D5.5 Case study evaluation report

Phase 1

MILESTONE:	31 MARCH 2019 (M24)
STATUS:	FINAL, VERSION 1.0
COORDINATOR:	TEK
CONTRIBUTORS:	TRT, CSY, IKER, TEK, NOK, BT, VCE, AINA, CAM, MDH, SICS, BUT
REVIEWERS:	ELIZABETA FOURNERET (SMARTESTING SOLUTIONS & SERVICES), LUCIANO BOZZI (RO TECHNOLOGY)
DISSEMINATION LEVEL:	PU (PUBLIC)
HANDLE:	HTTP://HDL.HANDLE.NET/20.500.12004/1/P/MMART2/D5.5
LAST EDITED:	29 MARCH 2019

ECSEL JU

MegaM@Rt2 Consortium

Participant organization	*Short name*	*Country*
SOFTEAM	SOFT	FR
Thalès RT	TRT	FR
Smartesting Solutions & Services	SMA	FR
ClearSy	CSY	FR
ARMINES — Association Pour La Recherche Et Le Developpement Des Methodes Et Processus Industriels	ARM	FR
Université de Pau et des Pays de l'Adour	UPAU	FR
Atos Spain	ATOS	ES
Universidad De Cantabria	UCAN	ES
Universitat Oberta de Catalunya / ICREA Fundacio Per A La Universitat Oberta De Catalunya	UOC	ES
IKERLAN	IKER	ES
Fent Innovative Software Solutions	FTS	ES
Tekne	TEK	IT
Università Degli Studi Dell'Aquila	UAQ	IT
Intecs	INT	IT
Ro Technology	RO	IT
Åbo Akademi University	ABO	FI
Telia Communication	AINA	FI
Space Systems Finland	SSF	FI
Nokia Solutions and Networks Oy	NOK	FI
VTT	VTT	FI
Conformiq	CON	FI
Bombardier Transportation Sweden AB	BT	SE
Volvo Construction Equipment	VCE	SE
Mälardalen University	MDH	SE
SICS Swedish ICT Västerås	SICS	SE
Brno University of Technology	BUT	CZ
Camea	CAM	CZ

Executive summary

The project — The MegaM@Rt2 Consortium envisions a *Framework* (an integrated suite of technologies, methods, and tools) that supports the *Model-Driven Engineering* (MDE) for "*enhancing productivity while reducing costs and ensuring quality in development, integration and maintenance of complex software systems*". The overall project goal is to provide and validate the Framework.

The Consortium includes partners with the role of *technology providers* and other ones with the role of *case study providers*. The former are universities and research centers, as well as companies that offer software development solutions. The latter are small, midsize, and large enterprises, for which software is an essential component of their products, if not the product itself.

In the MegaM@Rt2 project, the technology providers start from the technical solutions that they already have (*baseline version*) and arrive at Framework that integrates the final versions of the modeling tools. The organization of these activities in three work packages (WP) follows the broad division of the tools in three areas of application: *modeling at design time* (WP2), *modeling at runtime* (WP3), and *model management and traceability* (WP4).

The evaluation of the project results is *case studies* based. In the work package WP1, the case study providers propose and define the systems that they design, implement, and verify using the Framework during the project prosecution; then they detail and plan these activities and the project evaluation and organize them in *development and validation scenarios*; moreover, they express the requirements that as users expect the Framework to satisfy.[1]

Both technology providers and case study providers participate in the work package WP5. In the context of the task T5.1, the former integrate the results of WP2, WP3, and WP4 in the Framework. The latter design, implement, and verify the case studies in T5.2, while in T5.3 validate the framework and evaluate the project outcome.

The project adopts an iterative and incremental approach. WP2, WP3, WP4, and T5.1 provide the initial, intermediate, and final versions of the tools and of the integrated Framework. T5.2 and T5.3 evolve along two phases: the preliminary development, validation and evaluation activities take place during the Phase 1 that ends at month 24 of the project, the final ones during the Phase 2 that ends together with the project at month 36.

This document — This deliverable D5.5 reports on the task T5.3 at the end of Phase 1. Firstly, it recalls and, in the case changes occurred, updates the definition of the case studies, of the scenarios, and of the Key Performance Indicators (KPI). Quality enhancement, as well as cost and time reduction are quantified by a set of KPI whose values are related to the effort spent for the activities (design, implementation, verification) in the scope of each scenario. Secondly, it describes such activities: the use of the tools, the interactions between the case study providers and the tool providers, the development status. Finally, it accounts for the Phase 1 results: report the values of the KPI preliminarily measured, gives feedback and recommendations to the technology providers about the aspects of the tools that could be improved, plans the remaining activities to be carried out during the Phase 2.

There is a conclusion section for each of the nine case studies and globally for the document. During Phase 1, cases study providers made some changes in the choice of the best scenario to validate this or that tool and, conversely, in the judgement of the best way for using a tool in a certain scenario; many problems were encountered but solved thanks to the collaboration that the tool providers fully offered. Although it is likely that similar issues will be encountered in the Phase 2, the Consortium is

[1] It is worth mentioning that in WP1 the technology providers define the high-level architecture of the Framework in the task T1.4, and that the case study providers carry out a preliminary development using the baseline tools (those available when MegaM@Rt2 starts) in the task T1.3.

confident of reaching the objectives it set, because good KPI values were preliminarily obtained during the Phase 1 and the project is substantially in time with the plan.

Phase 2: the next and last year of the case studies — This document closes the Phase 1. The case studies will be completed during the Phase 2. In iterative incremental manner, the case study providers will finish to design, implement, and verify the systems. The modeling tools in their final version and integrated in the final version of the Framework will be used for the development, and they will be validated scenario by scenario. The definitive values of all KPI will be measured. At the end of the project (M36), the deliverable D5.7 “Case study evaluation report — Phase 2” will give the final validation of the Framework with focus on its industrial applicability, and the final evaluation of the MegaM@Rt2 project outcome.

Table of contents

Acronyms

API	Application Programming Interface
BB	Baseband
BTS	Base Transceiver Station
CAF	Common Architecture Framework
CI	Continuous Integration
CNN	Convolution Neural Network
CSV	Comma Separated Values
DAL	Design Assurance Level
DDS	Data Distribution Service
EATS	Engine After-Treatment System
EMF	Eclipse Modeling Framework
ETM	Embedded Trace Macrocell
FDE	Feature Development Efficiency
FMS	Flight Management System
GW	Gateway
HW	Hardware
IDL	Interface Definition Language
IR	Infrared Radiation
JVM	Java Virtual Machine
KPI	Key Performance Indicator
L2	Layer 2 (the second layer of the IP communication stack)
LP	License Plate
MAC	Medium Access Control
MB	Main Branch
MBSE	Model Based Systems Engineering
MBT	Model Based Testing
MDE	Model Driven Engineering
MQTT	Message Queue Telemetry Transport)
OCL	Object Constraint Language
OCR	Optical Character Recognition
PC	Personal Computer
PCB	Printed Circuit Board
PS	(Nokia) Platform SW
QoS	Quality of Service
R&D	Research and Development
REST	Representational State Transfer

SoC	System on Chip
SMS	Short Message Service
SUT	System Under Test
SW	Software
TCMS	Train Control and Management System
UDS	Urea Dosing System
UML	Unified Modeling Language
UWB	Ultra-Wideband
WoW	Way of Working
WP	Work Package
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

1. Introduction

This document is the deliverable result of the task T5.3 “Case study execution and validation” that belongs to the work package WP5 “Integration, Case Study Development and Evaluation” of the ECSEL project *MegaM@Rt2*.

The integrated suite of model based technologies, tools, and processes at which MegaM@Rt2 aims to arrive is the *Framework*. This document reports the results of the preliminary Framework *verification* and *validation*, obtained during the Phase 1 that ends at the month M24 of the project. (The Phase 2, which ends at M36 together with the project, provides the final results.)

The *tool provider* partners participating in the project develop the Framework (starting from solutions that they already have) in the work packages WP2, WP3, WP4, and in the task T5.1.[2]

The *case study providers* have the role of end-users of the Framework, which they verify and validate through the *case study* method.

The case studies consist in experimenting the Framework by using it to develop (design, implementation, and verification) certain *systems* (that is what the case study providers do in the task T5.2). The experimentation results enables also the *evaluation* of the MegaM@Rt2 outcome (and that is what the case study providers do in the task T5.3).

The case studies are structured in *scenarios*. These express the system development aspects that are the most important ones from the point of view of what the case study providers need for their industrial activities, and of which are the benefits that MegaM@Rt2 is expected to bring. The system development as well as the Framework validation are organized around them.

In the deliverable D1.1 (work package WP1), the case study providers defined the scenarios and, on their basis, gave the *user requirements*. These were analyzed by the tool providers that defined the Framework requirements[3] and the requirements of the individual tools[4]. This document closes the loop by reporting about the requirements verification[5].

This deliverable recalls the scenarios definitions and updates them in the case changes occurred.

The Full Project Proposal (FPP) document specifies the broad classes of *Key Performance Indicators* KPI (Table 1.2) for measuring the quality enhancement, as well as cost and time reduction that the Framework brings, in order to validate it with respect its industrial applicability. The FPP states that *“these indicators will be refined and updated in the course of the project in order to reflect the detailed requirements and the project environment”*. Accordingly, the deliverable D5.4 (task T5.2) defined the KPI.

This deliverable reports the KPI definitions and updates them in the case changes occurred.

---

[2] The organization of the work packages follows the broad division of the tools in three areas of application: *modeling at design time* (WP2), *modeling at runtime* (WP3), and *model management and traceability* (WP4); the task T5.1 deals with the integration of the tools in the Framework.

[3] WP2 defined the design time requirements (whose identifiers start with the string “SYS” that stays for *System Engineering*), WP3 defined the runtime requirements (identifiers: “RTA” for *Runtime Analysis*), and WP4 defined the megamodeling requirements (identifiers: “MTM” for *Model & Traceability Management*).

[4] The identifiers of the tools requirements start with the acronym of the tool; they can be easily distinguished from the system requirements (see the previous note) and the user requirements that start with the acronym of the case study provider (see the table “MegaM@Rt2 Consortium”).

[5] The relations (mapping and traceability) between the requirements of the three sets (user, Framework, and tools) are given in the deliverables D2.2, D3.2, and D4.2 (respectively).

The KPI definition describes how to collect data by experimenting the Framework in each scenario for obtaining the KPI values: V_1 in Phase 1 and/or V_2 in Phase 2. Moreover, it states what follows:

- The phase when the indicator is measured, preliminarily in Phase 1 and/or finally in Phase 2.
- The choice of the reference value V_{REF}, against which to compare V_1 and V_2 for obtaining the improvement given by $\Delta_i\% = 100 \cdot (V_i - V_{REF}) / V_{REF}$:
  - *Comparison with the state of the practice*: $V_{REF} = V_0$, where V_0 is the pre-MegaM@Rt2 value of a parameter (e.g. work, cost, number of defects) of certain activities in a typical and comparable project, which is already known to the partner that develops the case study.
  - *Clean experiment*: $V_{REF} = V_B$, where is V_B is the parameter evaluated when the case study was preliminary developed during the Task T1.3 by using the baseline version of the tools (those available at the beginning of MegaM@Rt2).
- The total improvement $\Delta_T\%$ that the partner foresees for the Phase 2.

Document organization — For each case study there is an *individual chapter* whose sections have the following common structure:

- The § "Case study overview" recalls the system that the partner develops using the *Framework*, as well as the scenarios and KPI definitions.
- The § "Development and validation activities" reports the work that the partner did during the Phase 1 and that allowed the KPI evaluation, the interactions with the tool providers, and the development status.
- The § "Results of the first evaluation phase" regards the KPI values obtained during the Phase 1 and the plan for improvements. The latter is given in terms of Framework requirements that the case study provider expressed in WP1, the state of the art that the partner perceives and feedbacks to the tool providers, and the activities planned for the Phase 2.

The chapter "Conclusions" closes the document.

1.1. Case study and KPI lists

The case studies are listed in the Table 1.1 and the KPI, in the Table 1.2.

Table 1.1: Case studies

§	*Acronym*	*Technological domain*	*Application specific*	*Partner*
2	01_TRT	Avionics	Flight Management System	TRT
3	02_CSY	Railway	Platform screen doors control	CSY
4	03_IKER	Smart warehouse	Deployment and supervision of agents	IKER
5	04_TEK	Short range communications	Indoor positioning	TEK
6	05_NOK	Telecom	Base Transceiver Station	NOK
7	06_BT	Transportation	Train Control and Management System	BT
8	07_VCE	Automotive	Engine Control	VCE
9	08_AINA	ICT Services	SMS Gateway	AINA
10	09_CAM	Traffic Monitoring	Intelligent Traffic Surveillance System	CAM

The Tables 1.2 lists the KPI with their definitions and target ranges given by the FPP.

Table 1.2: Key Performance Indicators

KPI	*Definition*	*Range*	
		Min	*Max*
KPI_1.1	Reduction of design time/design effort by design artefacts reuse.	10%	50%
KPI_1.2	Improvement of the time required for identification of design problems.	10%	30%
KPI_1.3	Reduction of time/efforts for requirements validation.	10%	50%
KPI_1.4	Improvement in the predictability of system deviations comparing to reference projects (to be determined in empirical evaluation).	10%	30%
KPI_2.1	Reduction of validation effort by automated trace collection and analysis.	10%	30%
KPI_2.2	Reduction of time for monitors setup by automated probes injection.	50%	80%
KPI_3.1	Reduction in time/effort required for managing and handling all the involved models (e.g. time for model retrieval and access).	10%	40%
KPI_3.2	Improvement in number of linked models to be actually managed and handled in existing engineering practices (e.g. both system and NFRs models).	10%	30%
KPI_3.3	Reduction of integration effort for the modelled system artefacts.	10%	50%
KPI_4.1	Reduction in time/effort required for tracing and handling all the involved models at design and runtime levels (e.g. creation of and access to relations between system and traces models).	10%	50%
KPI_4.2	Improvement in number of models to be actually traced and handled in existing engineering practices.	10%	30%
KPI_5.1	Productivity improvement in requirement validation.	10%	50%
KPI_5.2	Quality Improvement by improving predictability and conformance to specifications.	10%	30%
KPI_5.3	Gains in productivity compared to the baseline development process that does not include megamodelling with continuous development.	10%	30%
KPI_5.4	Costs savings for development and maintenance of large complex systems in European companies.	5%	10%

Note: KPI_5.3 and KPI_5.4 don't have specific numbers in the Full Project Proposal (see § B2.2.1 "Impact objectives and management", Table 2.3 "Performance indicators for MegaM@Rt2 impact objectives"). In this document we numbered them for ease of reference.

2. Case study 01_TRT – Flight Management System

2.1. Case study overview

2.1.1. System description

The avionics use case provided by Thales TRT is a safety-critical application: The Flight Management System (FMS). The purpose of the FMS in modern avionics is to provide the crew with centralized control for the aircraft navigation sensors, computer based flight planning, fuel management, radio navigation management, and geographical situation information. Taking charge of a wide variety of in-flight tasks, the FMS allows us to reduce the workload of the flight crew. The FMS is especially responsible for the flight management services allowing in-flight guidance of the plane. From pre-set flight plans (take-off airport to landing airport), the FMS is responsible for the plane localization, the trajectory computation allowing the plane to follow the flight plan, and the reaction to pilot directives. The FMS is an industry level application characterized as a safety critical application typically DAL B or C (Design Assurance Level), dealing with different kinds of real-time requirements.

The FMS is decomposed into multiple tasks. It computes various data (i.e., exact location, trajectories, and nearest airports list among many others) and it is supposed to send guidance instructions to the autopilot and to send the different results to a display.

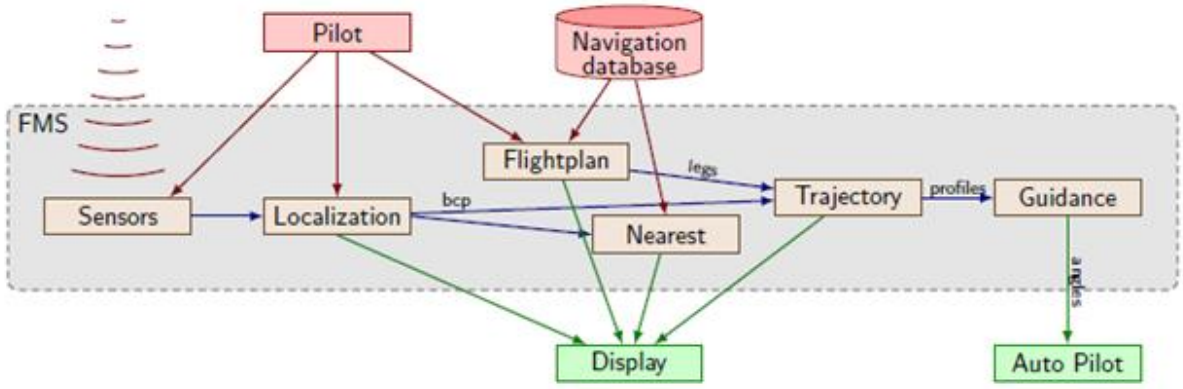

Figure 2.1: Functional description of the FMS application

The FMS inputs consist of:

1) Sensor data: The FMS receives data from different sensors installed in a plane like: the GPS, inertia sensors, etc. The data from these sensors and state data conserved in the FMS is used to estimate an accurate position (including latitude, longitude and altitude) of the plane.
2) Pilot commands: The pilot can configure most of the tasks in the FMS. For that purpose, the FMS is able to receive configuration commands.
3) Navigation database: The FMS includes a read-only Navigation Database that contains the elements from which the flight plans are constructed, such as: airport locations, runway locations, departure procedures, airways (similar to sky-level highways), arrival procedures.

Likewise, the FMS outputs consist of:

1) The display: All the data computed by the FMS could be displayed on a console or through lights/LEDs on panels. Each FMS task can send the data independently of the other tasks to the display.

2) The autopilot: The role of the autopilot is to translate the attitude output information from the FMS (delta to be applied to the airplane roll, pitch and yaw angles) to actions to be issued by the aircraft actuators like aps, slats, spoilers and the rudder.

2.1.2. Development and validation scenarios

The global objective of Thales from MegaM@Rt2 is improve its performance design practices by linking the design with the runtime aspects (performance data collected at runtime). The main expectations of Thales from the MegaM@Rt2 framework are the following:

1) To be able to select the most suitable execution platform for Thales applications (based on an efficient technique that collects the timing characteristics of the application corresponding to each platform).
2) To be able to efficiently deploy the Thales application on the selected execution platform.
3) To perform the verification of the timing behaviour of the Thales applications using simulation or analysis techniques.
4) To allow fine grain modelling and verification of the Thales application which will lead to tighter estimated performance bounds and consequently less over dimensioning of the needed execution resources to run the application.

Following scenarios were defined in the Thales case study in order to fulfill the above mentioned expectations.

2.1.2.1. Scenario № 1: S3D modeling methodology

The scenario 1 aims at collecting feedback from the execution/simulation of the FMS in order to annotate the FMS design model with timing characteristics that depend on the considered execution platform. The timing characteristics include the execution times, the activation periods, the jitters, the overheads, etc. For each considered architectural mapping, the obtained annotated model is used to estimate the performance of the various FMS task groups using the performance verification techniques as described in the scenario 2. This will allow the selection of the most appropriate execution platform for the execution of the FMS application.

S3D, the Single-Source System Design Framework (http://umlmarte.teisa.unican.es/) developed by the University of Cantabria towards the mega-modeling of complex distributed systems, is used to fulfill the objectives of scenario 1. From a CAPELLA model of The FMS application, we aim at generating an S3D generic, platform-independent, reusable model. Then the next step is to automatically generate platform independent code. Then, we have to validate that S3D is able to capture the concepts represented in the CAPELLA model of the FMS case study. Then, we must identify the mapping between the CAPELLA concepts and the corresponding concepts in S3D. This mapping will be used to specify the automatic transformation from the CAPELLA model to the S3D model. From the S3D model, the next step is to explore different architectural mappings. For each one, the corresponding simulation model will be automatically generated. The simulation model includes the performance impact of the architectural mapping decided, thus supporting the performance analysis. In this way, it will be possible to compare different solutions and select the most appropriate in a short time, thus reducing design and verification efforts. The model will support the system test generation and verification as well.

2.1.2.2. Scenario № 2: Performance modelling and verification

Starting from the design model of the application (in our case the FMS model in the modelling tool CAPELLA), the designer must be able to generate a valid model that can be interpreted by the

performance verification tools (simulation tools, scheduling analysis tools). This may require defining a set of transformation rules to fill the semantic gap between the design model (e.g. in CAPELLA) and the performance verification model. If a modification is made in the design model, a new corresponding performance model has to be automatically generated. Once a valid performance verification model is generated, the designer must be able to run an appropriate performance verification technique to validate the performance of each relevant functional scenario in the design model. The performance verification shall be able to generate human understandable graphics and data automatically (e.g. representing the worst case response time and worst case latencies through Gantt charts) to assist the designer to understand timing errors.

For scenario 2, we use Xamber, the configuration and analysis tool for partitioned systems developed by Fentiss. As far as the temporal model is concerned, it uses a model based on partitions, tasks, end-to-end flows and mutual exclusion resources. The objective here is to add to Xamber more flexibility and support for other temporal behaviour concepts, not only partitioned systems. Extending Xamber with these concepts will allow to open it to other domains. Right now, Xamber is only used for scheduling and resources assignment for partitioned systems in aerospace domain. After the experiment in this case study scenario it will be possible to use Xamber for systems in the avionics domain.

2.1.3. KPI definition

The definitions of the KPI for the case study 01_TRT are reported in the Table 2.1; the column Δ_T% gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 2.1: KPI definition (case study 01_TRT)

S	*KPI*	*Definition*	*U*	V_0	Δ_T%	*P1*	*P2*
1	KPI_1.3	Reduction of time/efforts for requirements validation in the range of 10%-50%	D	5	50%	✓	✓
1	KPI_5.1	Productivity improvement in requirement validation	€	60	30%		✓
1	KPI_5.4	Costs savings for development and maintenance of large complex systems in European companies	D	30	5-10%		✓
2	KPI_1.3	Reduction of time/efforts for requirements validation in the range of 10%-50%	D	5	50%	✓	✓
2	KPI_5.1	Productivity improvement in requirement validation	D	10	30%	✓	✓
2	KPI_5.4	Costs savings for development and maintenance of large complex systems in European companies	D	30	5-10%		✓

S = Scenario number — *Unit: D = Day, H = Hour*
P1 = Phase 1, P2 = Phase 2 — *V_0 = Estimated pre-MegaM@Rt2 value*
Δ_T% = Foreseen Phase 2 improvement — *V_B = Value obtained in MegaM@Rt2 with baseline tools*

Note that after Phase 1 we were not able to measure all KPIs. This was mainly due to the fact that the S3D and Xamber tools are not yet connected to CAPELLA, i.e. there is a considerable amount of time and effort that needs to be spent in order to model the FMS application in these tools. We expect to be able to perform the measurements during the second experimentation phase. In addition, in scenario 1 we did not yet start the experimentation related to the selection of the most appropriate architectural platform for the FMS application using S3D (KPI_5.1).

Note also that the negative value for KPI_1.3 in scenario 1 (Table 2.2) is due to the fact that the 3D methodology is not yet integrated with the CAPELLA methodology (Arcadia). This increases even more the effort to map the CAPELLA concepts in S3D. But we believe that once the integration is

done, we will be able to benefit from the capacities of S3D and measure a positive value for KPI_1.3. This is also planned for the second phase.

2.2. Development and validation activities

2.2.1. Previous work

Scenario 1 — First, from the CAPELLA model of the FMS application, Thales has developed a Linux and a PikeOS C++ implementation. Then, during the baseline experiments, we modeled manually in S3D few components from the CAPELLA model and the application code. Then we were able to automatically generate in S3D platform independent code corresponding to these few components. After this step, we were able to validate that S3D is able to capture the concepts represented in the CAPELLA model of the FMS case study.

Scenario 2 — After running the baseline experiments for scenario 2 of the FMS case study, we noticed that some concepts in the FMS model cannot be expressed in the Xamber temporal model. We identified therefore the need to adapt/convert these concepts into a valid temporal model for Xamber.

2.2.2. Development and validation activities

Scenario 1 — In the phase 1 of the experimentation, based on the CAPELLA model of the FMS and the corresponding implemented Linux code, we have manually modeled the entire FMS application in S3D. The manual modelling of the application in S3D required a lot of time and effort. This proofs that the integration of the S3D methodology with the CAPELLA methodology (Arcadia) is required to reduce the time and effort and benefit from the capacities brought by the use of S3D.

Scenario 2 — In the phase 2 of the experimentation, we worked on the extension of the Time4Sys pivot model and the Xamber timing verification tool with the missing concepts needed to model the FMS application. These concepts were already identified during the baseline experiments. This step allowed us to run transformation rules on the FMS model in Time4Sys to fill the semantic gaps between CAPELLA and Xamber. Then we modeled manually the FMS application in Xamber according to the output of Time4Sys. Then we run the simulation in Xamber and we were able to produce Gantts illustrating the latencies.

2.3. Results of the first evaluation phase

2.3.1. KPI values

The Table 2.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 01_TRT. Note also that the negative value for KPI_1.3 in scenario 1 (Table 2.2) is due to the fact that the 3D methodology is not yet integrated with the CAPELLA methodology (Arcadia). This increases even more the effort to map the CAPELLA concepts in S3D. But we believe that once the integration is done, we will be able to benefit from the capacities of S3D and measure a positive value for KPI_1.3. This is also planned for the second phase.

Table 2.2: Phase 1 KPI values (case study 01_TRT)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_1.3	D	5	10	-100%	50%
2	KPI_1.3	D	5	3	40%	50%
2	KPI_5.1	D	10	7	30%	30%

S = Scenario number
Unit: D = Day, H = Hour
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
V_0 *= Estimated pre-MegaM@Rt2 value*
$\Delta_B\% = 100 \cdot (V_1 - V_0)/V_0$ (baseline improvement)
V_B *= Value obtained with MegaM@Rt2 baseline tools*
$\Delta_1\% = 100 \cdot (V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)
V_1 *= Phase 1 value obtained with Framework initial version*

2.3.2. Plan of improvements

2.3.2.1. Requirement coverage

The table below includes the requirements defined by Thales for the FMS use-case.

Table 2.3: Thales requirements for the FMS case study

ID	*TRT Requirement*
TRT_01	From the parsing and the analysis of the execution logs of the application, it should be possible to extract various timing properties of the application such as activation periods, maximum jitters, worst-case execution times, overheads, etc. (scenario 1)
TRT_02	The model of the application should be annotated with the obtained timing properties. (scenario 1)
TRT_05	The obtained timing performance results should be integrated in the model of the application. (scenario 1)
TRT_03	From the annotated model of the application, model-based timing performance verification should be automatically performed to assess the overall timing performance after integration of the different task groups (by comparing timing latencies and response times to deadlines). (scenario 2)
TRT_04	The timing performance verification should be able to generate human understandable graphics and data automatically to assist the architect to understand design errors. (scenario 2)
TRT_05	The obtained timing performance results should be integrated in the model of the application. (scenario 2)

2.3.2.2. Feedback to tool providers

Unican (S3D framework): Although the approach is useful for the selection of the most suitable execution platform to execute the FMS application, without the integration of the S3D methodology with the CAPELLA methodology (Arcadia) the effort to build the model in S3D is too costly and time consuming, which will negatively impact the KPI_1.3.

Fentiss (Xamber tool): The obtained results show a clear improvement compared to the current practices at Thales. An even greater productivity improvement could be obtained with a seamless connection between Xamber and the design model via Time4Sys.

2.3.2.3. Work planned for the Phase 2

Work planned with Unican in phase 2 (scenario 1): We plan to generate code for various execution platforms from the S3D model of the FMS application. From the code, we will collect the timing

characteristics of the application, which will allow to select the most appropriate platform. We also plan to study and develop the integration of S3D with CAPELLA since this is essential for the applicability of the approach.

Work planned with Fentiss in phase 2 (scenario 2): We plan to connect CAPELLA to Xamber via the open source pivot model Time4Sys. This will allow a seamless connection from the design model (CAPELLA) to the timing performance verification tool (Xamber).

2.3.3. Summary of the progress towards the roadmap

The first phase of the experimentation allowed us to already obtain very promising results, that will need to be confirmed in the second phase of the experimentation.

In scenario 1, we were able to manually model the entire FMS application in S3D. We were also able to generate code for few components in S3D. We plan in the next phase to generate code from the S3D model of the FMS application for various execution platforms. From the code, we will collect the timing characteristics of the application, which will allow to select the most appropriate platform. We also plan in the second phase to study and develop the integration of S3D with CAPELLA since this is essential for the applicability of the approach (the manual modelling takes a lot of time and effort).

In scenario 2, we were able to fill the semantic gap between the timing behavior of the FMS model in CAPELLA and the timing behavior in Xamber models. This was done using the pivot model Time4Sys. We have also extended the capacities of Time4Sys and Xamber to be able model the entire FMS application. Then we have run the simulation in Xamber and obtained graphical and numerical results regarding the latencies of the functional chains in the FMS application. In the second phase of the experimentation, we plan to connect CAPELLA to Xamber via Time4sys in order to avoid manual modelling in Xamber.

3. Case study 02_CSY – Platform screen doors control

3.1. Case study overview

3.1.1. System description

The CSY Coppilot system manages a metro, tram or train station. It uses a pair of Lidars at each extremity of the platform to measure the position and speed of trains entering the station and one Lidar over each door of the vehicle that detects the movement of the doors. On top of the lidars, several computation units are dispatched, one for each lidar that runs image processing algorithms and one central that calculates the safety critical outputs, i.e. opening authorization for each platform doors.

3.1.2. Development and validation scenarios

The goal of CSY is to improve the availability of the Coppilot System. Feedback from operational instances shows a higher than expected number of fallback situations. A fallback typically occurs when the system cannot make a safe decision because its input data (from the sensors) is inconsistent or missing.

The case study is structured according the scenarios defined below.

3.1.2.1. Scenario № 1: Model the Coppilot System network

The components of the Coppilot System are modelled in the B language. For use in the MegaM@Rt2 Framework it needs to be transformed. Two possibilities are studied.

First, modelling the whole Coppilot system in a standard exchange language such as statecharts.

A DSL is designed to represent the kind of messages or information that transits on the network. Then, each of the components is modelled as a state machine that periodically products messages or reacts to incoming messages by producing other messages. The network is modelled as simple buffers between components.

Second, integrating the original B model of the safety core of Coppilot in EMF, the Eclipse Modeling Framework. This solution focuses on the safety core, which original B model is used by CSY to generate executable code of the core automaton of the system. It uses the B-xml representation of B and its metamodel to generate an EMF model conform to the original.

3.1.2.2. Scenario № 2: Model Simulation

The Coppilot System as statecharts (first solution of scenario 1) can be simulated using the PauWare engine. With this scenario, we are able to get an estimation of the quantity of information that transit on the different networks in order to dimension it. By comparing it to existing systems or future projects it's possible to detect bottlenecks and flaws in their design. Using simple FIFO models of the network gives a good first information on the required bandwidth of the network but more could be obtained with richer network information.

The Coppilot safety core as B-model can be used to generate an executable (B to C transformation). This executable is similar to the one embedded in the final system, but in MegaM@Rt2 it will be run on a computer in simulations. This simulation is fed with inputs from collected logs (traces), it can run faster than the embedded system which allows running several versions of the model and provide new logs for safety analysis.

3.1.2.3. Scenario № 3: Knowledge Extraction from Logs

Scenario 3 is about knowledge extraction. The goal is to automatically learn the real system as a timed hidden Markov model from the logs. Such model would capture both the timed characteristics of the train (e.g. typical duration for doors opening/closing, and typical durations for the train entering the station to halt), and the rate of misdetection from sensors, dependent of the position of the train and the status of the doors.

3.1.3. KPI definition

The definitions of the KPI for the case study 02_CSY are reported in the Table 3.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 3.1: KPI definition (case study 02_CSY)

S	*KPI*	*Definition*	*U*	V_0	V_B	$\Delta_T\%$	*P1*	*P2*
1	KPI_1.1	Estimated time to develop a new system prototype of size comparable to Coppilot-m	D	80	70	25%	✓	✓
1	KPI_1.2	Estimated time to run system analysis	H	70	65	14%	✓	✓
2	KPI_1.2	Estimated time to run system analysis	H	210		14%	✓	✓
2	KPI_2.1	Estimated effort spent on Log Collect and analysis	H	280		20%	✓	✓

S = Scenario number — *Unit: D = Day, H = Hour*
P1 = Phase 1, P2 = Phase 2 — *V_0 = Estimated pre-MegaM@Rt2 value*
$\Delta_T\%$ = Foreseen Phase 2 improvement — *V_B = Value obtained in MegaM@Rt2 with baseline tools*

3.2. Development and validation activities

3.2.1. Previous work

Using Baseline tools, we were able to represent the kind of information that transits on each network of the system. By designing different versions of the components, we prepared for the different versions of the Coppilot system.

This model suited our use-case scenario, especially the possibility that we have to design modularity in the systems, with different implementations of the components for different versions of the system. It lacked the possibility to precisely describe the network component (scenario 1).

Preliminary analysis indicated that network overloads could delay the delivery of sensors data and was identified as a possible cause of fallbacks. CSY thus proposed to model the network of the system, hoping to better understand the dynamics of network loads. Soon after this scenario was proposed, further analysis at CSY showed that network loads was not a significant cause of fallback situations (scenario 2).

The scenario thus evolved to model the internal state dynamics of the system, in order to understand how to make it more resilient to sensor inconsistencies, while of course preserving the existing level of safety.

In parallel, knowledge extraction techniques have been used on the logs, using publicly available python libraries of the state of the art of the domain. Many experimentations have been done in many directions, building experience at CSY. The results showed that hidden Markov models were the more promising representation of the logs (scenario 3).

3.2.2. Development and validation activities

Modelling during phase 1 has been focused on exploiting the models that CSY already possessed: the B-model representation of the safety core. An effort has been made to translate the B-model to EMF. The classic B model is contained in a textual file with its grammar and syntax. This model is dedicated to parsing for compilation and the proof obligation generation. Recently, an XML-based representation of the B models has been created. This representation, defined by the B-XML metamodel, enriches the tree structure of the grammar to an object-based structure and recreates bonds between those objects (such as type, attributes, visibility).

For CSY, this B-XML representation was primarily used to feed a chain of compilation. In MegaM@Rt2, CSY used this B- XML model for representation in Eclipse. CSY used EMF View tool to display models of the system and studied the possibility of editing directly the EMF representation of the model. It has been demonstrated that the EMF representation of the B-model could be enriched, in our case with information about fallback situations.

During phase 1, a simplified model of the Coppilot safety core has been created, this model has all the safety features required by the industry standards and can use the same sensors as the original one. The differences are in the details of information contained in messages between the sensors and the safety core and in the fine tuning of this core. As a result, the simplified model is as safe as the original one but lacks in availability. Some safe situations that lead to the correct behaviour of the original system create fallbacks in the simplified system.

This new model has been written in B and translated to EMF, then the EMF model has been shared to the safety team. The benefits of the graphical view (using EMF View) for the developer are null, the developer can read and understand the B syntax and has no difficulty to navigate between the different files (11 for the simplified model) he will still prefer to use B the legacy editor. For the safety team, the problem is different. The safety principles usually have been defined by a committee formed by experts from CSY and the client. Those experts do not have the same understanding of the B language, some of them may require a formation prior to those meetings. Using a graphical view that highlight the links between components brings a comfort, it separates the principles of safety from the understanding of the language.

During phase 1 experiments, returns from the safety members at CSY (no client was involved) showed that using EMF View could save time in safety analysis. We estimated that 2 days of formation could be saved in the whole development time of the system (KPI_1.1) as well as 7 hours of meeting for safety analysis (KPI_1.2).

Simulation during phase 1 has been done using the executable created from the B model using partial logs as inputs. The benefits of MegaM@Rt2 has been found in the modelling of the output logs and their integration in EMF with the B-XML model. Using JTL[6], we managed to create connections between requirements in the B-XML model and events in the log model.

Position consistency example: One example of safety analysis that benefits from this log modelling is the consistency fallback analysis. In text logs like the one CSY had before MegaM@Rt2, the developer had to search a pattern in gigabytes of text to localise the events in which a fallback situation happened. In the following listing, we see that this event occurs at time `20161017_231136480`. Then he needs to look in the previous events to identify the last known positions sent by sensors M21 and M24. The last known position for M24 can be found 10 lines above the fallback event, but the one for M21 is 100 lines above and it is harder to find. Searching for that information is time consuming and needs to be done for every fallback event.

[6] Janus Transformation Language (JTL) is an Eclipse EMF-based tool provided by the University of L'Aquila, which is realized to design and manipulate models, maintain consistency, and synchronize software artefacts.

```
 [...]
20161017_231136106;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Pos
ition_M21;33139
20161017_231136106;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Pos
ition_M24;33116
20161017_231136106;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Vit
esse_M21;6
20161017_231136106;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Vit
esse_M24;2
[...]
20161017_231136377;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Pos
ition_M24;32766
20161017_231136377;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Vit
esse_M21;0
20161017_231136377;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Vit
esse_M24;1
[...]
20161017_231136480;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Pos
ition_M24;33117
20161017_231136480;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__Vit
esse_M24;65535
20161017_231136480;u;dbg;192.168.10.101;1;0;M11.LogiqueSecu.Algo_DonneesUtiles__fal
lback_M11;inconsistent_position
[...]
```

With the integration of the log model in EMF in addition to the B specification model, a fallback event occurring in the log will be linked to the "position consistency test" in the B model. This additional information will highlight the link of the fallback event with the events containing the last known positions. For the developer who analyses the fallback situation, every relevant information is directly available. It can be immediately identified and treated. Moreover, the treatment can easily be automatized, this will be one improvement for the next phase.

With the integration of the log model in a graphical view, we estimate that 21 hours over 210 (3 days of 30 days) can be saved in the log analysis (KPI_1.2) and 35 hours over 280 (5 days of 40 days) can be saved in the log analysis (KPI_2.1). Those days are also counted in the global development time (KPI_1.1)

3.3. Results of the first evaluation phase

3.3.1. KPI values

The Table 3.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 02_CSY.

Table 3.2: Phase 1 KPI values (case study 02_CSY)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_B	$\Delta_B\%$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_1.1	D	80	70	12.5%	70	12.5%	25%
1	KPI_1.2	H	70	65	7%	63	10%	14%
2	KPI_1.2	H	210			175	16.6%	14%
2	KPI_2.1	H	280			245	12.5%	20%

S = Scenario number
Unit: D = Day, H = Hour
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
$\Delta_B\% = 100 \cdot (V_B - V_0)/V_0$ *(baseline improvement)*
$\Delta_1\% = 100 \cdot (V_1 - V_{REF})/V_{REF}$ *(Phase 1 improvement)*
V_0 *= Estimated pre-MegaM@Rt2 value*
V_B *= Value obtained with MegaM@Rt2 baseline tools*
V_1 *= Phase 1 value obtained with Framework initial version*

It can be noted that $\Delta_B\%$ and $\Delta_1\%$ for scenario 1 are obtained through different means but it would not make sense to addition them since the improvement with baseline was obtained with statecharts

model and the improvement during phase 1 with B-XML model. Using both kind of model in the same project would introduce additional development time.

3.3.2. Plan of improvements

3.3.2.1. Requirement coverage

Requirements for CSY are the following:

Table 3.3: CSY requirements

ID	*Requirement*	*Coverage*
CSY_01	Models of the modules shall be created using their specifications, those models shall describe the communication, size, frequency and variance of the messages.	This requirement has been partially fulfilled, the focus during phase 1 being on the safety core and not the satellite components and network aspects.
CSY_02	Models are available in B language, they should be reused.	This requirement has been totally fulfilled with the complete usage of the B model of the safety core.
CSY_03	Holistic analysis of the network, based on models and/or logs shall be produced, with mean and worst-case scenario.	This requirement has partially been fulfilled, the examples of analysis run toward the scenario involve network and components failures, they are based on logs and model of the safety core. No worst case or mean scenario has been produced because the simulation was run by existing partial trace and not generated from parameters. In return, this allows to play realistic simulation and particularly play simulation of trace that failed in previous versions of the system and see if a new version behave better.
CSY_04	Simulations shall be run using phase one models and input provided by logs.	This requirement has been fulfilled
CSY_05	Simulations shall be compared to real executions (live or logged).	This requirement has been fulfilled, phase 1 simulations are live executions since they use the actual software of the safety core

3.3.2.2. Feedback to tool providers

During phase 1, collaborative work has been done with UAQ (log modeling and connection with JTL) and ARM (integration of B-XML and log models in EMF Views).

3.3.2.3. Work planned for the Phase 2

During phase 2, CSY will focus on a global simulation, introducing model of the whole system in interaction with the model of the safety components. This will require new techniques of model animation for those components and reuse the work done during the baseline phase.

3.3.3. Summary of the progress towards the roadmap

During phase 1, both scenario 1 and 2 have been greatly pushed forward. CSY was able to integrate models of the system in the MegaM@Rt2 Framework and run simulations of those models. The added value from this integration clearly appears in the analysis of the logs, where the identification of the cause requirement and the automatic collections of involved variables simplifies the analysis work. We think that this could be pushed further during phase 2, with an automatic collection of defaults in logs, sorted by patterns, cause requirements and values. This automatic collection could be globally

analysed by the experts, with the added information of frequency of occurrences. The model will benefit of a global modelisation with models of satellites components and integration of the B-XML model. We plan to be able to generate execution from forged scenario in addition to rerun of existing traces

4. Case study 03_IKER – Deployment and supervision of agents

4.1. Case study overview

4.1.1. System description

IKERLAN's interest in this project is focused on improving the whole SW lifecycle development process in order to ensure quality and efficiency of its projects. More specifically, IKERLAN wants to deepen on the following technologies or processes: model-based design, automatic code generation, model-based testing and runtime verification.

In this context at the beginning of the project Ikerlan proposed a case study related with the deployment and supervision of agents in a Smart Warehouse. During the Phase 1 IKERLAN has realized that this case study is not the most suitable one to validate all the defined scenarios. The lack of design specifics of the behavior of the supervision application (since this is quite simple) has led us to look for another case study where we can apply some of the solutions proposed by the tools providers in the MegaM@Rt2 framework.

The case study “Deployment and supervision of agents” is widely described in D1.3. This case study will be used in scenarios 1 & 4. In brief this case study will focus on the components which implement the infrastructure for the supervision system (represented by red color boxes on Figure 4.1). The agent business functionality is out of the scope of this case study.

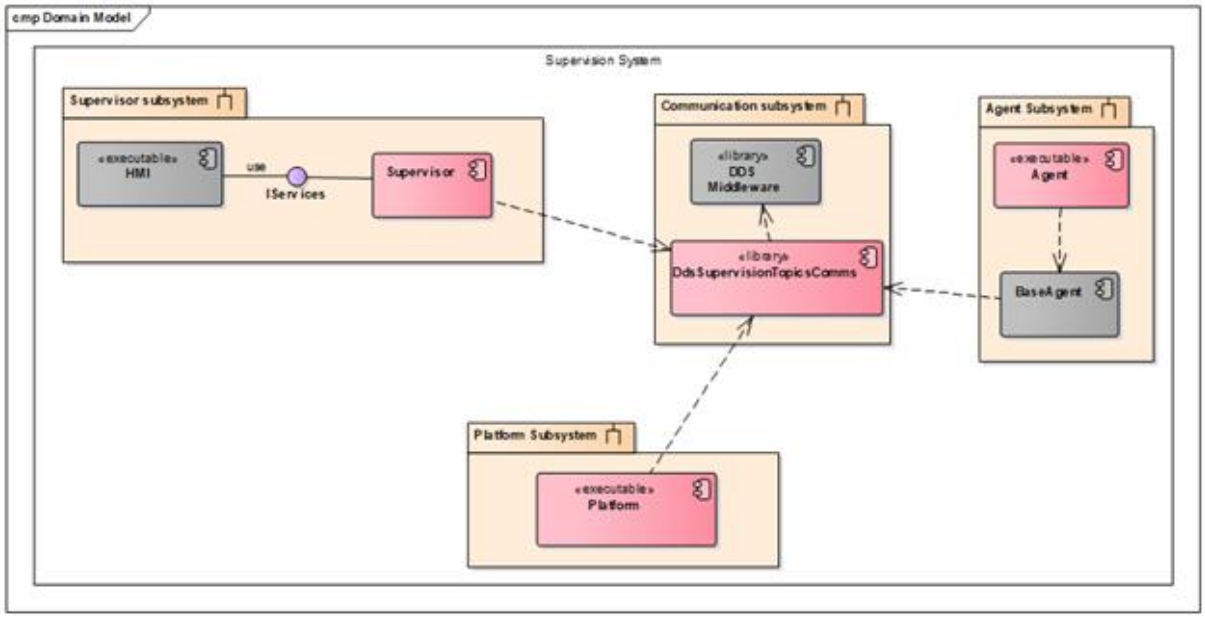

Figure 4.1: Supervision system architecture

The main functional requirements for the supervision system are the following ones:

- The system should be able to configure different layouts. A layout is an arrangement of different agents within different nodes. An agent is a software component prepared to be executed within a JVM or a Windows platform. Every agent has at least an infrastructure state machine (common for all agents) and implements a specific business logic related to its control function within the Smart Warehouse. These agents are supposed to be in some kind of structured repository.

- The system should be able to deploy a layout. Deploy is install & run agents in nodes as configured in the layout.
- The system should be able to undeploy a layout. Undeploy is kill & uninstall agents configured in the layout.
- The system should be able to activate a layout. Activate is to communicate all the agents in the layout to go to an operational state.
- The system should be able to deactivate a layout. Deactivate is to communicate all the agents in the layout to go to a ready state.
- The system should be able to monitor the infrastructure state of the agents.
- The agents should implement an infrastructure state machine.

The case study "IoT gateway to connect devices to cloud" is described next. This case study will be used in scenarios 3, 4 & 5. In brief, this system will be in charge of enquiring a device in order to obtain data of interest from it, such as telemetry, and later relaying this data to the cloud through an MQTT broker. Similarly, the end user on the cloud should be able to command the device to perform certain operations, such as software updates, following the route to the broker and from there to the gateway.

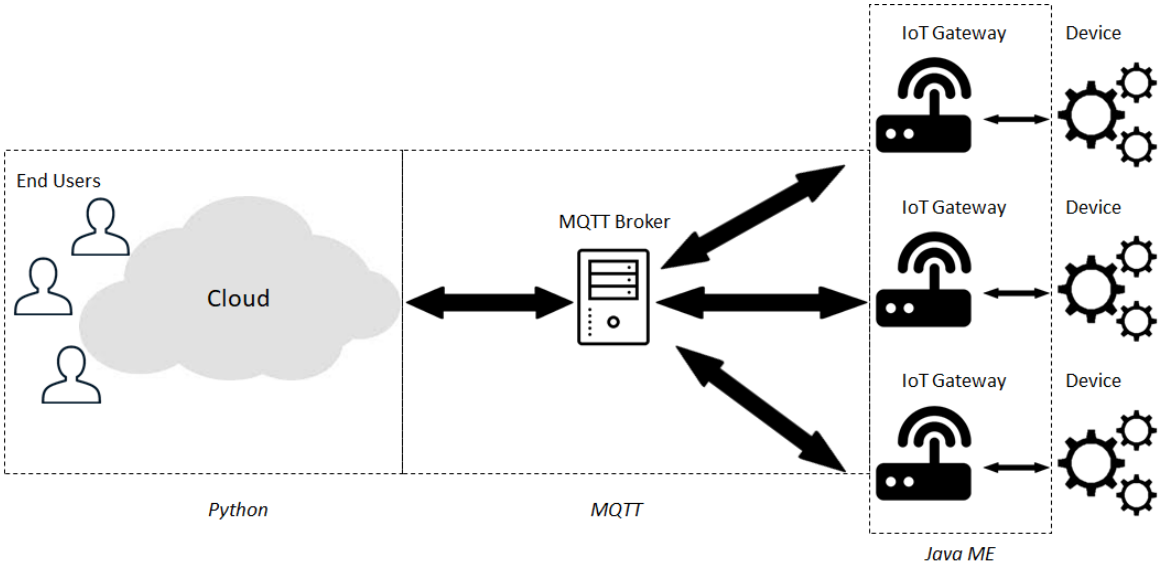

Figure 4.2: IoT Gateway scenario

The main design drivers for the IoT gateway system are:

- The system design should be as generic as possible, so that it can be reused in different IKERLAN's projects as much as possible.
- The system design and its state machine should be validated as autonomously as possible, so that the system behaviour is as expected.
- Code generation should be automated as much as possible to reduce the impact of system design modifications/enhancements.
- The system should allow different levels of runtime verification and log analysis, so that different users/project phases can verify the system accordingly to their requirements.
- The system should allow automatic system validation and runtime verification of its design, so that behaviour imperfections can be detected and correct them accordingly.

- The Gateway should allow bi-directional communication from/to both the device and the cloud.
- The communication protocol should follow a publish/subscribe paradigm, e.g. MQTT.
- The system design should allow a potential modification of the communication to allocate request/response protocols, e.g. REST APIs.

4.1.2. Development and validation scenarios

The case study is structured according the scenarios defined below.

4.1.2.1. Scenario № 1: Communication data modelling

The main objective in the scenario is the automation of the communication library (called DDSComms) for the supervision system. This library includes the third party DDS specification implementation (DDS community version implementation from PrismTech) and an IKERLAN component that allows management (read and write) of the data (topics) shared between the different subsystems.

IKERLAN uses one tool from the DDS implementation library in order to generate the necessary classes to read and write DDS data/topics from the application level. One topic is defined by means of an IDL (Interface Definition Language) file text. A compiler tool (IDL compiler which third party provides) allows to automate the generation of the implementation classes. Our library needs some wrapper classes around the automatically generated ones. Some of these classes are common to all DDS topics but others are specific to the topic.

Before MegaM@Rt2 we defined every topic in a different file and execute the compiler for each topic. In a following step the wrapper classes are added. All the classes are packaged as a library to be used for other subsystems. We consider of great interest, thanks to MegaM@Rt2, to have a design tool with which we define this data and which we can customize to generate all the implementation classes (wrapper classes included).

4.1.2.2. Scenario № 2: Automata modeling and simulation

The behavior of the IoT gateway is driven by a state machine. This state machine is subject to slight modifications depending on the final application in which it is incorporated. The main objective in this scenario is that the designer uses the tools provided by the modelling environment to simulate state machines and activity diagrams, validating the requirements at an earlier stage of the development. This will help the designer to troubleshoot problems that were not initially considered, as well as predicting possible future ones before they can derail the project. Once the model is validated, source code skeleton of the automata can be automatically obtained, reducing in a great way the effort of implementing their logic and keeping the source code synchronized with the model if any change is done.

4.1.2.3. Scenario № 3: Model management and traceability

The scenario description follows: “A designer working with a model wants to compare the changes being introduced with a previous version of the model. For all kinds of data, changes between both versions are highlighted and clearly understood, pointing which parts are being affected by the changes that are being introduced. The designer shall have the possibility of reverting to a certain revision. Different designers should be able to work sharing the model somehow.”

The main objective in this scenario is version management in system modeling.

4.1.2.4. Scenario № 4: Automatic test generation and verification

This scenario is focused on the tests automation and particularly on model-based testing of the supervision system.

Every time a change is made in any of the supervision subsystems, the developer shall run and pass all unit tests in its local environment before committing any code to the mainline. But this is not enough, in addition to automated unit tests, when committing changes, an integration test shall be executed to check that the correctness of the whole system is not affected.

Moreover, if the model changes new unit tests have to be automatically created to validate the updated component. We are looking for a tool that let us to define a test model with different test scenarios that can be run every time a modification is made in the supervision system.

The main objective in this scenario is to achieve traceability between requirements and tests.

4.1.2.5. Scenario № 5: Runtime trace analysis

The main objective in this scenario is to explore the scope of the runtime analysis tools and look for tools (i.e. based on parsing log files) that can match our requirements and improve the validation procedure in the IoT gateway described earlier in this document.

4.1.3. KPI definition

The definitions of the KPI for the case study 03_IKER are reported in the Table 4.1; the column ΔT% gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 4.1: KPI definition (case study 03_IKER)

S	*KPI*	*Definition*	*U*	V_0	Δ_T%	*P1*	*P2*
1	KPI_1.1	Time in automate DDSComms library for 11 topics	H	24	33%	✓	
1	KPI_1.2	Number of files to save the DDS data topics	Q	11	91%	✓	
1	KPI_1.1	Time in modifying one topic and getting the supervision system ready again	H	10	50%		✓
2	KPI_1.3	Time to develop the state machine from scratch	H	24	33%		✓
2	KPI_1.3	Time to add/remove new states or transitions and generate code	H	4	25%		✓
3	KPI_3.1	Time spent to obtain and compare different model versions	H	4	25%		✓
3	KPI_3.1	Time spent to merge two versions of models	H	4	66%		✓
4	KPI_1.3	Traceability report between requirements & tests	Q	0	100%		✓
4	KPI_1.3	Time to get the tests running after requirement modification	H	4	50%		✓
5	KPI_2.1	Time to run tests manually or semi automatically vs. Time to run them based on the trace analysis	H	24	33%		✓
5	KPI_2.1	Time to detect bugs through manual or semi automatically done tests vs. Time to detect bugs based on the trace analysis	H	6	33%		✓

S = Scenario number — *Unit: H = Hour, Q = Quantity*

P1 = Phase 1, P2 = Phase 2 — *V_0 = Estimated pre-MegaM@Rt2 value*

Δ_T% = Foreseen Phase 2 improvement — *V_B = Value obtained in MegaM@Rt2 with baseline tools*

4.2. Development and validation activities

In the following, the section 4.2.1 “Previous work” reports on the case study up to the month M22 and the section 4.2.2 “Development and validation activities” on what Ikerlan did during the last two months of phase 1.

4.2.1. Previous work

4.2.1.1. Scenario 1

During the baseline experiments it was proven that Modelio not only allowed to access to its design elements but also incorporated the execution of scripts. A first test (during Helsinki hackathon) was carried out to see the feasibility of the problem. Softeam implemented a first version of a new module (DDS Modeler version 0) to add in Modelio. This module allowed to design a class in Modelio, tag it under the <Topic> stereotype and launch an operation to execute a script that generates the rest of the necessary classes. In this first version class attributes cannot be of composite type.

Figure 4.3 shows a topic used for testing. Figures 4.4 and 4.5 show snapshots from Modelio with the new module added and the result of executing the DDS operation.

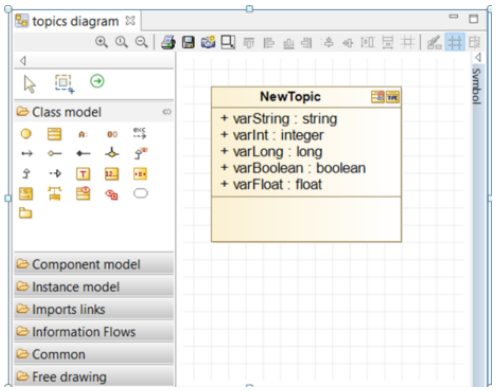

Figure 4.3: Topic description in UML

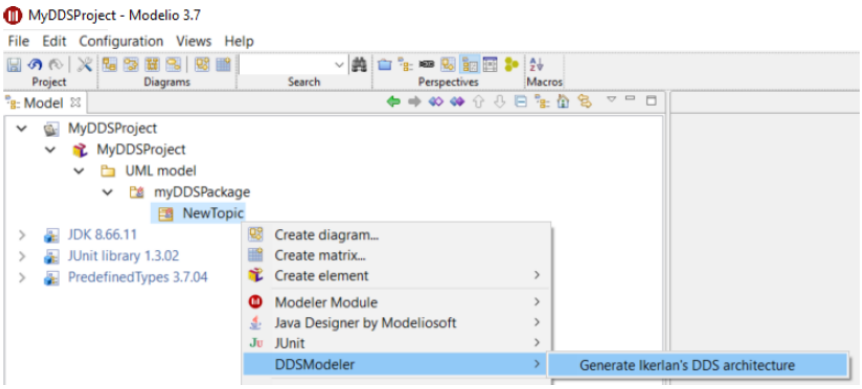

Figure 4.4: DDSModeler plugin

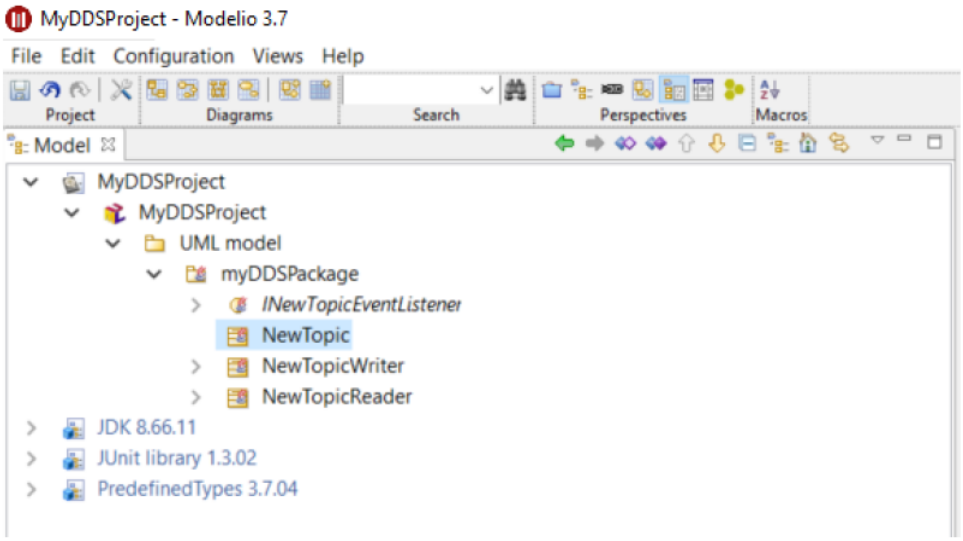

Figure 4.5: Automatic generated classes from topic description

After the baseline experiments, Modelio was selected as a design tool to develop and validate this scenario.

Starting from the first version of the DDSModeler module, Softeam and Ikerlan have been working together. The collaboration has been carried out in the following way: Ikerlan included new detailed requirements necessary for scenario validation, Softeam updated the module and Ikerlan tested the modifications and sent the feedback to Softeam.

During phase 1 the module has been improved to allow new features required by Ikerlan:

- The topic attributes can be structured types.
- These structured types can be defined in the same IDL file as the topic or can be defined in another IDL file only for types.
- Special treatments for attributes which are enumerate type.

Softeam has incorporated these features into its module and has also made a number of improvements related to the module usability. As a result, the version 1 of the DDS Modeler has been released.

4.2.1.2. Scenario 2

During baseline experiments Moka (from ATOS) and Pauware (from UPAU) tools were selected as candidates to develop and validate this scenario. After reviewing the existing documentation and tools that were available from the start of the project, Ikerlan selected the Pauware tool to start the evaluation. The reason for this choice was that finally the tool would not only allow us to run the model but also offer us the possibility to generate Java code for the state machines. Due to the type of projects that Ikerlan develops, we found interesting to have a Java engine for the execution of state machines.

At Helsinki hackathon Ikerlan worked with UPAU, for learning how to use the PauWare Java API to code and run a state machine. During the baseline experiments the implementation of a basic agent was carried out, according to the state machine shown in the following figure.

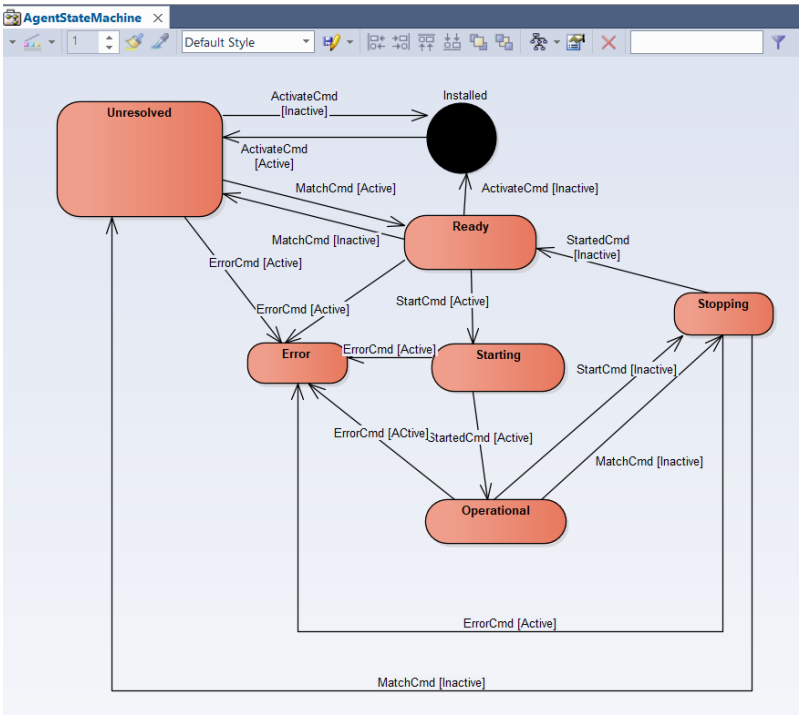

Figure 4.6: Basic agent state machine

During phase 1 we have come to the conclusion that, in order to validate this scenario, it is better to use case study "IoT gateway to connect devices to cloud". In this case study the behavioural models are more complex and allows us to take greater advantage of the MegaM@Rt2 tools.

At Versailles hackathon we shared with UPAU the first ideas and an initial draft version of the automata.

4.2.1.3. Scenario 3

After the baseline experiments some tools (ModelioConstelation, EMFViews and NeoEMF) were selected as candidate tools. However, we weren't able to check whether the tools fit our requirements or not, due to a prioritization of other scenarios. During phase 1 we have not worked on this scenario.

4.2.1.4. Scenario 4

After the baseline experiments, Certifyit tool from Smartesting was selected to develop and validate this scenario. At Helsinki hackathon Ikerlan showed Smartesting the supervision system and defined some preliminary requisites (Version 0) for testing. The requirements to be validated were the following:

- The supervisor system should be able to deploy a layout configured in the persistence.
- Agents will be deployed on platforms as long as they exist in the binary code repository.
- The agents once deployed will remain in READY state until they are activated.
- The supervisor system must be able to activate a previously deployed layout.
- The activated agents will modify their status from READY to OPERATIONAL.
- The supervisor system must be able to deactivate a previously activated layout.
- Deactivated agents will change their status from OPERATIONAL to READY.
- The supervisor system must be able to undeploy a layout.

Smartesting designed a first model to test some of these requirements. The following images show different views of the model for the automatic generation of tests covering the above requirements:

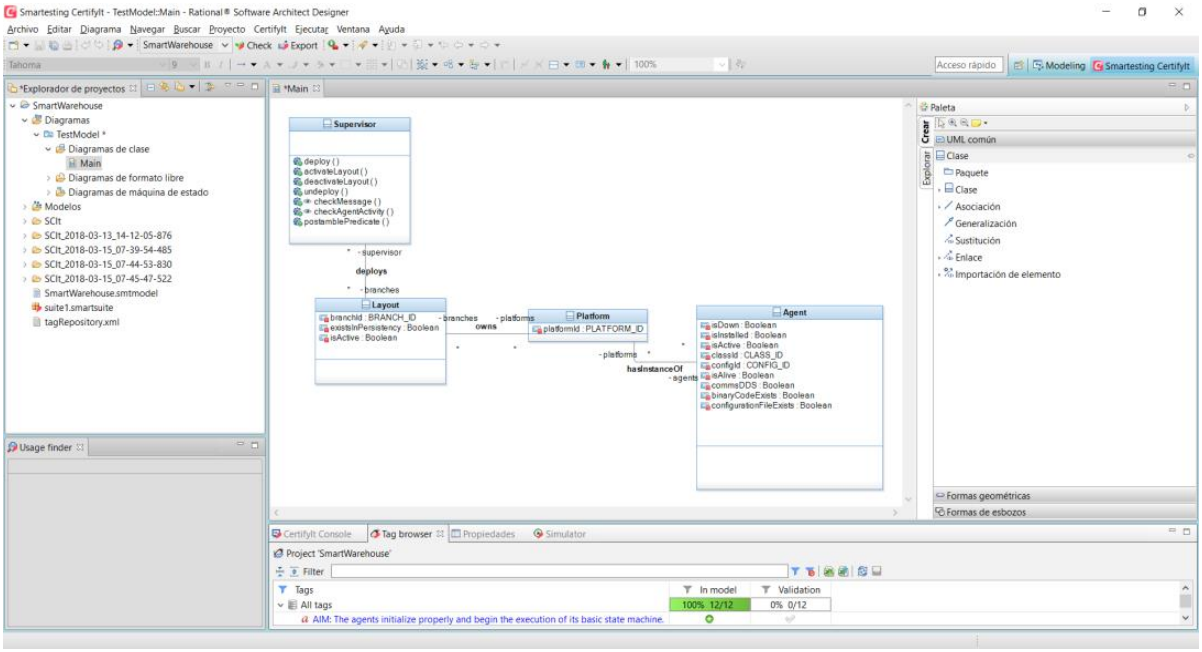

Figure 4.7: Test Model Class Diagram

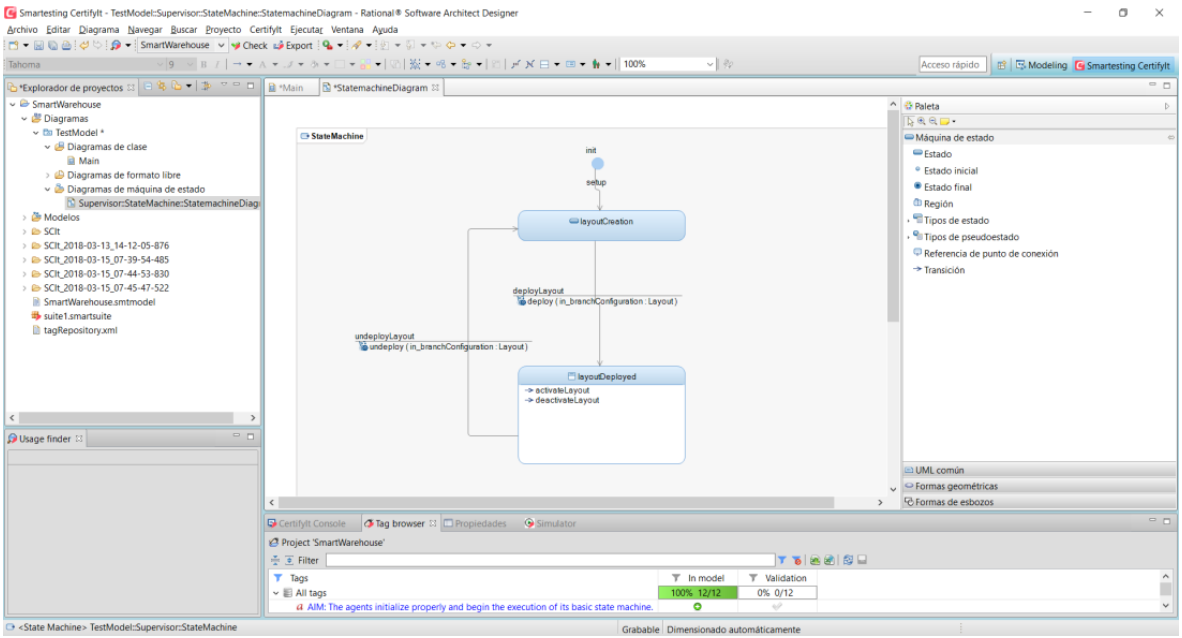

Figure 4.8: Test Model State Machine

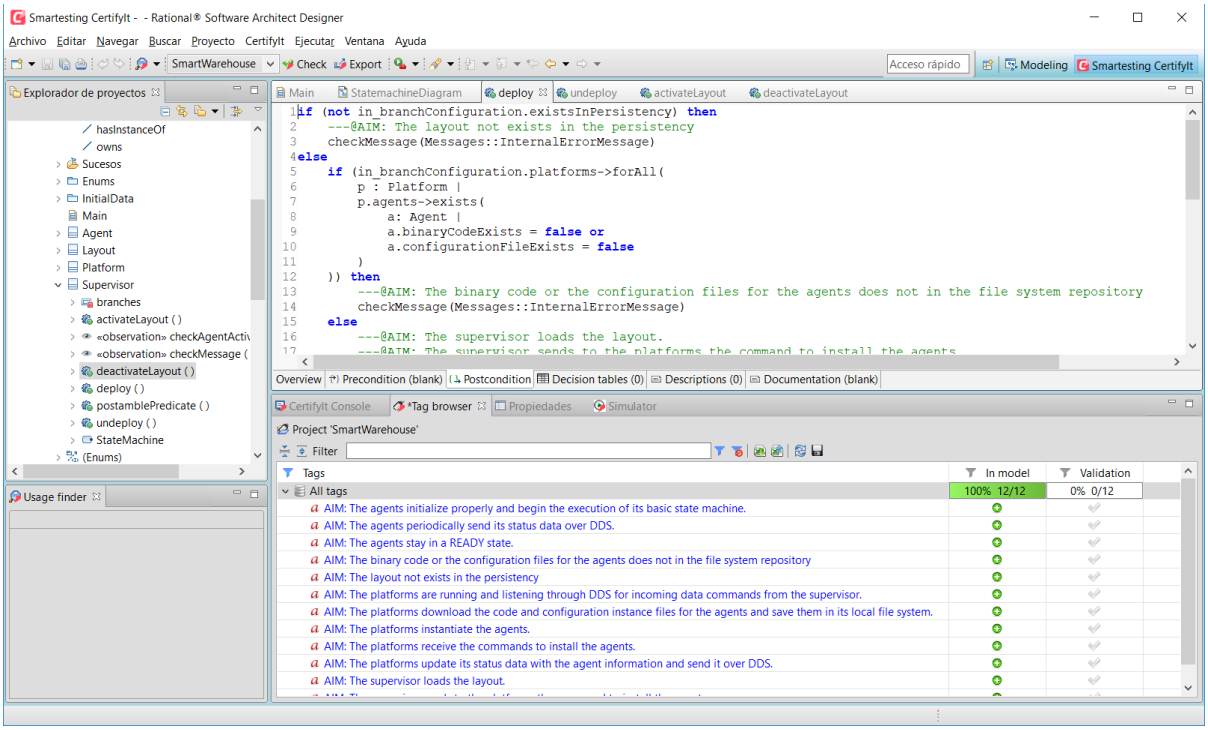

Figure 4.9: Test Model precondition/postcondition operation description

With the above model as input, Certifyit generates the tests automatically. The following image shows the result of the generation.

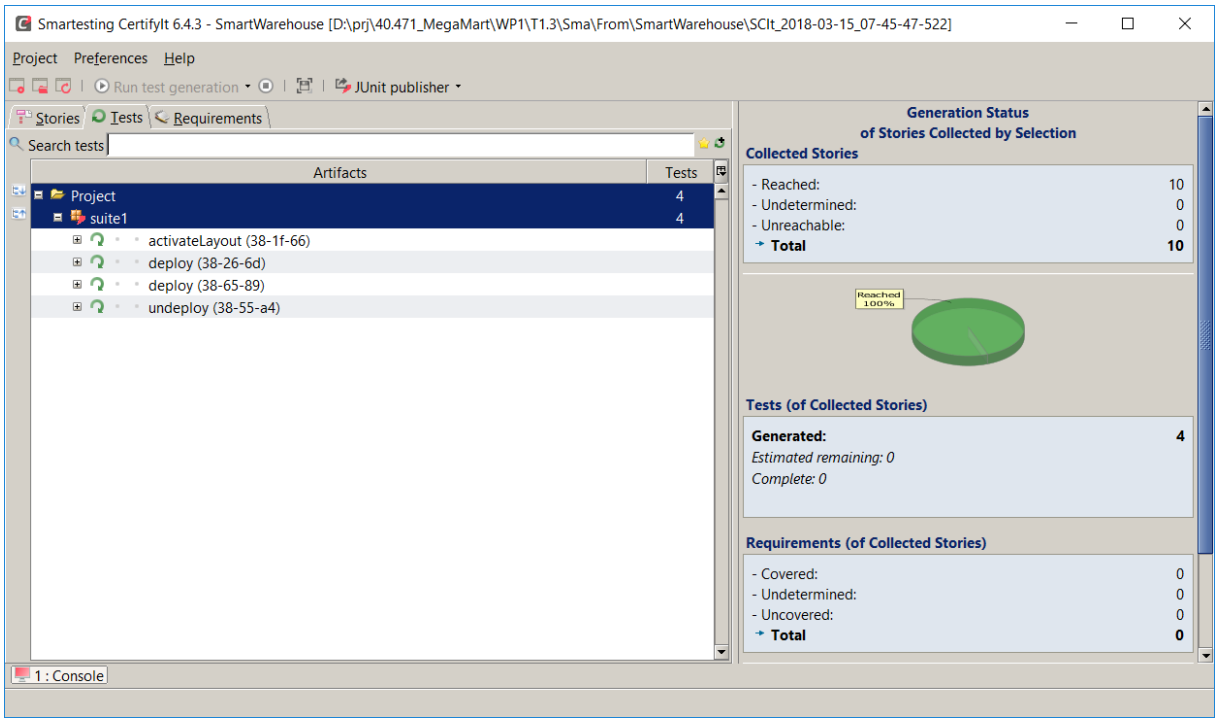

Figure 4.10: Tests generated

During phase 1 Ikerlan has checked Certifyit with a preliminary test model that covers some functional requirements (version 0) of the supervision system to make offline testing. Ikerlan has implemented the adaptation layer of the tests, which have been generated automatically by Certifyit and has executed them.

4.2.1.5. Scenario 5

During the baseline experiments ConvexHull was selected as a good tool to check in order to analyze logs but this tool was not available until later phases in the project.

During phase 1, when reviewing the deliverables of the tool providers, we found it very interesting to focus also on this runtime analysis scenario, not only to validate the behavior of the system but also to explore issues of traceability between the runtime and the design or requirements. As in the case of scenario 2, we have considered that the IoTgateway use case is the right one for this scenario.

4.2.2. Development and validation activities

4.2.2.1. Scenario 1

During the last two months of phase 1, Softeam has resolved the issue related with synchronization between code and data model and a new update of the DDSModeler has been provided, with which the measure of KPIs has been carried out.

The next figure summarizes the validation plan:

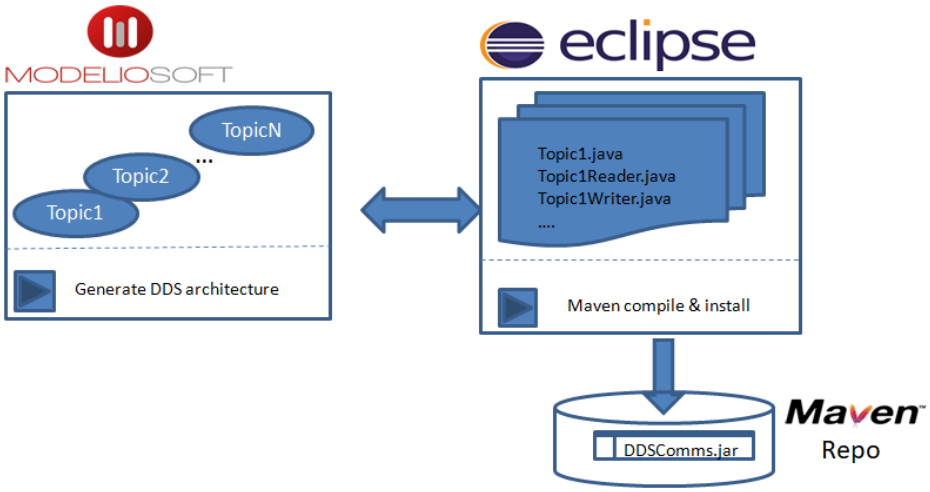

Figure 4.11: Experimentation plan description

The validation plan to measure the KPIs planned for phase 1 has followed these steps:

1) Create an empty Maven eclipse project for the infrastructure library (com.ikerlan.supervisor.ddsinfrastructure.jar).
2) Design the library of basic classes with Modelio (until now we use Enterprise Architect).
3) Using the Modelio with the DDSModeler last release, Java designer generates the Java files for the basic classes in the Eclipse workspace.
4) Add to the model all the infrastructure topics used in the DDSComms library. The next figure shows the diagram with the data model.

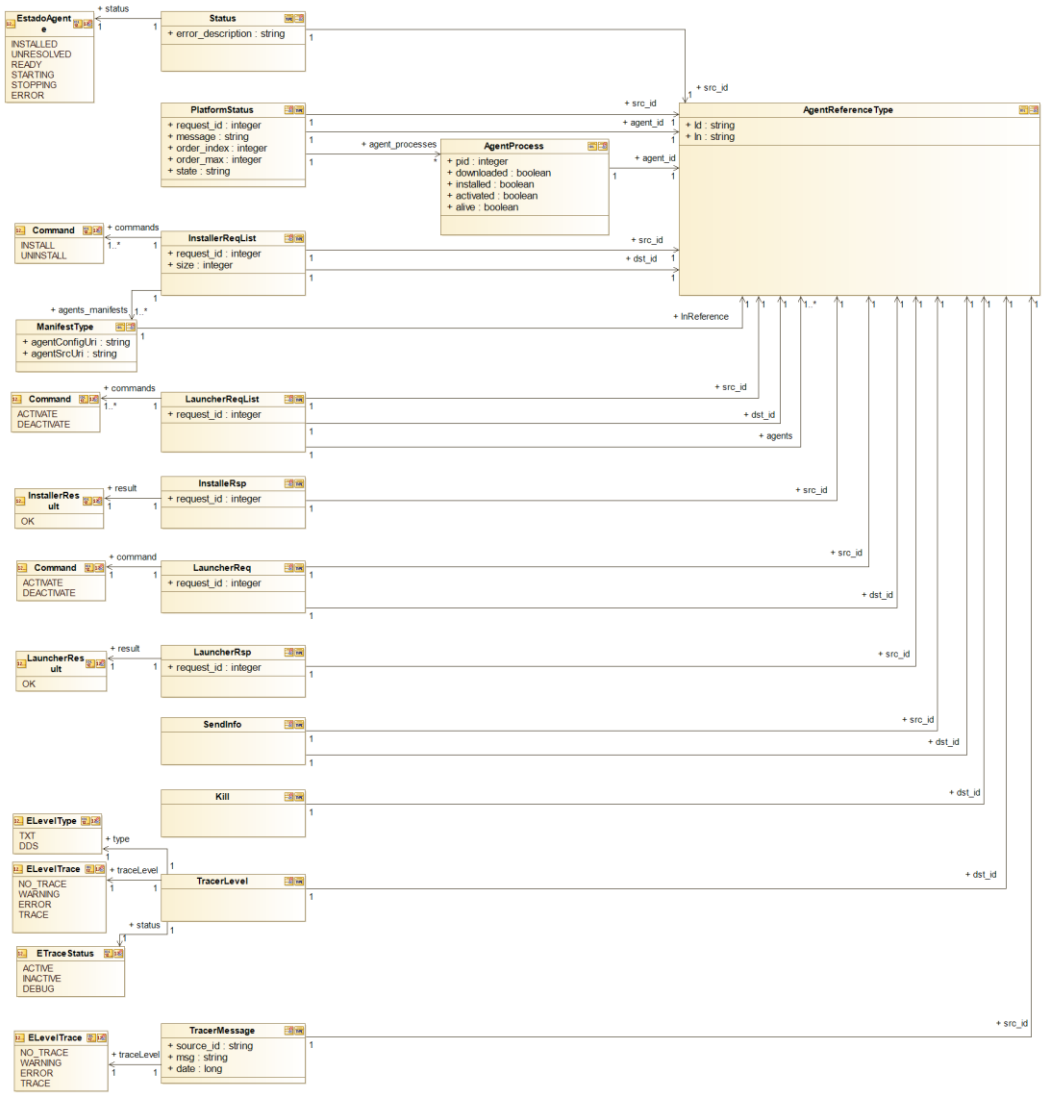

Figure 4.12: Class diagram with infrastructure data

5) Using the DDSModeler, execute the IDL compiler for all the DDS data (call topics) defined in the model. All the generated classes related with every topic will be added to the Eclipse workspace.

6) From the Eclipse workspace use Maven to package the library.

7) Measure the time it takes starting defining all the topics until we get the library ready to be used by the other components of the application. Compare this time with what it took when topics were not included in any model.

On the other hand, Ikerlan has begun a new collaboration with UOC in order to explore the way of modelling the messages used in the case study “IoT gateway to connect devices to cloud”.

4.2.2.2. Scenario 2

The collaboration with UPAU has been resumed and we have sent them the behavioural diagrams modeled with the tool we have used until now. Our first goal is to share with UPAU the characteristics of our activity-based state machine in order to improve it and share with them our concerns regarding the current state machine (moving from an activity-based machine to an event-based one). The issues we are facing right now are related with different tool compatibilities. UPAU’s tool needs XMI files in

version 2.1, which can be generated with the export tool provided by Enterprise Architect (which is the tool we used for modelling). However, these have a singular pattern compared to all other editors, UPAU has worked with (Modelio, Papyrus, Star UML), which make the XMI file incompatible. We are currently researching other possibilities to solve this issue and continue exploring UPAU's tool.

On the other hand, ATOS has also offered us its collaboration to use Moka tool in this scenario. The state we are in is similar, as the diagrams need to be modelled in Papyrus in order to simulate their behaviour and progress with the objectives of the scenario.

4.2.2.3. Scenario 3

During phase 1 we have not worked on this scenario.

4.2.2.4. Scenario 4

Ikerlan has redefined and added new requirements to the ones used in phase 1. We are currently modeling the tests to validate these requirements. We are receiving help from SMA in this task. We are carrying out working telephone meetings. In this way we hope to speed up the learning curve in the preparation of the test model as well as to know better the tools (Certifyit & MBeeTle) for the generation and execution of tests.

This work will go on during phase 2 so no KPIs measurements have been done during phase 1.

4.2.2.5. Scenario 5

Ikerlan is collaborating with ATOS in the use of aspects to inject non-functional code to the business logic. These aspects will allow the injection of both logging and verification functions which could be used to assess the performance of the system during runtime. The first stages of this collaboration have consisted on refactoring the code that Ikerlan had already developed, so that the injection of aspects could be accomplished easier. Following on this code, ATOS is now in the process of developing a small PoC (Proof of Concept) to show what benefits this functionality will add to the scenario's objectives, allowing the execution of it in future phases.

In parallel to this, Ikerlan is exploring with UAQ the interest in including traceability concepts in this scenario in order to improve the design or the validation.

4.3. Results of the first evaluation phase

- A module (DDSModeler) has been developed by Softeam and the feasibility of using it for the modelling and automation of the DDSComms library has been proven.
- Model Based Testing (MBT) has proven to be a good approach to system verification, especially in scenarios where effort invested in automation can be leveraged; such as systems with hundreds of requirements or systems where continuous improvements need to be added during an extensive period of time. Tools like Certifyit & MBeeTle are of great help to address tests generation and requirements validation. These tools considerably reduce the implementation time of the tests.

4.3.1. KPI values

The Table 4.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 03_IKER.

Table 4.2: Phase 1 KPI values (case study 03_IKER)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_1.1	H	24	12	50	33%
1	KPI_1.2	Q	11	1	90	91%

S = Scenario number
Unit: Q = Quantity, H = Hour
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
V_0 *= Estimated pre-MegaM@Rt2 value*
$\Delta_B\% = 100\cdot(V_B - V_0)/V_0$ (baseline improvement)
V_B *= Value obtained with MegaM@Rt2 baseline tools*
V_1 *= Phase 1 value obtained with Framework initial version*
$\Delta_1\% = 100\cdot(V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)

4.3.2. Plan of improvements

4.3.2.1. Requirement coverage

The Table 4.3 shows the user requirements coverage.

Table 4.3: User requirements experimentation in the case study 03_IKER

Identifier	*Definition*	*Experimentation*	*P1*	*P2*
IKER_01	A modeling tool supporting DDS UML profile, for the representation of DDS related concepts in order to enable code generation based on it.	Scenario 1: In progress. We have addressed the addition of DDS Topic to the design model. Modelio tool is being used.	✓	✓
IKER_02	An automated model-based code generation tool for different programming languages: mainly Java, C and C++ and optionally .NET.	Scenario 1: In progress. We have addressed the java code generation. Modelio and DDSModeler module tool is being used.	✓	✓
IKER_05 & IKER_06	A modeling tool that supports linking of automatic generated source code with the model keeping them synchronized.	Scenario 1: In progress. Issues related with DDSModeler has been solved by Softeam in order to cover this requirement.	✓	✓
IKER_04	A tool that supports definition, simulation and debugging of automata, based on an intuitive and easy-to-use user interface, which enables the analysis of different automata executions, pointing clearly any correctness error detected in the model.	Scenario 2: In progress. PauWare Api has been checked. Integration with our case study expected at Phase 2.		✓
IKER_03	A modeling tool that supports the importation of existing data from other common modeling tools that are currently being used in our organization (e.g. Enterprise Architect, Doors, Rhapsody, etc.)	Scenario 3: there is a risk that this requirement will not be covered		✓
IKER_07	A versioning tool that stores all the data in the modeling environment with branching and merging support	Scenario 3: expected to be covered at Phase2		✓

Identifier	*Definition*	*Experimentation*	*P1*	*P2*
IKER_08	A versioning tool that supports the possibility of working at the same time with different versions of the model data, being able to easily visually compare changes between their single elements.	Scenario 3: expected to be covered at Phase2.		✓
IKER_09	Automated model-based automatic unit test generation tool based on the component definitions. A continuous integration process that supports the detection of new commits and changes (in Git, TFS and SVN like source control systems) triggering this way the integration process.	Scenario 4: In progress. Certifyit and MBeeTle tools are being used.	✓	✓
IKER_21	A runtime monitoring tool that is able to gather log information produced by the system	Scenario 5: expected to be covered at Phase2		✓
IKER_24	A runtime monitoring tool that supports the definition of the desired behavior of the system, so it can be used to compare it against the real log information	Scenario 5: expected to be covered at Phase2		✓
IKER_25	A runtime monitoring tool that is able to gather log information produced by the system	Scenario 5: expected to be covered at Phase2		✓

4.3.2.2. Feedback to tool providers

- Continuous collaboration with Softeam to adjust the DDSModeler to our needs. The work is being carried out through mail and teleconference.
- Continuous collaboration with SMA to model for testing and resolve doubts on how to use Certifyit and MBeeTle. Periodic teleconferences are held.
- The case study “IoT gateway to connect devices to cloud” has been introduced to UPAU, UOC, ATOS and UNIVAQ. Designed documentation has been submitted and several teleconferences have been planned.

4.3.2.3. Work planned for the Phase 2

- Scenario 1:
  - Incorporate QoS attributes in the design of the topics (currently we use fixed attributes in code).
  - Make changes in the attributes of any of the topics. Regenerate the library and evaluate the time it takes to have the whole application ready for execution.
  - Evaluate the advantages of incorporating messaging data into the design model in case study “IoT gateway to connect devices to cloud”.
- Scenario 2
  - Compare the design process of the initial state machine (designed with Enterprise Architect) vs. the final state machine (using MegaM@Rt2 Framework).
  - Evaluate the performance and quality of the new generated code (from Pauware) vs. the initial state machine (from the Enterprise Architect templates).

- Scenario 3
    - Select a tool from Megam@Rt2 Framework in order to carry out design version control, check it and compare it with our current practices.
- Scenario 4
    - Go on with the modelling for testing for new requirements.
    - Implement the adapter classes for these new requirements and run the tests.
    - Get traceability report for our requirements.
    - Explore the advantages that online testing can offer in relation to the supervisor application and evaluate the MBeeTle tool from Smartesting in this scenario.
- Scenario 5
    - Use aspects to assess the performance of the system during runtime.
    - Explore traceability concepts.

4.3.3. Summary of the progress towards the roadmap

After finishing phase 1 we would highlight that some of our validation scenarios (1 & 4) are well focused. Messaging data modelling and MBT techniques to validate requirements are allowing to improve our software design, development and test processes. There doesn't seem to be any problems in reaching the target for the KPIs.

During this phase we had some difficulty mapping Megam@Rt2 tools with some of our scenarios. In scenario 2, execution of models and generation of code, the problem has been related with the simplicity of the behavior of our use case. In scenario 5 our use case was not the most appropriate for the use of some of the tools and methods offered from Megam@Rt2 project. That is the reason why during this phase we have dedicated considerable effort to the selection of another use case that allows to test, learn and take advantage of the improvements that certain tools can offer. Thus, we have defined a new use case “IoT gateway to connect devices to cloud” that is described at the beginning of this chapter. We are currently working on the redefinition of these scenarios and resuming some of the collaborations that we detected during the baseline experiment as well as opening some new ones. During phase 2 we hope to meet our expectations in these scenarios.

Although we have not yet worked in scenario 3, we expect to do so during phase 2.

5. Case study 04_TEK – Indoor positioning

5.1. Case study overview

5.1.1. System description

The Tekne case study 04_TEK is centered on a wearable mote that is a mobile network node, based on the Ultra-Wideband (UWB) technology, with short range communications and indoor positioning capabilities. The Tekne case study experiments the MegaM@Rt2 Framework in the development of the UWB network Medium Access Control (MAC).

5.1.2. Development and validation scenarios

5.1.2.1. Scenario № 1: Modeling, design time verification, functional verification

The Scenario 1 is experimented in the Phase 1. It consists of the following steps:

1) System modeling: requirements, design, and verification of the state machine diagrams.
2) Preliminary functional verification on the basis of modeling results (model design-time verification with different coverage strategies and script files).
3) Evaluation of the effort for deriving the test models from the system models.
4) Tracing of requirements along the different development phases.

The Scenario 1 will be refined in the Phase 2 through the integration of Conformiq with AIPHS. The latter is a tool developed by UAQ conceived to support designers in the development of On-Chip Monitoring Systems able to satisfy given monitorability requirements, namely requirements about the possibility to observe the behavior of a system for collecting metrics. AIPHS uses the Embedded Trace Macrocell (ETM). ETM is a debug component of the ARM processor that enables reconstruction of program execution and that is designed as a high-speed, low-power debug tool for instruction tracing. AIPHS offers two modalities: instrumentation, which is suitable for functional verification, and ETM initialization only, which being overhead free is suitable for non-functional verification and runtime monitoring of the running system. AIPHS is provided in the form of static libraries with which to link the application.

5.1.2.2. Scenario № 2: Runtime monitoring and non-functional verification

Scenario 2 is planned for the Phase 2. It consists of the following steps for non-functional verification through runtime monitoring of the target processing platform.

1) Automatic generation of the code that must be adapted to specific software environment and a specific hardware, to the test models, and to the structure of the runtime monitor.
2) Bridge from the models to the testing: specifications of the desired system behavior, then generation of the monitor from specifications.
3) Runtime monitoring: connecting the runtime monitor to the SUT through code instrumentation or, for not intrusive verification, through hardware sniffers—in our case by using the Embedded Trace Macrocell (ETM) of the ARM processor of the target system.
4) Functional verification and non-functional verification.

5.1.3. KPI definition

The definitions of the KPI for the case study 04_TEK are reported in the Table 5.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 5.1: KPI definition (case study 04_TEK)

S	*KPI*	*Definition*	*U*	V_0	$\Delta_T\%$	*P1*	*P2*
1	KPI_1.2	Early identification of design problems R=r/t, where r is the number of defects that require to rework the design or requirements, and t is the total number of detected defects.	Q%	5%	15%	✓	✓
1	KPI_1.3	Worktime T for requirement verification (test) as development time percentage.	W%	33%	20%	✓	✓
1	KPI_5.2	Predictability and conformance as (the reciprocal of) the rework effort ratio R=r/t, where r is the rework effort and t is the total effort.	W%	20%	20%	✓	✓
2	KPI_1.1	Design and implementation gain G as the complement of the percentage of models reuse and automatic code generation with respect to the total design and implementation.	Q%	100%	15%		✓
2	KPI_2.1	Worktime T for test case execution as development time percentage.	W%	16%	15.0%		✓
2	KPI_2.2	Worktime T for test preparation as development time percentage. T = test cases design + environment setup (automatic monitor setup versus developing simulator/emulator to stimulate the SUT and collect events).	W%	17%	40.0%		✓

S = Scenario number
Unit: W% = Work rate, Q = Quantity rate
P1 = Phase 1, P2 = Phase 2
V_0 = Estimated pre-MegaM@Rt2 value
$\Delta_T\%$ = Foreseen Phase 2 improvement

As reference value the case study 04_TEK takes V_0 that is the average value for typical and comparable project and that is used for planning new projects; consequently, the Phase 1 improvement is evaluated as $\Delta_1\% = 100 \cdot (V_1 - V_0) / V_0$.

With respect to the deliverable D5.4, now the case study does not consider KPI_5.1 “Effort for test cases design” that is included in KPI_2.2.

5.2. Development and validation activities

5.2.1. Previous work

Baseline period (task T1.3) activities:

1) preliminary model’s design;
2) execution and verification with the OMNeT++ simulator.

Task T5.2 activities:

1) The design of the first version of the MAC state machines was completed. Conformiq Designer was used for modeling and for the design-time verification with different test coverage strategies.

2) The test cases were manually designed based on the design-time verification whose results are given in graphical mode and through script files. Then the test environment with real targets (actual processing platform: UWB motes) was prepared and the functional verification was executed.

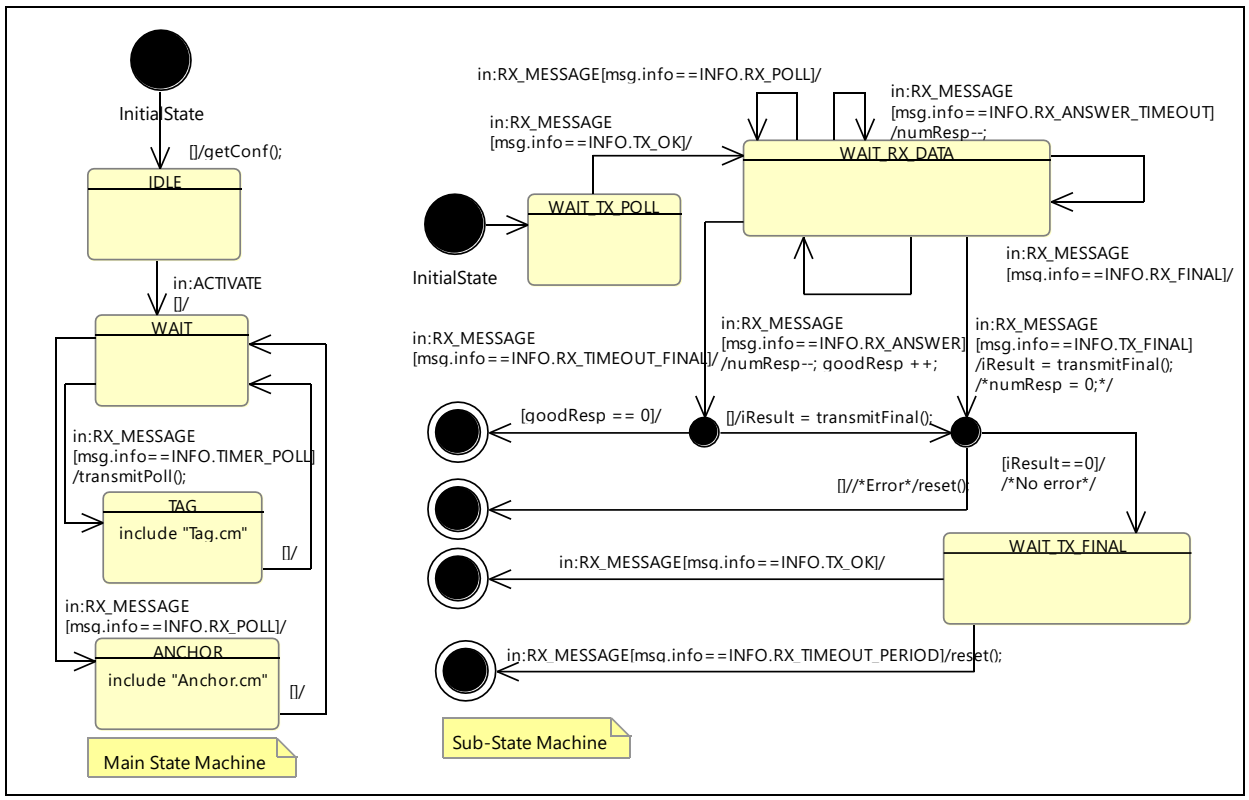

Figure 4.1: Statechart diagram example

With respect to the Scenario 2 (test automation and runtime monitoring) we provided the MAC models designed with Conformiq to RO Technology (RO) and University of L'Aquila (UAQ) and supported them for the initial experimentation of the AIPHS tool on which they are working. During Helsinki hackathon within Phase 1, preliminary activities have been started on an ARM-based target board (Zedboard Zynq-7020), similar to the one foreseen for the UWB node, in order to access, configure and manage ETM registers for runtime tracings.

5.2.2. Development and validation activities

The test/verification process needs the elements for stimulating the SUT (System Under Test) and collecting the events that the latter generates. These elements can be developed in the context of the verification phase, or some steps could be automated in the process that starts with the models and the test design, takes in account requirements traceability and coverage, parametrizes and executes the verification, and interprets the results. Our expectation is for tools that from the design artefacts build such elements that simulate the rest of the world with respect to the SUT (being this a component, or a sub-system, or the whole system).

The state of the art is much more advanced for Web applications and high-level programs that run on a uniform underlying level (e.g. the Java virtual machine). Embedded systems, even if there a convergence toward Linux/Unix like operating systems, pose more problems if only for the required layers to interface peripheral, timers, communication channels. We suppose that the first step is the generation of code that in some manner can be adapted and integrated.

We searched for a tool that takes in input the specifications given by UML statecharts and associated classes definitions and that automatically outputs code for real-time embedded applications in C programming language. We experimented two tools, namely *Yakindu*[7] and *QP*[8] even if they are not among those the project proposes.

Both tools generate C and C++ code that can be integrated with other parts of the application and with the underlying operating system. Quantum Platform provides a more complete framework to support state machines in the case the target is a bare metal system without operating system, moreover it provides also a lightweight solution (named *QP-nano*, in C language) for constrained targets (the complete solutions in C and C++ language are *QP/C* and *QP/C++*).

Yakindu is a Papyrus extension; it was developed firstly for state machines design and verification and then extended for code generation; it offers the advantage of complying with well-established standards. Instead, QP was developed firstly as framework for state machines development in the form of libraries (source code available) and functions templates, then it was completed with a visual tool for statechart design; it has the disadvantage of using a proprietary file format that doesn't allow importing/exporting the models; it addresses practitioners' needs.

Both tools cannot be used independently, especially due to the lack of connection towards model-based testing techniques.

5.3. Results of the first evaluation phase

5.3.1. KPI values

The Table 2.1 shows the preliminary KPI values obtained by the case study 04_TEK during Phase 1.

Table 2.1: Phase 1 KPI values (case study 04_TEK)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_1.2	Q%	5%	4.33%	13.40%	15%
1	KPI_1.3	W%	33%	30.00%	9.09%	20%
1	KPI_5.2	W%	20%	18.10%	9.50%	20%

S = Scenario number — *Unit: W% = Work rate, Q% = Quantity rate*

$\Delta_T\%$ *= Foreseen Phase 2 improvement* — V_1 *= Phase 1 value obtained with Framework initial version*

V_0 *= Estimated pre-MegaM@Rt2 value* — $\Delta_1\% = 100 \cdot (V_1 - V_{REF}) / V_{REF}$ (Phase 1 improvement)

5.3.2. Plan of improvements

5.3.2.1. Feedback to tool providers

The Table 5.3 shows the tool requirements on which the Tekne user requirements were mapped.

Table 5.3: System requirements experimentation in the case study 04_TEK

Identifier	*Definition*	*Experimentation*	*P1*	*P2*
CQDESIGN 110	Conformiq Designer shall support model analysis and debugging facilities.	Model verification (design-time verification).	✓	✓
CQDESIGN 080	Conformiq Designer shall support test generation from models based on user selected testing goals.	Different test coverage strategies.	✓	✓

[7] https://www.itemis.com/en/yakindu/state-machine/

[8] https://www.state-machine.com/

Identifier	*Definition*	*Experimentation*	*P1*	*P2*
CQDESIGN 090	Conformiq Designer shall support visualization of test cases to provide the user an understanding of the test cases purpose and origin.	Usability of the tool.	✓	✓
CQDESIGN 050	Conformiq Designer shall support generation of test plans for manual test execution.	Functional properties manual verification.	✓	
CQDESIGN 010	Conformiq Designer shall provide GUI to support automated testing.	Usability of the tool.		✓
CQDESIGN 020	Conformiq Designer shall support fully automatic generation of test cases from models.	Integration of Conformiq with AIPHS.		✓
CQDESIGN 030	Conformiq Designer shall support generation of executable test scripts.			✓
CQDESIGN 040	Conformiq Designer shall support generation of human readable test documentation.	Usability of the tool.	✓	✓
AIPHS 020	Runtime generation of logs for WCET analysis.	Non-functional properties verification.		✓
AIPHS 030	Runtime generation of logs for performance measurements on targets with multi-core processors, running bare-metal and Linux based applications.	Bare-metal with BIOS to manage the ARM and the UWB transceiver.		✓
AIPHS 040	Generalization of the concept among monitoring infrastructures by defining a general reference architecture that can be adapted to different applications.	Embedded C language for communications.	✓	✓
AIPHS 050	Develop a methodology to suggest the best monitoring mechanisms to be used in a given system, depending on data of interest and constraints to be satisfied.	Integration of AIPHS with other tools.		✓
AIPHS 060	Provide case study support to select the best monitoring infrastructure.	The case study experiments different configurations of the target system.	✓	✓

The Table 5.3 show also the aspect of the tool that the case study 04_TEK experiments (column Experimentation) and in which phase of the project.

5.3.2.2. Work planned for the Phase 2

Additional collaboration with the tool providers — Analysis of Framework capabilities for models import/export and requirements traceability.

Scenario № 1 (design and functional verification) — System requirements refinement; completion of models and their design-time verification (Conformiq), with a focus on how tests case can be derived automatically; test script generation, implementation, functional verification.

Scenario № 2: (runtime and non-functional verification) — Analysis and experimentation of the model-based testing capabilities:

- Produce the test script from the models (Conformiq) and process them with the Generator part of AIPHS (UAQ) for generating the Monitor from the specifications. The Monitor is a library that uses the ARM Embedded Trace Macrocell (ETM) and so is overhead free.

- Connect the Monitor to the SUT. AIPHS configures the Monitor on the basis of the information it extracts from the scripts.
- Collect the events generated by the SUT and analyze them: the Oracle part of AIPHS collects the event generated by the SUT and compares them against the expected behavior.

Both scenarios — The expectation is the automatic generation of (part of) the code. For that we will analyze models import/export possibilities. This capability, which is valuable by itself in terms of development effort reduction, can be exploited halfway between the design and the runtime. We think of the automatically generated code as an executable model. It can run on an emulator (e.g. an ARM processor emulator on a main computer, be it Windows or Linux) and so on one side it enables a model verification closer to the final system, on the other side it allows an early functional verification.

Project wide activities — Evaluate all proposed KPI.

5.3.3. Summary of the progress towards the roadmap

During the Phase 1, we focused mainly on design time and evaluated three out of six KPI working on the first one of the two planned scenarios. On that basis we positively evaluated the impact that MegaM@Rt2 can have on our software in terms of cost, time, and quality. The project is in time with the plan.

We are confident to reach the objective we set, and that better results can be obtained thanks to a full introduction of runtime techniques and a more integrated use of the Framework.

6. Case study 05_NOK – Base Transceiver Station

6.1. Case study overview

6.1.1. System description

The base station architecture enables various HW and SW configurations from small size picocell to huge server clouds. The goal is to create an integrated workflow tool chain from a platform independent perspective, to facilitate the integration of various subsystems and verification of the overall system performance, to meet a set of desired system requirements and, at the same time, taken into account the ever-increasing system complexity. In addition, there will be formal linking between the requirements, the architecture/design documents, test and verification results.

Nokia DevOps brings the word "continuous" to the centre stage of agile development in the form of continuous planning, continuous requirements, continuous programming, continuous integration, continuous testing, continuous monitoring, continuous feedback, continuous delivery, continuous deployment, and continuous operations with a focus on delivering continuous value to the customer, faster!

In Nokia Way of Working (WoW) process evolution towards DevOps (Figure 6.1), Nokia SW production follows Test Driven and Continuous Integration approaches. The SW development is shared across the Feature teams, where all tasks related to the feature development from the requirement specification, SW design, implementation and to the testing are shared across the members of this feature team.

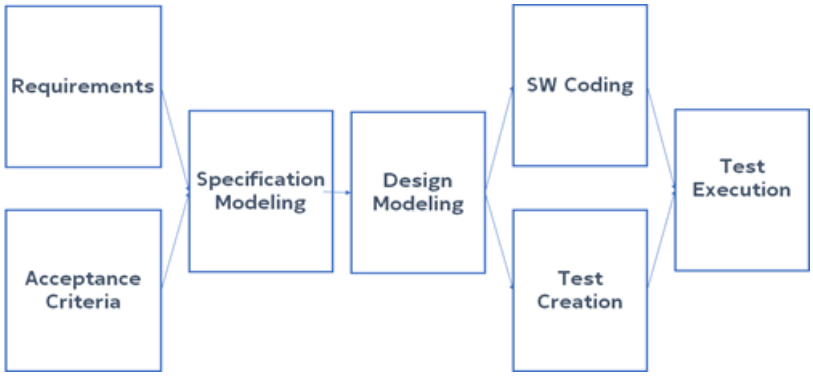

Figure 6.1: Current Nokia Feature Team SW development process

In Megam@Rt2, Nokia's target is to improve the feedback mechanism between Feature teams and between team resources towards Continuous feedback (Figure 6.2).

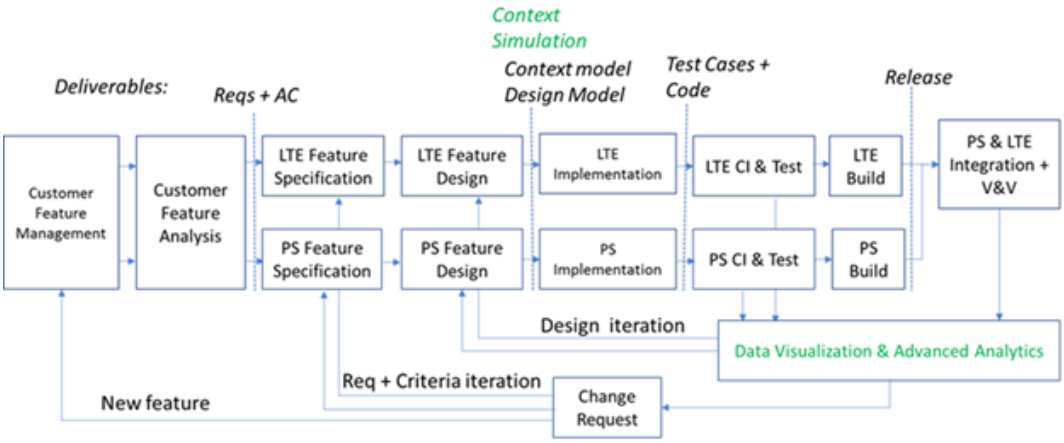

Figure 6.2: Feature team process improvement

Nokia is studying to use data/regression modelling for feedback communication and to link several design environments and their way of working process. Data/regression modelling will be used as visual feedback mechanism between R&D teams.

The Figure 6.3 depicts how Nokia is combining L2 SW and Baseband thermal design using data/regression modelling as fast feedback for iterations and optimization.

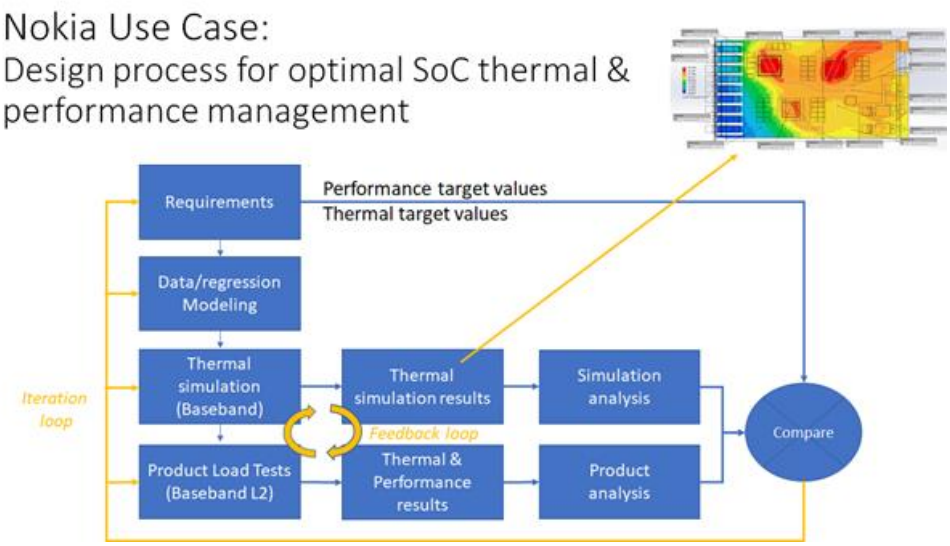

Figure 6.3: L2 user plane performance effects on thermal performance of eNB

6.1.2. Development and validation scenarios

Figure 6.4 depicts how Nokia's use case (Figure 6.3) is divided iin two scenarios, which are linked together by product test data.

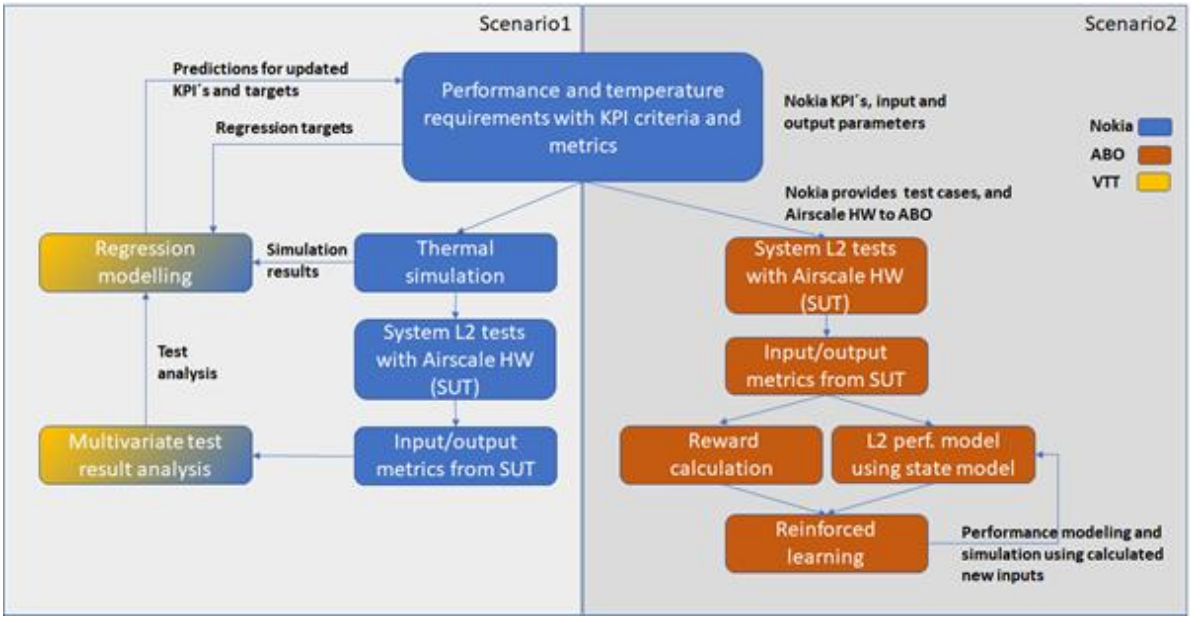

Figure 6.4: Overview of Nokia use case including two scenarios.

The case study is structured according the scenarios defined below.

6.1.2.1. Scenario № 1: Thermal management in small size products

Scenario 1 concentrates on Runtime Validation and specifically on Runtime analytics. Scenario 1 compares and analyses performance and temperature reference/target values from legacy test results against measurement values from latest SW build. Later, when Scenario 1 includes also simulation results, comparison and analysis are done between reference/target, simulation and product test results.

6.1.2.2. Scenario № 2: System mega-Modeling and SW synthesis

Scenario 2 provides modelling methods to find optimal SW and HW performance configurations.

Performance modelling is used to study unknown and interesting parameter conditions/combinations to improve architecture performance.

6.1.3. KPI definition

The definition of the KPIs for the case study 05_NOK are reported in the Table 6.1; the column Δ_T% gives the total improvement that Nokia foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 6.1: KPI definition (case study 05_NOK)

S	*KPI*	*Definition*	*U*	V_0	V_B	Δ_T%	*P1*	*P2*
1	KPI_5.2	Amount of corrected PS faults all releases	Q	148	176	1	✓	✓
1	KPI_5.2	Daily BB SW releasing success rate (PS MB)	%	77	75	23	✓	✓
1	KPI_5.2	Daily Platform SW Main Branch promotions	Q	0.7	1	43	✓	✓
2	KPI_5.1	Specification hours for new development	H	4361	4755	26	✓	✓
2	KPI_5.2	Open Faults in specification	Q	6	5	67	✓	✓
2	KPI_5.2	FDE (Feature Development Efficiency)	%	65	70	23	✓	✓
2	KPI_5.1	Element Feature Specification maturity	Q	33	53	112	✓	✓

S = Scenario number
Unit: % = Percentage, H = Hours, Q = Quantity
P1 = Phase 1, P2 = Phase 2
V_0 = Estimated pre-MegaM@Rt2 value (V_0 at M_0 = Apr 2017)
Δ_T% = Foreseen Phase 2 improvement
V_B = Value obtained in MegaM@Rt2 with baseline tools (M_{15} = Jun 2018)

KPI improvements were mainly based on Nokia internal actions (methods and tools development by several master thesis workers). Amount of corrected PS faults all releases improvement % is low correction amount point of view due to good reference value in *V_0 at M_0 = Apr 2017* when the project started.

6.2. Development and validation activities

6.2.1. Previous work

In Veikka Grekula´s Master Thesis work the service-based modeling approach was studied as a potential alternative to current component-based modelling approach.

MetaEdit+ was studied in thesis work to raise the level of abstraction using a modeling language. Domain Specific Modelling (DSM) and Domain Specific Modelling Language (DSML) were used for modeling the System of Systems (SoS).

As conclusion, the complexity of current BTS system configuration and architecture, SW components models are becoming too extensive to be efficient by means of modeling. To tackle this, SW should be decomposed into a reasonable sized set of services. The future work should take the proposed service-based model into broader use.

6.2.2. Development and validation activities

Nokia has the following task plan for scenarios 1 and 2 in Phase 1:

Target setting for modelling

- Non-functional targets are set in co-operation with Åbo Akademi.
- Targets are specified in requirement analysis phase as acceptance criteria.
- Specific measurements (status, state, warnings, errors, values, KPIs) are selected for each acceptance criterion to be used in simulation and testing.

Model creation

- To specify ReefShark SoC thermal and L2 user plane performance KPI target values.
- In cooperation with Åbo Akademi non-functional L2 performance model will be created with selected non-functional requirements. This model will be used as test case for L2 user plane load in Scenario 2.

- Create data/regression model, combining L2 KPIs and ReefShark SoC Thermal KPIs.

Model Simulation & 1st phase analysis

- Execute simulations to collect simulation results (specific measurements).
- Simulation results are compared against target settings using multivariate methods.
- Iterations on target settings and/or on simulation are done according to analysis results.

Model verification with real HW

- Test execution in real environment (real SW + real HW) & 3rd phase analysis
- Execute tests to collect test results (specific measurements).
- Test results are compared against target settings, simulation and virtual tests using multivariate methods.
- Iterations on target settings and/or on simulation are done according to analysis results.
- To setup measurable acceptance criteria of interfaces between architecture blocks by selecting specific temperature sensors (inside ReefShark SoC, on baseband and system PCB).
- In cooperation with VTT, create short feedback loop using analytics algorithms and advanced data visualization.

During Phase 1, Nokia has integrated two equivalent test systems to L2 performance test environment (see Figure 6.5 for "ÅBO" HW). These test systems are executing L2 performance tests 24/7 controlled by Jenkins. Every night Jenkins checks if new L2 SW release is available, installs that release to both target HWs, and executes the test suite for ~10 times on both target HWs during 24 hours. Test suites include two different traffic profiles, and each profile includes six different input parameter combinations.

Figure 6.5: One of two L2 performance test environments (HW "ÅBO").

After each test suite execution, Jenkins executes Python macro, which collects performance test result to CSV file for regression modelling.

To understand interoperability with selected tool purposes and current Nokia component modelling processes and tools, following modelling trials were made. Nokia tested transferring existing UML models from MagicDraw to Papyrus to enable fast verification, model transfer trial was made with one small component, which was delivered with MagicDraw 2.5 XMI file format towards Papyrus tool. Data transfer was working, but model graphics and block level description were lost. Due to this, there are no benefits to use Papyrus tool, because current activity diagram based modelling do not include necessary activity descriptions (like action language, e.g. ALF), which is necessary for simulations and

for designers, when model based approaches will be taken into use. To get any benefits from usage of Papyrus, Moka and Conformiq tools, it would mean way of working process change in Nokia. For L2 performance evaluation, we would need activity descriptions in functional models for L2 control to enable usage of selected tools.

6.3. Results of the first evaluation phase

6.3.1. KPI values

The Table 6.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 05_NOK.

Table 6.2: Phase 1 KPI values (case study 05_NOK)

S	*KPI*	*U*	$V_0 = M_0$ *Apr 2017*	$V_B = M_{15}$ *Jun 2018*	Δ_B%	$V_1 = M_{22}$ *Jan 2019*	Δ_1%	Δ_T%
1	KPI_5.2	Q	148	176	-19	154	-4	1
1	KPI_5.2	%	77	75	-3	90	17	23
1	KPI_5.2	Q	0.7	1	43	1.2	20	43
2	KPI_5.1	H	4361	4755	9	5181	19	26
2	KPI_5.2	Q	6	5	17	3	50	67
2	KPI_5.2	%	65	70	8	70	8	23
2	KPI_5.1	Q	33	53	61	79	139	112

S = Scenario number
Unit: H = Hour, % = Percentage, Q = Quantity
Δ_T% = *Foreseen Phase 2 improvement*
V_0 = *Estimated pre-MegaM@Rt2 value*
$\Delta_B\% = 100\cdot(V_B - V_0)/V_0$ (baseline improvement)
V_B = *Value obtained with MegaM@Rt2 baseline tools*
$\Delta_1\% = 100\cdot(V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)
V_1 = *Phase 1 value obtained with Framework initial version*

Phase 1 improvements are given with respect to the corresponding values $V_{REF} = V_0$.

Please notice that V_1 values are from the end of January 2019 (instead of end of March 2019).

6.3.1.1. Scenario 1 results

Nokia had first fully automated test results from two L2 performance test environments available during February 2019, so Nokia has only preliminary regression models available to evaluate Scenario 1 results.

Figure 6.6 shows minimum and maximum CPU-load behaviour as a function of activity (percentage of active mobile users sending data compared to maximum number (active + passive) of mobile users in the cell).

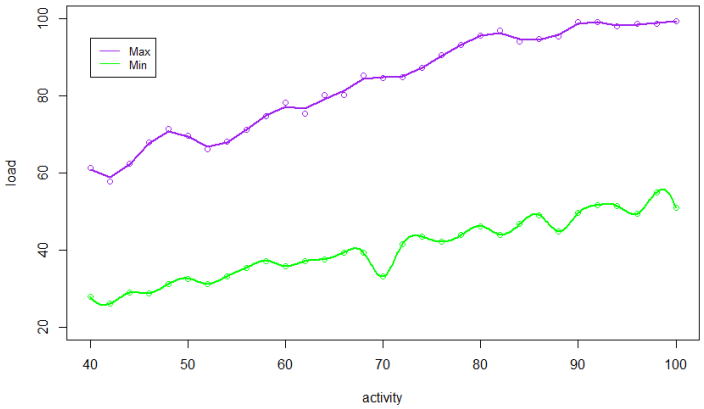

Figure 6.6: Example test results showing minimum and maximum CPU-loads

Figure 6.7 shows when regression model if fitted between maximum and minimum values. Regression model is repeatable data model (i.e. not based on random data), which will be used:

- as Physical Twin (data from physical product);
- to compare against simulation model (Digital Twin);
- to compare against performance targets (typically only maximum or minimum specified);
- to improve simulation model to be more realistic. Typically this improvement needs multivariate analysis to indicate which parameters/attributes cases deviations between Physical and Digital Twins.

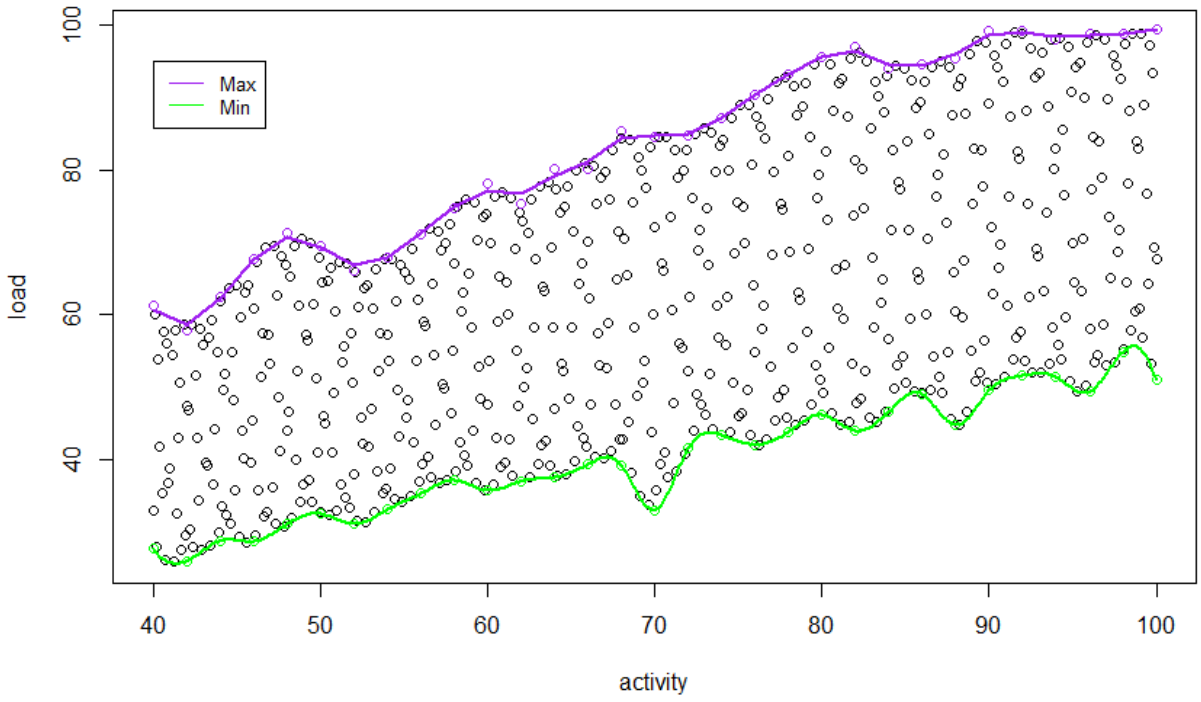

Figure 6.7: Regression model fitting between minimum and maximum loads

Figure 6.8 shows scenario 1 quartal based KPI development from beginning of MegaM@Rt2 project (04/2017) until the latest available official Nokia KPIs (01/2019):

- Daily Platform SW Main Branch promotions:

- MagicDraw tool usage as specification modelling tool is global practise in specification team.
- UML based specification modeling resources have been increased.

- Daily Baseband releasing success rate:
  - Release CI is piloting Code Change Based Test Selection to minimize feedback time from designer's commit to integration test result.
- Amount of corrected Platform SW faults:
  - Code Change Based Test Selection has not yet improved feedback loop so that designers could have more time for SW corrections.
  - Negative percentages during 09/2017-09/2018 were mainly caused by additional work load from platform SW for 5G. Vo reference value (at Mo, 4/2017) is based on 4G (LTE) SW.

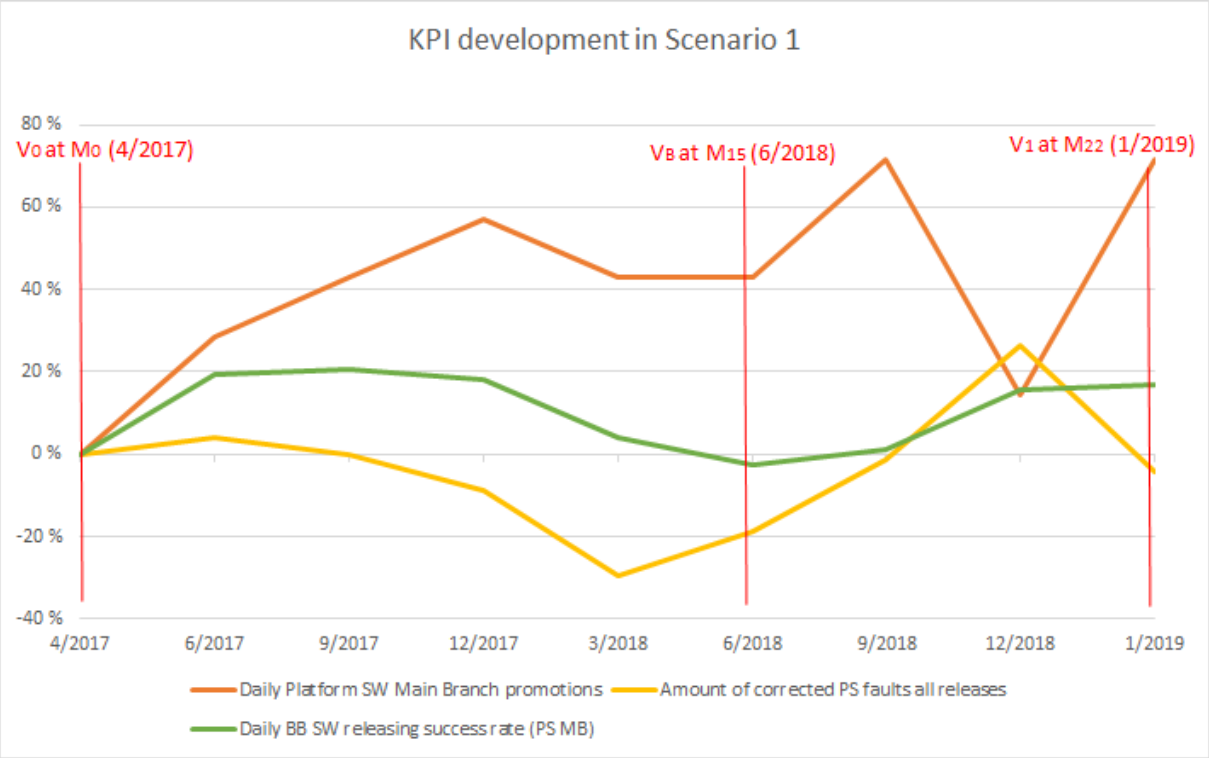

Figure 6.8: KPI development in Scenario 1

6.3.1.2. Scenario 2 results

Figure 6.9 shows scenario 2 quartal based KPI development from beginning of MegaM@Rt2 project (04/2017) until the latest available official Nokia KPIs (01/2019):

- Specification hours for new development:
  - Specification resourcíng capacity has been increased globally to left shift design process.
- FDE (Feature Development Efficiency):
  - Has been improved with usage of MagicDraw tool globally transferring specification creation process from textual to graphical format ensuring better quality and faster information transfer between teams.

- Element feature specification maturity:
    - Quality of SW development process has been improved with modelling tools. Tools provides automated checking results to improve quality of reviews between teams ensuring better SW quality in SW component commits and releasing.
- Open Faults in specification:
    - is not included in Figure 6.9, because it's huge variations in percentage is caused by small numerical values. Similar behavior (huge percentages) is also visible in Element Feature Specification maturity.

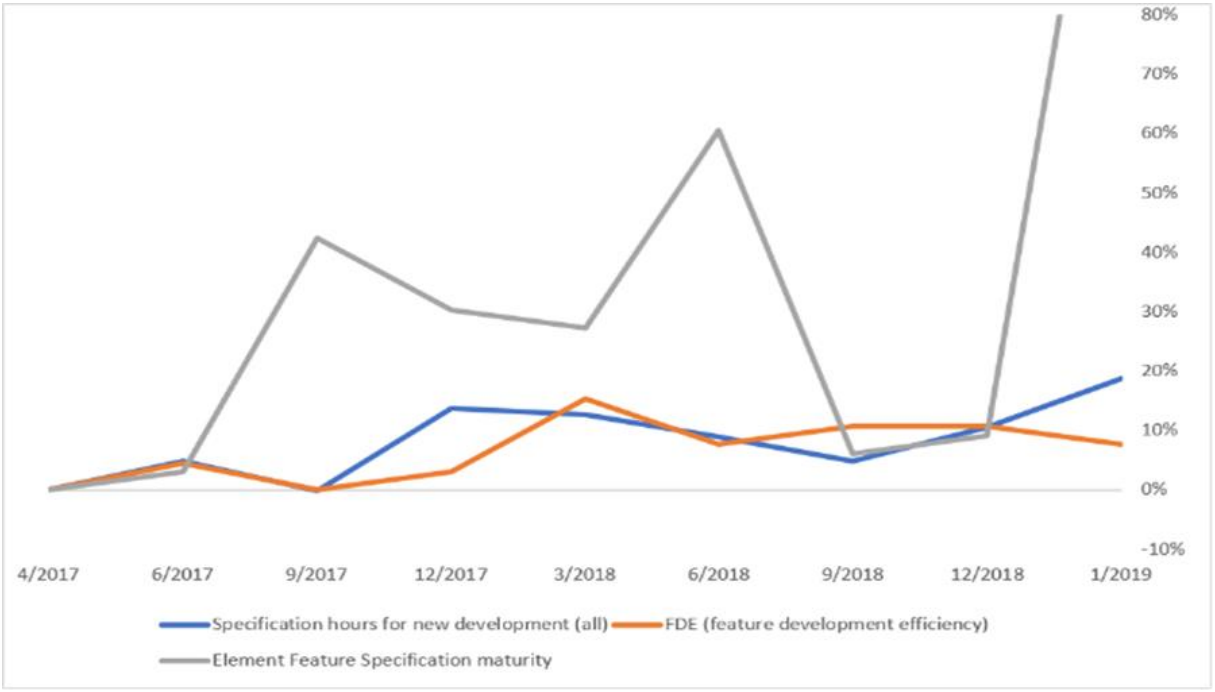

Figure 6.9: KPI development in Scenario 2

6.3.2. Plan of improvements

6.3.2.1. Requirement coverage

Covered in chapter 6.3.2.2.

6.3.2.2. Feedback to tool providers

Nokia is still waiting for deliverables from VTT and Abo Akademi, so there is no possibility to evaluate requirement coverage against tool purposes.

For future projects (e.g. MegaM@Rt3) it is necessary to have at least interim tool purposes available for Phase 1, so that the case study providers are able to give initial feedback to tool providers.

In Nokia case majority of necessary tool purposes are available in M32, so evaluation of Phase 2 results on risk.

6.3.2.3. Work planned for the Phase 2

There are no changes planned for Scenario 1 work, so initial Phase 2 plan as described in Chapter 8.3.2.4 of D5.4 is still valid.

6.3.3. Summary of the progress towards the roadmap

Nokia´s initial MegaM@Rt2 scope (Nokia Vision), see Figure 6.10, was planned to improve process flow for R&D. This improvement starts from Acceptance criteria specification phase and takes a critical look at its contents. It must be verified that SW Design domain model contains the information that is needed in testing as an input for automated test generation. That information does not necessarily include testing specific data, which is typically modelled in test phase domain models. From the process flow point of view possible automated transformations shall be developed towards automated transformation when regarded feasible. The focus is in testing and traceability, because the requirements information is needed in automated test result analysis. Nokia is expecting a creation of efficient Model Based System Design (MBSD) Way-of-Working (System of Systems), which enables Executable Acceptance Test Driven Development (EATDD) for Lean SW development.

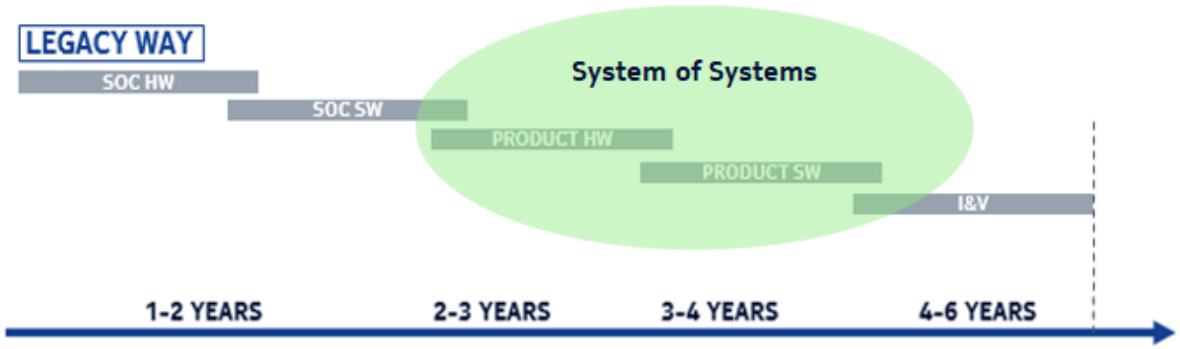

Figure 6.10: Nokia initial vision for MegaM@Rt2 (System of Systems)

During Phase 1, Nokia WoW focus has changed to improve Time to Market aspects and Scaled Agile Framework (SAFe), see Figure 6.11, so that products could get out to our customers in 15 months after product content decision instead of 4-6 years.

Having Modeling & Simulation as a de-facto design technique as a part of Nokia R&D process, we could achieve this. That would need a totally new and different approach – Nokia calls it as virtual platform initiative or modeling and simulation platform for both SW and HW.

Desired state:

- The core of the modeling and simulation is not the models and simulation themselves, but 'Problem Statement', i.e. we have a problem (or requirement) and we want to solve it early enough by modeling and simulating and analyzing the results.
- We have created a process framework for modeling and simulation with tight interlock with operation research (discipline that deals with the application of advanced analytical methods to help make better technical and management decisions).
- Interoperability across different HW/SW simulation tools.
- We have defined the specific phases horizontally/vertically, when tools can be and must be used with expected results and data for decision making.
- Nokia expects to find and produce algorithms for automated test result and root cause analysis. Nokia's assumption is that 30% less time (efficiency gain) is needed during R&D, because of modelling, leading to 40% improvement in productivity. Similarly, 60% less time (efficiency gain) is needed in testing, because of model based approach, leading to 50 % improvement in productivity.

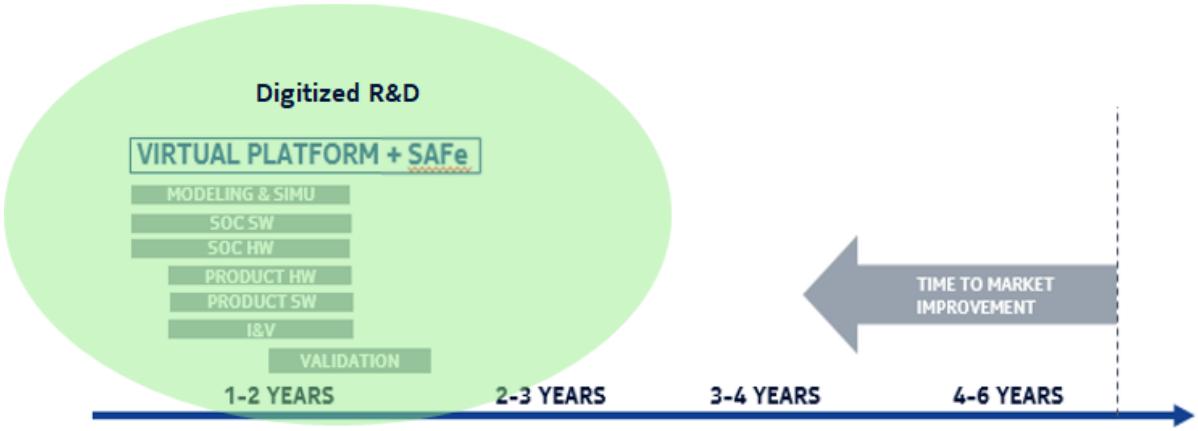

Figure 6.11: Nokia Digitized R&D vision for MegaM@Rt2 and MegaM@Rt3

7. Case study 06_BT – Train Control and Management System

7.1. Case study overview

7.1.1. System description

Bombardier Transportation Sweden AB (BT) proposes the case study 06_BT "Train Control and Management System" that is centered around the BOMBARDIER MITRAC Train Control and Management System (TCMS) platform. MITRAC is a high capacity infrastructure backbone, built on open standard IP-technology, that allows easy integration of all control and communication requiring functions on-board the train. More details can be found in deliverable D1.1.

The main TCMS systems challenges that are investigated in MegaM@Rt2 are related to (1) variability, (2) an efficient traceability support between architecture, design and runtime artefacts, and (3) testing.

7.1.2. Development and validation scenarios

The aspects of the development cycle that Bombardier aims to improve thanks to its participation in MegaM@Rt2 are summarized below.

As described in the deliverables D5.4 and D1.3, the main Bombardier expectation from MegaM@Rt2 is to provide methods and tools to support the develop a Platform Management System that can handle the configurability of the overall TCMS system, including requirements documentation, design documentation, test artefacts, artefacts needed for assessment, etc. The Bombardier expectation corresponds to the following MegaM@Rt2 framework requirements:

- MTM-01200 – Model versioning capabilities over large sets of interrelated models.
- MTM-01300 – Model access-control capabilities over large sets of interrelated models.

The Bombardier expectation to provide an efficient traceability support (between architecture, design and runtime artefacts) corresponds to the following MegaM@Rt2 framework requirements:

- MTM-01000 – Base model management capabilities.
- MTM-02000 – Traceability capabilities between design/system models and runtime models.

The Bombardier expectations for TCMS systems testing correspond to the following MegaM@Rt2 framework requirements:

- RTA-40001 – The framework should provide support for test selection.
- RTA-40002 – The framework should provide support for test generation.
- RTA-40003 – The framework should provide support for test execution.
- RTA-40004 – The framework should provide support for test analysis, including test coverage.
- RTA-40007 – The system should provide support for runtime simulators.

One of the big challenges regarding platform development today is to handle the varying interfaces. Modules are highly depending of each other. Changing a parameter in one module influences highly the behaviour of other modules. Nowadays this prevents to develop and test the MITRAC platform independently from the specific project.

The case study is structured according the scenarios defined below.

7.1.2.1. Scenario № 1: Platform and Variability Management

Platform and Variability Management concerns the investigation of innovative and automated ways to cope with variability in the development of the TCMS system. The expectation of MegaM@Rt2 is to provide methods to support to develop a Platform Management System that can handle the configurability of the overall TCMS system, including requirements documentation, design documentation, test artefacts, artefacts needed for assessment, etc. More specifically, Bombardier goal is to analyse how MITRAC TCMS platform can exploit variant and variability management of product lines tools. The aim is to develop a variability model that can be applied to TCMS. In order to describe and handle the development of the TCMS platform, a TCMS platform management system is needed. The TCMS System is a complex system which can consist of up to 20 subsystems. Each subsystem is allocated to several vehicle functions, each function can have many different variability points: for example, number of doors, strength of brakes, types of brakes, number of cars, etc. These variability points go down to very technical details that define the behaviour of the train for very different customers and environments.

7.1.2.2. Scenario № 2: Traceability management - software development

This scenario is related to the following Bombardier expectations.

- Developing methods to enable platform design (Scenario 2.1). As described in D1.3, the main Bombardier expectation (from MegaM@Rt2) is to provide a method that establishes guidelines on how to structure the building blocks, how to define interfaces, how to integrate platform building blocks with project specific building blocks by minimizing the impact that project specific code has on the platform building block.
- Design and automatic test code generation (improvement of automatic test code generation, Scenario 2.2). The vision is that the gap between the requirements, design and testing can be closed. In order to achieve this, the aim is to transform the customer documents into software requirements and then into test code.

7.1.2.3. Scenario № 3: System Testing

Testing concerns the investigation of innovative methods for test case generation, test case selection, and test results evaluation. More specifically, this scenario is related to the following Bombardier expectations.

- Generate test cases / test scenarios out of requirements models (Scenario 3.1). As described in D1.3, the expectation of Bombardier is that MegaM@Rt2 provides methods on: (1) how to design requirement models such that test cases can be derived; (2) how test cases for module testing can be derived; (3) how test cases for higher level integration tests can be derived.
- Generate test simulator based on the models (Scenario 3.2). Bombardier aims at designing TCMS systems models in an efficient way such that they provide the information that is required to derive the subsystem simulator. The TCMS systems design needs to satisfy time and budget constraints in project, involving subsystems to be simulated with a number of requirements in the scale of 3000-5000 and number of signals in the scale of 40000.
- Prioritize test scripts (Scenario 3.3). Bombardier expectations are related to methods: (1) on how to choose code modules to test that are under high risk; (2) on how to choose the best test cases (correct level of detail testing) that with a high probability would find errors in the code; (3) on how to schedule test cases over the time of the project such that at any given time the correct prioritization is ensured; (4) on how to schedule test cases over the time of the project such that at any given time the coherence of the code is given (overall integration test).

- Generate test scripts (Scenario 3.4). The expectation is that MegaM@Rt2 provides a method on how the generation of test scripts out of test cases can be automated.
- Automatic test result evaluation (Scenario 3.5). The expectation is that MegaM@Rt2 provides a method on how to improve the analysis of test results and test error root cause analysis based on the test results.

7.1.3. KPI definition

The definitions of the KPI for the case study 06_BT are reported in the Table 7.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 7.1: KPI definition (case study 06_BT)

S	KPI	Definition	U	V_0	V_B	$\Delta_T\%$	P1	P2
1	KPI_1.1	Design Effort: Time needed to design a pantograph function	H	50	40	20	✓	✓
1	KPI_5.2	Review Effort: Time needed to review a pantograph function	H	20	15	75	✓	✓
2	KPI_1.1	Time to design the architecture of TCMS	H	100			✓	✓
2	KPI_1.2	Requirement bug detection: Requirement bugs found in the design of the pantograph function	N	2			✓	✓
2	KPI_1.3	Time to review the design of the pantograph function	H	50			✓	✓
2	KPI_3.1	Time to manage the model of the pantograph function during development	H	50			✓	✓
2	KPI_3.3	Time to integrate the pantograph function with other functions	H	30			✓	✓
3	KPI_5.2	Fault detection: number of injected or naturally occurring faults detected in each sw component	M	60			✓	✓
3	KPI_1.1	Time spent on software testing for a master controller function	H	8			✓	✓

S = Scenario number
P1 = Phase 1, P2 = Phase 2
$\Delta_T\%$ = Foreseen Phase 2 improvement
Unit: H = Hour, N= Number of bugs, M= Average Mutation Score (%)
V_0 = Measured pre-MegaM@Rt2 value
V_B = Measured Value in MegaM@Rt2 with proposed tools (M_{24} = 3/2019)

As reference value the case study 06_BT takes V_0 that is the pre-MegaM@Rt2 value and that results in baseline measurements using the environment, tools and processes at BT; consequently, the Phase 1 improvement is evaluated as $\Delta_1\% = 100\cdot(V_1 - V_0)/V_0$. A given KPI can be measured more than once but on different scenarios.

7.2. Development and validation activities

7.2.1. Previous work

Tool 1: Pure::variants tool covers the requirements variability. The goal was to design a variability model that could be used to model several aspects of variability, in other words, is it possible to create one model that can be used for several different but similar projects. We managed to create such model and we showed, using analysis, that it is possible to design a model which can be used for handling several aspects of requirement variability. The method developed does not provide any means for directly processing the requirements written in natural language and then extracting

relations and dependencies from it. Future work should address this problem and try to find the way of using machine learning algorithms to automatically identify relations, dependencies, and attributes in the requirements documents and then extract them and import into the Pure::variants model. The measurement of KPIs has been performed before applying the tools in scenario 1.

Methodology 1: MetaEdit+ (by using EAST-ADL architectural models) and Modelio (SysML). Constellation/Modelio has been selected because it has capabilities to create SysML/EAST-ADL models.

Methodology 2: Matlab Simulink and SysML. An open challenge that we are facing is tool integration with Matlab Simulink. More specifically, we are working to resolve the issue that SysML-based tools do not provide any direct mechanism for solving or evaluating constraint equations and model simulation. We are looking at linking these two specifications or just use Simulink/Stateflow if possible for modelling high level requirement.

7.2.2. Development and validation activities

The tools used are evaluated with regard to important KPIs for variability management in the development of complex safe-critical control systems. The real-world Bombardier MITRAC TCMS platform will be used. Such platform allows to create complex TCMSs which can consist of up to 20 subsystems. Each subsystem is allocated to several vehicle functions, each function can have many different variability points: for example, number of doors, strength of brakes, types of brakes, number of cars, etc. These variability points go down to very technical details that define the behavior of the train for very different customers and environments.

7.3. Results of the first evaluation phase

The previous results can be summarized as follows:

1) An approach to automatically generate block diagrams (e.g. Function Block Diagram language or Simulink) from requirement documents and then with the use of pure::variants functions, to create various aspects which are then further transformed into feature models.
2) The results of an automatic transformation made possible the identification between core and variant features presented in textual requirements making it easy to define what parts of the software are project specific and what are common for all generated models.
3) The results indicate that variability modelling using the pure::variants tool is applicable for requirement variant handling in the railway domain.
4) The results show that for the two requirement models considered 39% of the features are consistent and similar to each other. The requirement models in two different projects contain a high number of variation points.
5) No direct and fully automatic mechanism have been found to link Simulink artefacts and SysML models.
6) Semi-automatic and manual traceability between diagrams is supported by Modelio for SysML, MetaEdit+ for EAST-ADL and MATLAB Simulink. Three types of diagrams have been considered: structural (22%), functional (63%), and extra-functional (15%).
7) We applied model-based testing using three different tools: Simulink Design Verifier, Conformiq Designer and Conformiq Creator.
8) MDH implemented two tools for model-based testing: LMBT and ClassTree and these tools have shown to be applicable and efficient in terms of test generation time and number of generated test cases compared to manual testing.

9) The test generation time depends on the complexity of the model, the test design configuration and the Conformiq Creator algorithmic settings. In the time spent on creating automatic test cases, we also consider the upfront time for learning the modeling tool and modeling language. To conduct this case study, a student spent 5 weeks on learning the modeling language. Approximately, for the Master Controller 3–4 hours are spent to write the test cases, understanding the system models and the test goals. The Master Controller requirements are covered by 2 test suites, and roughly 8 hours are spent on writing these test cases.

10) Smartesting, MDH and Bombardier worked on creating a test model in UML/OCL for runtime test generation. An example of a part of the test model is shown in Figure 7.1.

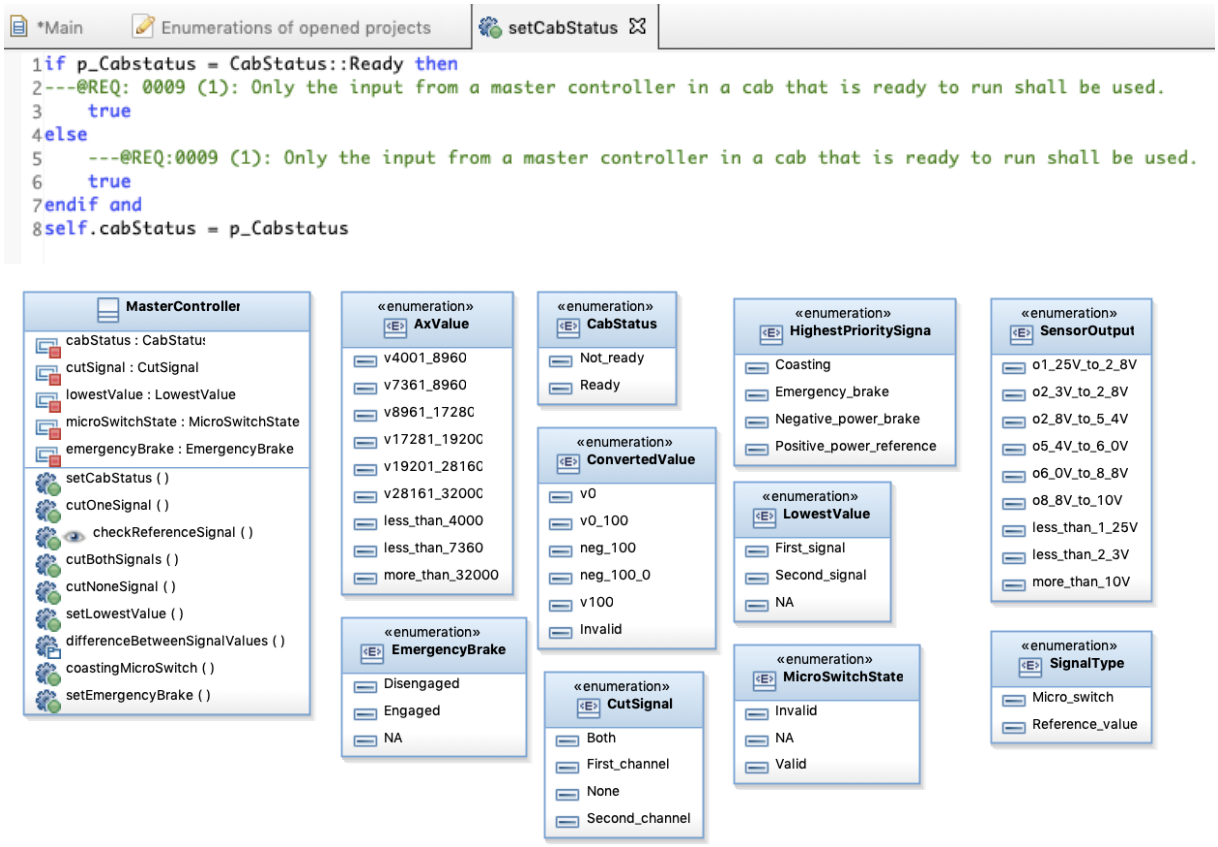

Figure 7.1: UML Class Diagram, and a Master Controller Example of an OCL code for an operation.

7.3.1. KPI values

The Table 7.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 06_BT.

Table 7.2: Phase 1 KPI values (case study 06_BT)

S	*KPI*	*U*	V_0	V_B	$\Delta_B\%$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_1.1	H	50	40	20	25	50	50
1	KPI_5.2	H	20	15	25	10	50	50
2	KPI_1.1	H	100			50	50	50
2	KPI_1.2	N	2			10	80	80
2	KPI_1.3	H	50			30	40	50
2	KPI_3.1	H	50			10	80	80
2	KPI_3.3	H	30			20	33	50
3	KPI_5.2	M	60			90	50	50
3	KPI_1.1	H	8			7	12	10

x = not measured
S = Scenario number
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
$\Delta_B\% = 100\cdot(V_B - V_0)/V_0$ (phase 1 improvement)
$\Delta_1\% = 100\cdot(V_1 - V_0)/V_0$ (Phase 2 improvement)
Unit: D = Day, H = Hour
V_0 *= measured pre-MegaM@Rt2 value*
V_B *= Measured Value in MegaM@Rt2 with proposed tools* (M_{24} *= 3/2019)*
V_1 *= Estimated Value in MegaM@Rt2 with proposed tools* (M_{36} *= 3/2020)*

7.3.2. Plan of improvements

7.3.2.1. Requirement coverage

The Bombardier expectation corresponds to the following Model & Traceability Management (MTM) requirements:

- MTM-01200 – Model versioning capabilities over large sets of interrelated models.
- MTM-01300 – Model access-control capabilities over large sets of interrelated models.
- MTM-01000 – Base model management capabilities.
- MTM-02000 – Traceability capabilities between design/system models and runtime models.

And to the following Runtime Analysis Tool Set (RTA) requirements:

- RTA-40001 – The framework should provide support for test selection.
- RTA-40002 – The framework should provide support for test generation.
- RTA-40003 – The framework should provide support for test execution.
- RTA-40004 – The framework should provide support for test analysis, including test coverage.
- RTA-40007 – The system should provide support for runtime simulators.

7.3.2.2. Feedback to tool providers

Bombardier is evaluating several model-based testing tools, variability management tools as well as MBSE tools. We would like to extend the use of these tools for runtime models and runtime verification. In this way we could measure other relevant KPIs during the later stages of development.

7.3.2.3. Work planned for the Phase 2

For the Phase 2 we will use the real-world Bombardier MITRAC TCMS platform, as we did in Phase 1.

The tools used in Phase 2 will be evaluated with regard to the KPIs.

Scenario 3.1 – Generate test cases / test scenarios out of requirements models. As described in D1.3, the expectation of Bombardier is that MegaM@Rt2 provides methods on: (1) how to design requirement models such that test cases can be derived; (2) how test cases for module testing can be derived; (3) how test cases for higher level integration tests can be derived.

Scenario 3.2 – Generate test simulator based on the models. Bombardier expectation is that MegaM@Rt2 provides a method on how design system models in an efficient way such that they provide the information that is required to derive the subsystem simulator, where 'efficient' is to be understood that they can be designed given the time and budget constraints in project - involving subsystems to be simulated with a number of requirements in the scale of 3000-5000 and number of signals in the scale of 40000.

Scenario 3.3 – Prioritize test scripts. Bombardier expectation is that MegaM@Rt2 provides methods: (1) on how to choose code modules to test that are under high risk; (2) on how to choose the best test cases (correct level of detail testing) that with a high probability would find errors in the code; (3) on how to schedule test cases over the time of the project such that at any given time the correct prioritization is ensured; (4) on how to schedule test cases over the time of the project such that at any given time the coherence of the code is given (overall integration test).

Scenario 3.4 – Generate test scripts. The expectation is that MegaM@Rt2 provides a method on how the generation of test scripts out of test cases can be automated.

Scenario 3.5 – Automatic test result evaluation. The expectation is that MegaM@Rt2 provides a method on how to improve the analysis of test results and test error root cause analysis based on the test results.

The tools used in Phase 2 will be evaluated with regard to the KPIs for traceability management - software development - of complex safe-critical control systems.

Scenario 2.1 – Developing methods to enable platform design. As described in D1.3, currently, the writing of requirements, the design document and the implementation using FBD (Function Block Diagram) design language (part of the IEC 61131-3 standard) are manually handwritten. The requirement document and the design document can be manually linked to the Bombardier requirement management tool, DOORS, the implementer adds manually requirements or design requirements references to the code. There is thus a gap between requirements, design and code. Any changes made in the requirements implies a manual update to the design which implies a manual change of the IEC 6113 model, and a manual change of the test artefacts. As described in D1.3, the main Bombardier expectation (from MegaM@Rt2) is to be provided with methods and guidelines on how to structure the building blocks, how to define interfaces, how to integrate platform building blocks with project specific building blocks by minimizing the impact that project specific code has on the platform building block.

Scenario 2.2 – Design and automatic code generation (improvement of automatic code generation). The vision is that the gap between the requirements, design, coding and testing can be closed. In order to achieve this, the aim is to break the customer documents down to software requirements. These models shall be refined in design models. These design models shall then be refined to a level that code can be generated automatically.

7.3.3. Summary of the progress towards the roadmap

The progress measured during Phase 1 has been done in pilot projects inside Bombardier from variability management, and modelling at design and runtime to the process of generating test cases from specification models representing system requirements and the desired functionality. For example, the generated test cases were executed on the system under test in an attempt to obtain a pass or fail verdict. While different MBT techniques have been evaluated, only few of them target the

real-world industrial embedded system domain and show evidence on its applicability. We showed during Phase 1 that there is a serious need to investigate the use of the tools in this project and the evidence on how modeling and verification and validation can improve the current way of manually creating test cases based on natural language requirements. The ongoing investigations was carried out to improve the current processes by using an MBSE approach within an industrial context. Our results suggest that the evaluated tools are useful for describing components and functions in TCMS. One of the fundamental results from these evaluations is that the use of model-based testing results in a smaller number of test cases compared to manual testing performed by industrial engineers, leaving more time for exploratory testing.

8. Case study 07_VCE – Engine Control

8.1. Case study overview

8.1.1. System description

The Volvo Construction Equipment (VCE) 07_VCE “Engine Control” case study concerns the engine platforms that are used in Volvo CE machinery. Engines based on the same platform are reused between different machinery. The base engine is developed by a technology provider within the Volvo Group. Back at Volvo CE, the engine platform is then adapted for specific applications. The complete engine platform is a highly complex system and therefore we focus on a subsystem of the engine platform. The Engine after-treatment System (EATS) is part of the engine system with the responsibility to reduce exhaust emissions based on the legislation region in which the machine is used.

8.1.2. Development and validation scenarios

In Megam@Rt2, VCE focuses on the following challenges it encounters while developing the engine systems:

- Variability creates many challenges, including variability in requirements, design specifications, software configurations and especially in software calibrations. Furthermore, test cells in which physical engines are tested and calibrated need to be adapted for each application variant.
- Model-based testing. This work proposes a methodology for effective model-based testing at the architectural level for the automotive domain. For example, EAST-ADL architecture specification provides a solid foundation for developing a plan for testing at this level. We use architecture-based test models, criteria, and techniques tailored to the automotive domain. Architectural (integration) -based tests, developed using these architectural models, can be used to assess the architecture itself or to test the implementation conformance with the architecture. This methodology aims at detecting defects earlier in the development lifecycle and reducing testing costs.
- The third scenario, model comprehensibility analysis will allow us to analyse various SysML diagrams developed on different levels. The aim is to analyse how different stakeholders interact with the models. The usability of models will be evaluated with respect to correctness of comprehension and time to accomplish certain comprehension tasks.

The case study is structured according the scenarios defined below.

8.1.2.1. Scenario № 1: Variability modeling

The initial object for scenario 1 was the Urea Dosing System (UDS), now the scenario is scaled up to include the complete engine EATS.

Based on requirements and design specifications, adaptations of the current system are created. Currently, there is no systematic method to define when the adaptation or new feature is specific only for one application or for the complete range of product platforms. The lack of mechanisms for variability management causes situations in which requirements and design specifications are recreated and defined several times for very similar applications. It is at the discretion of system architects to plan reuse and predict whether a function might be used in other applications as well. The mentioned specifications, before MegaM@Rt2, were mostly natural language documents supported by graphical illustrations.

8.1.2.2. Scenario № 2: From modeling to test case generation

This scenario considers the stakeholder's requirements, needs, and concerns written in some natural language text as an input and develops models that can be used to automatically create test cases. We intend to use an MBT tool, Conformiq, to model the behavior and structure of the functions existed in VCE.

EAST-ADL Model Testing Concept used in this scenario:

- Feature Model Testing (Vehicle Testing). Testing as it is perceived externally.
- Abstract Functional Testing (Analysis and Design Levels). Testing from a functional point of view.
- Timed Model-Based Testing (Analysis Design Level). Testing from a timing point of view.
- Functional Model-Based Testing (Design Level). Testing from a detailed architectural functional allocation to the hardware architecture.
- Performance Testing (Design and Implementation Level). Stress testing and non-functional testing.
- Code-Based Testing (Implementation Level). Code coverage-based testing and mutation testing.

8.1.2.3. Scenario № 3: Model comprehensibility

In this scenario, VCE focuses on understanding the steep learning curve and interpretation of the SysML diagrams to derive various artefacts at different levels of the development process. This helps us to investigate whether we are capitalizing the models (e.g., analysis, structural, behavioral) in different levels of the test process. With this scenario we want to understand if the model and model views address needs of different stakeholders. We need to make sure that we are not neglecting some of the stakeholders and not utilizing models in the right context.

8.1.3. KPI definition

The definitions of the KPI for the case study 07_VCE are reported in the Table 8.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 8.1: KPI definition (case study 07_VCE)

S	*KPI*	*Definition*	*U*	V_0	V_B	$\Delta_T\%$	*P1*	*P2*
1	KPI_1.1	Reduction in time for specifying auxiliary engine components	W			20.0%		✓
1	KPI_1.2	Reduction of design problems	Q%			10.0%		✓
1	KPI_1.4	Reduction of specification analysis time during reuse	W			20.0%		✓
2	KPI_5.2	Fault Detection: Number of injected or naturally occurring faults detected	Q%	60%	90%	33%	✓	✓
2	KPI_1.3	Number of test cases Created	Q	77	8	90%	✓	✓

S = Scenario number
Unit: W = Work, Q = Quantity, Q% = Quantity Rate
P1 = Phase 1, P2 = Phase 2
V_0 = Estimated pre-MegaM@Rt2 value
$\Delta_T\%$ = Foreseen Phase 2 improvement
V_B = Value obtained in MegaM@Rt2 with baseline tools

8.2. Development and validation activities

8.2.1. Previous work

Scenario 1. initially for the baseline experiment, the Urea Dosing System was modeled and analyzed in order to familiarize with the MegaM@Rt2 technologies. The initial experiment was to use Modelio (SOFTEAM) and EMF Views (ARM) and Feature IDE (not part of MegaM@Rt2) in order to model variability in the UDS. The technology was not available to fulfill the VCE requirements at that time. Then the same study was performed with a Volvo CE internal tool, which provided the necessary technology. The study was then expanded to a larger scope, to include other subsystems other than the UDS, however, several challenges have been faced.

8.2.2. Development and validation activities

Based on the report in D5.4, Softeam (Modelio) reached out to VCE and presented their current implementation of variability modeling which should fulfill VCE requirements. Initial efforts are being done to investigate the Modelio tool capabilities. Furthermore, work is being done in order to measure the KPI's defined for Scenario 1.

Scenario 3 does not define any KPI measurements, however, work is ongoing to set-up an experiment for model comprehensibility analysis. This will help VCE to understand how different stakeholders interact with models and how modeling affects their day to day activities.

8.3. Results of the first evaluation phase

The Baseline experiment (initial setup documented in D1.3) was performed on a subsystem of the EATS, the Urea Dosing System (UDS). That was run as a pilot study for VCE and baseline experiment for MegaM@Rt2. Only then it became obvious the scale of variability within the engine system.

The baseline experiment had shown some limitations in the MegaM@Rt2 toolchain regarding variability modeling. The study was then expanded to include more engine subsystems; however, it was done with the Volvo internal tool for modeling. There, several challenges have been identified and reported in D5.4:

- complexity;
- scalability;
- asset library;
- evolution of assets.

Conformiq MBT Tool: Conformiq Creator and Designer (CONFORMIQ) has been used for creation of SysML models and automated generation of test cases. It is suitable as it fully supports SysML and UML models which were necessary for the generation of test cases and also ensures coverages. Our initial investigations suggested that the methodology applied for deriving the test cases is applicable.

For Scenario 3, the evaluation has not yet started as modeling activities are still ongoing. When a representative model of the system is created, it will be possible to perform the comprehensibility analysis.

8.3.1. KPI values

The Table 8.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 07_VCE.

Table 8.2: Phase 1 KPI values (case study 07_VCE)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_B	$\Delta_B\%$	V_1	$\Delta_1\%$	$\Delta_T\%$
2	KPI_5.2	Q%	60	90	33%	x	x	33%
2	KPI_1.3	Q	77	8	90%	x	x	90%

S = Scenario number
Unit: W = Work, Q = Quantity, Q% = Quantity Rate
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
V_0 *= Estimated pre-MegaM@Rt2 value*
$\Delta_B\% = 100\cdot(V_B - V_0)/V_0$ (baseline improvement)
V_B *= Value obtained with MegaM@Rt2 baseline tools*
$\Delta_1\% = 100\cdot(V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)
V_1 *= Phase 1 value obtained with Framework initial version*
x= no measurements are performed

For Scenario 1, baseline tools within the MegaM@Rt2 toolchain did not provide the functionality which was needed for the scenario. Therefore, no Phase 1 measurement has been performed. The functionality has been provided recently and the KPI evaluation with MegaM@Rt2 needs to be done in Phase 2.

8.3.2. Plan of improvements

8.3.2.1. Feedback to tool providers

Modelio (SOFTEAM) has addressed the requirements from VCE for variability modeling regarding the variability modeling (*MODELIO-160*). VCE now needs to evaluate the tool functionality and provide further feedback.

8.3.2.2. Work planned for the Phase 2

Scenario 1 (Variability modeling). Based on the results presented in D5.4, a methodology for variability modeling needs to be defined which will address the challenges mentioned in 8.3

The MegaM@Rt2 tool chain received capabilities (Modelio – SOFTEAM) based on VCE requirements for Scenario 1. The Phase 1 study will be, if possible, performed in the new version of Modelio and an analysis will be performed. The main focus, however, will be to define a variability modeling methodology suitable for the engine domain.

Scenario 2 (From modeling to test case generation). Our initial investigations suggested that the methodology applied for deriving MBT test cases is applicable and efficient, and future investigations needs to be conducted to:

- Investigate the effectiveness and efficiency of the MBT test-case generation.
- Semi-automatically generate diagrams out of CAF specification to reduce the effort of creating test models.
- Investigate the use of complex data types and timing aspects into the test model.

Scenario 3 (Model comprehensibility). For this scenario, the aim of VCE is to understand how different stakeholders interact with models, how modeling affects their day-to-day activities and estimate the long-term effects of models by assessing the needs of different stakeholders.

8.3.3. Summary of the progress towards the roadmap

Until now work has been focused on Scenario 1 and 2. For Scenario 1, we have been working to define a product line engineering development process and method suitable for the Engine Controls. The initial experiments for this scenario were performed with Volvo CE internal tools since the MegaM@Rt2 toolset was not ready until now. Now that we have received the functionality which was needed, we are setting up experiments to evaluate the new tool capabilities. Scenario 2 is focusing on

model-based testing and using MegaM@Rt2 tools shows good initial results. Scenario 3 has been paused until now as we have been working extensively on migrating form document-centric to model-driven development. We plan to work on work on Scenario 3 in the second phase of the project.

9. Case study 08_AINA – SMS Gateway

9.1. Case study overview

9.1.1. System description

The general purpose of the Aina SMS Gateway application is to deliver a mass of SMS messages from the service providers (public administration offices, supermarket chains, car service chains, etc.) to the SMS recipients (end users) and also deliver possible responses from the recipients to the service providers. The architecture of the Aina SMS Gateway application is shown in Figure 9.1 below.

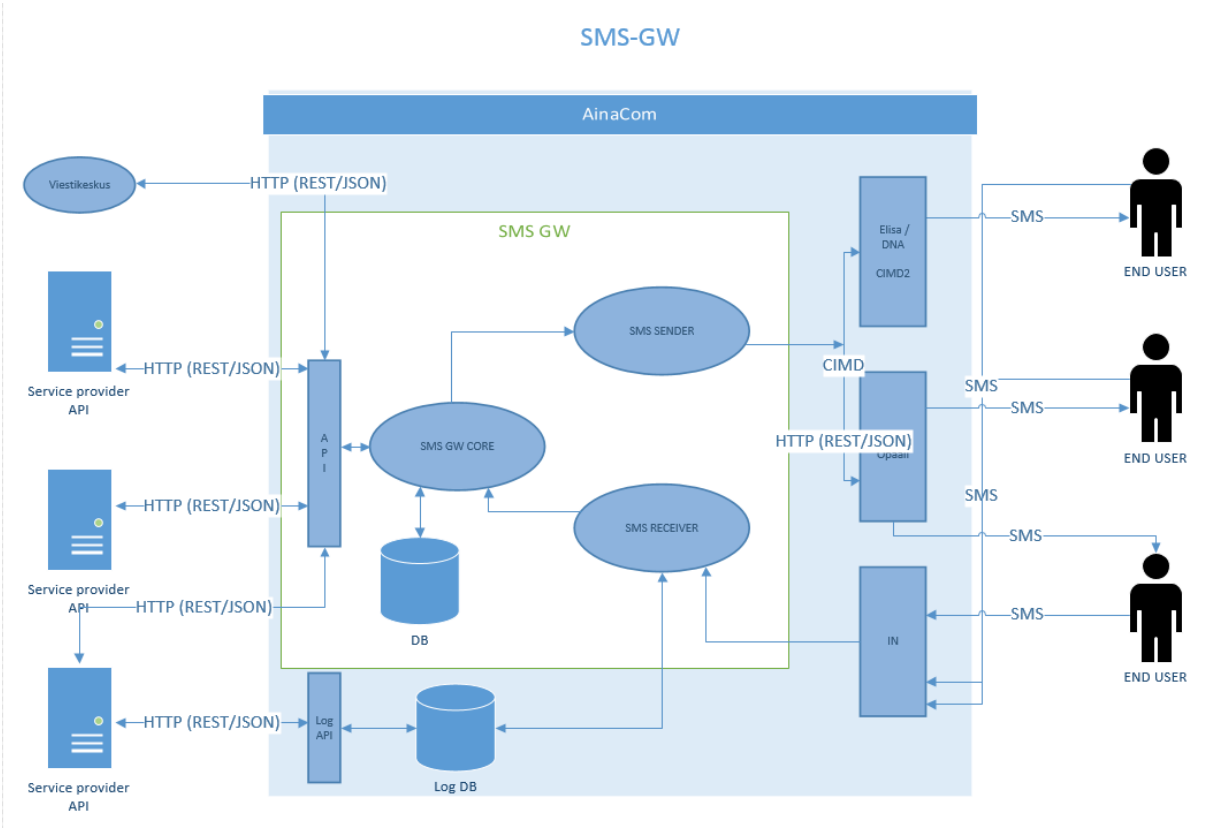

Figure 9.1: Aina SMS Gateway architecture

9.1.2. Development and validation scenarios

The goal of the ECSEL MegaM@Rt2 project lies in providing practical solutions to the following major issues:

1) system engineering incorporating current engineering practices;
2) design vs. runtime traceability, analysis and requirement verification/validation in the context of observed deviations;
3) overall scalability of model-based methods and tools for traceability.

The goal of Telia Communication SMS Gateway case study is to take advantage of the above mentioned model-based methods and tools and apply ECSEL MegaM@Rt2 project results for the selected case study. The goal in this case study is to model the usability and redundancy of services with data analytics related to the real-time activities and traceability issues, which will be realized in the requirements, design, implementation, verification and validation, and dissemination tasks. The

case to be tested with the theory would be related to the functionality of Telia Communication SMS Gateway and especially with the critical parts (see the sequence diagram in the Figure 9.2). Overall, this case study will be also a case study for other Telia Communication development projects and especially how to implement a new test strategy.

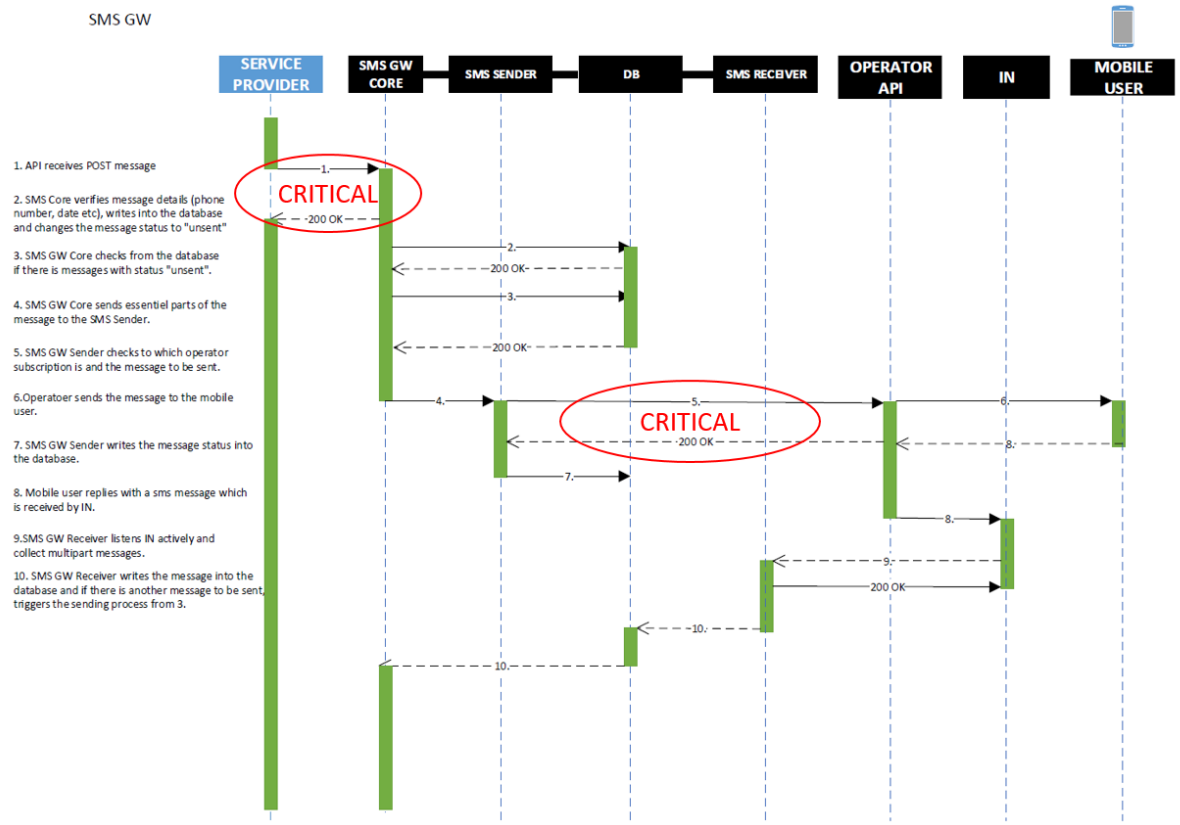

Figure 9.2: Aina SMS Gateway sequence diagram

The main problems with SMS Gateway that we expect MegaM@Rt2 to solve are following:

- Unexpected behavior with tele operator API's (message delivery delays, message delivery failures etc.)
- Performance issues with unexpected data load from service providers.
- Unable to automate rerouting message deliveries if one of the tele operator API is not responding.
- Root cause analysis of failures are done manually at the moment. Current basic alert process doesn't support automatic analyses of the root cause and recovery.
- Unable to simulate service providers, tele operator API's and mobile users unexpected behavior while testing.
- Test execution is semi-manual.

The case study is structured according the scenarios defined below.

9.1.2.1. Scenario № 1: Automation of system builds and self-testing

In scenario 1, enabling the automation of the system builds and make the builds self-testing will have major positive impact to our way of developing software and especially in this case study, the SMS Gateway. We aim to improve product quality, eliminate redundant tasks and eliminate dependencies

of the key personnel. As a result of automation of building the system will also produce a history of builds and releases supporting the developing tasks and the investigation of issues and errors.

9.1.2.2. Scenario № 2: Test automation

In scenario 2, main target is to replace current manual test process with test automation according to work package (WP) plans. It is vital to decide the main functions of test process and test automation tools. Test process should include correct terminology to be used, test roles, test design and templates of test plan, testing tools, responsibilities and tasks between test roles and testing tools. At the end we expect that the whole testing process is fully automated.

9.1.2.3. Scenario № 3: Simulation environment

In scenario 3, we aim to develop a simulation environment to simulate the SMS-traffic process with following simulators: service provider API (AINA), IN-system (AINA) and operator APIs (SSF). SMS-traffic process also includes Shredder-functionality which is optional and will be developed into the simulation environment if necessary.

9.1.2.4. Scenario № 4: Runtime analysis and monitoring

In scenario 4, MegaM@Rt2 should deliver methods and tools for the runtime analysis (runtime monitoring) to improve finding root causes of failures and predict possible future failures based on runtime behavior of service provider and tele operator API's. This can help us to improve functionality of SMS GW towards auto recovery of failures.

9.1.3. KPI definition

The definitions of the KPI for the case study 08_AINA are reported in the Table 9.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 9.1: KPI definition (case study 08_AINA)

S	*KPI*	*Definition*	*U*	V_0	V_B	Δ_T%	*P1*	*P2*
1	KPI_5.3	Enable automation of system builds.	H	42	143	610	✓	✓
1	KPI_5.3	Make builds self-testing.	H	63	214	610	✓	✓
1	KPI_5.4	Aim to improve product quality, eliminate redundant tasks and eliminate dependencies of the key personnel. This will gain us time and cost savings	H	35	119	610	✓	✓
2	KPI_5.1	The manual test process is replaced with test automation process with test cases taking advantage of test mutation method.	H	234	635	540	✓	✓
2	KPI_5.3	Enable automation of system builds.	H	31	85	540	✓	✓
2	KPI_5.3	Make builds self-testing.	H	86	233	540	✓	✓
2	KPI_5.4	Aim to improve product quality, eliminate redundant tasks and eliminate dependencies of the key personnel. This will gain us time and cost savings.	H	39	106	540	✓	✓
3	KPI_1.3	Aim to develop following simulators: service provider, operators, IN-system and mobile user behavior. It's also important that these simulators interact with each other.	H	349	959	470	✓	✓
3	KPI_5.4	Aim to improve product quality, eliminate redundant tasks and eliminate dependencies of the key personnel. This will gain us time and cost savings.	H	87	240	470	✓	✓
4	KPI_1.2	MegaMaRt2 should deliver methods and tools for the runtime analysis (runtime monitoring) to improve finding root causes of failures and predict possible future failures based on runtime behavior of service provider and tele operator API's.	H	56	191	2100	✓	✓
4	KPI_5.4	Aim to improve product quality, eliminate redundant tasks and eliminate dependencies of the key personnel. This will gain us time and cost savings	H	11	38	2100	✓	✓

S = Scenario number — *Unit: H = Hour*
P1 = Phase 1, P2 = Phase 2 — *V_0 = Estimated pre-MegaM@Rt2 value*
Δ_T% = Foreseen Phase 2 improvement — *V_B = Value obtained in MegaM@Rt2 with baseline tools*

9.2. Development and validation activities

9.2.1. Previous work

9.2.1.1. Scenario 1

We have set up a test automation environment which will be part of the whole process enabling automation of system builds and testing. We are using Azure DevOps as a source control of our software development and we have setup Jenkins as an automation server. Although we consider replacing Jenkins with Azure Pipelines. So, there is no automation between source control and automation server yet or between automation server and testing/production environments. This actually requires more configuration of automation server depending on decision of the platform and also configuration of testing/production environments.

9.2.1.2. Scenario 2

We have put a lot of effort for evaluating different partner tools and currently we are using Test Designer (CON) for modelling purposes. Most important selection criteria were that it will help us produce test scripts based on test mutation methods. We have produced a basic model of the SMS-GW and we are reviewing the model architecture and revising the goals for testing in order to find the best practices to the actual testing and how the final model will support that. We have set up a test automation environment with Jenkins and Cucumber. We also had a demo session of Pauware (UPAU) but the technology behind software was not suitable for our development environment. Although we still have plans to evaluate other similar tools until the end of Phase 2.

9.2.1.3. Scenario 3

We have developed a first version of the simulators: service provider (AINA), IN-system (AINA) and operator APIs (SSF). We decided not to continue with Shredder simulator because it has no relevant role in the simulation scenario. With the first test runs with the simulators, we have already identified how to improve SMS-GW functionality.

9.2.1.4. Scenario 4

The preliminary results of the first simulator test runs shows that it is possible to identify critical areas of SMS-GW functionality and how to improve them. These critical areas are:

- Operator APIs: the production environment includes four different operator APIs. Three of the APIs are based on CIMD-protocol and one is based on Opaali-technology. The difference between these technologies have to be taken into account when taking advantage of machine learning. Also, the pricing differentials per API has to be recognised.
- Service provider APIs: there are several service provider APIs implemented into the production environment. The differences between different service provider APIs have to be taken into account when designing and implementing machine learning.

We have developed basic log functionality into the simulators which will support runtime monitoring and analysis during test runs. We have had preliminary talks with VTT and ABO how to proceed according to the WP plans.

9.2.2. Development and validation activities

9.2.2.1. Scenario 1

We have continued to study how to enable automation of system builds and make system self-testing according to WP plans. We have also continued the tests with Jenkins and Azure Pipelines to decide which one should be the tool to be chosen as our source control software.

9.2.2.2. Scenario 2

We have continued to study how to enable test automation according to WP plans. We also have made preliminary actions to evaluate CertifyIt (SMA) and Lime Testbench (SSF) tools. Generic model of entire SMS Gateway is done, but now we have enhanced current model with detailed functionality to support various testing cases especially with test mutation methods. This was done with the support of Conformiq who provided a handful of tailored examples that demonstrate various modeling techniques and approaches that could be applied in the final model.

9.2.2.3. Scenario 3

We have continued with MegaM@Rt2 partners (SSF) enhancing simulators functionality according to WP plans. We also have made preliminary actions how to implement these simulators into the test automation process.

9.2.2.4. Scenario 4

We have continued to plan with MegaM@Rt2 partners (VTT and ABO) how to enable runtime monitoring and analysis according to WP plans. We have made preliminary actions to decide whether the runtime monitoring and machine learning will be based on some tools provided by tool providers or are we going to develop a tool by ourselves with the help of MegaM@Rt2 partners. We have provided first datasets of the logs generated during simulator runs and we have delivered them to VTT to be analyzed. This will support us to decide how to develop the process finding root causes of the failures.

9.3. Results of the first evaluation phase

- After evaluation of different testing tools, we have decided to work with Test Designer (CON). We managed to generate a general model of SMS-GW, but we needed support from CON to produce a detailed model to support Scenario 1 and Scenario 2 targets and KPI's.
- We (AINA and SSF) have designed and developed first version of the simulation environment (Scenario 3) and with the results of the simulation runs we have improved the functionality of SMS Gateway. We have still been evaluating following tools: MBeeTle (SMA), CertifyIt (SMA) and MBPet (ABO) and we have plans to continue these evaluations.
- We have created and provided first datasets of the log data of SMS-GW to VTT to be analyzed. This is the preliminary action to start evaluating VTT-tools, Clusterability and Convex Hull.

9.3.1. KPI values

The Table 9.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 08_AINA.

Table 9.2: Phase 1 KPI values (case study 08_AINA)

S	*KPI*	*U*	V_0	V_B	$\Delta_B\%$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_5.3	H	42	143	240,5	271	545,2	610
1	KPI_5.3	H	63	214	239,7	407	546,0	610
1	KPI_5.4	H	35	119	240,0	226	545,7	610
2	KPI_5.1	H	234	635	171,4	1068	356,4	540
2	KPI_5.3	H	31	85	174,2	142	358,1	540
2	KPI_5.3	H	86	233	170,9	392	355,8	540
2	KPI_5.4	H	39	106	171,8	178	356,4	540
3	KPI_1.3	H	349	959	174,8	1723	393,7	470
3	KPI_5.4	H	87	240	175,9	431	395,4	470
4	KPI_1.2	H	56	191	241,1	338	503,6	2100
4	KPI_5.4	H	11	38	245,5	68	518,2	2100

S = Scenario number
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
$\Delta_B\% = 100\cdot(V_B - V_0)/V_0$ (baseline improvement)
Unit: D = Day, H = Hour
V_0 *= Estimated pre-MegaM@Rt2 value*
V_B *= Value obtained with MegaM@Rt2 baseline tools*
V_1 *= Phase 1 value obtained with Framework initial version*
$\Delta_1\% = 100\cdot(V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)

9.3.2. Plan of improvements

9.3.2.1. Requirement coverage

The Table 9.3 shows the present of the AINA requirements.

Table 9.3: AINA requirements

ID	*AINA Requirement*	*Coverage*
AINA_01	MegaM@Rt2 should enable the automation of building the system and make the builds self-testing.	90%
AINA_02	MegaM@Rt2 should enable the possibility to automate testing and getting test results automatically.	71%
AINA_03	MegaM@Rt2 should enable the possibility to simulate service providers, tele operators and mobile users in the test environment especially with the critical data load.	86%
AINA_04	MegaM@Rt2 should deliver methods and tools for the runtime analysis (runtime monitoring) to improve finding root causes of error situations and predict possible future problems.	27%

9.3.2.2. Feedback to tool providers

We have put a lot of effort for evaluating different partner tools. Biggest problem choosing the tools has been the development platform which these tools supports. Our development is heavily based on Python and MegaM@Rt2 tools are mostly Java-based, so the list of tools have been narrowed to a few tools to be evaluated.

9.3.2.3. Work planned for the Phase 2

- *Scenario 1* — We will continue to study how to enable automation of system builds and make system self-testing according to WP plans. We also have to finalize the studies whether to

choose Jenkins and Azure Pipelines our source control software. This will support the target to complete the test automation environment and make builds self-testing.

- *Scenario 2* — We will continue to study how to enable test automation according to WP plans. We also plan to complete the evaluations of CertifyIt (SMA)- and Lime Testbench (SSF) -tools. Depending on the results these tools will support the scenario KPI's. We also plan to complete modelling with the Test Designer tool (CON). This will help us to produce test script based on the mutation testing methodology. These test scripts will be run in the test automation environment alongside with the simulators (scenario 3).
- *Scenario 3* — We continue with MegaM@Rt2 partners (SSF) enhancing simulators functionality according to WP plans. We also need support from MegaM@Rt2 partners (SSF and ABO) how to implement these simulators into the test automation process.
- *Scenario 4* — We continue to plan with MegaM@Rt2 partners (VTT and ABO) how to enable runtime monitoring and analysis according to WP plans. We have to decide whether the runtime monitoring and machine learning will be based on some tools provided by tool providers (e.g. VTT) or are we going to develop a tool by ourselves with the help of MegaM@Rt2 partners. Preliminary plan is to take advantage of the logs generated during simulator runs. This will also help us to decide how to develop the process finding root causes of the failures.

9.3.3. Summary of the progress towards the roadmap

During Phase 1 we have mainly been focusing on scenarios 1, 2 and 3. These scenarios are well under way to reach the KPI's for scenarios in timetables set for them. During this phase we were a bit concerned of the situation with the evaluations of the MegaM@Rt2 tools. Mainly because the tool platforms (Eclipse) were not supporting our development environment. Anyhow we are satisfied with the tools we are using now and we hope that ongoing evaluations of MegaM@Rt2 tools will bring us more options. In phase 2 we are going to be mainly focused in Scenario 4.

10. Case study 09_CAM – Intelligent Traffic Surveillance System

10.1. Case study overview

10.1.1. System description

The CAMEA (CAM) case study 09_CAM "Intelligent Traffic Surveillance System" is focused on the UnicamVELOCITY system that is composed as a combination of a local processing on site (LP detection and OCR) with all the infrastructure around (video cameras, IR flashes, PC, networking, etc.), and background processing running on server side. Those systems than can serve for average speed check, red light enforcement, or travel time estimation, etc.

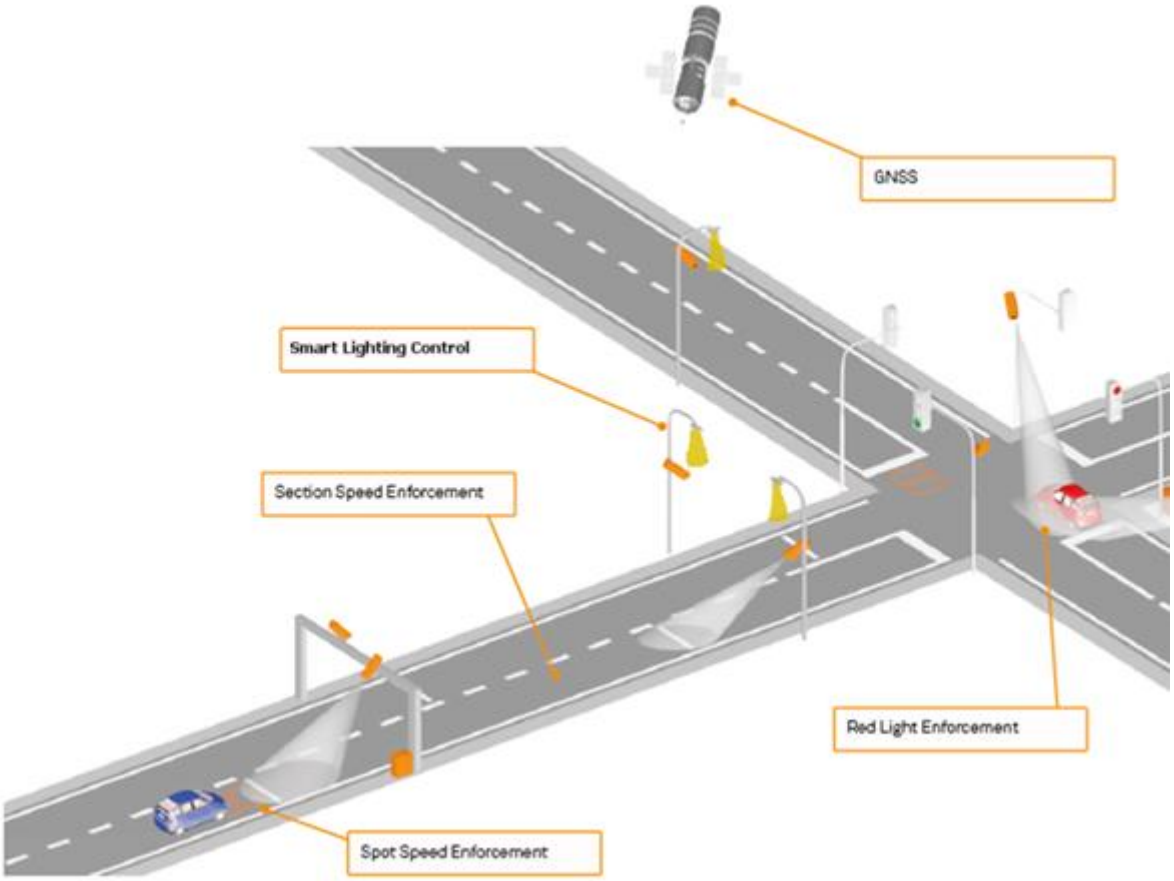

Figure 10.1: Example of CAMEA Unicam systems combined

CAMEA uses multiple levels of processing in its system. Part of the computation is done locally and the rest is done on the server. Currently, on the sites Unicam systems use Ethernet cameras (manufactured by CAMEA) wired to computationally powerful PCs capable of running powerful video detection algorithms. The detector also supports additional logic e.g. for controlling IR flashes on demand/trigger. The detection results are then sent to the server computer and processed in the meaning of the matching corresponding detection and calculating average speed. When speed limit is violated, detailed evidence images can be requested from the site (which saves network bandwidth).

Powerful computational machine is often advantage, however, there are often needs for decreasing power consumption of the system as a whole. The site does not always have a stable power supply and the system has to be then battery powered. In case of public lightning present, the batteries can

be recharged overnight. Sometimes there is no power supply at all and batteries have to be big enough to allow a couple of days of system's operation and then the whole battery pack has to be changed. Even heat dissipation could be a big problem when the computer running in hermetically sealed box (weather resistant) without active cooling.

At this point, use of smaller computers or even all-in-one solutions in form of cameras with embedded intelligence (in case of CAMEA build on the basis of Xilinx Zynq chip) is big advantage. The question is, how to port all the field-proven applications to the new platforms including often different division of algorithm parts over the system. For the smart camera, most time-critical part of the algorithm, such as LP detection, can be moved to the FPGA (e.g. FPGA part of popular Zynq chip) and simple post processing and sending detection results will be then executed in CPU (or CPU part of Zynq). On the server side, OCR algorithm can be executed (over the cropped image with LP present) and all the matching is then done in the same manner.

It is obvious that this way the energy consumption of the on-site system part has been reduced and part of the computation load has been transferred to the server (this way the low network load is kept). The biggest question is how to easily transfer all the algorithms and services to other (often hybrid) platforms including moving part of the processing load to the cloud.

10.1.2. Development and validation scenarios

For the purpose of the scenario definition, we are focused on the embedded version of UnicamVELOCITY system is based on all-in-one cameras of heterogeneous structure running various components of the detection chain. The local sub-system (e.g. embedded processing in camera) can be built on the basis of modular Framework for embedded image processing. This consists of following parts:

- **Detector** trained for desired objects (in our case license plates - LP).
- **Tracker** that keeps track of the detected object and helps with filtering and selection of the best sample (e.g. best LP for subsequent post-processing).
- Image **post processing** (e.g. CNN).

Based on the framerate and the computation load, the individual parts of the sub-system can be mapped at suitable HW such as FPGA, GPU or ARM CPU.

Selecting appropriate processing elements with subsequent mapping of the design of the system to the hardware is very challenging task repeating for most of the projects (starting from scratch or being adapted from existing solutions). Automating or guiding this part of development process can significantly reduce the time-to-market and development costs.

As well optimization phase of the system development can be automated or guided in same manner resulting in the same reduction of the time-to-market and development costs. Both will give to SMEs like CAMEA valuable competitive advantage.

The case study is structured according the scenarios defined below.

10.1.2.1. Scenario № 1: Optimal division of functionality among available HW

Algorithms for individual parts of sub-system are prepared and verified as pure SW components. Then particular methods and tools are used for the mapping individual algorithms (parts of the sub-system) to the target HW (usually hybrid platform - combination of embedded CPU, GPU and FPGA). As an input, model of the system is prepared. The constraints defined will be met and better solution/combination will be found. Then they should be mapped to individual HW components with regards to defined constraints such as price, maximum frame rate, power consumption etc.

10.1.2.2. Scenario № 2: Optimization of code runtime and profiling

Based on mapped sub-system division, the code/components should be profiled and the result will be used for optimization of code runtime in form of static scheduling of tasks and setting priorities. Particular methods and tools are used for the profiling (estimation) of a sub-system based on the model prepared or measuring real values during runtime. Subsequently, based on the results of profiling, tasks are scheduled and prioritized in an optimal way. This will result in optimal and effective run of a sub-system.

10.1.3. KPI definition

The definitions of the KPI for the case study 09_CAM are reported in the Table 10.1; the column $\Delta_T\%$ gives the total improvement that the partner foresees for the Phase 2; the columns P1 and P2 show if the KPI is evaluated in Phase 1 and/or in Phase 2.

Table 10.1: KPI definition (case study 09_CAM)

S	*KPI*	*Definition*	*U*	V_0	$\Delta_T\%$	*P1*	*P2*
1	KPI_5.1	Productivity improvement in requirement validation.	H	120	40%	✓	✓
1	KPI_5.3	Gains in productivity compared to the baseline development process that does not include mega-modelling with continuous development	H	2240	40%	✓	✓
2	KPI_4.1	Reduction in time/effort required for tracing and handling all the involved models at design and runtime levels (e.g. creation of and access to relations between system and traces models).	H	440	40%	✓	✓
2	KPI_5.3	Gains in productivity compared to the baseline development process that does not include mega-modelling with continuous development	H	975	40%	✓	✓

S = Scenario number
Unit: H = Hour
P1 = Phase 1, P2 = Phase 2
V_0 = Estimated pre-MegaM@Rt2 value
$\Delta_T\%$ = Foreseen Phase 2 improvement

As reference value the case study 09_CAM takes V_0 that is the pre-MegaM@Rt2 value and that results internal records of labour spent on HW/SW development tasks performed during UnicamVELOCITY system; consequently, the Phase 1 improvement is evaluated as $\Delta_1\% = 100\cdot(V_1 - V_0)/V_0$.

10.2. Development and validation activities

10.2.1. Previous work

10.2.1.1. Scenario № 1: Optimal division of functionality among available HW

Previously, the development process was based on preparing SW proof of concept based on suitable state-of-the-art algorithms. Very basic profiling was done, and experts then went through the code and identified (based on their own knowledge and experience) parts of code that are critical for acceleration. Based on their experience, they selected suitable HW matching computation power, timing and power consumption constraints.

10.2.1.2. Scenario № 2: Optimization of code runtime and profiling

Previously, the part of the iterative development process was simple timing analysis. Developers had to identify critical parts of algorithms and transfer them to the HW (FPGA, GPU, ...). Then timing data

was collected and evaluated offline. The evaluation results served for the next iteration when the accelerated parts can be adjusted (e.g. another part of the method is accelerated as well).

10.2.2. Development and validation activities

10.2.2.1. Scenario № 1: Optimal division of functionality among available HW

Currently, the process of profiling has been updated by adding better offline analysis (based on the models prepared). This includes advanced and iterative profiling of heterogeneous code after division to hardware at the end of first iteration is done. It apparently reduced demand on expert's knowledge (less experienced people), and it made development time shorter and development process cheaper.

10.2.2.2. Scenario № 2: Optimization of code runtime and profiling

As mentioned in Scenario 1, the process of profiling has been updated by adding better runtime analysis (based on the running code). This helps with iterative profiling of heterogeneous code after division to hardware at the end of first iteration. It reduced demand on experts and made development iterations shorter and development process cheaper.

10.3. Results of the first evaluation phase

10.3.1. KPI values

The Table 10.2 shows the Phase 1 results obtained by experimenting the MegaM@Rt2 Framework (initial version) in the different scenarios of the case study 09_CAM.

Table 10.2: Phase 1 KPI values (case study 01_TRT)

S	*KPI*	*U*	$V_0 = V_{REF}$	V_1	$\Delta_1\%$	$\Delta_T\%$
1	KPI_5.1	H	120	98	18.3%	40%
1	KPI_5.3	H	2240	1440	35.7%	40%
2	KPI_4.1	H	440	370	15.9%	40%
2	KPI_5.3	H	975	630	35.4%	40%

S = Scenario number
Unit: D = Day, H = Hour
$\Delta_T\%$ *= Foreseen Phase 2 improvement*
V_0 *= Estimated pre-MegaM@Rt2 value*
V_1 *= Phase 1 value obtained with Framework initial version*
$\Delta_1\% = 100\cdot(V_1 - V_{REF})/V_{REF}$ (Phase 1 improvement)

10.3.2. Plan of improvements

10.3.2.1. Requirement coverage

The MegaM@Rt2 Framework requirements related to the use-case scenarios were narrowed to:

Table 10.3: CAMEA requirements

ID	*CAMEA Requirement*	*Coverage*
CAM_03	Modelling framework that supports optimal division of functionality between available HW (distribution within platform) and the cloud (offloading functionality).	68%
CAM_04	CAM_04 - Modelling framework that supports optimisation of code runtime, e.g. using static scheduling of tasks/processes (based on profiling or expertise) using priorities.	69%
CAM_05	Profiling tool with ability of monitoring software code run and running FPGA components	82%

10.3.2.2. Feedback to tool providers

The development and runtime support valuable for both our scenarios is still not well embedded to the MegaM@Rt2 Framework. This needs some tunes.

10.3.2.3. Work planned for the Phase 2

In the next phase, this should be integrated in MegaM@Rt2 Framework enabling runtime analysis. Based on the results of profiling, the tasks scheduling and prioritization in an optimal way should be also estimated using the Framework.

10.3.3. Summary of the progress towards the roadmap

It has been already proven that offline analysis, profiling and runtime-support has very positive impact on the effort (in term of hours) spent during development. Thus time-to-market and costs are reduced which can give CAMEA significant competitive advantage. The last step needed is better integration with MegaM@Rt2 Framework.

11. Conclusions

This document reports the preliminary results of the MegaM@Rt2 project, those that were obtained during the Phase 1 (from the month M15 to the month M24 of the project).

By developing (design, implementation, verification) the systems they proposed as case studies, the case study providers experimented the tools—individually and integrated in the Framework—that the tool providers made available according to the plan of the Full Project Proposal document (see the Table 11.1 that shows also the previous baseline period and the successive Phase 2).

Table 11.1: Tools, Framework, and case studies timing

Project month	*M06*	*M15*	*M20*	*M22*	*M24*	*M32*	*M36*
Tools baseline version	◆						
Case studies baseline period		■					
Tools initial version		◆					
Tools intermediate version			◆				
Framework initial version				◆			
Case studies Phase 1		■	■	■	■		
Tools final version						◆	
Framework final version							◆
Case studies Phase 2					■	■	■

The development is organized in scenarios. These put the focus on the benefits that MegaM@Rt2 is expected to bring; they allow to better understand the aspects that the case study providers found most important for their industrial activities; they structure and organize the tools verification and validation, which are based on the requirements and the KPI defined by MegaM@Rt2.[9]

During the Phase 1, certain scenarios were used to carry out the verification and validation activities. The same scenarios along with other ones will be used for the same purpose during the Phase 2 (Figure 11.1).

[9] The Chapter 1 “Introduction” gives more details on the types of requirements (user, systems, and tools requirements) as well as on the KPI (Key Performance Indicators).

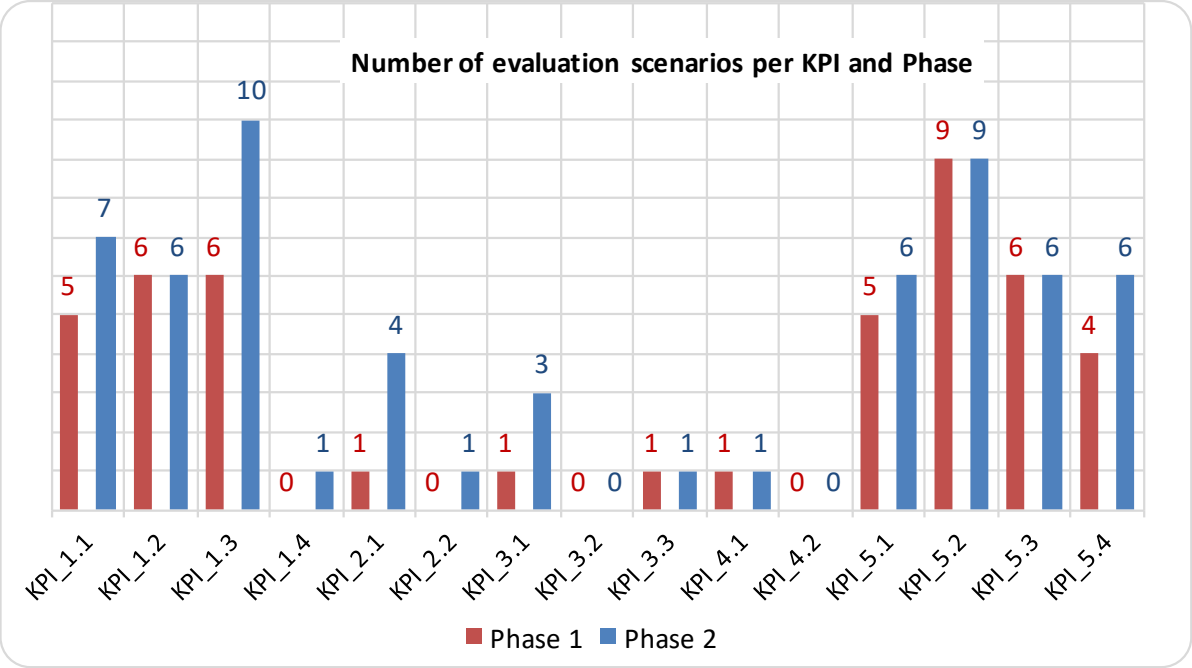

Figure 11.1: Number of scenarios for evaluating each KPI in Phase 1 and Phase 2

The case study providers gave feedback and recommendations to the tool providers on the basis of the verification.

Moreover, the case study providers measured the KPI, which express quality enhancement as well as cost and time reduction, for evaluating the industrial applicability of MBE solutions that MegaM@Rt2 aims at providing. The Figure 11.2 shows the KPI ranges: the expected ones that the Full Project Proposal document set, and the actual ones that the MagaM@Rt2 Consortium attained during the Phase 1.

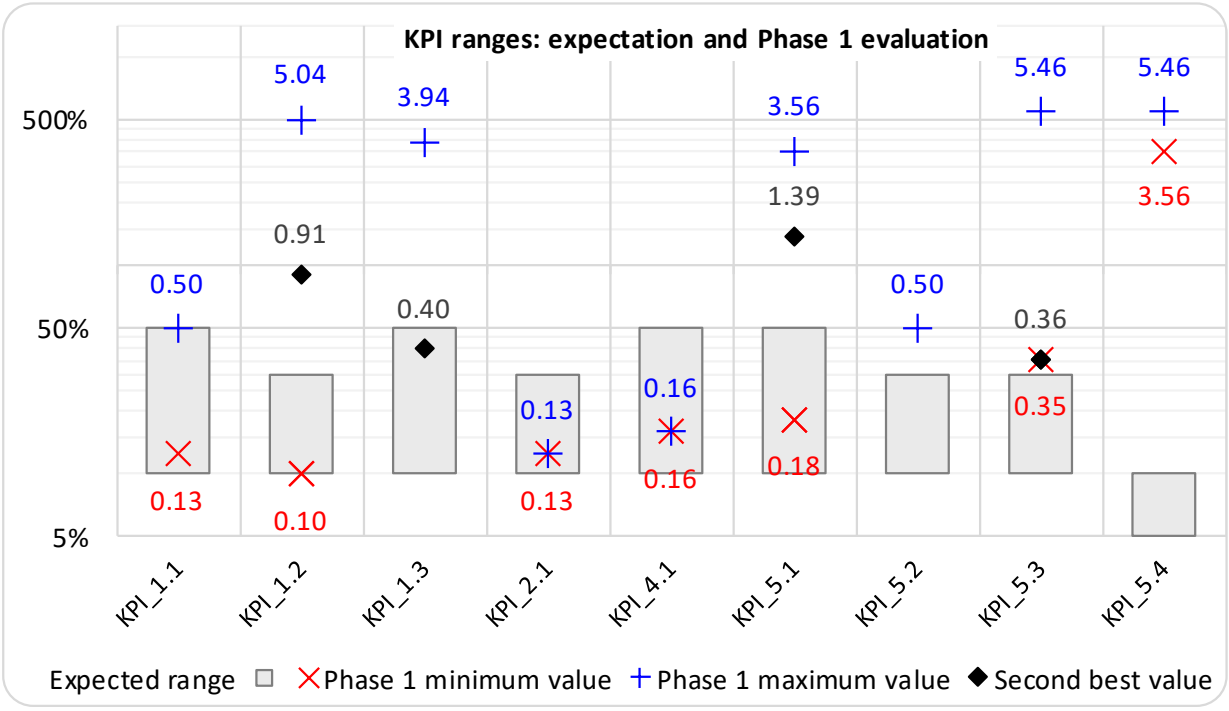

Figure 11.2: KPI ranges attained by the MagaM@Rt2 Consortium during the Phase 1

The Figure 11.2 doesn't show two negative values (KPI_1.3 and KPI_5.2) that were due to an incomplete integration—an issue that will be solved in Phase 2.

The highest values (KPI_1.2, KPI_1.3, KPI_5.1, KPI_5.3, and KPI_5.4) of the Figure 11.1 were obtained on the AINA Information Technology system that doesn't have aspects of direct real-time control of field devices (sensors, actuators) and equipment (communication). For completeness, the Figure 11.2 shows also the second-best value of these KPI, when available.

In this document there are also the plans for the Phase 2 that will complete the scenarios and KPI. Phase 2 will also give the final validation of the tools integrated in the Framework as well as the evaluation of the project outcome.

The MegaM@Rt2 Consortium is confident to reach the objectives because good KPI values were obtained during the Phase 1 and the project is substantially in time with the plan.

References

[FPP] MegaM@Rt2 Full Project Proposal

[D1.1] MegaM@Rt2 “D1.1: Industry Requirements Specification”. Published as a consortium internal document [September 2017]

[D1.3] MegaM@Rt2 “D1.3: Case studies scenarios definition”. Published as a consortium internal document [August 2018]

[D2.2] MegaM@Rt2 “D2.2: Design Tool Set Specification”. Available: [http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.2] [February 2018]

[D3.2] MegaM@Rt2 “D3.2: Specification of the MegaM@Rt2 Runtime Analysis tool set”. Available: [http://hdl.handle.net/20.500.12004/1/P/MMART2/D3.2] [February 2018]

[D4.2] MegaM@Rt2 “D4.2: Specification of the Model Management & Traceability tool set”. Available: [http://hdl.handle.net/20.500.12004/1/P/MMART2/D4.2] [December 2018]

[D5.4] MegaM@Rt2 “D5.4: Case study development report – Phase 1”. Published as a Consortium internal document [January 2019]

[ReefShark] Nokia ReefShark Baseband SoC, https://networks.nokia.com/5g/reefshark . Visited 13.12.2018.

[Time4Sys] PolarSys Time4Sys provides a framework that fills the gap between the capture of timing aspects in the design phase of a real-time system and the ability of specific/dedicated tools to verify the consistency and performances of a given scheduling. https://www.polarsys.org/time4sys/. Visited 25.1.2019.

❑