



D4.4: Model & Traceability Management (MTM) Tool Set – Final version

Accompanying note

MILESTONE: 30/11/2019 (M32)

STATUS: FINAL

COORDINATOR: SOFT

CONTRIBUTORS: SOFT, ARM, UAQ, UOC, INT

REVIEWERS: CSY, MDH

DISSEMINATION LEVEL: PUBLIC

HANDLE: <http://hdl.handle.net/20.500.12004/1/P/MMART2/D4.4>

LAST EDITED: 29/11/2019

Executive summary

The overall objective of MegaM@Rt2 WP4 is to provide traceability and tool-supported techniques and methods for model management. These and the related developed tools' aim to create and manage large-scale models at runtime in order to support efficient verification and testing, including means for traceability of requirements. To address and reach this objective, WP4 investigates and develops relevant and efficient tool-supported solutions that are able to automate and generalize the traceability approaches, defined within the project, that address the project case-study challenges.

This deliverable D4.4 specifies the final version at M32 of the MegaM@Rt2 global model and traceability management tools developed within tasks T4.2, T4.3 and T4.4 to support model management and traceability activities. Therefore, this deliverable contains the current status of the traceability demonstrators developed within the MegaM@Rt2 project.

Table of Contents

Executive summary	2
Table of Contents	4
Acronyms	6
Introduction	10
Progress status of the traceability analysis tool set	11
NeoEMF	12
Update on delayed features at initial and intermediate release	12
Features planned for Final release (M32)	12
EMF Views	13
Update on delayed features at initial and intermediate release	13
Features planned for Final release (M32)	13
Modelio	14
Update on delayed features at initial and intermediate release	14
Features planned for Final release (M32)	14
Constellation	15
Update on delayed features at initial and intermediate release	15
Features planned for Final release (M32)	15
PADRE	17
Update on delayed features at initial and intermediate release	17
Features planned for Final release (M32)	17
JTL	18
Update on delayed features at initial and intermediate release	18
Features planned for Final release (M32)	18
Other tools from WP2 and WP3	18
Traceability framework status, the roadmap profile	19
Base Generic model management Roadmap	19
Inter-model traceability Roadmap	20
Inference support Roadmap	21
Relationship with other work packages	22
WP2, WP3 and WP4 harmonization	22
A common Traceability metamodel for WP4 tools integration	22
Model-driven Design-Runtime Interaction in Safety Critical System Development: an Experience	
Report on the CLEARSY Use Case	26
On a common Runtime metamodel for WP4-WP3 tools integration	32
Conclusions	33
References	34
Appendix: Hackathons and Tools Fair reports	36
Prague tools fair	36

Prague Hackathon results	37
CSY Challenge: Log interpretation	37
Santander Hackathon results	40
BT Challenge: Variability Modelling Using Requirements on Different Design Levels: Using High-level Requirements, HL Design, LL Requirements, and HL Test Specifications	40
CSY Challenge: Log and test interpretation	43

Acronyms

AADL	ARCHITECTURE ANALYSIS AND DESCRIPTION LANGUAGE
ADL	ARCHITECTURE DESCRIPTION LANGUAGE
AL	ARCHITECTURAL LANGUAGE
ALEX	AUTOMATA LEARNING EXPERIENCE
ALF	ACTION LANGUAGE FOR FOUNDATIONAL UML
API	APPLICATION PROGRAMMING INTERFACE
APL	APACHE PUBLIC LICENSE
ARINC	AERONAUTICAL RADIO INCORPORATED
ASCET	ADVANCED SIMULATION/SOFTWARE AND CONTROL ENGINEERING TOOL
ASIL	AUTOMOTIVE SAFETY INTEGRITY LEVEL
ASL	ACTION SPECIFICATION LANGUAGE
ATL	ATLAS TRANSFORMATION LANGUAGE
AUTOSAR	AUTOMOTIVE OPEN SYSTEM ARCHITECTURE
BMM	BUSINESS MOTIVATION MODEL
BPMN	BUSINESS PROCESS MODEL AND NOTATION
CBSE	COMPONENT-BASED SOFTWARE ENGINEERING
CDO	CONNECTED DATA OBJECTS
CP	CONSTRAINT PROGRAMMING
CPS	CYBER-PHYSICAL SYSTEMS
CSP	CONSTRAINT SOLVING PROBLEM
CSR	CASE STUDY REQUIREMENT
CTS	CONCEPTUAL TOOL SET
DL	DESCRIPTION LOGICS
DMA	DIRECT MEMORY ACCESS
DSE	DESIGN SPACE EXPLORATION
DSL	DOMAIN-SPECIFIC LANGUAGE
DSML	DOMAIN-SPECIFIC MODELLING LANGUAGE
EAST-EEA	ELECTRONICS ARCHITECTURE & SOFTWARE TECHNOLOGIES - EMBEDDED ELECTRONIC ARCHITECTURE
EMF	ECLIPSE MODELING FRAMEWORK
EMOF	ESSENTIAL MOF
EPL	ECLIPSE PUBLIC LICENSE
FBD	FUNCTION BLOCK DIAGRAM
FMI	FUNCTIONAL MOCK-UP INTERFACE
FOSS	FREE OPEN SOURCE SOFTWARE
fUML	SEMANTICS OF A FOUNDATIONAL SUBSET FOR EXECUTABLE UML MODEL
GMF	GRAPHICAL MODELING FRAMEWORK
GPL	GENERAL-PURPOSE (MODELLING) LANGUAGES
GPL	GNU PUBLIC LICENSE

GUI	GRAPHICAL USER INTERFACE
HUT-TCS	HELSINKI UNIVERSITY OF TECHNOLOGY - LABORATORY FOR THEORETICAL COMPUTER SCIENCE
HW	HARDWARE
IEC	INTERNATIONAL ELECTROTECHNICAL COMMISSION
IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
INCOSE	INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING
ISO	INTERNATIONAL ORGANIZATION FOR STANDARDISATION
ITEA	INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT
LBT	LEARNING-BASED TESTING
LGPL	LESSER GNU PUBLIC LICENSE
M2C	MODEL TO CODE
M2M	MODEL TO MODEL
M2T	MODEL TO TEXT
MAENAD	MODEL-BASED ANALYSIS & ENGINEERING OF NOVEL ARCHITECTURES FOR DEPENDABLE ELECTRIC VEHICLES
MAF	MAJOR FRAME
MARTE	MODELING AND ANALYSIS OF REAL-TIME EMBEDDED SYSTEMS
MAST	MODELING AND ANALYSIS SUITE FOR REAL-TIME APPLICATIONS
MBD	MODEL-BASED DEVELOPMENT
MBSE	MODEL-BASED SYSTEM ENGINEERING
MBT	MODEL-BASED TESTING
MDE	MODEL DRIVEN ENGINEERING
MDSD	MODEL DRIVEN SOFTWARE DEVELOPMENT
MDT	ECLIPSE MODELING DEVELOPMENT TOOLS
MFR	MMRT FRAMEWORK REQUIREMENT
ML	MODELLING LANGUAGE
MoC	MODEL OF COMPUTATION
MOF	META OBJECT FACILITY
M&S	MODELLING AND SIMULATION
NFP	NON FUNCTIONAL PROPERTY
OAL	OBJECT ACTION LANGUAGE
OCL	OBJECT CONSTRAINT LANGUAGE
OCRA	OTHELLO CONTRACTS REFINEMENT ANALYSIS
ODM	ONTOLOGY DEFINITION METAMODEL
OMG	OBJECT MANAGEMENT GROUP
OMT	OBJECT-MODELING TECHNIQUE
OOA	OBJECT-ORIENTED ANALYSIS
OOM	OBJECT-ORIENTED MODELLING
OOSA	OBJECT-ORIENTED SYSTEMS ANALYSIS
OOSE	OBJECT-ORIENTED SOFTWARE ENGINEERING

OS	OPERATING SYSTEM
OWL	WEB ONTOLOGY LANGUAGE
PAL	PLATFORM-INDEPENDENT ACTION LANGUAGE
PDM	PLATFORM DESCRIPTION MODEL
PIM	PLATFORM INDEPENDENT MODEL
PLC	PROGRAMMABLE LOGIC CONTROLLERS
PSCS	PRECISE SEMANTICS OF UML COMPOSITE STRUCTURES
PSM	PLATFORM SPECIFIC MODEL
PSSM	PRECISE SEMANTICS OF UML STATE MACHINES
PTA	PRICED TIMED AUTOMATA
QVT	QUERY/VIEW/TRANSFORMATION
RC	RESOURCE-CONSTRAINED
REAL	REQUIREMENT ENFORCEMENT ANALYSIS LANGUAGE
RTES	REAL-TIME EMBEDDED SYSTEMS
S3D	SINGLE SOURCE SYSTEM DESIGN
SAT	PROPOSITIONAL SATISFIABILITY PROBLEM
SBSE	SEARCH-BASED SOFTWARE ENGINEERING
SCADE	SAFETY-CRITICAL APPLICATION DEVELOPMENT ENVIRONMENT
SCRALL	STARR'S CONCISE RELATIONAL ACTION LANGUAGE
SWRL	SEMANTIC WEB RULE LANGUAGE
SMALL	SHLAER-MELLOR ACTION LANGUAGE
SMC	STATISTICAL MODEL CHECKING
SMM	STRUCTURED METRICS METAMODEL
SMT	SAT MODULO THEORIES
SMUML	SYMBOLIC METHODS FOR UML BEHAVIOURAL DIAGRAMS
SOA	SERVICE-ORIENTED ARCHITECTURE
SUT	SYSTEM UNDER TEST
SW	SOFTWARE
SWRL	SEMANTIC WEB RULE LANGUAGE
SysML	SYSTEM MODELLING LANGUAGE
TADL	TIMING AUGMENTED DESCRIPTION LANGUAGE
TCP	TOOL COMPONENT PURPOSE
TCTL	TIMED COMPUTATION TREE LOGIC
TIMMO	TIMING MODEL
TS-MM	MEGAMODELLING TOOL SET
TS-RT	RUNTIME TOOL SET
TS-SE	SYSTEM ENGINEERING TOOL SET
TSC	TOOL SET COMPONENT
TSP	TIME AND SPACE PARTITIONING
T&E	TEST AND EVALUATION

UML	UNIFIED MODELING LANGUAGE
UCAN	UNCONVENTIONAL COMPUTER ARCHITECTURE AND NETWORKS
UTA	UPPAAL TIMED AUTOMATA
UTP	UML TESTING PROFILE
V&V	VERIFICATION AND VALIDATION
W3C	WORLD WIDE WEB CONSORTIUM
XMCF	XTRATUM CONFIGURATION FILE
XMI	XML METADATA INTERCHANGE
XML	EXTENSIBLE MARKUP LANGUAGE
XSD	XML SCHEMA DEFINITION
xTUML	EXECUTABLE TRANSLATABLE UML
xUML	EXECUTABLE UML

1. Introduction

This deliverable covers the provision of the final version of the MegaM@Rt2 Model & Traceability Management (MTM) framework consisting of a set of tools developed in Task 4.2, Task 4.3, and Task 4.4 to support scalable model management, model-based techniques as well as traceability and provenance.

The objective of Task 4.2 is to propose scalable model management techniques required for the integration and use of the different modelling artefacts in a continuous development toolchain. The objective of Task 4.3 is to provide other complementary scalable model-based techniques, such as model storage and model transformation facilities.

Finally, Task 4.4 aims at supporting the generation/building and handling of traceability information, notably between design time (WP2) and runtime (WP3) models.

The project has evolved through a set of phases:

- by evaluating and defining a set of consistent requirements and tools implementing them - based on the case study providers' requirements and the current state of the art in the relevant system-engineering area;
- by identifying the tools' improvements, beyond the state of the art, required to address the MegaM@Rt2 objective related to the continuous development process of large and complex systems;
- by defining the roadmap to implement such improvements;
- by initiating baseline experiments to apply tools to case studies' scenarios and to provide preliminary feedback.

This deliverable presents the current status of the project evolution based on the defined roadmap. [Section 2](#) reports, for each tool, the detailed status of the planned activities highlighting the ones, i.e. the relevant tool purposes, not in-line or deferred with respect to the time plan.

[Section 3](#) summarizes and applies the tool set status to the roadmap, which maps the status, at M32, of the whole Model & Traceability Management (MTM) framework.

[Section 4](#) identifies the dependencies and relationships of the WP4 activities with the ones developed by other WPs. The deliverable is structured in a practical form that will be reused to report the status of the "final" milestone, simplifying the evaluation and the comparison of the relative progress between them.

2. Progress status of the traceability analysis tool set

This section gives an overview of the implementation achievements and development status of the model & traceability management tools with regards to the intermediate milestone, as stated in the deliverable D4.2 [1]. The achievements of each tool are thus presented, and commented upon, to make explicit the achievements of the tool requirements (also called *purposes*), both in the baseline and the intermediate version of the framework release.

To briefly synthesise the reported status, it should be noted that:

- **6 tools** are related to the model & traceability management tool set framework,
- **38 tool purposes** are related to the tools participating in model & traceability management issues:
 - **All 14 tool purposes** related to the initial release are “**DONE**”.
 - **All 8 tool purposes** related to the intermediate milestone are “**DONE**”.
 - Among **the 16 tool purposes** related to the final milestone (M32)
 - **13 purposes are “DONE”**;
 - **3 purposes NeoEMF-040, EMFVIEWS-070 and M.CSTEL-130 are “CANCELLED”**.

All the cancelled purposes have been justified by the evolution of the use case requirements or the framework architecture. Of course, **the related effort has been redirected on other relevant and more important capabilities.**

It should also be noted that this deliverable D4.4 focuses on the final milestones. To avoid misunderstanding and to facilitate the review, issues and progress statuses related to the intermediate version, which have been already fulfilled, are not presented here.

2.1.

2.2. NeoEMF

2.2.1. Update on delayed features at initial and intermediate release

No delayed features.

2.2.2. Features planned for Final release (M32)

Table 1: NeoEMF - M32

Purpose	Properties	Comments / Release notes	Affected CSR
NeoEMF-020: NeoEMF shall support integration with other Eclipse-based modeling tools (e.g. EMF Views) for collaborative modeling, access control, etc.	Criticality: High Release: Final Status: Done	The integration with EMF Views has been successfully prototyped and evaluated. Results have been published in the ACM/IEEE MODELS 2018 international conference. Further developments have been made for integration with the Epsilon framework as well. Complementary results are in the process of being published as an extended journal publication.	CAM_01, CSY_02, IKER_03, IKER_08, NOK_04, NOK_05, VCE_01, VCE_06
NeoEMF-030: NeoEMF shall support the integration of new data-stores, in addition to the NoSQL ones already being implemented.	Criticality: Medium Release: Final Status: Done	Directly related to the experiments described for NeoEMF-020, the integration with other kinds of data stores has been realized by the implementation of a NeoEMF connector for the Epsilon framework. This latter framework already comes with the support for multiple complementary data stores.	BT_01, BT_02, BT_03, CAM_01, NOK_11, NOK_21, TEK_03
NeoEMF-040: NeoEMF shall support the combination of several data-stores for a same model (improvement of the multi-database architecture).	Criticality: Medium Release: Final Status: Cancelled	This tool capability is not provided yet. It has finally not been developed as not strictly required by any of the considered use cases in the context of MegaM@Rt2.	BT_01, BT_02, BT_03, CAM_01, NOK_11, NOK_21, TEK_03

2.3. EMF Views

2.3.1. Update on delayed features at initial and intermediate release

No delayed features.

2.3.2. Features planned for Final release (M32)

Table 2: EMF Views - M32

Purpose	Properties	Comments/Release notes	Affected CSR
EMFVIEWS-050: EMF Views shall support the verification and validation of such viewpoints/views (e.g. reusing VeriATL).	Criticality: High Release: Final Status: Done	EMF Views core operations have been formalized, preparing for future work on the V&V side. However, we did not implement an actual V&V support due to the focus finally put on other more proprietary features.	BT_01, IKER_18, NOK_16, NOK_19, NOK_20, TEK_05, TRT_05, VCE_01
EMFVIEWS-060: EMF Views shall support editable and/or storable views, and provide related update strategies to base models.	Criticality: Medium Release: Final Status: Done	Views can be fully stored but only partially edited so far. The provided view update support is currently incomplete. This is a difficult and scientifically challenging topic we plan to continue addressing in future research efforts.	BT_01, IKER_07, IKER_08, IKER_18, NOK_09, NOK_16, NOK_17, NOK_18, NOK_19, NOK_20, NOK_25, TEK_05, TRT_05, VCE_01
EMFVIEWS-070: EMF Views shall provide access-control mechanism(s) for such viewpoints/views (e.g. specific user profiles).	Criticality: Medium Release: Final Status: Cancelled	EMF Views has already been used in order to implement some kind of access control capabilities over regular models. However, the definition and provisioning of user profiles (and corresponding access control rules) on top of model views is not yet implemented within EMF Views itself.	BT_01, IKER_08, IKER_18, NOK_16, NOK_19, NOK_20, TEK_05, TRT_05, VCE_01

2.4. Modelio

2.4.1. Update on delayed features at initial and intermediate release

No delayed features.

2.4.2. Features planned for Final release (M32)

Table 3: Modelio - M32

Purpose	Properties	Comments/Release notes	Affected CSR
MODELIO-080: Modelio shall manage traceability on holistic system engineering level.	Criticality: High Release: Final Status: Done	Holistic system engineering support is postponed till further progress on the methodological aspects in WP2 and clarification of requirements and scenarios by the case study providers.	BT_01, BT_05, BT_06, IKER_16, IKER_18, NOK_12, NOK_14, NOK_17, NOK_18, NOK_19, NOK_20, VCE_01, TRT_05, TEK_01, TEK_05

2.5.

2.6. Constellation

2.6.1. Update on delayed features at initial and intermediate release

No delayed features.

2.6.2. Features planned for Final release (M32)

Table 4: Constellation - M32

Purpose	Properties	Comments/Release notes	Affected CSR
M.CSTEL-100: Modelio Constellation shall provide Mega Modelling capabilities for model storage	Criticality: High Release: Final Status: Done	Modelio SaaS development includes a first tentative to fulfill this requirement.	BT_01, BT_02, BT_03, CAM_01, NOK_11, NOK_21, TEK_03
M.CSTEL-110: Modelio Constellation shall provide Mega Modelling capabilities for model query	Criticality: High Release: Final Status: Done	Modelio SaaS development includes a first tentative to fulfill this requirement.	BT_01, BT_02, BT_03, CAM_01, NOK_11, NOK_21, TEK_03
M.CSTEL-120: Modelio Constellation shall provide Mega Modelling capabilities for model transformation	Criticality: High Release: Final Status: Done	Modelio SaaS development includes a first tentative to fulfill this requirement.	BT_01, BT_02, BT_03, CAM_01, NOK_11, NOK_21, TEK_03
M.CSTEL-130: Modelio Constellation shall provide Mega Modelling capabilities for traceability	Criticality: High Release: Final Status: Cancelled	This functionality has been implemented at the Modelio level.	BT_01, BT_05, IKER_16, IKER_17, IKER_18, NOK_12, NOK_14, NOK_16, NOK_17, NOK_18, NOK_19, NOK_20, NOK_21, NOK_25, VCE_01, TRT_05, TEK_05
M.CSTEL-140: Modelio Constellation shall integrate to Mega Modelling infrastructure	Criticality: High Release: Final Status: Done	Modelio SaaS development includes a first tentative to fulfill this requirement.	BT_01, BT_02, BT_03, BT_05, CAM_01, CSY_02, IKER_03, IKER_07, IKER_08, IKER_09, IKER_10, IKER_15, IKER_16, IKER_17, IKER_18, IKER_27, NOK_04, NOK_05, NOK_09, NOK_10,

			NOK_11, NOK_12 NOK_13, NOK_14, NOK_16, NOK_17, NOK_18, NOK_19, NOK_20, NOK_21, NOK_23, NOK_24, NOK_25, TEK_03, TEK_05, TRT_05, VCE_01, VCE_06
--	--	--	---

2.7.

2.8. PADRE

2.8.1. Update on delayed features at initial and intermediate release

No delayed features.

2.8.2. Features planned for Final release (M32)

Table 5: PADRE - M32

Purpose	Properties	Comments/Release notes	Affected CSR
PADRE-050: PADRE shall enable different types of refactoring sessions at different levels of automation.	Criticality: High Release: Final Status: Done	This tool capability is planned for a future release.	IKER_01, NOK_01, TEK_02, CSY_02, CSY_01, TEK_01, BT_04, CAM_02, NOK_22, IKER_16, IKER_17, VCE_01, TRT_05, TEK_05, NOK_16, NOK_21, NOK_25
PADRE-060: PADRE shall integrate runtime traces in the performance antipatterns detection and model refactoring activities.	Criticality: High Release: Final Status: Done	This tool capability is planned for a future release.	IKER_01, NOK_01, TEK_02, CSY_02, CSY_01, TEK_01, NOK_07, NOK_08, CSY_05, TRT_02, TRT_05, IKER_16, NOK_25, IKER_16, IKER_17, VCE_01, TRT_05, TEK_05, NOK_16, NOK_21, NOK_25

2.9.

2.10. JTL

2.10.1. Update on delayed features at initial and intermediate release

No delayed features.

2.10.2. Features planned for Final release (M32)

Table 6: JTL - M32

Purpose	Properties	Comments/Release notes	Affected CSR
JTL-060: JTL shall support the application of changes deducted from runtime models to the correspondent design models by using traceability models	Criticality: High Release: Final Status: Done	JTL shall be able to apply changes on design models by using traceability models and inference information previously calculated from the runtime models	NOK_25, IKER_16, IKER_17, VCE_01, TRT_05, TEK_05, NOK_16, NOK_21, NOK_25
JTL-070: JTL shall support the integration with other Eclipse-based modeling tools.	Criticality: High Release: Final Status: Done	JTL shall support the integration with other MegaM@Rt tools required to achieve their purposes (e.g., tools able to perform inference computations, monitoring, non-functional analysis).	BT_01, BT_05, IKER_16, IKER_17, IKER_18, NOK_12, NOK_14, NOK_16, NOK_17, NOK_18, NOK_19, NOK_20, NOK_21, NOK_25, VCE_01, TRT_05, TEK_05

2.11. Other tools from WP2 and WP3

S3D is a framework developed by UCAN, which is specially focused on WP2 and WP3, but also in relation with WP4. Its goal is to provide a tool with capability to generate executables from UML models, execute them, collect metrics from execution and feed the model with the results. In that context, although S3D does not have a specific feature focused on WP4, its internal operation has elements covered in this WP. Traces are not planned to be a goal by itself, since there is no work planned on trace viewing or similar. However, with the evolution of the project and after evaluating the work done on WP4 and the proposals from other partners, UCAN has decided to generate traces to later obtain the information about the execution of the system. Thus, the requirement "S3D-060: S3D Framework shall support continuous integration process that supports test execution" (Release Intermediate) has been affected and involved in the activities of WP4. UCAN has decided to take advantage of the proposal of using CTF as the selected format for trace generation.

3. Traceability framework status, the roadmap profile

This section tracks, for the final milestone at M32, the MegaM@Rt2 Framework implementation plan and the enhancements or upgrades of the tool features regarding WP4 challenges. For each framework requirement related to WP4, the roadmap table is reported with the status of the tool developments. Therefore, this section defines a synthesis, from a WP4 requirements point of view, of the implementation achievements, introduced per tool in the previous chapter of this document.

For each category of framework requirements related to model and traceability management, the table describes the implemented achievements regarding each milestone (baseline, intermediate at M20 and final at M32). Work that has been achieved before M20, is in unbolded black format. For readability issue, purposes that have been **cancelled** are colored in **bold red** and the purposes **achieved** between M20 and M32 are colored in **bold green**.

3.1. Base Generic model management Roadmap

Table 7: Status of Base Generic model management Roadmap

FRAMEWORK FEATURES	BASILINE	INTERMEDIATE	FINAL
MTM-01000: MTM TOOL SET SHALL PROVIDE BASE MODEL MANAGEMENT CAPABILITIES.	NEoEMF-010 M.CSTEL-030		NEoEMF-030 [done] NEoEMF-040 [done] M.CSTEL-100 [done] M.CSTEL-110 [[done] M.CSTEL-120 [done] M.CSTEL-140 [done] XPM-060 [done]
MTM-01010: MTM TOOL SET SHALL PROVIDE MODEL INDEXING/REFERENCING CAPABILITIES FOR LARGE SETS OF MODELS.	NEoEMF-010 M.CSTEL-030		NEoEMF-030 [done] NEoEMF-040 [done] M.CSTEL-100 [done] M.CSTEL-110 [done] M.CSTEL-120 [done] M.CSTEL-140 [done] XPM-060 [done]
MTM-01020: MTM TOOL SET SHALL PROVIDE MODEL STORAGE CAPABILITIES FOR LARGE SCALE (SETS OF) MODELS.	NEoEMF-010 M.CSTEL-030		NEoEMF-030 [done] NEoEMF-040 [done] M.CSTEL-100 [done] M.CSTEL-110 [done] M.CSTEL-120 [done] M.CSTEL-140 [done] XPM-060 [done]
MTM-01030: MTM TOOL SET SHALL PROVIDE MODEL QUERYING CAPABILITIES FOR LARGE SCALE (SETS OF) MODELS.	NEoEMF-010 M.CSTEL-030		NEoEMF-030 [done] NEoEMF-040 [done] M.CSTEL-100 [done] M.CSTEL-110 [done] M.CSTEL-120 [done] M.CSTEL-140 [done] XPM-060 [done]

3.2. Inter-model traceability Roadmap

Table 8: Status of Inter-model traceability Roadmap

FRAMEWORK FEATURES	BASILINE	INTERMEDIATE	FINAL
MTM-01100: MTM TOOL SET SHALL PROVIDE MODEL CARTOGRAPHY/VIEW CAPABILITIES OVER LARGE SETS OF INTERRELATED MODELS.	EMFVIEWS-010 EMFVIEWS-020 PAPHYRUS-010	EMFVIEWS-030 [done] EMFVIEWS-040 [done]	EMFVIEWS-050 [done] EMFVIEWS-060 [done] EMFVIEWS-070 [cancelled] M.CSTEL-140 [done]
MTM-01200: MTM TOOL SET SHALL PROVIDE MODEL VERSIONING CAPABILITIES OVER LARGE SETS OF INTERRELATED MODELS.	M.CSTEL-070 M.CSTEL-080		EMFVIEWS-060 [done] M.CSTEL-140 [done]
MTM-01300: MTM TOOL SET SHALL PROVIDE MODEL ACCESS-CONTROL CAPABILITIES OVER LARGE SETS OF INTERRELATED MODELS.	M.CSTEL-010 M.CSTEL-020 M.CSTEL-030 M.CSTEL-040 M.CSTEL-050 M.CSTEL-060		N _{Eo} EMF-020 [done] EMFVIEWS-070 [cancelled] M.CSTEL-140 [done]
MTM-01400: MTM TOOL SET SHALL PROVIDE MODEL IMPORT/EXPORT CAPABILITIES FROM/TO WP2-WP3 & OTHER MODELING SOLUTIONS.	MODELIO-150 EMFVIEWS-010 EMFVIEWS-020 JTL-010 S3D-120 CHESS-101		N _{Eo} EMF-020 [done] M.CSTEL-140 [done]
MTM-01500: MTM TOOL SET SHALL PROVIDE INTEGRATION WITH SOURCE CODE REPOSITORIES & CONTINUOUS DEVELOPMENT SOLUTIONS.		XPM-010 [done] XPM-020 [done]	M.CSTEL-140 [done] XPM-060 [done]
MTM-02000: MTM TOOL SET SHALL PROVIDE TRACEABILITY CAPABILITIES BETWEEN DESIGN/SYSTEM MODELS AND RUNTIME MODELS.	MODELIO-070 EMFVIEWS-010 EMFVIEWS-020 JTL-010	JTL-020 [done] JTL-030 [done] JTL-040 [done] JTL-050 [done]	MODELIO-080 [done] M.CSTEL-130 [cancelled] M.CSTEL-140 [done] JTL-070 [done]
MTM-02010: MTM TOOL SET SHALL PROVIDE INTER-MODEL TRACE STORAGE CAPABILITIES FOR LARGE SCALE (SETS OF) MODELS.	MODELIO-070 EMFVIEWS-010 EMFVIEWS-020 JTL-010	JTL-020 [[done] JTL-030 [done] JTL-040 [done] JTL-050 [[done]	MODELIO-080 [done] M.CSTEL-130 [cancelled] M.CSTEL-140 [done] JTL-070 [done]
MTM-02020: MTM TOOL SET SHALL PROVIDE INTER-MODEL TRACE QUERYING CAPABILITIES FOR LARGE SCALE (SETS OF) MODELS.	MODELIO-070 EMFVIEWS-010 EMFVIEWS-020 JTL-010	JTL-020 [done] JTL-030 [done] JTL-040 [done] JTL-050 [done]	MODELIO-080 [done] M.CSTEL-130 [cancelled] M.CSTEL-140 [done] JTL-070 [done]

3.3. Inference support Roadmap

Table 9: Status of Inference support Roadmap

FRAMEWORK FEATURES	BASILINE	INTERMEDIATE	FINAL
<p>MTM-03000: MTM TOOL SET SHALL PROVIDE INFERENCE CAPABILITIES FROM RUNTIME MODELS TO DESIGN/SYSTEM MODELS.</p>		PADRE-020 [done] EMFVIEWS-040 [done] PADRE-030 [done] JTL-030 [done]	M.CSTEL-130 [cancelled] M.CSTEL-140 [done] PADRE-050 [done] PADRE-060 [done] JTL-060 [done] JTL-070 [done]
<p>MTM-03010: MTM TOOL SET SHALL PROVIDE AUTOMATED INFERENCE COMPUTATION CAPABILITIES (FROM RUNTIME TO DESIGN MODELS).</p>		PADRE-020 [done] EMFVIEWS-040 [done] PADRE-030 [done] JTL-030 [done]	M.CSTEL-130 [cancelled] M.CSTEL-140 [done] PADRE-050 [done] PADRE-060 [done] JTL-060 [done] JTL-070 [done]
<p>MTM-03020: MTM TOOL SET SHALL PROVIDE INFERENCE INFORMATION ANALYSIS CAPABILITIES (FROM RUNTIME TO DESIGN MODELS).</p>		PADRE-020 [done] EMFVIEWS-040 [done] PADRE-030 [done] JTL-030 [done]	M.CSTEL-130 [cancelled] M.CSTEL-140 [done] PADRE-050 [done] PADRE-060 [done] JTL-060 [done] JTL-070 [done]

4. Relationship with other work packages

This section gives an overview of the dependencies and relationships of the WP4 activities with the other WPs in particular with:

- WP1, related to the framework architecture definition and the use case scenarios implementation;
- siblings WP2 and WP3, that complete the framework capabilities;
- WP5, concerning the integration of the whole set of participant tools in the MegaM@Rt2 framework.

4.1. WP2, WP3 and WP4 harmonization

For the M32 period the coordination and collaboration between WP2, WP3 and WP4, has continuously evolved. The WP and deliverable leaders worked together to maintain and ensure a significant level of consistency regarding all the outputs of the project. For this final milestone, the coordination involves the final status reports deliverables (D2.5, D3.5 and D4.4) and the tools guidelines deliverables (D2.6, D3.6 and D4.5).

In particular, a common traceability format (more details can be found in the next section) has been seen as an integration facility between the workpackages, WP2, WP3 and WP4. Interaction between tools providers

4.1.1. A common Traceability metamodel for WP4 tools integration

In order to provide interoperability between the WP4 tools, WP4 defined a generic traceability metamodel. It has to fulfill the needs of the MegaM@Rt2 project, but also to be reusable in other contexts/projects. By adopting a common traceability metamodel, the concerned tools are able to produce models that represent *Trace Links* between software artifacts while all conforming to the same metamodel. According to the WP4 purposes, the involved tools (e.g. the ones mentioned in Section 2) are interested in specifying correspondences between system runtime information (as obtained from the running system) and the different system design artefacts (notably design models). Such correspondences can be used to make consistent and exploitable relations between system design and runtime information. Furthermore, this integration allows a successive analysis of the traceability models (e.g. via model views interconnecting them in a transparent way) to discover system properties deviations and affected components. Such a feedback loop from runtime is highly relevant at design time, the most suitable level for system engineers to analyse and take impactful decisions accordingly.

As a directly related project result, the workshop MDE@DeRun 2018 (Model-Driven Engineering for Design-Runtime Interaction in Complex Systems), co-located with the STAF 2018 conferences, has been initiated and organized for the first time in June 2018. The main objectives, content and intermediate results of the event are explained in the report paper WP4 published after the workshop [6]. Following-up the success of the first edition, the second edition of the MDE@DeRun workshop took place in July 2019 still co-located with the STAF 2019 conferences [7], notably including a presentation of the main MegaM@Rt2 WP4-related research results [8].

The developed traceability support relies on a dedicated *Traceability Metamodel*, which facilitates the definition of traceability links with multiple sources and targets while considering different types of links, artefacts and additional information (as the development context of the *Trace Links*). The different concepts which are part of our traceability reference metamodel are graphically depicted in Figure 1.

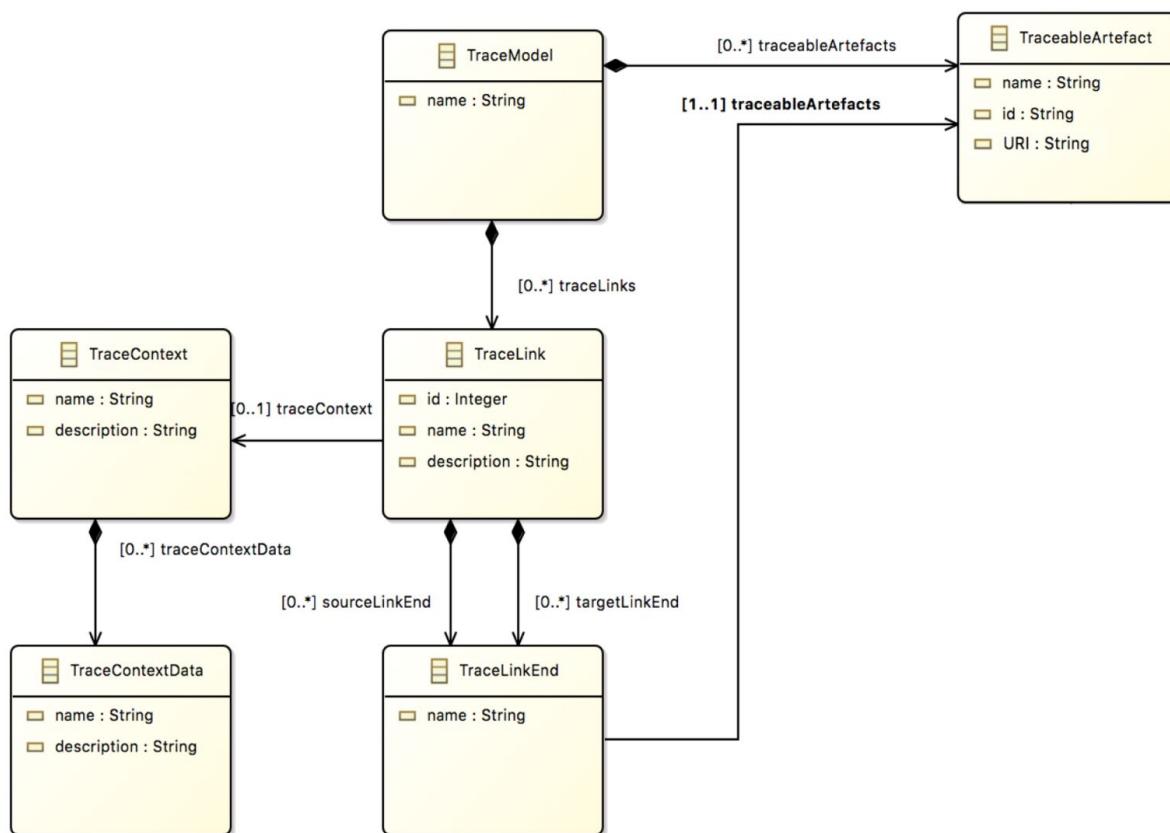


Figure 1 - Traceability Metamodel

This traceability reference metamodel basically defines the notion of *TraceModel*, which is the root element of any traceability model. A *TraceModel* stores all the traceability data regarding the mapping of a set of source artefacts into a set of target artefacts, i.e. two sets of *TraceableArtefacts*. For instance, *TraceLinks* can be defined between a design model and a model containing corresponding run-time information.

A *TraceModel* stores both *TraceableArtefacts* and *TraceLinks* between these artefacts. These *TraceableArtefacts* will normally be references to actual artefacts that live in source or target model. Moreover, the *TraceableArtefact* metaclass contains the attribute *URI* to refer to the location where the actual artefacts are stored, e.g., a text document, a UML model, (in fact it represents a generic link/path to the actual model element, e.g., a XMI link). Furthermore, *TraceableArtefact* contains an *id* that allows modelers to identify the traceable element inside this location (e.g., a reference to a template number inside a text document, the name or id of a model element inside a UML model). Moreover, this class can be specialized on the basis of specific needs; for instance, in EMF the metamodel can be extended in order to link a specific object of type `EObject` (`org.eclipse.emf.ecore.EObject`). Note that, it is possible to link elements that are in the same models and in different ones.

TraceLink elements represent explicitly trace relationships between a set of source and target artefacts. It contains several *TraceLinkEnds*. It also associates to a number of contexts through which it can capture custom information. *TraceLinkEnd* represents an end of a traceability link and it represents a specific *TraceableArtefact*.

The *TraceContext* metaclass is used to represent the role of the context metaclass, i.e., enable traceability metamodel designers to attach custom information to traceability links. Each context defines several *TraceContextData* that captures additional information. *TraceContextData* can be specialized on the base of specific needs of the MegaM@Rt2 project. For instance, relevant inference information calculated from the runtime models or non-functional analysis results that may be then further used at the system/design model-level.

We started to experiment the proposed Traceability metamodel by focusing on two tools from WP4, namely **EMF Views** (from ARM) and **JTL** (and UAQ). In particular:

- **EMF Views** is an Eclipse/EMF-based framework that allows specifying viewpoints gathering/interrelating concepts from one or several existing metamodels and obtaining views on corresponding sets of models. Such views can then be navigated and queried transparently as regular models. For more information on the particular topic of model views, please refer to the detailed study of the state-of-the-art we realized and published during MegaM@Rt2 [9].
- **JTL** is an Eclipse/EMF-based framework that allows to maintain consistency and synchronize software models via bidirectional transformations, and to keep traceability during design.

Within the project purposes, the WP4 tools are involved in the realization of a runtime-design feedback loop. Two explanatory examples showing the possible adoption of the proposed traceability metamodel by means of EMF Views and JTL are described below.

EMF Views example: Let us consider the realization of a runtime-design feedback loop via a view gathering 4 different models covering both runtime and design time aspects of a given system in the project. As shown in Figure 2, this view relies on a runtime log model (that conforms to a simple trace metamodel), a Java code model (that conforms to the Java metamodel from MoDisco), a component model (that conforms to OMG UML) and a requirement model (that conforms to OMG ReqIF). Thanks to this view, these 4 complementary models can be interconnected together.

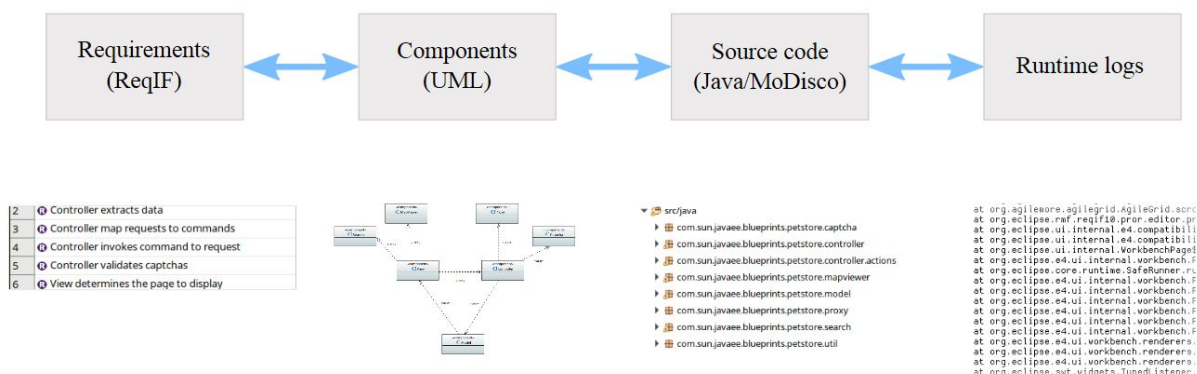


Figure 2 - Complementary models

A concrete example of the view from the previous figure is given in Figure 3. By using this view, an engineer can navigate transparently within and between the four contributing models as if they were all part of the same single model. For more details on this view example from MegaM@Rt2, and on how we realized it in practice using EMF Views, please refer to our corresponding ACM/IEEE MODELS 2018 publication [10].

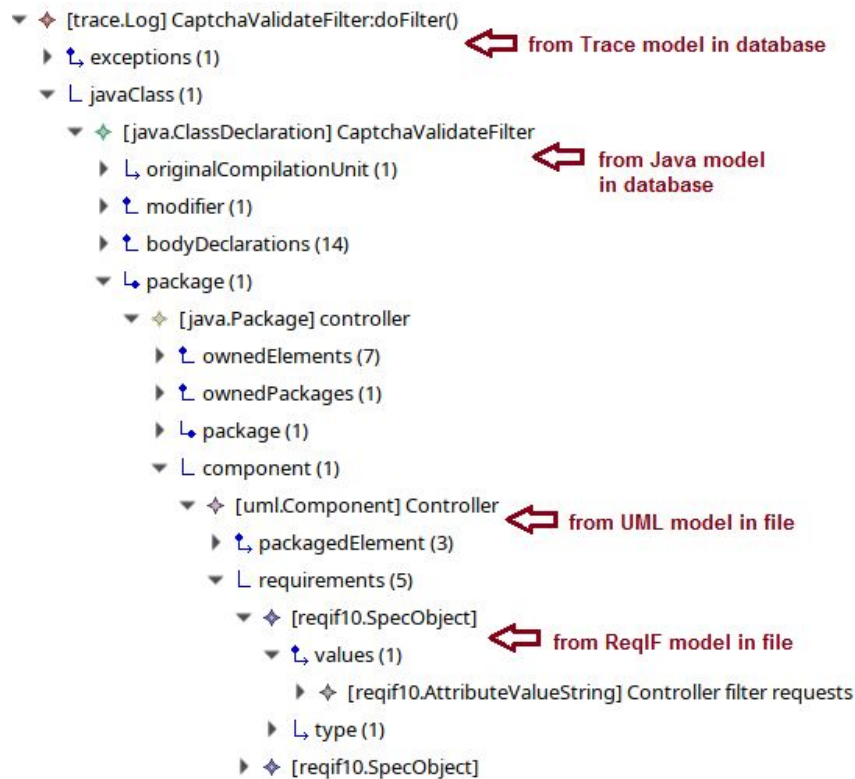


Figure 3 - A view combining the models

Thus, from a particular runtime information collected at system execution (here a *trace.Log* element), one can move back to the originating Java code instructions (here *java.ClassDeclaration* elements). One can then follow the links to the components (here *uml.Component* elements) the code implements, and up to the actual requirements these components fulfil (here *reqif10.SpecObject* elements).

Such a view combining different models can also be queried as any regular model, in order to extract relevant data out of it. For example, one can obtain all the requirements that are related to a given execution trace (runtime to design time traceability). Or, the other way around, one can get all the execution traces that correspond to a particular requirement (design time to runtime traceability).

JTL example: Let us consider the realization of the runtime-design feedback loop via an example that cover both runtime and design time aspects of a given system in the project. The design model consists of a sample UML Sequence Diagram that describes some interactions among different elements (as depicted in Figure 4).

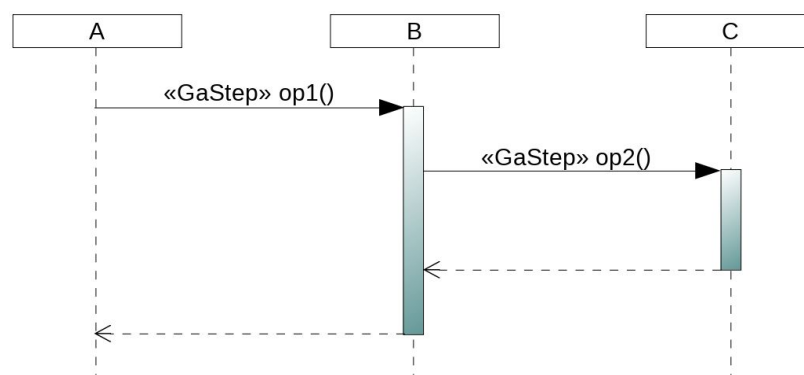


Figure 4 - A sample Sequence Diagram

Let us suppose that we are interested in analyzing the runtime logs produced by using the *Log4j* framework (a sample of log is depicted in Figure 5). To this end, logs are maintained in runtime models and related to the correspondent design models by means of trace links.

TIMESTAMP	CALLER CLASS.METHOD	LEVEL	TRACE NUMBER	SEQUENCE NUMBER	CALLED CLASS.METHOD	MESSAGE
2018-06-01T07:22:07.000+0200	[it.univaq.disim.sealab.traceability.A.main]	TRACE	trace:0	sequence:0	it.univaq.disim.sealab.traceability.B.op1	- Calling operation B.op1()
2018-06-01T07:22:00.100+0200	[it.univaq.disim.sealab.traceability.B.op1]	TRACE	trace:0	sequence:1	it.univaq.disim.sealab.traceability.C.op2	- Calling operation C.op2()

Figure 5 - A sample textual log

Figure 6 depicts the implementation of our vision of runtime-design models. In particular, the left-hand side of the figure depicts the design model of the considered Sequence Diagram in XML format within EMF; the right-hand side of the figure depicts the runtime model representing the considered logs (to this end we defined a specific metamodel representing the logs). In the middle, an instance of the used Trace metamodel is proposed to connect runtime and design model elements. It is described by trace links, expressed by source and target trace link ends. A trace link end is an external reference to elements belonging to the design or runtime models (as graphically represented by the blue arrows in the “Traceability between design and runtime models” figure).

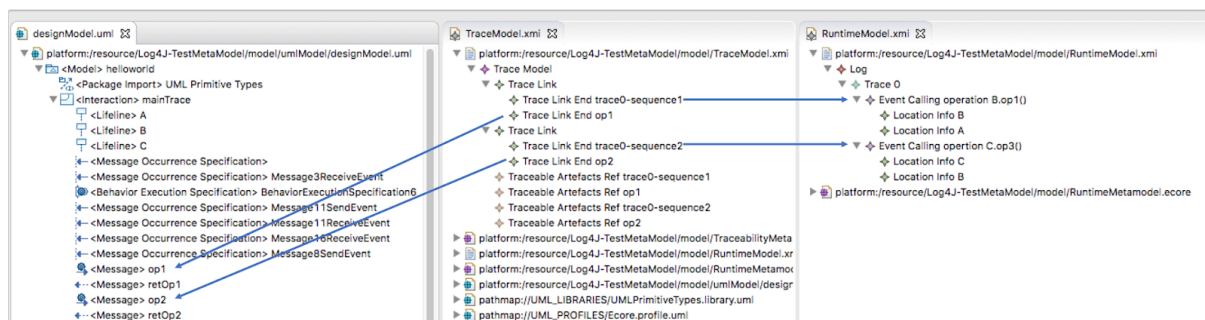


Figure 6 - Traceability between design and runtime models

In order to execute a full model-based performance antipattern evaluation, performance indices (e.g. response time, utilization, etc.) can be stored in the event class by processing runtime logs. These may help in detecting and removing performance antipatterns into the design model. For instance, the execution time of an event element is mapped to the response time of the *GaStep* stereotype of MARTE. To this purpose, the JTL framework can be combined with the **PADRE** framework (also involved in the WP4) that is able to perform a model-based performance antipattern evaluation.

4.1.2. Model-driven Design-Runtime Interaction in Safety Critical System Development: an Experience Report on the CLEARSY Use Case

During this last period of MegaM@Rt², we performed a practical experience of using a model-based approach and related techniques to deal with the interactions between design time and runtime in the development of a safety-critical software system [11]. Notably, the novelty resides in the combination of different complementary solutions for model traceability and model views in order to provide support for such a practical use case. Using the proposed approach, we also show how we can automatically infer some design deviations, and identify elements affected by these deviations, from a possibly large spectrum of runtime system configurations or conditions.

The reported model-based experiment is based on a real industrial use case from MegaM@Rt2, a Railway system developed by CLEARSY, one of the industrial partners of the MegaM@Rt2 project. In this context, CLEARSY aims at improving the robustness of its system by integrating the use of model-based techniques in its development cycle. Their goal is 1) to determine whether environmental conditions are met, and 2) to detect variations in the behaviour of the system in order to anticipate possible failures. In this paper, we propose both a conceptual model-based approach and an implementing solution that relies on Eclipse and EMF-based tools. The core components of our solution are 1) the definition of correspondences between design and runtime elements (as traceability models), and 2) the building of related views aggregating design and runtime models in a transparent way. To this intent, our implementation leverages the existing JTL and EMF Views tools from this MegaM@Rt2 Model & Traceability Management Tool Set.

Our model-based approach facilitates the establishment and exploitation of correspondences between design and runtime elements of a system. In particular, the system behavior at runtime is monitored, logged and then related to the initial system design whenever appropriate. The final objective is to connect fallback critical situations (at runtime) with their corresponding potential causes (in the system design and runtime). We specify design-runtime correspondences by means of a traceability model that links design and runtime information. These correspondences, along with the design and runtime models, are used as input to an integrated view that transparently relates the runtime logs with the initial system design. Navigating and querying this view can help the system engineer to discover fallback situations, identify their causes without manually analyzing very verbose logs, and navigate back to related safety requirements.

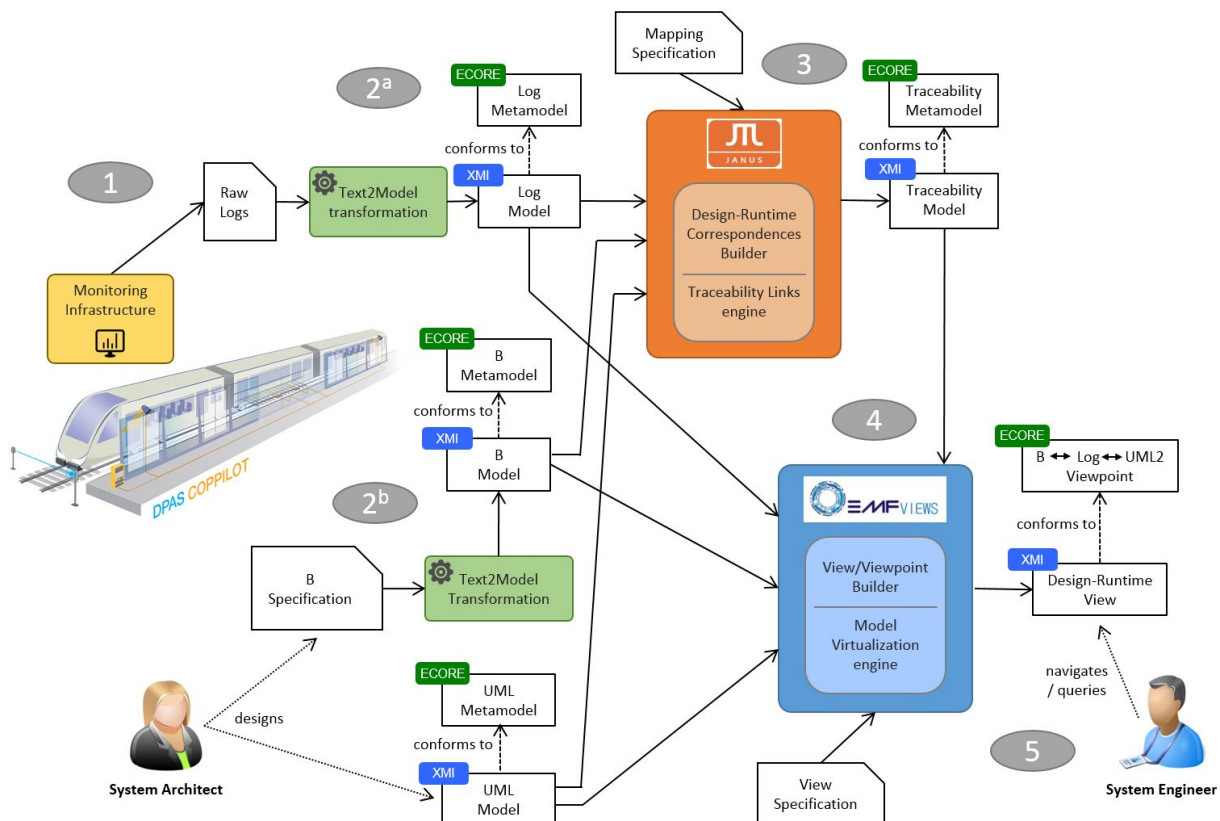


Figure 7 - Model-driven Design-Runtime Interaction in Safety Critical System Development: an Experience Report on the CLEARSY Use Case

Components developed specifically for this experiment, and existing solutions that had to be extended, are underlined in the figure right before. The main steps of our approach are the following:

1 - Monitoring. The considered system first has to be correctly instrumented in order to generate usable traces of its runtime execution. In practice, execution traces are often serialized in specific and (semi-)structured textual formats (e.g. comma-separated values). However, for large volumes of data, a binary format may be more efficient (e.g. the Common Trace Format (CTF) standard). In any case, the runtime traces must contain the relevant information required to later relate the runtime events with design elements. Moreover, we considered UML diagrams describing a static view of the system.

2 - Discovering design and runtime models. When not already stored as models that conform to an explicit metamodel, the design and runtime artefacts (requirements specification, runtime traces, etc.) should be converted to models. This may require specifying corresponding metamodels if not available. In our approach, we specified both the *Log Metamodel* that describes the runtime traces and the *B Metamodel* that covers B specifications (see *Log Model* and *B Model* in the figure, respectively). To convert runtime traces and B specifications into models, we use metamodel-driven *model discoverers*.

3 - Computing design-runtime traceability links. Once the design and runtime models are available, they can be linked together in several ways. We use traceability relationships which have been designed to help users understand associations and dependencies of heterogeneous models. In MDE, a traceability link is a relationship between one or more source model elements and one or more target model elements, whereas a *trace model* is a structured set of traceability links, e.g., between source and target models. For some previously identified cases, this can be performed automatically thanks to the definition of a list of patterns (using an appropriate formalism or language) to be detected from the analyzing the runtime data. Thus, the generated traceability model relates together the previously obtained runtime and design models according to these defined patterns

4 - Building the design-runtime view. In order to provide a transparent and integrated access to the runtime-design traceability information, we build a model view based on the previously obtained runtime, design and traceability models. This view acts as a “virtual” model that refers to these input models and connects them together according to the traceability information computed in the previous step. It has to be able to handle possibly large input models (e.g. verbose runtime traces) [10]. Moreover, it is important to note that this view conforms to a viewpoint that specifies how the different corresponding metamodels (*B*, *Log* and *UML* in our present case) are interconnected together. Such a viewpoint can also filter the element types that are not required by the users/engineers for the targeted engineering activities.

5 - Navigating and querying the design-runtime view. The view is an integrated interface provided to the system engineer in charge of analyzing and diagnosing the system. It offers a single entry point to the engineer, thus hiding the unnecessary complexity of the individual models (including the Traceability one) contributing to the view. From the view, the engineer can transparently navigate and query all the information relevant to the system, its runtime behavior and design specification. She/he can notably specify her/his own particular queries, and also rely on predefined libraries of queries that are adaptable in the context of her/his scenario. This way, the view can be used to diagnose the system and conduct its evolution (in collaboration with the system architect for example).

In this approach, when the system engineer wants to build a view on another runtime trace, only two steps need to be run again: first building a model from the new trace (2a), then recomputing the traceability links (3). The view can then be navigated and queried accordingly.

If the engineer wants to analyze a new fallback situation, then the only change required is to add a pattern for this fallback in the traceability link computation step (3). If the view should contain more information, e.g. specific to this new fallback situation, then the view configuration should be altered, or a new view should be created to reflect the updated situation (step 4).

4.1.3. Exploiting Architecture/Runtime Model-driven Traceability for Performance Improvement

In order to manage software complexity, ever more companies are considering Model-Driven Engineering (MDE) approaches, with the perceived benefit of enabling developers to work at a higher level of abstraction and to rely on automation throughout the development process. In this context, non-functional properties (e.g. performance, power consumption or memory footprint) are becoming ever more relevant for the success of a software application, and the early identification of problems induces lower cost solutions. By connecting runtime information and architectural design, developers can suggest architectural changes needed to meet performance requirements before the system actually experiments certain scenarios (e.g., some specific workloads). We propose a model-driven approach to support designers in their performance analysis and model refactoring processes by exploiting design/runtime interactions [12]. In particular, the system behavior at runtime is related to the architectural design in order to investigate potential performance issues in design and to suggest possible model refactoring actions. The approach has been realized within Eclipse EMF (Eclipse Modeling Framework: <https://www.eclipse.org/modeling/emf/>) and it integrates a model-driven framework for the definition of correspondences between design and runtime with another one for performance analysis and model refactoring, respectively JTL and PADRE. The approach has been applied on an e-commerce application based on microservices that has been designed by means of UML software models (profiled with MARTE).

The idea underlying our approach exploits the correspondences between the architectural design and the runtime aspects of a software system, with the aim of improving its performance. In particular, the system behavior at runtime is matched with the architectural design in order to connect performance critical situations at runtime with their corresponding potential causes in design. We specify such design-runtime correspondences by means of traceability models to make the relationships between system design and runtime information consistent and exploitable. Consequently, the analysis of such traceability models can help to discover system properties deviations and to identify the affected components.

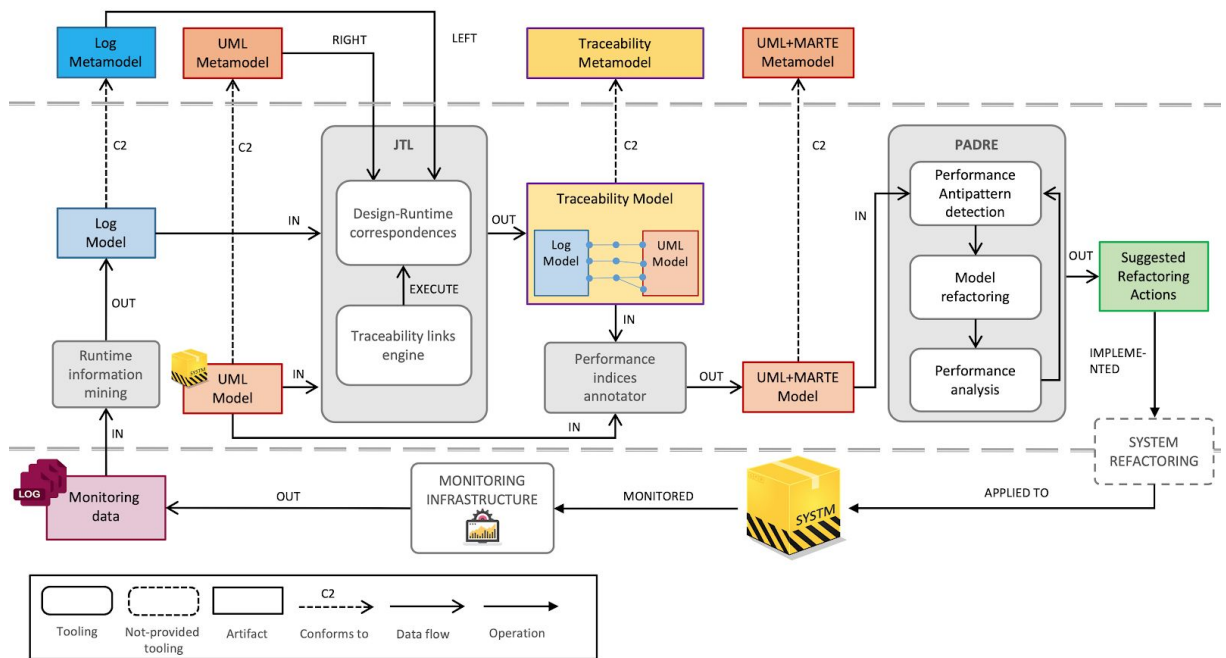


Figure 8 - Overall architecture

The overall architecture of our approach is depicted in Figure 8, where the top side represents the flow of activities on design artefacts, whereas the bottom side contains runtime activities. The large gray boxes represent the JTL and the PADRE frameworks that have been introduced in the previous section. The system under analysis is represented by the yellow box in the bottom part of the figure. The flow starts with an architectural design (represented by a UML Model) and Monitoring data (represented as Log files), and it follows a sequence of steps that are described in the following. As depicted in Figure 8, runtime data (i.e., logs/traces) are obtained through a monitoring infrastructure over a running system. The collected runtime information is then integrated in the EMF-based environment and translated in EMF artifacts. In this step, Logs are automatically transformed in Log Models that conform to a specific Log Metamodel (properly defined in Ecore format) representing monitoring data.

In this work, we propose to generate traceability links between UML and Log Models by means of JTL. In particular, the tool allows to specify Design-Runtime correspondences in a declarative way at metamodel level, as bidirectional model transformations (i.e., between design and runtime metamodels). The JTL Traceability engine is able to execute such bidirectional model transformations and automatically generate the correspondent traceability links between elements of the UML design model and the log model ones. Traceability links are collected in an explicit way as in Traceability Models conforms to a dedicated metamodel, namely the reference JTL Traceability Metamodel. In general, performance parameters (e.g., workload, resource demands) represent input values requested to solve the performance model derived from the software model, whereas indices (e.g., throughput and response time) represent the output of performance analysis. In this step, both parameters and indices are obtained and then annotated on the UML+MARTE Model that will be used within PADRE (as described in detail later) for the sake of Performance Antipattern detection.

PADRE is used to perform a performance antipattern analysis and to suggest the most promising refactoring actions that shall remove detected antipatterns and then improve the overall system performance. As illustrated in Figure 8, starting from an UML+MARTE Model, Performance Antipattern detection is first executed, followed by a Model refactoring step. The following

Performance Analysis step in the figure collapses two sub-steps, that are: (i) automated transformation of the refactored model into a performance model (i.e. a Queueing Network), and (ii) execution of Mean-Value Analysis on the performance model to obtain the current performance indices. PADRE is also able to back-annotate the UML+MARTE Model with the obtained performance indices, so that it can undergo a new refactoring loop. The process lasts until either the user stops it or no more performance antipatterns are detected on the model. The output of this process is represented by a set of well-formed refactoring actions that, at the design level, have demonstrated to effectively improve the system performance. The system refactoring step is represented as a dashed box between the design and the runtime areas of Figure 8, because the propagation of these actions on the running system is still to be automated.

We consider an e-commerce web application based on microservices. According to the microservice architecture, the application is developed as a suite of small services, each running in its own process and communicating with RESTful HTTP API. The application is composed by 9 microservices, each requiring a different database to operate, and developed on top of the Spring Cloud(Spring Cloud: <https://spring.io/projects/spring-cloud>) framework. Starting from the Log and UML models, a Traceability Model is automatically generated by means of JTL, as shown in Figure 9. In particular, a set of trace links between left and right elements and the rules that enforced their mapping are collected. A TraceLink relates one or more elements belonging to the left domain and the correspondent (one or more) elements belonging to the right domain.

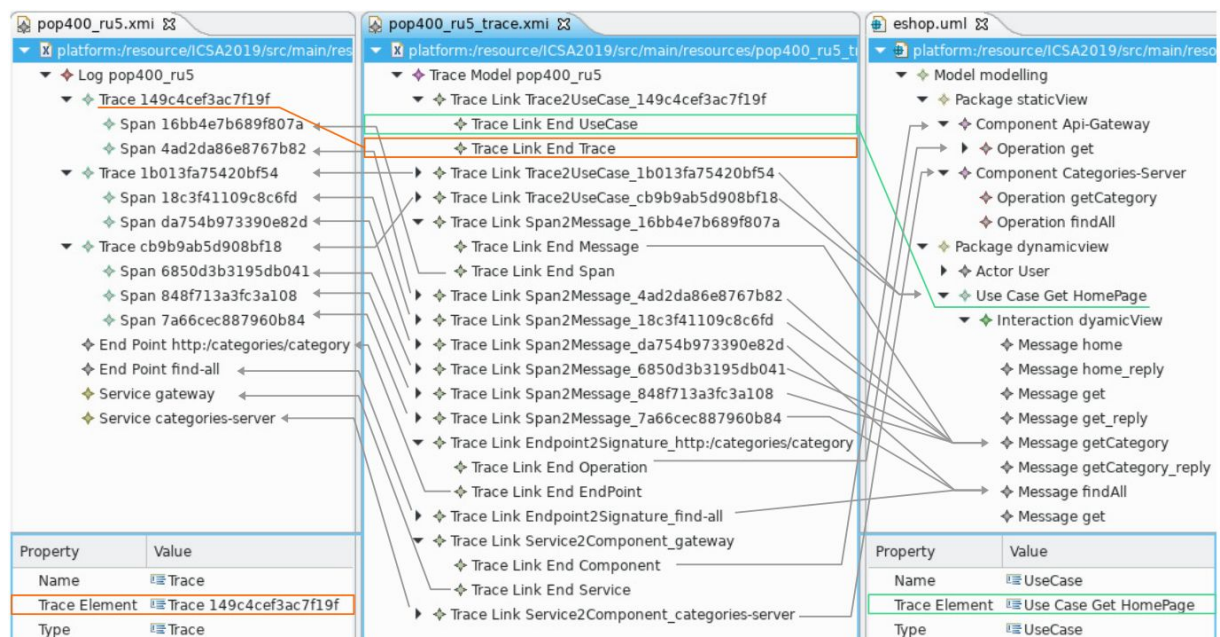


Figure 9 - Generated traceability model

The correspondences between the design and runtime concepts are defined by means of JTL and specified between the corresponding metamodels. The specification is defined by means of relations between elements of the two involved domains. The Traceability Model generated above is used to extract performance properties and incorporate them into the UML Model by means of the MARTE profile. In particular, the runtime information obtained by the monitoring infrastructure is used to obtain the performance input parameters, whereas the relationships between runtime information and software model are used to identify the proper UML elements that have to be annotated with these parameters.

The obtained UML+MARTE Model is given as input to PADRE. The Performance Antipattern detection step on our case study has identified some performance antipatterns. Furthermore, PADRE proposes a set of refactoring actions for each detected performance antipattern. For instance, among the actions proposed for a PaF performance antipattern, PADRE suggests to move the most demanding operations either to a new component deployed on a new node or to a component with less demanding operations. Moreover, PADRE suggests to deploy the critical component on a new and more powerful node, or on the node with the lowest utilization. By executing the detection and removal of performance antipatterns, we showed that the system performance has been improved as reported in the following table.

Refactoring Action	Web (Response Time = 1.615s)		Warehouse (Response Time = 53.332ms)	
	New RT	Improvement %	New RT	Improvement %
Move operation	1.402s	13.34	50.347ms	5.04
Increasing device capability	1.281s	20.73	53.332ms	0

Table 10: System performance

4.1.4. On a common Runtime metamodel for WP4-WP3 tools integration

Similarly to what has been done on identifying/defining a common metamodel for representing traceability information, WP4 also discussed a common metamodel for representing runtime information (e.g. execution logs, etc.). This ongoing work is in direct relation with WP3 but is very relevant in the context of the WP4 tools that will consume the corresponding models.

In some performed experiments, e.g. the ones presented in the previous section, partners had to define their own (simple) specific metamodels for dealing with the runtime information they had to consider in their respective contexts. However, in terms of reusability and interoperability it is preferable to be able to rely on a commonly shared runtime data representation format. Notably, some MegaM@Rt2 partners used the Common Trace Format (CTF, <https://diamon.org/ctf/>).

5. Conclusions

This document presented the final status of model and traceability management tool set component of the MegaM@Rt2 Framework. It provides an overview of the major achievements, features and work done during the project.

The “tools roadmaps” provides a detailed picture of the development activities per each tool, which shows that **the delivery plan was fundamentally respected** during the project even if **slight delays** has been managed without any impacts on UC providers activities.

All the cancelled purposes have been justified by the evolution of the use case requirements or the framework architecture. Of course, **the related effort has been redirected on other relevant and more important capabilities.**

To conclude, **the project development plan for WP4 successfully realised.** All the main project objectives have been fulfilled by adapting the tools during the development progress based on the concrete users needs but providing as well advanced features that are expected to be exploited in future applications.

References

- [1] MEGAM@Rt2, "D4.2: SPECIFICATION OF THE MODEL & TRACEABILITY MANAGEMENT TOOL SET," MARCH 2018. [ONLINE]. AVAILABLE: [HTTP://hdl.handle.net/20.500.12004/1/P/MMART2/D4.2](http://hdl.handle.net/20.500.12004/1/P/MMART2/D4.2)
- [2] MEGAM@Rt2, "D1.1: INDUSTRY REQUIREMENTS SPECIFICATION," SEPTEMBER 2017. [ONLINE]. AVAILABLE: PUBLISHED AS A CONSORTIUM INTERNAL DOCUMENT.
- [3] MEGAM@Rt2, "D1.3: CASE STUDIES SCENARIOS DEFINITION," JUNE 2018. [ONLINE]. AVAILABLE: PUBLISHED AS A CONSORTIUM INTERNAL DOCUMENT.
- [4] MEGAM@Rt2, "D1.2: ARCHITECTURE SPECIFICATION AND ROADMAP," NOVEMBER 2017. [ONLINE]. AVAILABLE: PUBLISHED AS A CONSORTIUM INTERNAL DOCUMENT.
- [5] MEGAM@Rt2, "D1.4: ARCHITECTURE SPECIFICATION AND ROADMAP" SEPTEMBER 2018[ONLINE]. AVAILABLE: PUBLISHED AS A CONSORTIUM INTERNAL DOCUMENT.
- [6] HUGO BRUNELIERE, ROMINA ERAMO, ABEL GOMEZ, VALENTIN BESNARD, JEAN-MICHEL BRUEL, ET AL.. MODEL-DRIVEN ENGINEERING FOR DESIGN-RUNTIME INTERACTION IN COMPLEX SYSTEMS: SCIENTIFIC CHALLENGES AND ROADMAP. MDE@DeRUN 2018 WORKSHOP, CO-LOCATED WITH THE SOFTWARE TECHNOLOGIES: APPLICATIONS AND FOUNDATIONS (STAF 2018) FEDERATION OF CONFERENCES, JUN 2018, TOULOUSE, FRANCE. (10.1007/978-3-030-04771-9_40).
<[HTTPS://HAL.ARCHIVES-OUVERTES.FR/HAL-01890878](https://hal.archives-ouvertes.fr/hal-01890878)>. SLIDES AT
<[HTTPS://WWW.SLIDESHARE.NET/HUGOBRUNELIERE/MODEL-DRIVEN-ENGINEERING-FOR-DESIGNRUNTIME-INTERACTION-IN-COMPLEX-SYSTEMS-SCIENTIFIC-CHALLENGES-AND-ROADMAP-STAF-2018-TOULOUSE-FRANCE](https://www.slideshare.net/HUGOBRUNELIERE/MODEL-DRIVEN-ENGINEERING-FOR-DESIGNRUNTIME-INTERACTION-IN-COMPLEX-SYSTEMS-SCIENTIFIC-CHALLENGES-AND-ROADMAP-STAF-2018-TOULOUSE-FRANCE)>
- [7] ALESSANDRA BAGNATO, HUGO BRUNELIERE, LOLI BURGUEÑO, ROMINA ERAMO, ABEL GOMEZ. STAF 2019 Co-LOCATED EVENTS JOINT PROCEEDINGS: 1ST JUNIOR RESEARCHER COMMUNITY EVENT, 2ND INTERNATIONAL WORKSHOP ON MODEL-DRIVEN ENGINEERING FOR DESIGN-RUNTIME INTERACTION IN COMPLEX SYSTEMS, AND 1ST RESEARCH PROJECT SHOWCASE WORKSHOP CO-LOCATED WITH SOFTWARE TECHNOLOGIES: APPLICATIONS AND FOUNDATIONS (STAF 2019), EINDHOVEN, THE NETHERLANDS, JULY 15 - 19, 2019. SOFTWARE TECHNOLOGIES: APPLICATIONS AND FOUNDATIONS (STAF 2019) FEDERATION OF CONFERENCES, JUL 2019, EINDHOVEN, NETHERLANDS. 2019. <[HTTPS://HAL.ARCHIVES-OUVERTES.FR/HAL-02180382](https://hal.archives-ouvertes.fr/hal-02180382)>
- [8] HUGO BRUNELIERE, ROMINA ERAMO, ABEL GOMEZ. A MODEL-BASED FRAMEWORK FOR CONTINUOUS DEVELOPMENT AND RUNTIME VALIDATION OF COMPLEX SYSTEMS (MEGAM@Rt2) - MDE@DeRUN 2019 - STAF 2019. SLIDES AT
<[HTTPS://WWW.SLIDESHARE.NET/HUGOBRUNELIERE/A-MODELBASED-FRAMEWORK-FOR-CONTINUOUS-DEVELOPMENT-AND-RUNTIME-VALIDATION-OF-COMPLEX-SYSTEMS-MEGAMRT2-MDEDERUN-2019-STAF-2019](https://www.slideshare.net/HUGOBRUNELIERE/A-MODELBASED-FRAMEWORK-FOR-CONTINUOUS-DEVELOPMENT-AND-RUNTIME-VALIDATION-OF-COMPLEX-SYSTEMS-MEGAMRT2-MDEDERUN-2019-STAF-2019)>
- [9] HUGO BRUNELIERE, ERIK BURGER, JORDI CABOT, MANUEL WIMMER. A FEATURE-BASED SURVEY OF MODEL VIEW APPROACHES. SOFTWARE AND SYSTEMS MODELING, SPRINGER VERLAG, 2019, 18 (3), PP.1931-1952. (10.1007/s10270-017-0622-9).
<[HTTPS://HAL.INRIA.FR/HAL-01590674](https://hal.inria.fr/hal-01590674)>. SLIDES AT
<[HTTPS://WWW.SLIDESHARE.NET/HUGOBRUNELIERE/A-FEATUREBASED-SURVEY-OF-MODEL-VIEW-APPROACHES-SOSYM-FIRST-MODELS-2018-COPENHAGEN-DENMARK](https://www.slideshare.net/HUGOBRUNELIERE/A-FEATUREBASED-SURVEY-OF-MODEL-VIEW-APPROACHES-SOSYM-FIRST-MODELS-2018-COPENHAGEN-DENMARK)>
- [10] HUGO BRUNELIERE, FLORENT MARCHAND DE KERCHOVE, GWENDAL DANIEL, JORDI CABOT. TOWARDS SCALABLE MODEL VIEWS ON HETEROGENEOUS MODEL RESOURCES. ACM/IEEE 21TH INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS (MODELS '18), OCT 2018, COPENHAGEN, DENMARK. PP.334-344, (10.1145/3239372.3239408). <[HTTPS://HAL.ARCHIVES-OUVERTES.FR/HAL-01845976](https://hal.archives-ouvertes.fr/hal-01845976)>. SLIDES AT
<[HTTPS://WWW.SLIDESHARE.NET/HUGOBRUNELIERE/TOWARDS-SCALABLE-MODEL-VIEWS-ON-HETEROGENEOUS-MODEL-RESOURCES-MODELS-2018-COPENHAGEN-DENMARK](https://www.slideshare.net/HUGOBRUNELIERE/TOWARDS-SCALABLE-MODEL-VIEWS-ON-HETEROGENEOUS-MODEL-RESOURCES-MODELS-2018-COPENHAGEN-DENMARK)>
- [11] ROMINA ERAMO, FLORENT MARCHAND DE KERCHOVE, MAXIMILIEN COLANGE, MICHELE TUCCI, JULIEN OUY, HUGO BRUNELIERE, DAVIDE DI RUSCIO. MODEL-DRIVEN DESIGN-RUNTIME INTERACTION IN SAFETY CRITICAL SYSTEM DEVELOPMENT: AN EXPERIENCE REPORT. THE JOURNAL OF OBJECT TECHNOLOGY, 2019, THE 15TH EUROPEAN CONFERENCE ON MODELLING FOUNDATIONS AND APPLICATIONS, 18 (2), PP.1:1-22. (10.5381/jot.2019.18.2.a1). <[HTTPS://HAL.ARCHIVES-OUVERTES.FR/HAL-02170550](https://hal.archives-ouvertes.fr/hal-02170550)>. SLIDES AT
<[HTTPS://WWW.SLIDESHARE.NET/HUGOBRUNELIERE/MODELDRIVEN-DESIGNRUNTIME-INTERACTION-IN-SAFETY-CRITICAL-SYSTEM-DEVELOPMENT-AN-EXPERIENCE-REPORT-JOT-ECMFA-2019-EINDHOVEN-THE-NETHERLANDS](https://www.slideshare.net/HUGOBRUNELIERE/MODELDRIVEN-DESIGNRUNTIME-INTERACTION-IN-SAFETY-CRITICAL-SYSTEM-DEVELOPMENT-AN-EXPERIENCE-REPORT-JOT-ECMFA-2019-EINDHOVEN-THE-NETHERLANDS)>
- [12] DAVIDE ARCELLI, VITTORIO CORTELLESA, DANIELE DI POMPEO, ROMINA ERAMO, MICHELE TUCCI. EXPLOITING ARCHITECTURE/RUNTIME MODEL-DRIVEN TRACEABILITY FOR PERFORMANCE IMPROVEMENT. INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE (ICSA) 2019, PP. 81-90, [HTTPS://DOI.ORG/10.1109/ICSA.2019.00017](https://doi.org/10.1109/ICSA.2019.00017)

Appendix: Hackathons and Tools Fair reports

This appendix reports technical details and results of the hackathon challenges related to the Traceability Tool Set application.

A.1 Prague tools fair

Tool Posters

- Innopolis University: A.Naumchev, M.Khazeev, “Intelligent Formalization, Verification and Validation of Traceable Requirements”,
https://drive.google.com/open?id=1HaaO16o1mET8fTz_LTFsDGnyLKII5Uif

Use Case Posters

- CLEARSY System Engineering, JTL, “The railway case study: Model-Driven Design-Runtime Interaction in Safety Critical System development”,
<https://drive.google.com/open?id=18IAVts7CUU325b99KWPxcHOswel-hYt>

A.2 Prague Hackathon results

A.2.1. CSY Challenge: Log interpretation

CONTEXT

The Copilot system controls Platform Screen Doors in urban railway stations. The system has a safety fallback position, to ensure passenger safety in non-nominal situations. Increasing the availability of the system requires a careful analysis of the events sequences leading to the fallback position.

In the context of previous experiments, we focused on detecting one specific cause of fallback situations: when two sensors report inconsistent values. However, a fallback may also be caused by faulty sensors or other conditions. CSY was interested in detecting root causes behind different types of fallbacks. To detect these other causes in our current solution, we needed to define more general patterns when computing traceability links.

CSY provided log (trace) and a (simplified) specification of the system of one existing version of the Copilot system. Furthermore, CSY provided an informal description of possible fallbacks conditions and alleged causes.

Before the Hackathon, we were able to model the logs and the specifications and to link those models together.

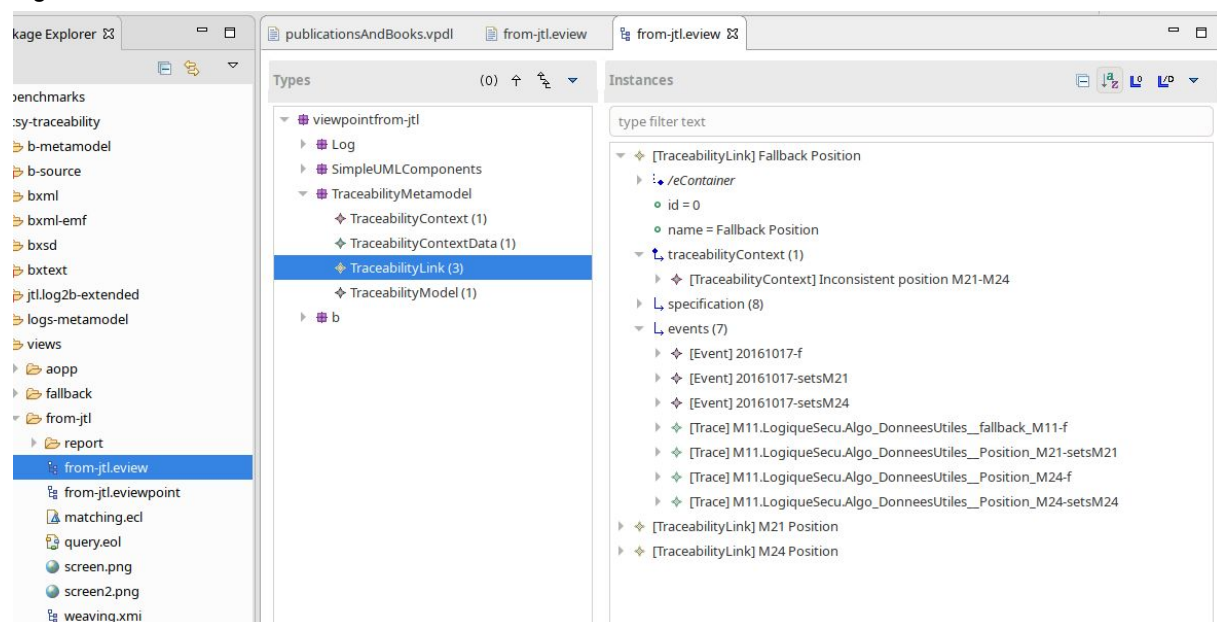


Figure 10 - Example of traceability model

This traceability model was rich and informative but difficult to handle, especially for someone who would not have created the model.

GOAL

- Link traces to the specification and identify those corresponding to different fallback situations.
- Use traceability links to identify root fallbacks causes.
- Format the analysis results into a user friendly form.

RESULTS

During the hackathon, we proposed an html interface to represent the results of the analysis. This interface is interactive, the analyst can click on a list of fallback and observe which events are involved in this fallback. He can also have a view of the original specification that triggered the fallback situation and compare it with the logged values.

Fallbacks	Timestamp	M21	M24
<p>Inconsistent position M21-M24 Difference between M21 and M24 positions: 373mm ▼ B specification source Treatment_i.imp:37-40:</p> <pre>IF train_in_par = INCONS_POSITION THEN fallback := INCONSISTENT_POSITION</pre> <p>Train_i.imp:66-90:</p> <pre>in_par <-- is_train_in_par = VAR dd IN dd <-- abs(Position_M21 - Position_M24); IF dd < POSITION_DELTA THEN VAR par1, par2 IN par1 <-- abs(Position_M21 - PAR_POSITION); par2 <-- abs(Position_M24 - PAR_POSITION); IF par1 < POSITION_DELTA & par2 < POSITION_D THEN in_par := IN_PAR ELSE in_par := NOT_IN_PAR END END ELSE in_par := INCONS_POSITION END END</pre>	231131709	33142	
	231135969	33136	
	231136025		33117
	231136077	33137	
	231136106	33139	33116
	231136181		33117
	231136227		33118
	231136256		
	231136331		33116
	231136377		32766
	231136480		33117
	231150677	33136	
	231150719		33117
	231150774	33138	
	231150828	33139	33115
	231150872	33137	
	231150931	33138	32766

Figure 11 - Example of B specification and list of events representation

In this example, we have one inconsistency on the left, with its B language specification and the list of events on the right. The focus is set on particular values, spread through the logs but connected through the specification.

We also looked at the possibility to send different requests to the model. In particular, we were able to request a series of values from the log and display it as a plot.

A.3 Santander Hackathon results

A.3.1. **BT Challenge: Variability Modelling Using Requirements on Different Design Levels: Using High-level Requirements, HL Design, LL Requirements, and HL Test Specifications**

CONTEXT

The Challenge will focus on requirements related a subsystem taking care of the smoke detection onboard a train. Here is a short description: When a smoke detector senses smoke in metro train passenger area, responses are triggered in a sequence involving up to 11 different subsystems/actors just to name a few: Smoke detection system, TCMS, Audible alarm system, Fire extinguisher system, Train driver. This case gives good possibilities to design such a system using common features in a generic kernel with just a few subsystems and low complexity, and add additional variants (feature packages) on top of that. For example additional packages can be added with extensions to the main common scenario. Alternatively, one can add extensions to the main scenario or add subsystems (e.g., using a state machines and or feature trees). Finally, it will be interesting to add quality attributes e.g. performance/timing, reliability, availability, maintainability, safety.

GOALS

The goal of the challenge is to use and experiment several modelling tools for the purpose of designing a smoke detector sub-system by taking into account the common features and variants.

RESULTS

The BT challenge permitted to apply the different MegaM@Rt2 tools for runtime analysis and combine them. As depicted in the figure below the process applies Modelio (Softeam), CertifyIt and MBeeTle (Smartesting) and CompleteTest (MDH). The system requirements expressed in words are synthesized and formalized in Modelio. On the one hand, from Modelio the requirements are imported semi-automatically into the Smartesting UML/OCL model, to maintain the traceability with the requirements. This model than can be used with both Smartesting tools : CertifyIt and MBeeTle to generate executable test cases. On the other hand, CompleteTest is used for model checking and generation of abstract test cases. These tests are complementary and are used for conformance checking.

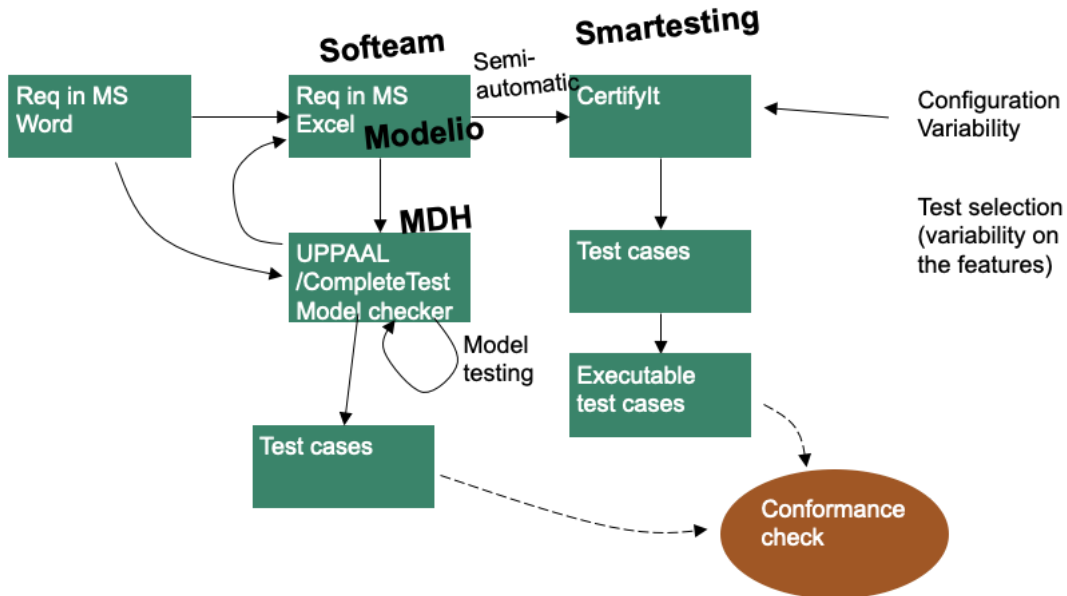


Figure 12 - MegaM@Rt process for BT challenge

Results for Modelio

Modelio has been used to extract the hierarchy in the requirement document from natural language to a model. As shown in the figure below, the REQ id and hierarchy for the smoke detectors are translated and used by the other tools in the toolchain.

ID	[2F-VehicleFuncs-VF-TRS.-C30-REQ-0198 (v.2)]
Description	The HVAC system shall be shut down in the car where smoke is detected.
Safety level	0
Safety argument	Fire hazards, however power supply to heaters is closed down directly by TCMS.
↑	
ID	[2F-VehicleFuncs-VF-TRS.-C30-REQ-0199 (v.2)]
Description	The driver shall have the possibility to close/open the fresh air dampers.
Safety level	0
Safety argument	Comfort function, as smoke density can't be more heavy than acceptable for operation in the tunnel.
↑	
ID	[2F-VehicleFuncs-VF-TRS.-C30-REQ-0200 (v.2)]
Description	OCC shall have the possibility to close/open the fresh air dampers.
Safety level	0
Safety argument	Comfort function, as smoke density can't be more heavy than acceptable for operation in the tunnel.

5.1.2 -> Concept

Smoke detected in the train

Both HVAC in the cab and the HVAC in passenger area will behave the same way in case of smoke.

To avoid smoke to migrate to other cars TCMS will order the HVAC, in the car where smoke is detected, to turn off (off mode) so the air circulation is stopped. Internal doors between cars will be closed by TCMS as well, but that is not part of this concept. (ref[4])

If the train stands at platform (ref[2]) when the fire alarm activated (not reset fire alarm), the HVACs in all cars of the affected train set will be turned off (off mode)

The cab is considered to be a separate part of the car. Smoke detected in the passenger area will not affect the HVAC in the cab. If smoke is detected within the cab the cab HVAC will go to off mode. CPT-0018 (v.2)

5.1.3 -> Derived Requirements

Req-ID	Description	Derived to	SIL
0039 (v.1)	If fire (smoke) is detected in the passenger area, when the train is between platforms, the HVAC in the affected car shall be put in off mode.	4S_09.03.05.-TCMS Software	0
0040 (v.1)	If fire (smoke) is detected in the passenger area, when the train	4S_09.03.05.-TCMS	0

Figure 13 - An example of requirement hierarchy extracted.

Results for CompleteTest

CompleteTest tool has been used for model checking and model testing of timing requirements in the smoke detector by manually modelling these using timed automata and using TCTL properties for generating test cases (as shown in Figure 16).

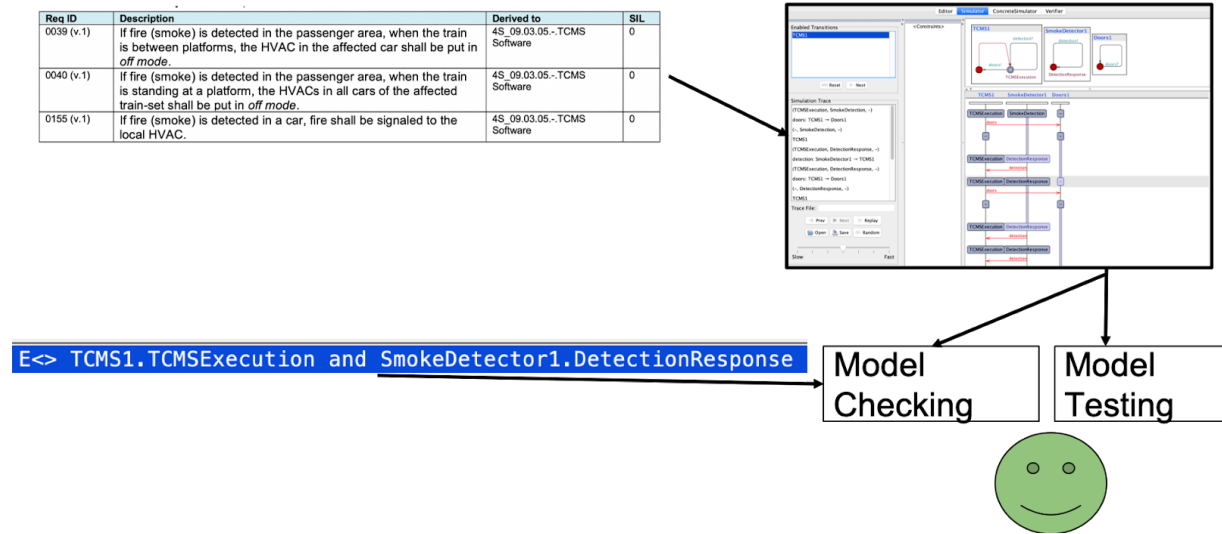


Figure 14 - The model testing process used for model testing natural language requirements.

Results for Smartesting

Based on the given specification a corresponding UML/OCL model was created with the necessary input test data, as depicted in the figure below

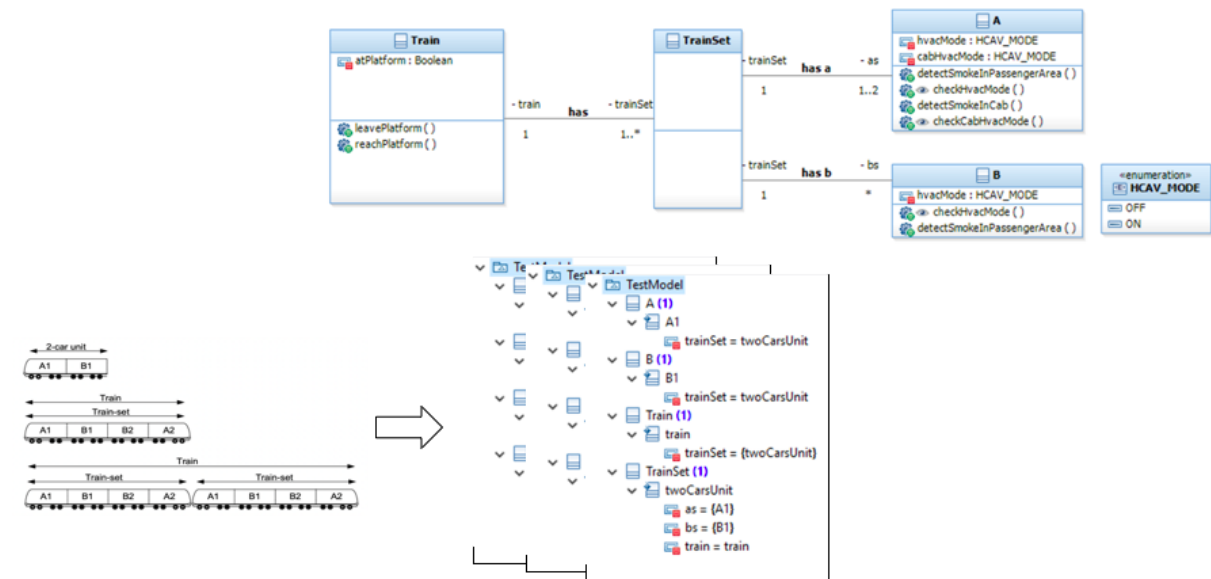


Figure 15 - UML/OCL test model

In this model, we imported the requirements from MODELIO, that ensured bi-directional traceability with the generated test cases, as illustrated in the figure below.

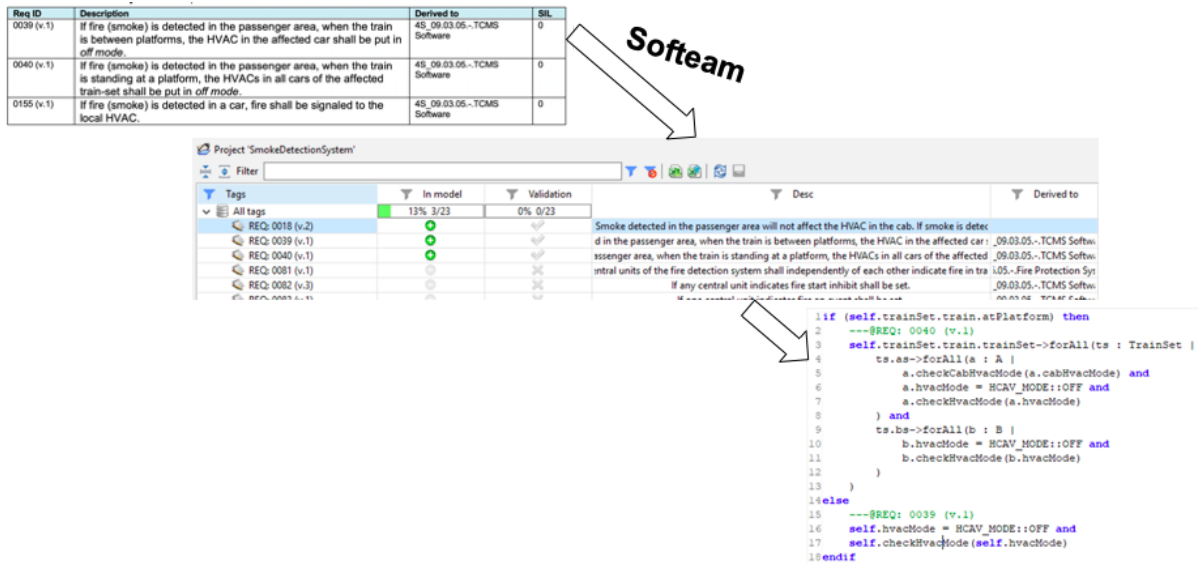


Figure 16 -Traceability of requirements from Modelio to Smartesting

Based on the model as proof of concept the CertifyIt tool was used to generate test cases using off-line generation. The generated tests are executable. Within the hackathon the system under test was not available thus, no adaptation layer was produced. But, as usual, as soon as the adaptation layer is finalized, the tests can be executed on the targeted system. MBeeTle can be used without any additional effort.

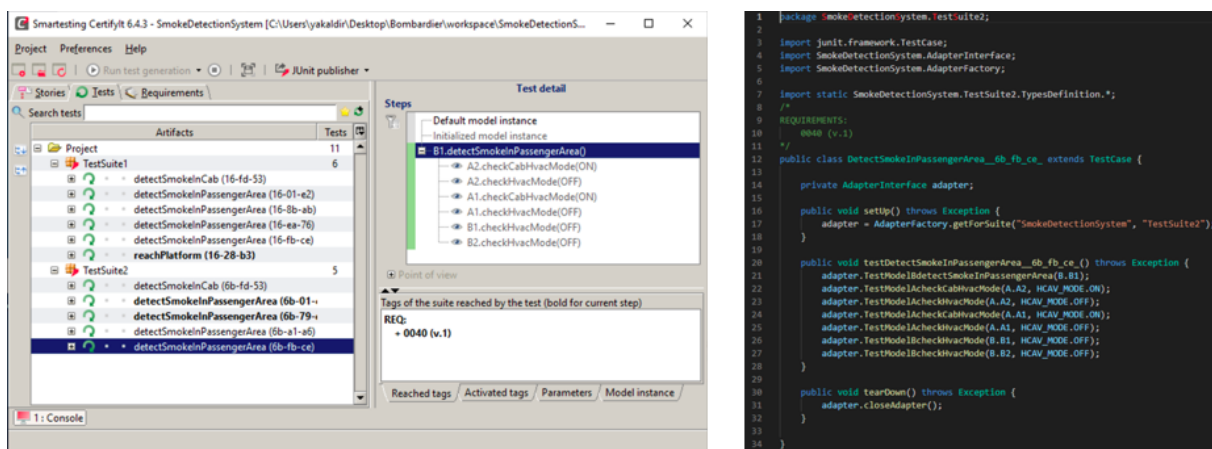


Figure 17 - Smartesting test generation

A.3.2. CSY Challenge: Log and test interpretation

CONTEXT

The Copilot system controls Platform Screen Doors in urban railway stations. The system has a safety fallback position, to ensure passenger safety in non-nominal situations. Increasing the availability of the system requires a careful analysis of the events sequences leading to the fallback position.

In the context of the previous experiment, we focused on detecting one specific cause of fallback situations: when two sensors report inconsistent values. However, a fallback may also be caused by faulty sensors or other conditions. CSY is interested in detecting root causes behind different types of fallbacks. To detect these other causes in our current solution, we would need to define more general patterns when computing traceability links.

During the last Hackathons we were able to detect the fallback cause and report information about fallback and context in a html page.

We also developed the frame of a test environment, using models and simulation. This framework is able to use several models of the environments to produce sets of tests that can cover large combination of specifications.

Now we would like to step back a little. Fallback detected during last experiments might be transient artefacts that appear only one one frame due to sensor or delay on the communications. The real bad condition appears at the sequence level: when a fallback appears on several frames or if there is no good frame to catch up, the system goes into a failsafe mode from which it cannot leave by itself. The information we would like to catch now is the fallback or set of fallback that provoked this failsafe mode.

GOAL

- Link traces to the specification and identify those corresponding to different fallback situations.
- Link the fallback that provoked a failsafe mode
- Automatize tests analyse

RESULTS

Concerning the analysis alone, we were able to examine a new B specification produced by CSY. CSY extended the B specification we used in the paper in order to include more fallback situations. The old specification was roughly 30% of the new one. We went through three new fallback situations that were included in the new B specification. We started to think about how these new fallbacks can be traced to the log and to design models.

Concerning the test environment, the new content produced by CSY (Dynamics of the train, statecharts of the sensors, complete model in external modeling tool) have been studied and are perfectly relevant for modeling the environment and generating test cases using data mutations on train model, sensor model or system model. The combinatorial explosion should be controlled by allowing only two mutations at a time.

Megamart has the tools to produce combinatorial test cases from requirements.

[CSY_2]: From Requirements to Combinatorial Test Case SEAFOX/ACTS

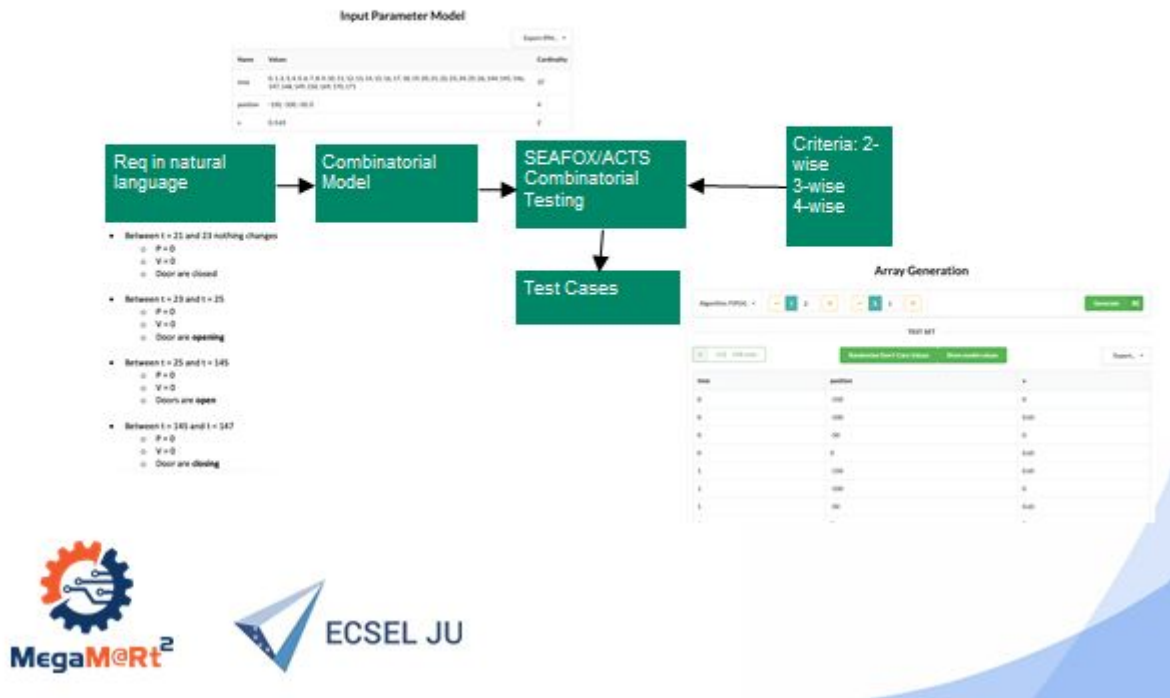


Figure 18 - From Requirements to combinatorial test cases

Finally, the possibility to connect the test generation with the log analysis has been studied, we expect to use the log analysis for testing results of simulation. This would allow for a completely automated test suit, that generates tests, run them and collect the results.