods could also greatly gain strength if they could be invalidated by new data arriving at a safety-database like it is envisioned for Use-Case 4.2.

14.1.2 Related Use Cases

UC4.2

Version	Nature	Date	Page
V1.0	P	2014-02-07	82 of 125

14.2 Specification

14.2.1 Requirements from the UC

14.2.1.1 Use-Case 4.2

Use-Case 4.2 is aimed at improving the efficiency of the product risk management of interventional X-ray systems (iXR) (cf. Figure 14-1). This medical device consists of a C-bow to allow instruments moving in a multidimensional space to be aimed at a patient and controlled by an operator, a table to move and position a patient in several directions and of a set of monitors and imaging facilities.



Figure 14-1. Example of an X-ray system developed in use-case 4.2

Product Risk Management (PRM) is a continuous process throughout the lifetime of a product addressing all risk management activities related to the health, safety, privacy and security of people. This includes product design, manufacturing, distribution, installation, service (maintenance, repair), de-installation, surveillance and where necessary timely corrective actions. Two phases are distinguished:

- pre market: activities during design and release of the product (project execution)
- post market: activities after release of the product.

Three case studies are identified as relevant areas for improving product risk management processes:

1. Analyzing risk profile related to an adverse event;
2. Impact analysis of design changes;
3. Comparing actual risk profile to residual risk profile (trending).

Part of the PRM is the development of separate FMEA's for reliability and safety once a new variant of an iXR is designed and operational field data (like customer feedback, e.g. "complaints") gives rise to evaluate assumptions underlying the FMEA. FTA-methods are not used by Philips in this process.

Aim is to detect intolerable risks that require technical or organizational countermeasures in order to reduce them to a tolerable level. In this context it is very important to draw a line between initial risks (without countermeasures) and residual risk (with countermeasures). Experience feed-back on the latter leads to review of the design base and the related FMEA as well as change proposals. Key of the analysis is to assess whether tolerable risk limits are exceeded.

Important problem was the difficulty to track results in Excel-sheets and related databases and to represent them in meaningful information for diverse users in the PRM as well as to have a systematic overview in a system to manage risk data of variants the system efficiently and prevent repetition. Several engineering methods are used for this and they need to be combined.

An overview of the interrelations between the parts of the current *safety risk management process* is depicted in the figure below. In the next section, each part in this figure is described in detail.

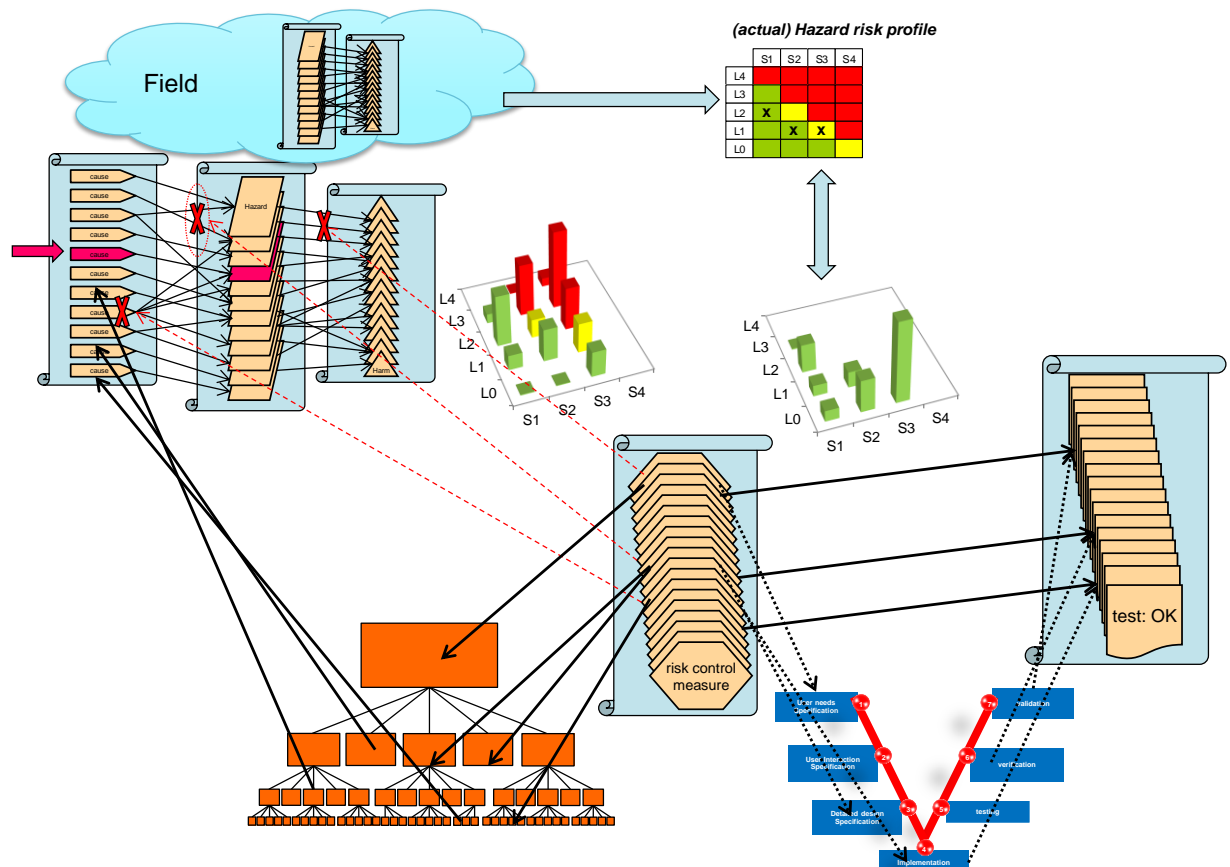
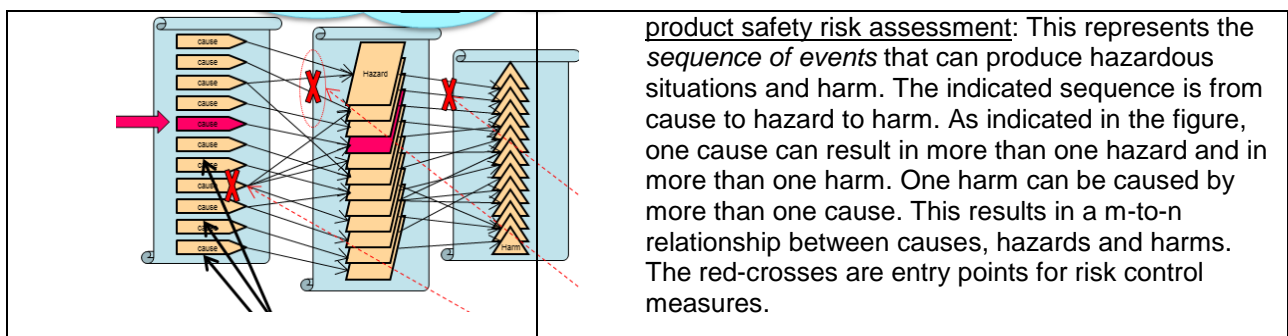
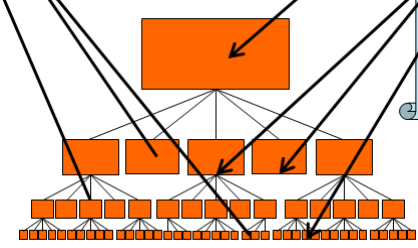
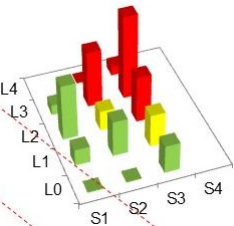
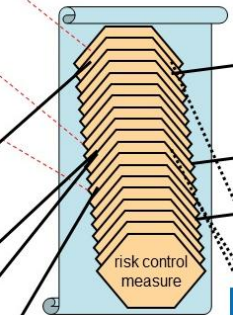
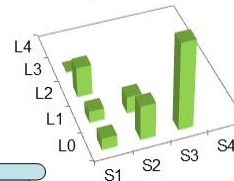
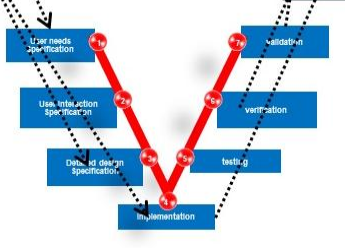
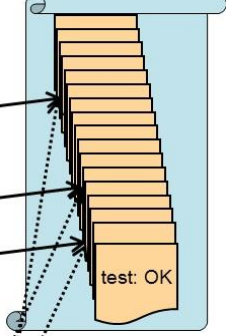
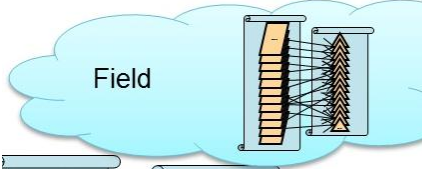



Figure 14-2: Overview of interrelations between parts of the safety risk management process.

In figure 14-2, the following parts can be distinguished:



	<p><u>system design</u>: The system is built up from hardware and software components and units. The corresponding design choices directly affect the possible causes for hazards and harms. The diagram represents the hierarchical build-up of the system design.</p>
	<p><u>initial risk profile</u>: based upon the <i>severity</i> of harm and <i>likelihood</i> of occurrence of the hazards, a risk profile of the complete product can be compiled. Sequences of events resulting in harms with high severity (e.g. S4) and high likelihood (e.g. L4) are unacceptable.</p>
	<p><u>risk control measure</u>: Within the <i>risk management process</i> risk control measures are defined and implemented to reduce the risk(s) to an acceptable level. As indicated with the connecting lines, risk control measures are preferable defined as safety concepts and specified in the top level of the system design. Other risk control measures are defined and implemented on unit level.</p>
	<p><u>residual risk profile</u>: This is the risk profile after implementing the risk control measures. The risk analysis process is repeated until sufficient risk control measures have been defined and implemented to reduce the risks to an acceptable level.</p>
	<p><u>development process</u>: The risk control measures are realized via the development process. Note that some measures have impact at the overall system requirements and design level and some only at the low-level detailed design level. For each <i>risk control measure</i>, test and verification results are collected at the corresponding design levels.</p>
	<p><u>test evidence</u>: For all risk control measures, test and verification evidence is collected from the development process.</p>

	<p><u>post market analysis</u>: customer complaints and service work orders are analysed with respect to occurrence of hazardous situations and adverse events. When needed additional risk control measures are defined and implemented.</p>
<p>(actual) Hazard risk profile</p> 	<p><u>actual risk profile</u>: using the data from the post market analysis, the actual product risk profile is compiled. This profile is compared to the estimated residual risk profile.</p>

Improvements sought are amongst others:

1. Supporting the PRM by giving easy access to data from several user perspectives including developers and enable transposing of risk data in new profiles
2. Structuring the description of events to enable to link them to hazards situation data and to clarify and manage interrelationships of data (e.g. interdependencies of causes, hazards, harm, and control measures)
3. To link these data with simulations, relevant test designs and safety cases.
4. Aligning hazardous situations in pre-market analysis with those experience in post market phase
5. To reconcile pre and post market risk profiles
6. To identify trends, automated reasoning mechanisms to connect safety cases, explore tools that enable risk models to become more adaptive.

The need expressed in this use case for cross domain tooling may go beyond traditional methods and needs exploration of new methods to support PRM at Philips. They may vary from innovative approaches like self-reporting machines on the one site and renewal of the present analysis techniques by (e.g. Isograph) suites including methods like fault tree, event tree or markov analysis on the other site. But also support in the work flow system of rich interlinked items which are under a firm work-flow control can be thought of.

Currently this is under construction, a full set of UML design diagrams has been written and is under review. A trial program was already implemented and is being analysed before being continued with tool implementation.

14.3 Implementation

Experience from the project will be condensed in a publicly available directive for safety analysts.

14.4 Evaluation

[This section is otherwise empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

15 Feature documentation in model based software

Provider: ITKE

Task #: T6.4.16

Brick #: B3.10

Category: Safety Methodology

15.1 Overview

During every development step the set of requirements is refined and transformed into design features which must be documented for various purposes. A great variety of documentation documents is typical for modern products. The variety of documents results from the variety of intentions behind them. There are installation guides, manuals, safety instructions for the end-user, documentation for other developers, mounting procedures, disposal remarks and maybe legally mandatory documents, like e.g. specific parameter sheets.

From our experience we observe a trend to rationalize this process. Many departments pursue the goal to derive documentation from technical requirement descriptions. However, there are philosophical obstacles to this approach:

- The first and foremost problem is that requirements describe a desired state and documentation is describing the actually achieved state. Despite all the efforts to bring the achieved state and the desired state together a remaining difference is often observed. The correct capturing of this difference can be crucial in terms of safety or perceived product quality.
- The second important problem speaking against this strategy is that automatic documentation compilers will not alter the content based on the intention behind the document. Such alterations can be: change of language, change of style, reduction according to importance, reordering according to priority or modification/insertion/deletion of diagrams and pictures. This process is a little bit of an art which requires deep understanding of the reader's reception of the document in a given situation.
- The third reason opposing an all too optimistic compilation of requirements into documentation is the way information is assigned to documents. Requirements can often express themselves in different ways and become relevant to different documentation activities. It is very difficult to know this in advance and to properly characterize requirements by their relevance to specific documents. For example, some products change their area of application and will require documentation which was not expected up front.

These three important reasons have the consequence that production of documentation is still a mostly human task. There are several important requirements from the point of view of the human documentation engineer which if satisfied will greatly help him to produce high quality documentation:

1. Access to relevant technical artefacts (requirements, architecture diagrams, technical notes, test protocols, etc.) should be easy. The probability to refer to a wrong artefact should be small.
2. Documentation is not affected by every change of design but sometimes it is. Nevertheless, the documentation engineer should be able to track changes to the product and to adapt the produced documentation as early as possible.
3. The tool-chain should support the production of a complete set of documentation artefacts. The necessary documentation could be part of the requirements.
4. The tool-chain should allow for a nice workflow between original developers and documentation engineers. These developers are sometimes on different projects and do not have immediate idea what the documenting colleague is meaning. There should be an easy way to create a basket of artefacts for a thread of discussion.

5. Documents produced for documentation should be standardized and instructions for creating them should be available.

Especially the OSLC based approach is adequate to create and track relationships among documents. From experience it is considered already as very assuring if the final documentation for a product release is collected automatically from such configurations.

Even if a fully automatic documentation is yet impractical some documentation could be updated or generated semi-automatically. For example it should be easy to update numerical values in documentation from requirements if they are exactly those which shall be documented (otherwise the values have to be measured or derived from other sources like e.g. tests or parameter databases). The scripts or programs responsible for such a merge should be easily exchangeable between documentation engineers. Therefore they should be somehow attached as meta-data to the documentation involved.

The production of documentation is also one of the final quality assurance steps in a project. There should be ways to trigger issues on the official change management platform if problems are detected.

15.2 Specification

This is a methodological brick.

15.3 Implementation

Use-Cases implement such ideas according to their internal requirements.

Experience collected during CRYSTAL should be condensed into a directive for documentation engineers involved in projects based on Crystal RTP.

15.4 Evaluation

[This section is otherwise empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

16 Isograph FaultTree+

Provider: ITKE

Task #: T6.4.7

Brick #: B3.55

Category: Safety Analysis Tools

16.1 Overview

The Isograph offers a product called Reliability Workbench which includes fault tree analysis (FTA). The Reliability Workbench also supports various safety analysis techniques. The most up-to-date Isograph Workbench delivers FTA, FMEA, RBD, Statistical Tests and various designers for management of the process of finding the best technical approach to confront the spectrum of failures and their possible expression. One such designer helps to design the reliability allocation because there is not a single solution to face a potential failure. The Reliability Workbench is coming with catalogues of safety relevant objects according to ISO 26262 and IEC 61508. ITK Engineering advises and accompanies CRYSTAL-partners in their respective use-cases.

16.2 Specification

16.2.1 General Description

The minimal requirement for this brick will be the exposition of analysis results as an OSLC resource which can be then referenced by other tools to manage safety requirements actualization. Use-Case 3.2 plans to introduce a specialized IOS-based tool to navigate project graphs. This tool has the purpose to make the most specific propositions for work given a selected set of project objects. It can be imagined that unfinished but required FTAs appear as a workflow item in this tool. However, this tool is not representing the workflow itself because it is implemented in PTC Integrity.

16.2.1.1 Related Use Cases

This was planned for UC4.2 Safety layer of interventional X-ray system.

16.3 Implementation

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

16.4 Evaluation

[This section is otherwise empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

Version	Nature	Date	Page
V1.0	P	2014-02-07	89 of 125

17 SAFETY ARCHITECT (ALL4TEC)

Provider: All4Tec

Task #: T6.4.12

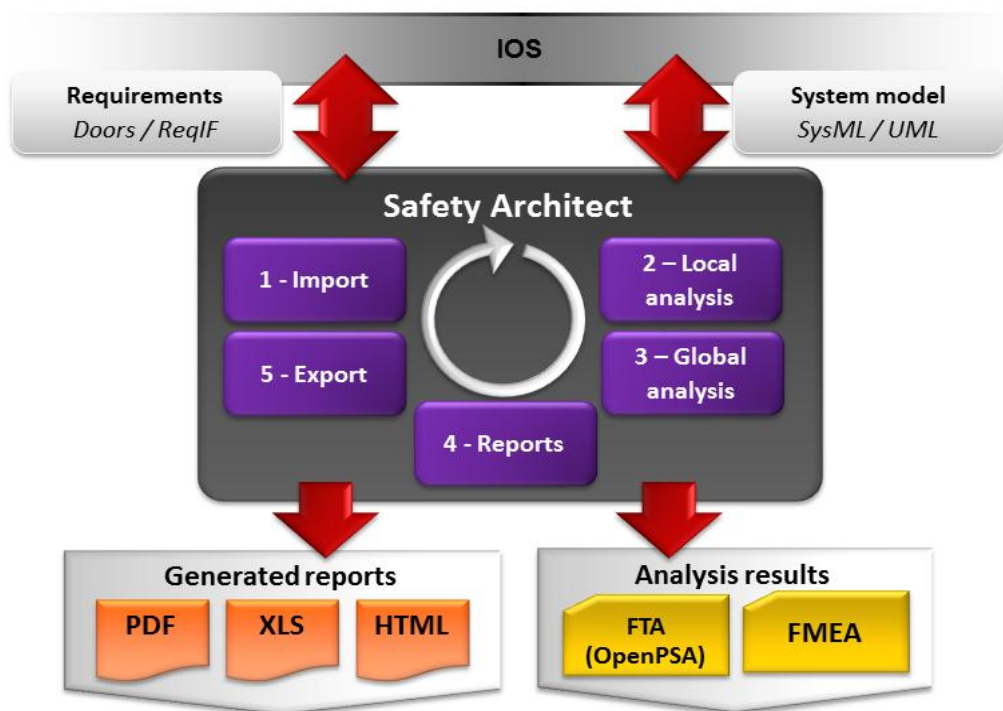
Brick #: B3.04

Category: Safety Analysis Tools

17.1 Overview

17.1.1 General Description

The aim of this brick is to support local FMEA (Failure Mode and Effects Analysis) on the model elementary components and to automatically generate Fault Trees. Using a system functional design or its physical architecture model, the user can perform a local analysis inside Safety Architect, by linking failure modes of the outputs of the block to the failure modes identified on the block inputs. In parallel, the user can also implement safety barriers, participating to the safety objectives compliance (cut the critical path). The user then defines what failure modes are the feared events (FE). These events will be studied by the analysis. Then a propagation is run, which consists in spreading in the system all the identified failure modes, and to trace those that reach a feared event. The results of this propagation are formulated through Fault Trees. In case of modification of the system or software model, Safety Architect is able to perform an impact analysis that reduces the rework costs that can be very high for a FMEA.



17.1.2 Related Use cases

The brick development is applied in the use case UC5.3 in order to support the safety analysis at different steps of the system design.

17.2 Specification

For the first iteration, and before the definition of IOS standards, the new functionalities that will be added to Safety Architect are the following:

- Possibility to handle new attributes of failure modes that are mandatory in the Alstom format,
- Automatic generation of the global analysis results in the Alstom format.

17.2.1 Requirements from the UCs

17.2.1.1 Use Case 5.3

[req-UC53-01]

The tool shall be able to import the functional and architectural specifications of a system or sub-system. In the use case, these specifications are formalized with SysML.

[req-UC53-02]

The tool shall be able to import existing safety (and functional) requirements defined in a Doors database or SysML model.

[req-UC53-03]

The tool shall be able to export newly defined safety requirement toward Doors or SysML.

[req-UC53-04]

The tool shall provide to safety engineers a safety specific viewpoint of the functions and components of the system, allowing them to consult, edit and validate the dysfunctional specification of these objects. By dysfunctional specification it is meant: failure modes, causes, local equations, input/output characterization...

[req-UC53-05]

During the import phase, the hierarchical description of functions and components shall be preserved by the tool and explicitly showed in the dysfunctional viewpoint.

[req-UC53-06]

The tool shall be able to detect gaps between the current system model and the newly imported one. These gaps shall be identified visually.

[req-UC53-07]

Models produced by the tool shall be "versionable" with mainstream versioning tools and support diff and to some extent merge.

[req-UC53-08]

The tool shall assist the safety engineer during the preliminary hazard analysis (PHA), by providing means to describe accident scenarios and to associate tolerable hazard rate (THR)

[req-UC53-09]

The tool shall be able to capitalize accident description, context description, barriers and scenarios description in libraries that can be reused for other analyses.

[req-UC53-10]

The tool shall produce a PHA description within a tabular view (e.g. excel sheet), one scenario per line.

[req-UC53-11]

The tool shall initiate the hazard log table that traces feared event, barriers, and requirements down to verification means (evidences) and status. At this stage only feared event, barriers and requirements are identified.

[req-UC53-12]

The tool shall assist the safety engineer during the system Hazard analysis (SHA), by providing means to describe the dysfunctional specification of every function of the system.

[req-UC53-13]

The tools shall produce FMEA and fault trees from the dysfunctional specification.

[req-UC53-14]

FMEA should be formalized within an excel sheet, fault tree should be either in the OpenPSA or Aralia format.

[req-UC53-15]

During the SHA phase, the tool shall provide support to SE for SIL definition and allocation given a global THR objective.

[req-UC53-16]

The tool shall ensure traceability between failure of functions and potential accident at system level.

[req-UC53-17]

During the SHA phase, the hazard log shall be updated with failures and associated safety requirements

[req-UC53-18]

The tool shall simulate the dysfunctional behaviour of the system, providing means to study failure propagation.

[req-UC53-19]

For the SSHA phase the tool shall verify the same requirements as the ones defined for SHA id: [req-UC53-12], [req-UC53-13], [req-UC53-14], [req-UC53-15], [req-UC53-17], [req-UC53-18]. Except for [req-UC53-16] that is redefined.

[req-UC53-20]

The tool shall ensure traceability between failure of functions at subsystem level and failure of functions at system level.

17.2.2 How will this brick be integrated in the UC

The system modelling process used is the Alstom Transport's ASAP process which is implemented with UML/SysML. ASAP is an advanced use case driven method that addresses requirement management, operational analysis, functional analysis and constructional analysis. Each phase of analysis is subjected to system safety analysis that will be supported by the Safety Architect brick.

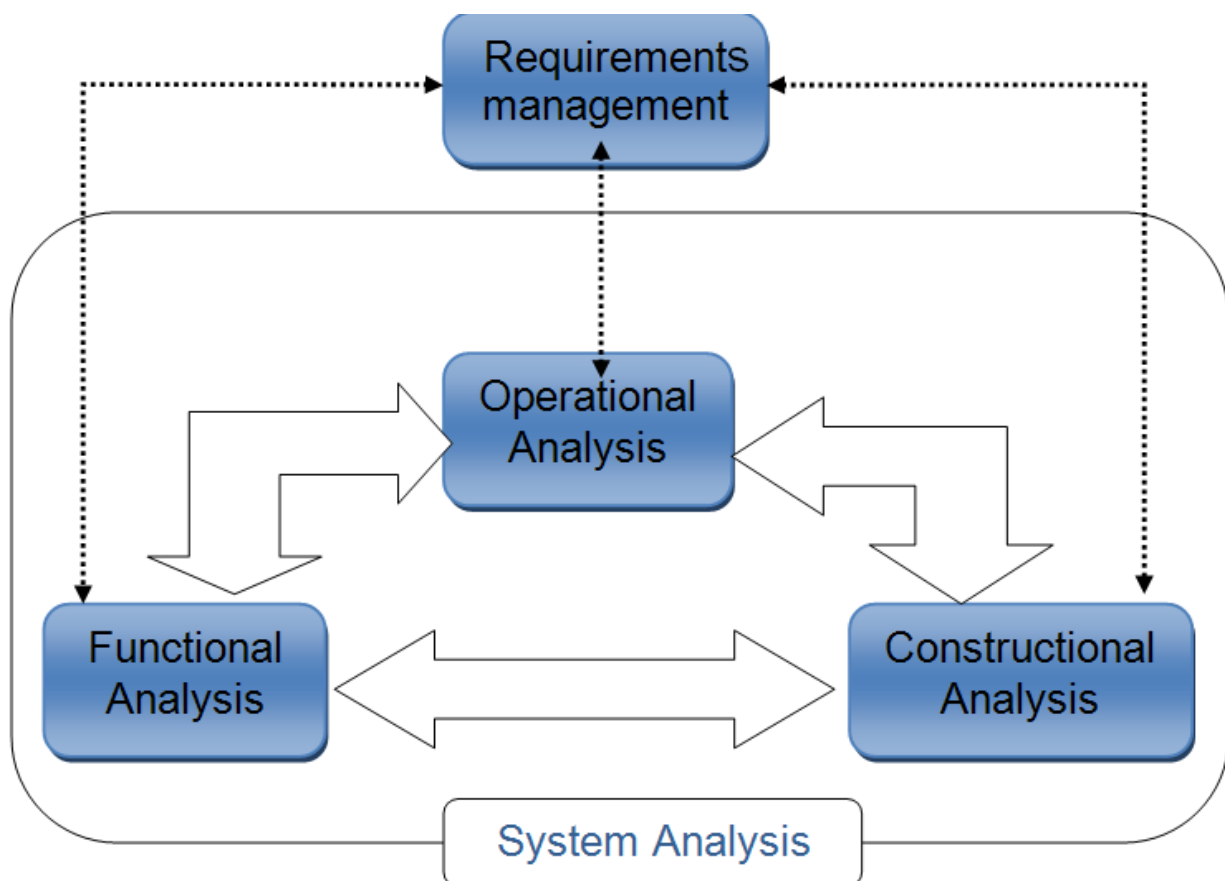


Figure 17-1 Relations between the requirements management process and design modelling process

17.2.3 Global process

The next paragraphs show the main processes needed for the global analysis. The activities are shown in terms of their order of execution and the conditions under which they are executed. Requirements engineering process is described in the section 17.2.3.1, system analysis process is described in section 17.2.3.2, safety analysis process is described in section 17.2.3.3, hazard log is then described in section 17.2.3.4.

17.2.3.1 Requirements analysis process

Figure 17-2 shows the requirements analysis process.

17.2.3.1.1 Inputs

Requirements analysis requires project description.

17.2.3.1.2 Description

Requirements activities are described as follows:

- Elicit requirements: The aim of this activity is to identify, gather and define the sources of elements that are used as a basis for the requirements.
- Identify context definition: The purpose of this activity is to identify and define the stakeholders and systems elements that are used as a basis for the context definition view.
- Analyse requirement: The aim of this activity is to understand the requirements of the system by looking at the use case for them.
- Define acceptance criteria: The aim of this activity is to consider how each use case is validated.
- Establish traceability: The aim of this activity is to check whether traceability between all the views has been defined.
- Document process: The aim of this activity is to produce requirements document.

17.2.3.1.3 Outputs

At the end of requirements analysis, new requirements are stored in DOORS.

17.2.3.2 System analysis process

The next stage of analysis focuses on system analysis process. System analysis encompasses operational analysis, functional analysis and constructional analysis. At each stage of analysis, we establish traceability to ensure that traceability between all the views has been defined.

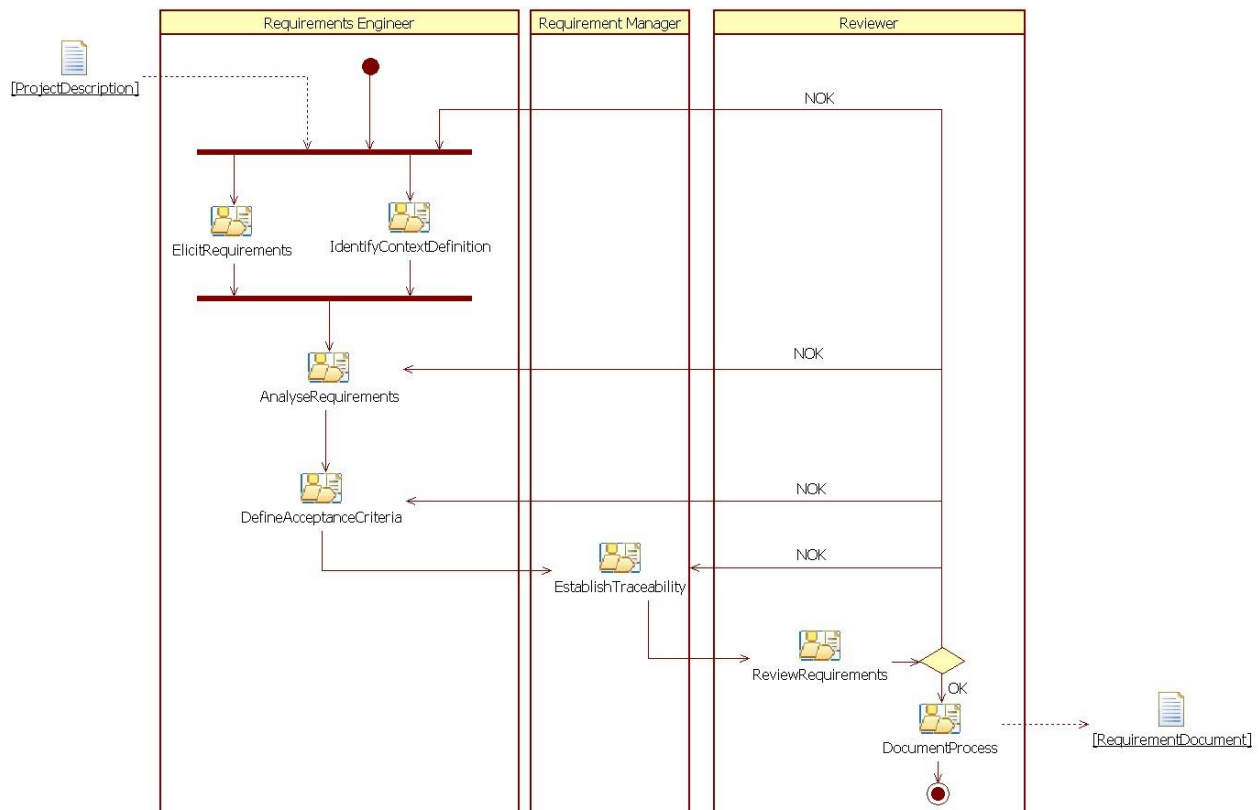


Figure 17-2 Requirements analysis process

17.2.3.3 Safety analysis process

The safety analysis process includes preliminary analysis, system hazard analysis and sub system hazard analysis. At the end of each process, we identify safety requirements. Safety analysis is supported by the Safety Architect tool.

System Engineering Method	Input	Description	Output
Operational Analysis	System requirements	Define the system environment. Understand actors intended uses. Develop operational scenarios. Develop the operational data model.	Internal block definition diagram, State machine diagram, Sequence diagram, Block Definition diagram
Function Analysis	System requirements	Identify the system functional dependencies with its environment. Define the system functional	Internal block definition diagram, Activity diagram, Block Definition diagram

System Engineering Method	Input	Description	Output
		break down structure. Define the system operation: How the system functions are executed. Develop the function data model	
Constructional Analysis	System requirements	Identify the system connection with its environment. Define the system breakdown structure. Define the system constructional behaviour. Develop the function data model.	Internal block definition diagram, Activity diagram, Block Definition diagram

Table 17-1 System analysis process

Safety Engineering Method	Input	Description	Output
Preliminary hazard analysis	Safety Plan, the system requirements and the Hazard Breakdown Structure	Identify protections necessary to eliminate or mitigate identified risks.	safety requirements at the system level
System hazard analysis	Safety Plan, the Preliminary Hazard Analysis, the Hazard breakdown structure, the System Functional Specification, the System Operational and Support Hazard Analysis.	Identify all failures leading to potential hazards through a Failure Mode and Effects Analysis (FMEA). Determine and assign the SIL of system functions. Identify barriers and safety requirements against hazardous situations. Identify the necessary sub-system hazard analysis, specific hazard analysis and interface hazard analysis and record this information in the SHL. Record identified hazards in the SHL.	new safety requirements

Safety Engineering Method	Input	Description	Output
Sub system hazard analysis	Safety Plan, System Hazard analysis, System Interface Hazard analysis and Sub-systems requirements specification.	Identify all failures leading to potential hazards through a Failure Mode and Effects Analysis (FMEA). Determine and assign the SIL of sub-systems functions. Identify barriers and safety requirements against hazardous situations.	safety requirements at sub system level

Table 17-2 Safety analysis process

17.2.3.4 Hazard Log

Hazard log	Input	Description	Output
	Hazard breakdown structure System Preliminary Hazard Analysis System Interface Hazard Analysis System Requirements Specification System and sub-system Requirements Tests Plans System and sub-system Requirements Tests Descriptions System and sub-system Integration Tests Descriptions System and sub-system Requirements Tests Reports Operational and Support Hazard Analysis System Hazard Analysis Sub-system Hazard analyses Specific Safety Studies Fault Tree Analysis Products and Software exported constraints	Record for each identified hazard the following attributes: An identification number, A complete description, Its consequences, Its estimated frequency, The components it involves, The protections, The associated actions, Its status (open, resolved, closed), The related safety requirements, Record people involved in safety related activities with their skills; Record methods, techniques and tools used for analysis; Record hypothesis used for analysis; Record known limits of analysis; Record level of confidence on used data for analysis.	System Hazard Log Sub-System Hazard Log

Hazard log	Input	Description	Output
		Verify the coverage of safety requirements by tests cases	

Table 17-3 Hazard log

17.2.4 Requirements fulfilled by initial tool/method version

The following requirements are directly addressed in the initial version of Safety Architect without additional work:

req-UC53-01 ; req-UC53-05 ; req-UC53-07 ; req-UC53-13 ; req-UC53-16 ; req-UC53-18.

The following requirements are partially addressed in the initial version of Safety Architect:

req-UC53-03 ; req-UC53-04 ; req-UC53-06; req-UC53-14.

17.2.5 What will be implemented/provided in the CRYSTAL project

System analysis is supported by Papyrus tool. Papyrus is a Modelling Tool that provides an implementation of the OMG standards (UML, SysML, Marte). We will provide the facility to import system or software models (SysML, UML) in Safety Architect and initial functionalities that cover the following requirements:

req-UC53-01 ; req-UC53-05 ; req-UC53-07 ; req-UC53-13 ; req-UC53-16 ; req-UC53-18.

In the first iteration, we will implement the possibility to support Alstom and to generate fault trees and FMECA at the SHA and SSHA levels of safety analysis.

Thus the first iteration will cover the following requirements:

req-UC53-04 : req-UC53-12 ; req-UC53-13 ; req-UC53-14.

The other requirements will be implemented in the next iterations.

17.2.5.1 New and improved features

Possibility to export the SHA and SSHA results in a “FMECA type” format which is mandatory for Alstom.

17.2.5.2 Interoperability requirements

[REQ-UC53-IO5-01]

Requirements [req-UC53-01], [req-UC53-02], [req-UC53-03], [req-UC53-14] define interoperability challenges and shall be verified in the RTP.

[REQ-UC53-IO5-02]

The Hazard Log gathers elements coming from different teams: requirement, hazards, functions, components and verification means (test case...). This artefact shall be interpretable between multiple tools and kept coherent regarding the different baseline of the system.

[REQ-UC53-IO5-03]

Artefacts defined within safety and system models shall be versioned and managed coherently with configuration management tools. Traceability links shall not be lost from one configuration to another.

17.2.5.3 New features specifications

As seen in the previous chapters (cf. 17.2.4, 17.2.5.1), Safety Architect addresses partially some requirements and must also integrate new features to support the Alstom process. The aim of this chapter is to specify what will be done in Safety Architect to be able to:

- Manage safety and functional requirements, and import them from an external tool, like Doors
- Generate analysis results corresponding to the Alstom needs (i.e.: SHA)

17.2.5.3.1 Requirements management

An important need in the Model Based Safety Analysis approach is to assure the requirements traceability. This implies different kinds of requirements: those identified in the upstream phases like the functional requirements, and those written during the safety analysis. These requirements are a key data used to exchange between the system and safety teams.

To include Safety Architect in a global process, the requirements management is mandatory. As each user may have already defined its requirements in its own format, the solution is to manage the requirements in a standard interchange format: [ReqIF](#). This format is an opened standard and it can be used to exchange requirements data base with Safety Architect and several tools, like Doors. Moreover Safety Architect will use the OSLC framework to communicate with external tools and share those requirements.

17.2.5.3.2 Requirements edition

To manage the requirements in Safety Architect, a new data library will be added to the projects. This library will be an independent XML file, respecting the ReqIF format. Each new project will be created with a default empty requirements library.

Safety Architect will be able to display and edit the requirements library content, using the ReqIF concepts. To ease the user work, the editor will include two points of view: a standard user presentation and an expert one. Indeed, ReqIF includes a lot of advanced concepts which are not useful for a standard use.

Moreover, this editor will use the [RME](#) project as base, to capitalise on this open-source project, already included in others Eclipse tools.

17.2.5.3.3 Requirements import / export

The first step of the safety analysis in Safety Architect is to model his system, or to directly import it from an external tool. To manage the requirements associated to an existing model, all the import wizards will provide in their advanced options the possibility to import a ReqIF file. Like this, the project created in Safety Architect will directly include the requirements found in the imported file.

Version	Nature	Date	Page
V1.0	P	2014-02-07	99 of 125

A new type of import will also be integrated, to import a ReqIF file in an existing project. This will merge the requirements existing in the project with the imported file content. This functionality can be used to import a new version of the requirements data base, to share standard requirements between several projects, etc.

Finally, the requirements library can be exported to an external XML file. Thus, any external tool which is compatible with ReqIF can import this file, which may include the safety requirements defined in Safety Architect during the safety analysis.

Safety Architect is planned to be able to interact with SysML/UML modellers with OSLC references, but not a full synchronization at the moment.

17.2.5.3.4 Requirements association

The requirement management added value is to assure their traceability. To assure the traceability during the system modelling and the safety analysis, the requirements may be linked to any element of the model. In Safety Architect, any object will may be linked to any requirement, this will be done by adding a 'requirements' property to all the objects manipulated.

Finally, once the requirements are associated to the different elements, the user can display a coverage report to see how any requirement is covered.

17.2.5.3.5 SHA generation, in Alstom format

Safety Architect is used to automate the global analysis of a system. The safety engineer can then concentrate is work on the added value tasks like the local analysis. Currently, the global analysis results are only displayed in the form of a critical paths report, or a failure tree. This is not enough to address the whole process of Alstom.

Indeed, the expected results are a custom representation of the SHA (System hazard analysis) format. The SHA considers the system as a whole and identifies how the systems, subsystem and operators interface and interact, and how the components fail.

17.2.5.3.5.1 SHA format

The SHA format used by Alstom includes some specific properties. To be able to generate a result which corresponds to this need, and which can also be used and understood by any other user, these are the property retained:

Property	Safety Architect implementation	Comment
ID	None	Generated automatically with the SHA
System function	Name of the Container	
Function	Name of the Bloc	
Failure mode	Name of the failure mode	
RRF	Risk Reduction Factor Property of the feared event	The value is selected amongst a limited enumeration

Property	Safety Architect implementation	Comment
Mode	Property of the failure mode	A global mode is created as a property of the Model. By default, this global mode is affected to all the failure modes, but it can be changed. The value is selected amongst a limited list, defined by the user in the Model
Option	None	Free text field, edited by the user after the SHA generation
Cause	Property of the failure mode.	The value is entered by the user, as a free text
Effect	None	Free text field, edited by the user after the SHA generation
Potential accident	Feared event associated to the failure mode	
ID Safety Req	ID of the safety requirement associated	
Safety Req	Detail of the safety requirement	
ID Functional Req	ID of the functional requirement associated	
Functional Req	Detail of the functional requirement	
Element involved	None	Free text field, edited by the user after the SHA generation
S	Property of the feared event	The value is selected amongst a limited enumeration
R	Property of the feared event	The value is selected amongst a limited enumeration
TAR	Property of the feared event	The value is selected amongst a limited enumeration
SIL	None	Automatically computed using RFF, S, R and TAR values
Comment	None	Free text field, edited by the user after the SHA generation

Table 17-4 SHA format in Safety Architect

17.2.5.3.6 SHA generation and export

The SHA generation is not automatically launched on each global analysis. Indeed, the basic result is still the fault tree. Once the analysis results are satisfying, the user can choose to generate the SHA arrays from the results files.

The SHA built is directly displayed in a specific view of Safety Architect, which allow the user to show/hide the columns and rows, edit some cells, etc. Finally, once the user has completely built its SHA, he can save it, and export it to an Excel file.

17.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

17.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

18 MU Safety Analysis Tool (MUSAT)

Provider: MU

Task #: T6.4.19

Brick #: B2.52

Category: Safety Analysis automation and verification

18.1 Overview

18.1.1 General Description

MU Safety Analysis Tool (MUSAT) will be built on top of DIVINE model checker. The DIVINE model checker is a tool for model-based development and system verification that covers multiple phases of system engineering, including requirement analysis and verification, verification of functional and non-functional requirements, safety verification and reliability assessment. It allows use of Linear Temporal Logic (LTL) for specification of temporal properties of systems under verification.

For this purpose, the extended and integrated version of the model checker will include several tools/components: a component for translation of MATLAB Simulink designs into CESMI (and CESMI extended with probabilistic choice), a component to build Markov Decision Process representation from a CESMI-based specification, and probabilistic and discrete, non-probabilistic LTL model checking core.

More in detail, MUSAT will support the following activities:

Requirements validation: checking the quality of a set of requirements using Linear Time Logic (LTL) as a formal specification language, using model checking techniques to discover errors such as inconsistencies, logical conflicts or missing requirements.

Non-functional requirement checking and reliability assessment: to check the compliance of a system model with respect to a set of probabilistic safety properties using explicit-state model checking for finite-state probabilistic systems.

Functional property checking: verify that a Simulink design complies with a temporal behavior specification, compiled from requirements for the given component.

18.1.2 Related Use cases

MUSAT will be integrated and evaluated within Use Case 2.6 - Multi-Mode Navigation System internally by Honeywell (CRYSTAL partner no. 69).

18.2 Specification

18.2.1 Requirements from the UCs

18.2.1.1 Use Case 2.6

At the current stage of the development of the UC 2.6, the requirements on MUSAT are as follows.

- The ability of checking coherence and sanity of a set of temporal LTL properties.
- Perform vacuity checking and vacuity witness generation for a given LTL formula.
- Qualitative and Quantitative model checking of selected probabilistic Simulink designs.

18.2.2 How will this brick be integrated in the UC

The tool will be integrated internally by Honeywell. The details of integration are not yet fully defined and therefore will be part of the next deliverable.

18.2.3 Requirements fulfilled by initial tool/method version

The DIVINE model checker already provides components for functional verification, safety checking and design verification. As for coherence and sanity checking of LTL formulae DIVINE is, at the moment, capable of checking satisfiability of a given LTL formula. All these capabilities will be directly exported by MUSAT or will be internally called by MUSAT as needed.

18.2.4 What will be implemented/provided in the CRYSTAL project

18.2.4.1 New and improved features

Relevant existing and newly implemented functionalities will be adapted to the proper use within the use case.

The following new features are planned:

- The DIVINE tool will be extended to handle probabilistic choice in its input models. This comprises handling of probabilistic MATLAB Simulink designs. On such inputs, the tool will implement probabilistic verification based on Markov Decision Processes (MDPs).
- An interface for exporting state spaces in the form of explicit-coded MDPs from DIVINE will be provided. This will allow integration with third-party probabilistic verification tools such as PRISM.

18.2.4.2 Interoperability requirements

DIVINE's existing interfaces will be extended with provisions for encoding probabilistic choice, based on the requirements coming from the UCs. Moreover, the backend probabilistic interface for integration with external probabilistic verification engines will be specified, in such a way as to allow interoperability with existing tools.

Version	Nature	Date	Page
V1.0	P	2014-02-07	104 of 125

18.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

18.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

19 Safety-analysis for Aerospace (ESA Standards)

Provider: GMV

Task #: T6.4.20

Brick #: B2.53

Category: Safety Methodology

The “Safety-analysis for Aerospace” brick evaluates the industrial applicability of safety-analysis frameworks in the scope of space systems. In particular, the output artefacts produced by the partners involved in the Aerospace Use Case will be assessed with respect to the dependability and safety requirements extracted from ESA standards (ECSS-Q-ST-30C and ECSS-Q-ST-40C) in order to prepare such artefacts for use in the Use Case.

19.1 Overview

This section summarises the dependability and safety process in the Aerospace Domain according to the applicable ESA standards. Currently, most of these activities (e.g., FMEA) are manually performed by the Safety Engineers. Therefore, the goal is to be able to automate this process and conduct some activities using one or more of the tools proposed in CRYSTAL. To achieve it, it will be necessary to evaluate the artefacts produced by each tool and determine their compliance with the ECSS standards.

19.1.1 General Description

Nowadays, the complexity and functionality of space systems is increasing more and more. Safety Critical Systems are those in which any misbehaviour could lead to an accident where the environment could be damaged or human life endangered. In these cases, the systems have to guarantee strong safety and dependability constraints.

In the Aerospace Domain, a dependability and safety process has to be conducted and properly assessed. In particular, in view of the growing complexity of the software used in space critical applications together with increasing cost and schedule constraints.

Systems are built from lower level subsystems, so that the system-level safety and dependability analysis needs inputs from lower-level subsystems. Systematic software and hardware failures are taken into account during the requirements, design, coding and testing processes, while requirements baseline errors are covered by the system safety and dependability analyses. The identification of suitable methods and the adoption of appropriate techniques are needed in order to ensure that the system behaviour is meant to be such that the system behaves according to its dependability and safety requirements.

These requirements are specified in the technical specification and considered in the subsequent design, implementation and verification/validation phases. The dependability and safety process allow the Verification and Validation (V&V) Manager to verify the implementation of the requirements in order to mitigate risks.

The following section details how the dependability and safety process is addressed in the ECSS standards at system and software level.

19.1.1.1 ECSS Standards

ECSS stands for “European Cooperation for Space Standardisation” and represents a cooperative effort of the European Space Agency (ESA), national space agencies and European industry associations for the development of a coherent, single set of consistent space standards for use by the entire European Space Community.

The result of this effort is the ECSS series of Standards (ST), Handbooks (HB) and Technical Memoranda (TM) organized in four branches:

- M: Management.
- Q: Product Assurance.
- E: Engineering.
- U: Sustainability.

Figure 19-1 depicts the ECSS structure highlighting the dependability and safety branches.

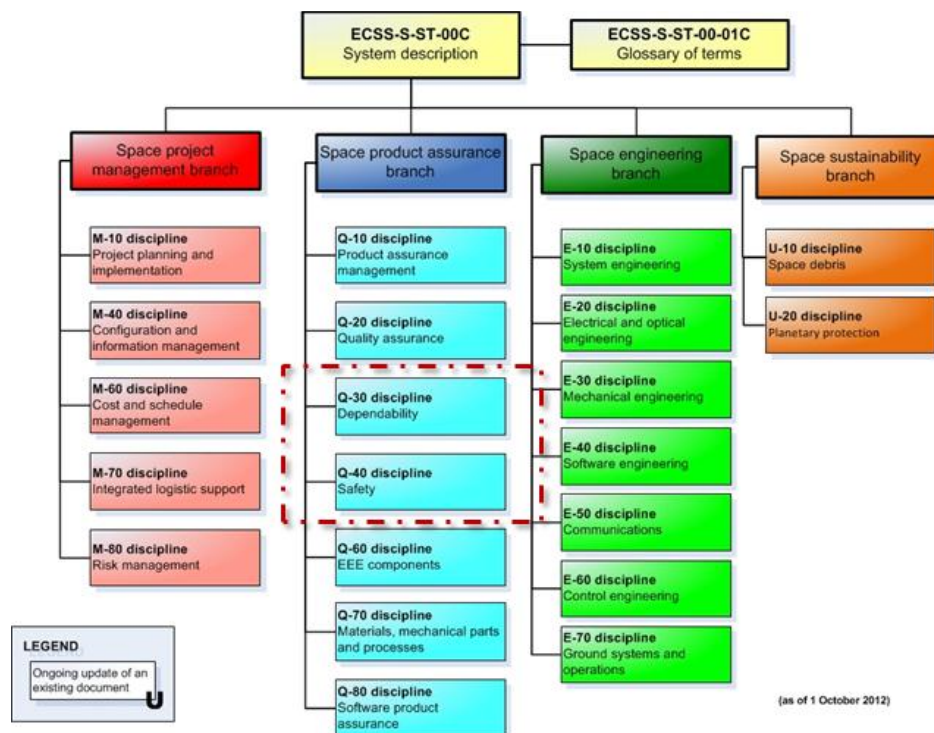


Figure 19-1: ECSS standards hierarchy

From the dependability and safety point of view, the following ECSS standards are applicable:

- **ECSS-Q-ST-30C** defines the requirements for a dependability assurance programme in space projects. This standard calls for the use of dependability analysis techniques, tailored to match the generic requirements in each project, to address the hardware, software and human functions composing the system.
- **ECSS-Q-ST-40C** defines the safety programme and the technical safety requirements for space projects.
- **ECSS-E-ST-40C** defines the principles and requirements applicable to space software engineering. In version C assets the need of specifying software RAMS (Reliability, Availability, Maintainability and Safety) requirements based on the System RAMS analysis result.

- **ECSS-Q-ST-80C** presents the software product assurance requirements to be met in a particular space project to provide confidence to the customer and to the suppliers. Namely, ECSS-Q-ST-80C presents:
 - Requirements to ensure that the software is developed to perform as expected and safely in the operational environment, meeting the quality objectives agreed for the project.
 - Requirements concerning “Software dependability and safety analysis” (subclause 6.2.2). These requirements (through the referred requirements of ECSS-Q-ST-30C and ECSS-Q-ST-40C) refer to the supplier carrying out a software dependability and safety analysis to assign criticality levels to software components, based on the criticality levels of the functions and the identification of safety functions. In addition, subclause 6.2.2 of ECSS-Q-ST-80C mentions that the software dependability and safety analysis is performed at every development milestone. It also expects that the list of software critical components is verified, reviewed and reduced and designed to facilitate dependability and safety analysis and software testing.
 - Requirements concerning “Handling of critical software” (subclause 6.2.3), regarding measures and activities to ensure dependability and safety of critical software components, the verification of the use of those measures, what to do regarding dead code and about non-critical code potentially affecting the critical code.

19.1.1.2 Dependability and Safety Process

Dependability and safety process is an iterative and continuous process that provides dependability and safety design guides.

The process has to be conducted along the whole project lifecycle at every development phase. Primary, the analysis focuses on the system behaviour (i.e., functionality) based on the objectives and requirements already defined. Subsequently, it focuses on concrete design criteria and coding rules. Additionally, it is important to consider that an entire project (system) consists of different parts (subsystems) which can be divided into elements composed of different components. Sometimes, different subsystems or elements are developed by different companies or contractors. In these cases, the complexity of the analysis increases. The supplier (upper-level) already provides a set of dependability and safety requirements, then they are scoped and tailored at subsystem level, and the analysis is performed again at subsystem level and additional requirements may be defined.

Two different approaches can be followed:

- *Top-Down approach*: in the top-down approach, dependability and Safety Engineering starts with analyses based principally on dependability and safety objectives and past experiences. On the basis of the system-feared events, an event tree is constructed (e.g. using a Fault Tree Analysis) to identify the worst-case event/failure at the boundaries between system, subsystem and equipment.
- *Bottom-Up approach*: in the bottom-up approach, Dependability and Safety Engineering starts with detailed analyses undertaken on product (e.g. using a Failure Mode Effect and Criticality Analysis). All failures modes are assessed for risk potential and event consequences are followed up to the next level of integration up to system level. Risks reduction actions are taken at the best suitable level.

Dependability and safety process is tailored (e.g., techniques applied) according to the software category. This category will be used to identify the suitable engineering and product assurance measures aiming at reducing the risks associated to the software criticality.

Version	Nature	Date	Page
V1.0	P	2014-02-07	108 of 125

Criticality Category	Definition
A	Software that if not executed or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: <ul style="list-style-type: none"> ▪ CATASTROPHIC consequences (safety or dependability).
B	Software that if not executed or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: <ul style="list-style-type: none"> ▪ CRITICAL consequences (safety or dependability).
C	Software that if not executed or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: <ul style="list-style-type: none"> ▪ MAJOR consequences (dependability).
D	Software that if not executed or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: <ul style="list-style-type: none"> ▪ MINOR or NEGLIGIBLE consequences (dependability).

Table 19-1: Software criticality categories

Systematic failures are taken into account during the requirements, design, coding and testing processes, while the requirements baseline errors are covered by the system safety and dependability analyses.

The severity categories utilised to perform the analysis are defined in Table 19-1 and can be a result of software that if not executed or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure. The severity of each failure is determined according to its consequences.

Severity	Level	Dependability [ECSS-Q-ST-30]	Safety [ECSS-Q-ST-40]
Catastrophic	1	Failures propagation	Loss of life, life-threatening or permanently disabling injury or occupational illness.
			Loss of system.
			Loss of an interfacing manned flight system.
			Loss of launch site facilities.
			Severe detrimental environmental effects.
Critical	2	Loss of mission	Temporarily disabling but not life-threatening injury, or temporary occupational illness.
			Major damage to interfacing flight system.
			Major damage to ground facilities.
			Major damage to public or private property.
Major	3	Major mission degradation	---
Minor or Negligible	4	Minor mission degradation or any other effect	---

Table 19-2: Criticality classification (extracted from ECSS-Q-ST-40C)

The severity categories must be assigned without consideration of existing compensating provisions (the worst case). The number identifying the severity category shall be followed by a suffix in the following cases:

1. The suffix 'R' shall be used to indicate redundancy.
2. The suffix 'S' shall be used to indicate safety in catastrophic and critical severities.

The criticality assigned to each component is the highest one of its associated failure modes.

In order to identify and evaluate failures, different methods and techniques (see section [19.1.1.3]) are applied and their results must be analysed. They could lead to (i) new requirements that reduce the risks identified; (ii) design constraints; and (iii) coding rules.

Version	Nature	Date	Page
V1.0	P	2014-02-07	109 of 125

19.1.1.3 Methods and Techniques

The main task behind the dependability and safety analysis is to identify which parts of the software are critical and which are not or at least with minor critical level. At the same time, mechanisms to mitigate and/or remove the effects of failures have to be proposed. To achieve it, the Space Domain uses well-established methods and techniques.

The techniques to be applied in each project depend on the criticality determined by the risks faced and the potential failure consequences. The objective of the whole analysis is to guarantee that the design, implementation and V&V processes are appropriate to ensure that any risk caused by failures is acceptable for the system.

To check the system behaviour in all possible situations and discover all potential failures is not an easy task. The definition of a dependability and safety strategy tries to ensure that the analysis performed has been done adequately and in a consistent manner.

The safety critical process covers the next analyses at various levels:

- **RAMS** (Reliability, Availability, Maintainability and Safety) process using FMECA (Failure Modes, Effects and Criticality Analysis), FTA (Fault Tree Analysis), HA (Hazard Analysis), etc.
- **Safety Case**. It is a documented body of evidences that provides a convincing and valid argument that a system is adequately safe for a given application or a given environment. The safety analysis presents the control and risk reduction measures, the safety critical elements, safety risk assessment status, etc. The evidences of the argumentations are based mainly in RAMS analysis. This includes: FMECA, FTA, etc.

The coordination among dependability and safety activities is essential from the very beginning of the project. Analyses are basically applicable to both fields and need to be performed in close synchronization.

The initial hazard analysis performed by the Safety Team, and the failure modes analysis accomplished by the Dependability Team, provides the safety information required to perform the initial safety risk assessment of the identified hazards. Without identified hazards and failure modes very little can be accomplished to improve the overall safety of the system. Identified hazards and failure modes become the basis for the identification of recommendations leading to the implementation of additional safety requirements.

Here are some possible techniques to conduct dependability and safety analyses but many other techniques can be used:

- **PHI** (Preliminary Hazard Identification). Identification of all possible situations that exposes people or environment to potential harm.
- **HA** (Hazard Analysis). It specially focuses on safety aspects. From a detailed identification of hazards and their associated accidents tries to eliminate and mitigate hazards that may affect to the system and environment.
- **HAZOP** (HAZard and OPerability). It is a qualitative process based on guide-words used to identify potential hazardous variations from design intent in components and in interactions between system components. In addition, operational problems are identified.
- **FMEA** (Failure Modes and Effects Analysis) represents a qualitative analysis method to identify failures and to investigate potential effects for every single function or component of a system. FMEA takes as input the requirements, the functional analysis and standards applicable. Then, for every component or functions identifies possible failure modes. Once failure modes are identified, failure causes are determined, and also the component and system effects. Finally, results are recorded in a table.
- **FMECA** (Failure Modes, Effects and Criticality Analysis) includes FMEA but also some extends. For each failure mode, FMECA also determines the probability of failure and the criticality level. So, different PA and QA requirements are assigned to that function or component under analysis.
- **FTA** (Fault Tree Analysis). From feared events, its potential causes are identified. The goal is to specify mitigation barriers that inhibit the occurrence of the top-level feared event.
- **ETA** (Event Tree Analysis) is used to determine the likelihood of potential consequences after the hazard has been realised. This technique starts from a hazard and obtains all possible subsequent events that could lead to specific consequences.

- **CCA** (Common Cause Analysis) identifies dependencies in a design and assures independence when it is needed (e.g. independence of failures of multiple systems). So, corrective measures for potential failures in multiple systems or multiple subsystems can be determined. CCA facilitates the identification of single causes which may lead to multiple failures.
- **HSIA** (Hardware-Software Interaction Analysis) is a method to analyse that software has been specified and designed to react to hardware failures and to ensure that it cannot overstress hardware.
- **PSSA** (Preliminary System Safety Assessment). Its purpose is to assist in validating the proposed system architecture and to allocate safety requirements to components of that architecture or even to identify derived safety requirements.
- **SSA** (System Safety Assessment) is used to confirm and provide evidences and arguments that safety requirements have been addressed. So that, the complete system will satisfy safety objectives. It demonstrates that all risks have been eliminated or minimised in order to be acceptable, and monitors the safety performance of the system in service.
- **Safety Case** is the final activity in the development process and uses SSA as input. It demonstrates that the entire safety requirements have been addressed and demonstrates that the system, in its operational environment, will not compromise agreed safety levels.

All these techniques are not used in the same project phase and they are the basis of hazards identification and the assessment of probabilities and severities. The outputs provided by each technique can be used as an output to refine the dependability and safety process.

All Dependability and Safety recommendations resulting from the various analyses performed are tracked, compiled and maintained. The recommendations represent mechanisms to mitigate the effects of the different system failure modes. All these techniques are complementary.

Tests specifications may also use the hazards analysis to prepare the test strategy for dependability and safety derived requirements. In turn, verification results will help to close the identified hazards and to support the safety related argumentation. This safety argumentation is built based on all project related evidences.

From safety point of view, risk reduction is formally documented in the hazard analysis and reviewed. Safety requirements are verified by testing as preferred verification method and analysis when testing is not sensible. The tests include the demonstration of all operational modes.

19.1.2 Related Use cases

This brick will be used in the Aerospace Use Case (“*WP205 – CRYSTAL space toolset applied to Avionics Control Unit Software generation, test, V&V and Certification*”). One or more tools will be proposed to be used in the use case based on the results extracted from this brick. Therefore, firstly an analysis has to be carried out to ensure that the artefacts produced will serve as safety artefacts for the qualification process.

19.2 Specification

19.2.1 Requirements from the UCs

19.2.1.1 UC2.5 – CRYSTAL space toolset applied to Avionics Control Unit Software generation, test, V&V and Certification

The following process has to be performed for Use Case 2.5:

1. Functional analysis

The functional analysis is a common basic task necessary to perform subsequent Dependability and Safety activities. Its purpose is to identify software critical functions. This analysis must also include the interfaces with other subsystems and with the underlying hardware. Then, the software context (the interaction of the software with its environment) is completely defined (e.g. system hardware and external commandability).

This task is primary based on the Use Case functional description. Later, it will be refined according to the Software Requirements Specification. Finally, when the software architecture is available, functions previously identified are mapped to software components.

2. Analysis of failure modes

The potential failure modes associated to each software function are identified.

Not only generic failure modes have to be included (e.g. incorrect function execution, non-execution, real-time constraints), but also those specific to the Use Case SW: interrupts, etc.

In addition their effects and corresponding recovery/mitigation actions are described and analyzed.

Software requirements document is used to identify software failure modes, whereas Software design document provides information about the causes of these failures. The effects require also information at system level to determine how a failure can affect other subsystems.

3. Criticality assessment

A criticality category is assigned to each software component based on the effects of the associated failure modes. The criticality of the software component corresponds to the highest severity of the potential failure modes of that component.

Compensation and recovery actions are extracted from SFMEA analysis. They are evaluated to decide their implementation or, if the final decision is for no actuation, a documented rationale has to be added.

The set of critical software components is listed but it shall be verified and reviewed at each software cycle review.

4. Verification of the implementation of compensation provisions

Recommendations/compensation provisions to the overall software life cycle are provided in order to fulfill the required measures and assure the required reliability.

The implementation of approved compensation provisions must be checked. A document containing the traceability matrix that traces compensation provisions to those requirements and software components that implement them has to be produced.

The SFMEA information must be provided according to the following procedure:

1. SFMEA analysis of each SW component: effects and observable symptoms.
2. Assignment of a severity category.
3. Identification of possible recommendations or compensation mechanisms.

The following SFMEA template is proposed:

Version	Nature	Date	Page
V1.0	P	2014-02-07	112 of 125

Column	Description
Item/ID	Sequential number identifying the item being analysed
SW Component(s)	Software component(s) where the functionality is performed
Function(s)	Statement identifying the functionality performed by the item
Failure Mode	Identification of the assumed failure mode of the item under consideration
Effect on sub-assembly	Short description of the consequences of the assumed failure on the functionality of the sub-assembly (local effects)
Effect on equipment	Short description of the consequences of the assumed failure on the functionality of the equipment (end effects)
Observable symptoms / Detection Principle	Provides the observability of the failure or its consequences
Compensation Provisions	Mechanisms to reduce or avoid the effects of a failure mode
Recovery Actions	Indicates potential means to recover the function or an acceptable degraded consequences
Criticality	Number categorizing the criticality of the failure effect
Remarks	Remarks about the failure mode

Table 19-3: SFMEA template

19.2.2 How will this brick be integrated in the UC

Dependability and Safety Activities are performed during the entire SW life cycle as described in the following figure:

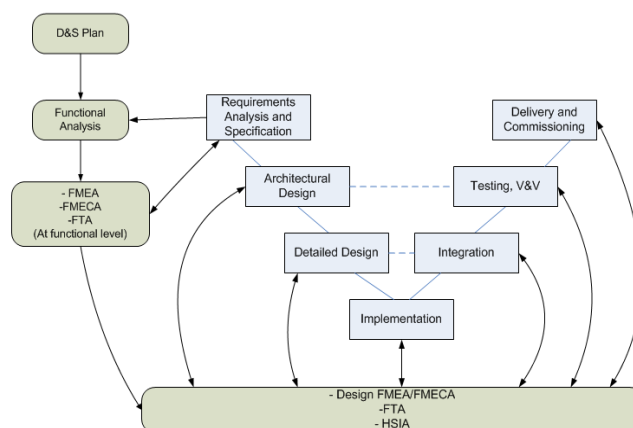


Figure: Integration of the brick in the use case

19.2.3 Requirements fulfilled by initial tool/method version

N/A

19.2.4 What will be implemented/provided in the CRYSTAL project

This brick will provide the requirements that CRYSTAL tools have to fulfil in order to apply them in the Aerospace Domain, and to be able use the output artefacts as part of the qualification process.

19.2.4.1 New and improved features

N/A

19.2.4.2 Interoperability requirements

N/A

19.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

19.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]

20 Autonomous Fault Tolerant System Design Methodology (AFTS DM)

Provider: Tecnalía

Task #: T6.4.21

Brick #: B2.54

Category: Safety Methodology

20.1 Overview

The increase of transistor integration density in conjunction with operation voltage reduction and operation frequency increase has resulted in more sensitive devices. In this regard, radiation is one of the major sources of faults primarily in aerospace domain applications where environment conditions are specially harsh.

In the case of SRAM FPGAs, the combination of traditional fault tolerance techniques such as Triple Modular Redundancy (TMR) (hardware triplication), temporal redundancy (time-delayed signal process repetition) and Scrubbing (device reprogramming) with modern design techniques based on Partial Reconfiguration (PR) is required in order to create a robust design, that is, an AFTS. Thus, PR arises as the key technology to leverage SRAM FPGA AFTS design.

AFTS-DM consists of a design methodology aimed at augmenting current reconfigurable device fault tolerance level, which is of crucial importance in FPGA-based critical system design. Since PR technology entails a number of design challenges a secure design methodology is therefore required. AFTS-DM is intended to combine recent advances in PR state-of-the-art to provide a reliable design methodology to implement PR technology as a solution for AFTS creation in those critical FPGA-based systems that require high security levels.

Depending on several factors such as target device, system architecture and environment conditions the AFTS DM will be adapted to the specific use case requirements (UC 2.5 leaded by TASE) in order to provide a reliable solution and an acceptable fault tolerance level.

AFTS DM will be a brand new development under the present project hinging on Tecnalía's expertise in the field of FPGA-based System on Chip Design and Partial Reconfiguration techniques.

20.1.1 General Description

Field-Programmable Gate Arrays (FPGA) were invented in the mid-eighties by Xilinx, currently the major FPGA vendor. FPGAs were conceived as a chip packed with transistors organized in regular-shaped logic blocks that could be configured and reconfigured by software tools. Earlier FPGAs were intended to be used as simple glue logic devices in larger systems. Thanks to the exponential increase in transistor integration density, modern FPGAs are now able to entirely implement complete digital systems, coining the so-called System on Programmable Chips (SoPCs).

The main advantage offered by the FPGAs is the ability to be reconfigured in the field. This feature results in a remarkable increase in design flexibility, which leads to the development of FPGA-based designs as the preferred option for prototyping before implementing the system in an Application Specific Integrated Circuit (ASIC).

Presently, a growing number of designs incorporate FPGAs in the final product. A step further in FPGA reconfigurability is the dynamic/run-time Partial Reconfiguration (PR) introduced by Xilinx in the early 2000s for its high performance Virtex family. Altera, the second major FPGA vendor, introduced its first partially reconfigurable devices a decade later. PR is the capability to reconfigure only a portion of the FPGA fabric

Version	Nature	Date	Page
V1.0	P	2014-02-07	115 of 125

while the remaining logic continues to operate without interruption. Thus, the inherent flexibility of the FPGAs is extended further, since the modules that compose the SoPC can be time-multiplexed. Therefore, a more efficient use of the silicon is possible.

Nevertheless, in order to implement a PR system, a strict design methodology must be followed. Such a methodology entails a steep learning curve which, in addition to the inherent complexity of the design methodology itself, results in a significant development effort. This fact has hindered the introduction of PR systems in the industrial sector, relegating PR to research applications. On the purpose of easing PR system design, Xilinx has evolved its PR Design Flow during the last years, taking significant advance in the field of PR. However, it still imposes important limitations to the implementation of PR systems.

Taking advantage of the Xilinx Design Language (XDL) a number of research developments have been accomplished aimed to circumvent, among others, the aforementioned issue. XDL is as an ASCII-based non-proprietary FPGA physical description language provided by Xilinx. The possibility to manage FPGA resources at the lowest level, in terms of both logic and routing, has brought about the development of new approaches for FPGA-based system design beyond the scope of traditional FPGA vendor commercial tools.

During the last years a remarkable effort has been made by the research community in this regard. This effort has resulted in the development of new applications and tools for reconfigurable system design.

New design possibilities enabled by XDL have resulted in a new paradigm for PR system design. In this regard, the most relevant related research works can be arranged according to the following areas:

- Algorithms for reconfigurable logic placement and routing
- Tools for XDL-based system design
- Frameworks for advanced PR system design
- Techniques for improved PR implementation

The analysis of presently available algorithms, tools, frameworks and techniques in the field of reconfigurable system design will provide a real view of the direction towards PR research is heading. Autonomous Fault Tolerant Systems (AFTS) or Software Defined Radio (SDR) are some examples of the key applications for PR technology where new advances in PR system design will significantly leverage its implementation in industrial applications.

Dynamic Partial Reconfiguration or Partial Run-Time Reconfiguration, hereafter referred to as Partial Reconfiguration (PR), is a process consisting of swapping parts or modules of a reconfigurable system while the rest of the systems remains running and therefore fully operational.

Whereas FPGA technology provides a high degree of flexibility by allowing onsite device circuit reconfiguration, PR takes a step further by enabling on-site circuit partial reconfiguration. That is, some portions of the FPGA logic, referred to as Reconfigurable Region (RR), are modified dynamically by downloading partial bitstream file through the configuration port. During the PR process, the rest of the system or logic, referred to as Static Region (SR), continues running without being affected. The reconfigurable logic is therefore replaced by the content of the partial bitstream. In Figure 20-1 a basic representation of the PR concept is shown.

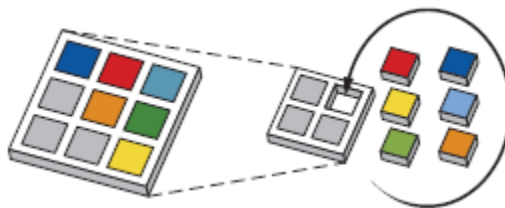


Figure 20-1: Partial Reconfiguration concept

There are several ways to download partial bitstreams into the FPGA. Apart from traditional methods, which consist in loading the bitstream externally through available configuration ports such as SelectMap or JTAG, there is an Internal Configuration Access Port (ICAP), which is essentially an internal version of the SelectMap. The ICAP therefore enables the Partial auto-Reconfiguration since a PR system is able to self-modify by accessing internally to the device configuration and load a new partial bitstream.

Partial bitstreams contain all the configuration commands and data necessary for PR. FPGA embedded configuration logic handles both full and partial bitstreams so the same type of programming information is processed in both cases in the same configuration engine. In addition, the task of loading partial bitstreams into an FPGA does not require knowledge of the physical location of the Reconfigurable Macro (RM) since the configuration frame addressing is already included in the partial bitstream.

XILINX PARTIAL RECONFIGURATION DESIGN FLOW

There exist two flows provided by Xilinx to design PR systems: Difference-Based PR and Module-Based PR. The main objective of the former is to allow small design changes to be performed dynamically. It is a useful method to make small on-the-fly changes to design parameters such as Look-up Table equations, Input-Output standards or filter parameters stored in a BRAM, among others. These design changes are made directly onto the placed and routed design making use of Xilinx FPGA Editor tool provided within the IDS software. The process output results in a partial bitstream that only contains information about the executed changes, that is, the differences between the original design bitstream and the modified design bitstream. Hence the name of Difference-Based PR.

Module-Based PR was originally based on Xilinx Modular Design methodology, which consists of defining distinct portions of an FPGA design in order to be one or more of them reconfigurable and the rest static. The initial design flow for Module-Based PR implied a great deal of challenges for PR system designers which hindered the development of PR designs. Nevertheless, based on the same modular design concept, Xilinx has provided significant improvements in PR design tools in the last years which has leveraged PR system design. Current Xilinx PR design flow can be considered as a mature and time-tested design option.

To design a PR system first of all the designer must define both the SR and at least one RR. With reference to the latter, the designer defines an RR in terms of both physical size and type of resources required. For each RR, a different set of RM is considered. It must be taken into account that the quantity and type of resources provided by the RR must be sufficient to host each of the selected RMs. The IDS software ensures that the resources used to construct the RM are completely contained within the selected RR and that no interference with the SR exists. Communication between static logic and reconfigurable logic is accomplished via the so-called Proxy Logic. A Proxy Logic is a single Look-up Table element automatically inserted by the software for each port of an RM (referred to as Partition Pin).

There are different PR styles depending on the way that RMs are implemented onto the RRs.

- *Island style*: this style allows swapping RMs exclusively in one RR on the FPGA. Although more than one RR may exist, each island is bound to host its individual set of RMs, thus not being possible to swap RMs between RRs. This configuration style is the only one supported by FPGA vendor tools such as Xilinx PR Design Flow
- *One-Dimensional Slot style*: in Island style, the largest RM defines the RR size. It results in a resource waste when a smaller RM is implemented in the RR. This effect is referred to as internal fragmentation. That is, a large RM cannot be replaced by multiple smaller RMs. Aiming at improving resource utilization; the RR is divided into a set of adjacent one-dimensional aligned resource slots. Hence, RMs can be implemented using required number of adjacent slots
- *Two-Dimensional Grid style*: even making resource slots as narrow as possible (1 CLB column wide in case of Xilinx FPGAs), it can still result in a waste of resources. It occurs in particular when dedicated primitives such as RAMs or multipliers are required. A step further consists in dividing the

resource slots so that they are organized in a two-dimensional grid. Thus, the internal fragmentation is reduced but in contrast RM placement becomes more complex.

PR system design support is provided via PlanAhead tool provided within the IDS software. All the elements required to build a PR system (SR, RRs and RMs) are managed in PlanAhead. Floorplanning, required to define RRs, and Design Rule Checks (DRC), established to guide designers on a successful path on design completion, are all accessed through the PlanAhead software environment.

In order to generate required full and partial bitstreams, Place and Route process is executed multiple times. Each placement and routing iteration results in a complete FPGA configuration. Once the design configuration meets all requirements, results from that successful implementation can be reused to create remaining configurations. Current IDS PR solution makes use of the so-called Partitions, a technology introduced in the ISE 8.2i, and enhanced in subsequent releases, to enable design reuse. After all configurations are implemented, verification routines validate consistency among all the versions. In addition, once all these design versions have been placed and routed, traditional timing, simulation and verification techniques can be used to validate results.

20.1.2 Related Use cases

Tecnalia will provide the implementation of the AFTS DM Technology Brick for Use Case UC2.5: the development of low level software for an Avionics Control Unit including autonomous navigation features based on GPS, inertial and/or image acquisition inputs.

The AFTS DM will consist of methodology adaptation to the specific aerospace application requirements (environment, application and architecture). It will result in the achievement of required fault tolerance level. In this regard, it should be noted that aerospace applications are subjected to the so-called Single Even Upsets. Hence, fault tolerance techniques are required in order to maintain SoC functionality without interruption.

20.2 Specification

20.2.1 Requirements from the UCs

As it has been said, PR is the ability to change design modules dynamically while the remaining system continues to run without interruption. In other words, PR is the ability to time multiplex hardware dynamically. Accordingly, FPGA flexibility is significantly enhanced which entails an important benefit in the SoPC design. In addition, since system modules and, therefore, system functionality can be modified on the fly, PR allows the designer to move designs to smaller devices resulting in a lower design cost. Consequently, power consumption may be reduced. Generally speaking, PR permits a more efficient use of the silicon by only loading a required functionality at a specific point of time.

It could be stated that PR addresses three main design requirements by allowing the designer to:

- Increase system design flexibility enabling to change a design in the field. Moreover, it is only required to run the full design flow for the RM (instead of on the entire design)
- Reduce cost and size by time-sharing functionality. That is, it is possible to fit either more logic into a given device or a design into a smaller and thus less expensive device
- Reduce power consumption permitting to use smaller devices to implement designs or by loading functions (by means of RM) on-demand (instead of designing exclusively for high performance)

Apart from the three aforementioned fundamental design needs, the Space Use Case defined in WP2.5 also defines some additional requirements to be accomplished by AFTS DM:

Version	Nature	Date	Page
V1.0	P	2014-02-07	118 of 125

- Improve the actual FPGA fault tolerance
- Enable the use of new techniques in design security
- Reduce bitstream storage requirements
- Accelerate configurable computing
- Alignment with space safety and dependability standards
- Allow on-flight FPGA reconfiguration/reprogramming triggered by an external event
- Compatible with real-time systems (schedulability)

The main goal of utilizing PR is therefore to reduce FPGA size and consequently cost and power consumption. The achievement of this objective will enable the implementation of FPGA technology in systems that due to its power and cost constraints would require ASIC implementation.

From the Use Case point of view, it is of course desirable to reduce the impact of a number of drawbacks that can come together with the implementation of PR:

- Resource overhead for providing both PR process management and communication infrastructure for RMs
- Timing penalties for the communication
- A challenging and more complex design process

20.2.1.1 Use Case Description

The application to be implemented for the Space domain is the Low Level Software for an Avionics Control Unit whose application software could include autonomous navigation features based on GPS, inertial and/or image acquisition inputs. This unit will be based in a LEON architecture running in multicore configuration inside an FPGA.

The aforementioned Software implements satellite's vital functions such as: attitude and orbit control in both nominal and non-nominal cases, telecommands execution or dispatching, housekeeping telemetry gathering and formatting, on board time synchronisation and distribution, failure detection, isolation and recovery, etc.

20.2.2 How will this brick be integrated in the UC

From the engineering process point of view, the use case comprises the tasks shown in Figure 20-2.

Specifically the PR technology brick influences three of the aforementioned tasks:

- *Architectural Design*: Partial reconfiguration requires a control and monitoring module. The architecture of the use case should be designed to include this module.
- *Coding*: PR imposes some requirements to the coding of modules to be reconfigured. Specifically, these modules should be implemented in a restricted area of the FPGA and their interfaces should be clearly defined. The coding phase should take into account all these requirements.
- *Integration Tests*: Finally, the reconfiguration monitoring and control module will be integrated in the use case at this stage.

From the implementation point of view, the space use case will employ a dual-FPGA board, whose block diagram can be observed in Figure 20 3.

This dual-FPGA architecture allows evaluating multiprocessor systems where a main (multi)processor embedded within one of the FPGAs distributes the processing load to a second device; it is the Low Level SW of this multiprocessor whose development is the main driven of the Aerospace Demonstrator.

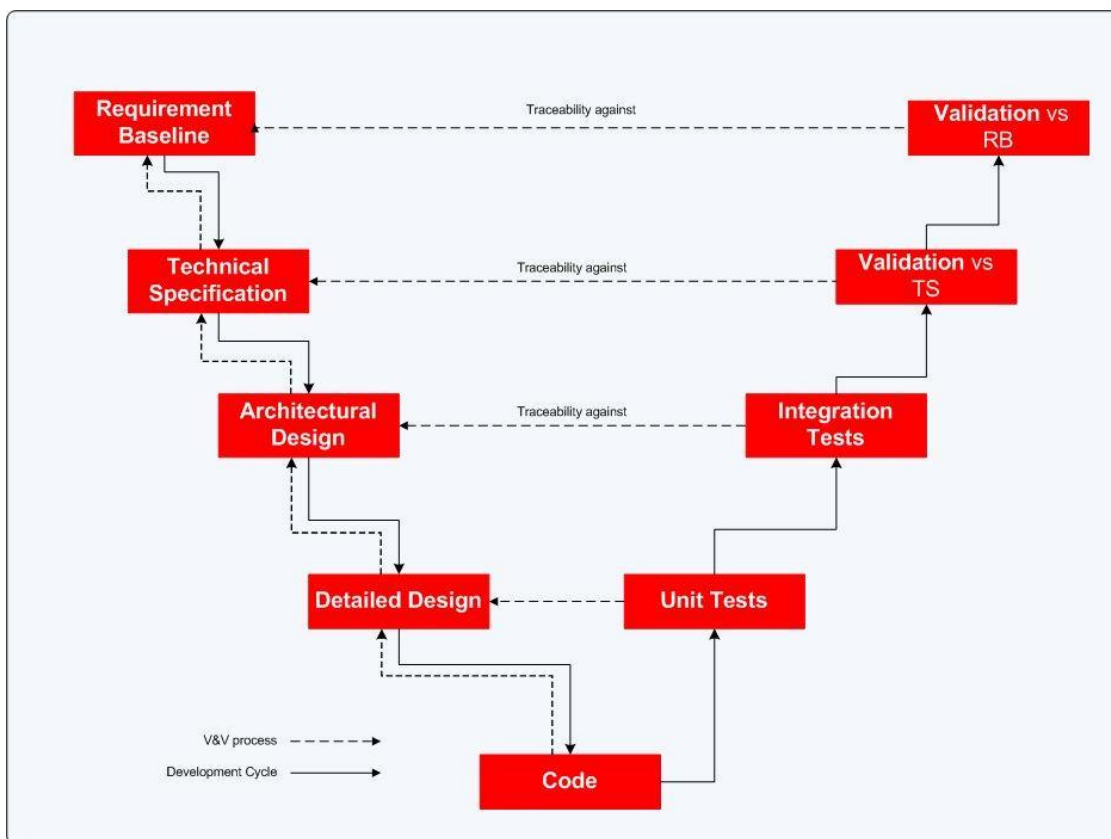


Figure 20-2: Software engineering process

In this case, the second FPGA can embed another processor, a DSP or just implement some hardware (VHDL) algorithms.

The PR methodology will be applied to some of the hardware modules implemented inside the Virtex 5 FPGA.

20.2.3 Requirements fulfilled by initial tool/method version

Without need of further development the PR technology brick tackles the following requirements:

- *Reduce cost and size of the FPGA devices:* PR allows reduce the number of modules to be implemented in the FPGA; making it possible, thus, to employ smaller devices.
- *Reduce power consumption* permitting to use smaller devices to implement designs or by loading functions (by means of RM) on-demand (instead of designing exclusively for high performance)
- *Reduce bitstream storage requirements*

20.2.4 What will be implemented/provided in the CRYSTAL project

Besides these requirements, PR can provide two new features to the Space use case:

- Fault-tolerance
- Flexibility and adaptability

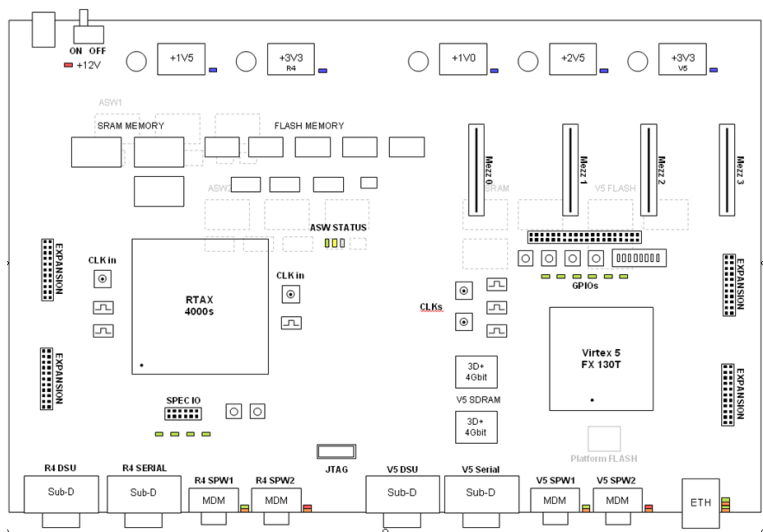


Figure 20-3: Dual-FPGA board potential lay-out

20.2.4.1 New and improved features

PR can be used to provide two new features to the Space Use Case:

- *Fault-tolerance*: PR might be employed to add fault tolerance capabilities to the application running in the FPGA. A monitoring system would be added to the FPGA design which would be in charge of controlling the smooth running of the application; were a failure be detected, this system would apply PR to reconfigure the damaged part of the FPGA design.
- *Flexibility and adaptability*: One of the main limitations of satellite system is their lack of reconfigurability once they are launched. If new standards or features appear when the system is already on space, there is no way to update the applications. In this regard, a system could be designed which, by means of partial reconfigurability, would updated a certain application on the controller's demand.

Adding these two new features to the use case is out of the scope of the CRYSTAL project, therefore only one of them will be selected. Unfortunately, at this stage of the use case definition it is not possible to determine which one is more appropriate for the use case.

20.3 Implementation/Elaboration

[This section is empty for this iteration of the document. In future iterations, it will give details on which requirements are successfully implemented and how they can be used]

20.4 Evaluation

[This section is empty for this iteration of the document. In future iterations, it will give details on how they fulfilment of the requirements was checked before integrating the brick into the SEE of the use case. For interoperability features, this might be done by pairwise interaction between some bricks.]



21 Summary

In the previous sections, the safety related bricks have been described and general as well as interoperability related requirements have been given. While some of the use cases are still in flux and there will be changes from lessons learned on the way anyway, the available information is sufficient to start development work on the bricks – both methodology bricks and tool bricks. For some of the bricks, development work has started in earnest already some time ago, the rest will follow now.

There are a lot of similarities between some of the bricks, e.g. there are three bricks related to FaultTree+. This will be analysed for synergies and possibilities for standardized interfaces for the next iteration of this document.

22 Terms, Abbreviations and Definitions

AFTS	Autonomous Fault Tolerant Systems
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AUT	Artefact Under Test
CENELEC	Comité Européen de Normalisation Électrotechnique - European Committee for Electrotechnical Standardization
CFT	Component Fault Trees
CMMI	Capability Maturity Model Integration
CO	Confidential, only for members of the consortium (including the JU).
CRYSTAL	CRITICAL SYSTem Engineering AcceLeration
CTF	Certification Tool Framework
D	Demonstrator
DCs	Design Components
ESA	European Space Agency
FTA	Fault Tree Analysis
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FPGA	Field Programmable Gate Array
HAZID	Hazard Identification
HAZOP	HAZard and OPerability
IOS	Interoperability Specification
MB	Model-Based
MDE	Model Driven Engineering
MSCs	Message Sequence Chart
MSDs	Modal Sequence Diagram
O	Other
OEM	Original Equipment Manufacturer
P	Prototype
PHI	Preliminary Hazard Identification
PP	Restricted to other program participants (including the JU).
PRM	Product Risk Management
PU	Public
R	Report
RAMS	Reliability, Availability, Maintainability and Safety
RE	Restricted to a group specified by the consortium (including the JU).
RTP	Reference Tool Platform
SAC	Safety application conditions
SafeCer	ARTEMIS Project "SAFETy CERTification"
SCADE	Safety Critical Applications Development Environment, product by Esterel Technologies

SDR	Software Defined Radio
SEE	Systems Engineering Environment
SP	Subproject
SPICE	Software Process Improvement and Capability Determination (ISO/IEC 15504)
SysML	Systems Modeling Language
TCG	Test Case Generation
UML	Unified Modeling Language
WP	Work Package
URML	a UML-Based Rule Modeling Language
V&V	Verification and Validation

Table 22-1: Terms, Abbreviations and Definitions



23 References

There are no external references in this document, except the CRYSTAL deliverables already listed in section 1.3.