A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimisation

D1.5 Meta model-driven abstraction and model-driven extensions and use-case enhancements

Version 1.0

Documentation Information

Contract Number	871669
Project Webpage	https://www.ampere-euproject.eu/
Contractual Deadline	30.09.2022
Dissemination Level	Public (PU)
Nature	R
Authors	Alexandre Amory, Tommaso Cucinotta(SSSA)
Contributors	Sara Royuela (BSC) Harald Mackamul (BOS) Massimiliano Polito, Olivier Constant (THALIT) Tiago Carvalho (ISEP)
Reviewer	Miguel Pinho (ISEP)
Keywords	AMALTHEA, CAPELLA, meta-model, automotive/railway use-cases

AMPERE project has received funding from the European Union's Horizon 2020 research and innovation programme under the agreement No 871669.

Change Log

Version	Description Change
V0.1	Initial version with the original deliverable description.
V0.2	Initial description of the augmentations for performance and heterogeneity.
V0.3	Extensions in the AMALTHEA DSML.
V0.4	Final description of the augmentations for performance and heterogeneity.
V0.5	Complete draft. 1st SSSA review.
V1.0	Complete document, after ISEP review.

Table of Contents

1 Executive Summary

This document constitutes Deliverable *D1.5. Meta model-driven abstraction and model-driven extensions and use case enhancements*, built upon D1.3 and several contributions from the other WPs. D1.5 reports on with activities carried out in WP1 *System Model Description and Use-cases*, targeting milestone MS3, and due originally on project month 27, extended to month 33 as agreed with the EC.

This deliverable presents the final release of the meta model-driven abstractions upon which multi-criteria optimization is being applied, and a description of the proposed DSML extensions. These include AMALTHEA/-Capella extensions for middleware communication (ROS) and execution behavior, for modeling runnable specializations, for capturing performance metrics, for modeling heterogeneouos and DVFS-enabled platforms, and for tagging modeling elements with safety requirements. It also refines the description of the Automotive and Railway use-cases provided in previous WP1 deliverables, detailing their enhancements based on the presented DSML extensions, and their resulting benefits from the toolchain under development in AMPERE.

2 Introduction

This deliverable provides results from the activities in the AMPERE *Task 1.4: Meta model-driven abstraction and model-driven extensions*. Task 1.4 (m7:m33) contributes to both D1.3 [1] and D1.5 (this document), and it designs and develops the meta model-driven abstractions incorporating the key non-functional semantics features from the AMALTHEA/Capella Domain-Specific Modeling Languages (DSMLs). These features enable automated model transformations that enable: multi-criteria optimization, evaluation of the correctness with respect to non-functional constraints, automatic generation of parallel code skeletons complying with OpenMP, the underlying parallel programming model of choice/reference within the AMPERE project, and all of these with the ability to exploit the acceleration capabilities of parallel heterogeneous architectures. The extensions presented below capture constraints related to end-to-end timeliness of parallel applications, energy constraints, fault tolerance and safety/security attributes. This task also investigates extensions needed for the integration of the timing models when deploying on heterogeneous platforms with such components as CPUs with heterogeneous capabilities (e.g., Arm big.LITTLE or DynamIQ), General-Purpose GPUs (GP-GPU), and FPGA acceleration.

This deliverable provides the following contents, where we also link the provided software components to the requirements defined in the traceability matrix from the amended version of D1.1 [2] (all the presented components dealing with AMALTHEA address implicitly requirements SYS-PCC-REQ-111 and SYS-PCC-REQ-112):

- Chapter 3 concentrates the final DSML extensions for both AMALTHEA and Capella. This chapter is subdivided by extension categories, namely:
  - Section 3.1 describes the middleware communication extensions to support publish/subscribe communication, used by ROS, and execution behavior extension to model conditional and loop constructs in the AMALTHEA Runnables (addressing requirements SYS-ODAS-REQ-106 and SYS-ODAS-REQ-114);
  - Section 3.2 presents the extensions which enable modeling runnable specializations, a new concept that allows runnable target different types of processing units, such as multicore, GPU, or FPGA;
  - Section 3.3 describes extensions to capture performance metrics out of a profiling campaign (addressing requirements SYS-PCC-REQ-101, SYS-PCC-REQ-103 and SYS-ODAS-REQ-112);
  - Section 3.4 describes AMALTHEA hardware model extension to represent hardware platforms with DVFS capabilities, like the platforms supported by the AMPERE project (addressing requirements SYS-PCC-REQ-105, SYS-ODAS-REQ-116 and SYS-ODAS-REQ-115);
  - Section 3.5 presents a Capella extension to capture runnable/task safety requirements and translate these definitions into an AMALTHEA model (aka, Capella/AMALTHEA bridge).
- Section 4.1 presents model extensions regarding publish-subscribe middleware, and useful for example, in the automotive use-case; these address also specifically messaging and component-based requirements in SYS-PCC-REQ-111, but also those in SYS-ODAS-REQ-106, SYS-ODAS-REQ-107 and SYS-ODAS-REQ-114.
- Section 4.2 shows the model extensions for the railway use-case; these modeling elements are key in specifying system characteristics that are needed to address requirements SYS-ODAS-REQ-*;
- Finally, Section 4.3 presents the plan to apply the use-cases into the AMPERE target platforms.

3 DSML extensions

This section describes the model extensions introduced in the AMALTHEA and CAPELLA DSML to support the AMPERE use-cases.

3.1 AMALTHEA Extensions for middleware communication and execution behaviour

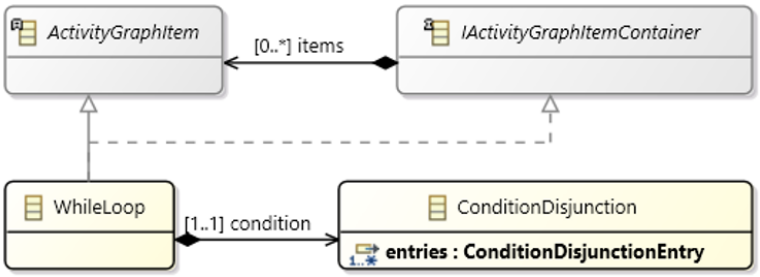
In order to specify the execution semantics of current middleware architectures the new elements *WhileLoop* and *ChannelFillCondition* were introduced in the latest releases of APP4MC. It is now possible to model a control flow path based on data available in a channel. For example: take one element as long the queue is not empty.

3.1.1 Loops

The new element *WhileLoop* can be used in the activity description of tasks, interrupt service routines and runnables (inherited from *ActivityGraphItem*). It can also contain (nested) activities in the loop body (inherited from *IActivityGraphItemContainer*).

- Proposal: Deliverable 1.3 Section 7.1.1
- Release: Eclipse APP4MC 1.2.0 (Jul 2021)

Figure 1: New while loop.

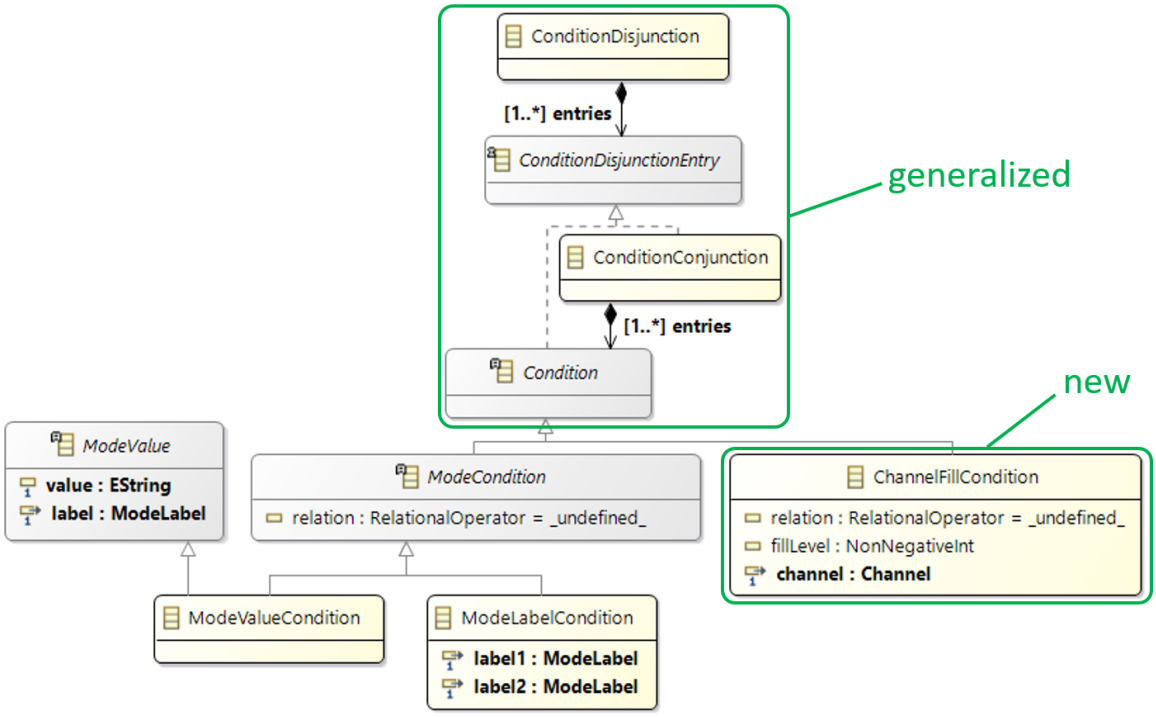


3.1.2 Conditions on channel fill level

Amalthea conditions are structured in a boolean *disjunctive normal form* (DNF) with an OR-condition on top level and AND-conditions on the next level. This part of the model was generalized to allow all kind of conditions instead of the former mode conditions. The new element *ChannelFillCondition* allows to specify conditions on the number of elements in a channel.

- Proposal: Deliverable 1.3 Section 7.1.1 / 7.1.2
- Release: Eclipse APP4MC 2.0.0 (Nov 2021), 2.1.0 (Apr 2022)

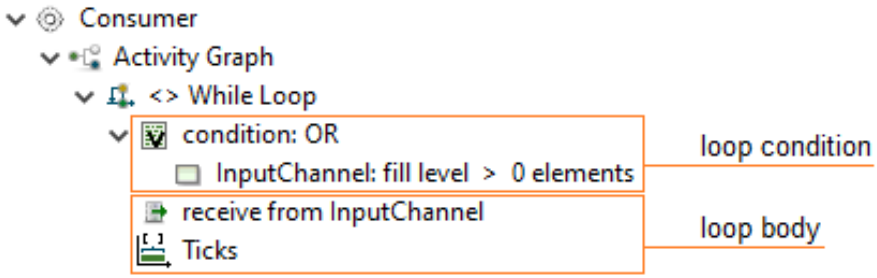
Figure 2: Generalized and extended conditions.



3.1.3 Example: Loop over Channel inputs

The example shows how inputs from a queue can be processed in a while loop. The loop condition checks if at least one element is available in the channel. The loop body contains a *channel receive operation* that is specified as *FIFO_Take* with *number of elements* $= 1$.

Figure 3: Example: Take from input queue as long as elements are available.



The semantics to handle channel fill conditions and loops is necessary for complex data handling between applications. For example, an application needs the n data points of a sensor data to start calculation. These modeling extensions allows the designer to specify conditions when and how to access channels. The task could be activated, when the channel reaches a certain fill level, a callback function can loop of the contents of a channel to take all data points and execute a functionality until it is empty. These extensions allow us to model the execution semenatics of modern pub/sub communication schemes.

3.2 AMALTHEA Extensions for parallelism and heterogeneity

This section describes the features introduced in the AMALTHEA DSML to expose parallelism and heterogeneity. The mechanisms to transform these new features into OpenMP code to effectively exploit performance are described in D2.3 [3].

3.2.1 Parallelism and specializations

D2.2 [4] proposed exposing parallelism in AMALTHEA at three different levels:

1. *among tasks*, to define coarse-grained parallelism handled by the operating system (OS) scheduler;
2. *among runnables*, to define a fine-grained parallelism handled by the parallel programming model runtime, i.e., OpenMP; and
3. *inside runnables*, exploiting an even finer-grained parallelism handled by a parallel programming model (not necessarily the same) in either the host or a dedicated accelerator (e.g., GPU or FPGA).

The former is already supported in AMALTHEA, and the latter is transparent to the AMALTHEA model. This section describes the second option, as *parallelism among runnables is the key to bridge the gap between the model-driven engineering (MDE) techniques used for the development of the complex cyber-physical systems (CPS) targeted in the project, and the parallel programming models supported by the underlying parallel and heterogeneous processor architectures*.

The preliminary support for parallelism presented in D2.2 [4] was based on two new *custom properties* that, added to a runnable, expose its parallel nature. These properties are:

- *host parallelism*, to describe a potentially concurrent unit of work to be in the host system, and
- *accelerator parallelism*, to describe a potentially concurrent unit of work to be executed in an accelerator device (e.g., GPUs or FPGA).

The goal of AMPERE is to develop an integrated ecosystem for low-energy and highly parallel and heterogeneous computing architectures. Consequently, the mechanism used to expose parallelism in the AMALTHEA model shall offer an extensible and modular approach to target all platforms considered in the project, including shared-memory (host parallelism) and heterogeneous systems (accelerator parallelism). For this reason, we modified the original approach of exposing fine-grained parallelism in AMALTHEA to also exploit the concept of *specialization*, i.e., a specific implementation to be executed under certain conditions. The main idea is that a runnable can be defined by a number of specializations (e.g., one for the host and other for FPGA), which are also concurrent units of work defined to be executed in the host or a particular accelerator device depending on the conditions assigned to each specialization.

3.2.2 New components in AMALTHEA

The execution path of a runnable/task can now depend on a local execution context. The context is represented by local mode labels that can be modified in their activity graph. Context information can also be propagated in the call tree (set in the context of a runnable call).

3.2.2.1 Execution context / Local Mode Labels

Execution Context

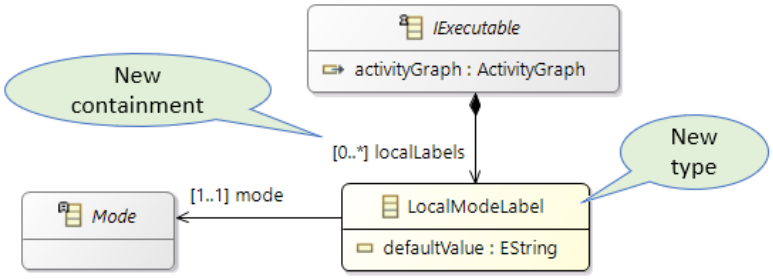
- is defined in the scope of an executable (task, ISR, runnable)
- values are represented by local mode labels
- values can be ...

- set/modified in an activity graph
- passed when calling a runnable

Local Mode Labels

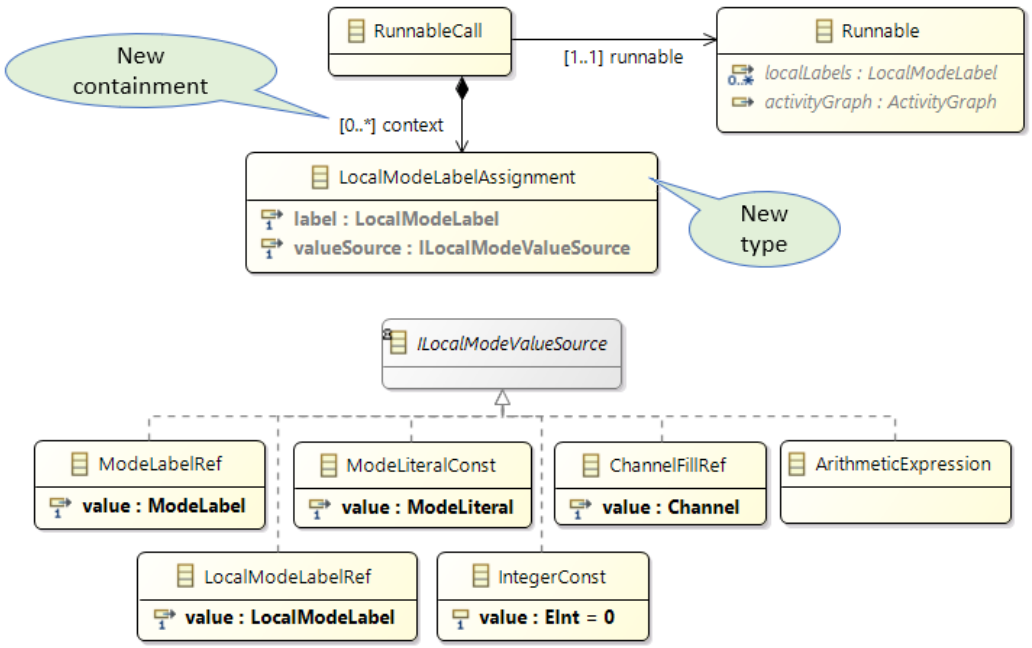
- can save a local copy of a global state at a specific point in time (e.g., when the function is called)
- can be modified (incremented, decremented, . . .)
- are used in conditions (switches, loops) in an activity graph
- can be set in a runnable call (“execution context”)
- have the same attributes as global mode labels

Figure 4: New execution context.



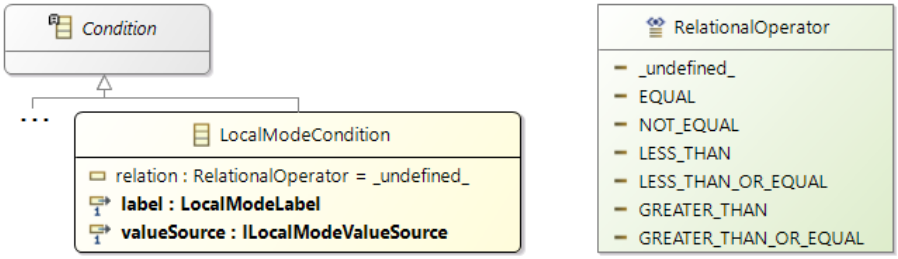
3.2.2.2 Setting an execution context

Figure 5: Setting an execution context.



3.2.2.3 Execution/implementation specializations

Figure 6: New local mode label conditions.



3.2.3 Example: Propagation of specializations

In order to address parallelism and specializations in a portable and productive manner, an AMALTHEA software model can be then increased as follows:

1. Introduce a new *mode* called *specializations* with the possible values *host_seq*, *host_omp*, *device_omp* and *device_cuda*, where *host_seq* and *host_omp* will define work to be executed in the host (sequentially or potentially in parallel, respectively), and *device_omp* and *device_cuda* will define work to be executed in a GPU (via OpenMP or CUDA, respectively). The extensions for FPGA support, described in D1.3, are not yet integrated with the specializations presented in this deliverable. We envision to include a *device_fpga* mode to integrate FPGAs in the common mechanism to support parallelism and heterogeneity.
2. Define a *local label* with mode *specializations* in each runnable containing parallel and/or heterogeneous behaviour. Then, introduce in the activity graph a *switch* indicating the available implementations and the conditions (based on the local label) to be fulfilled for executing each version.
3. Define the *context* for each parallel/heterogeneous runnable call in the tasks. This context uses the local label corresponding to each runnable.

As an illustration, Figure 7 depicts the AMALTHEA model of the DAPHNE [5] *Points2Image* use case[1] augmented with the new extensions for parallelism and heterogeneity. This application runs six times a pipeline of functionalities (runnables) consisting in (1) *read_case* to read a specific input image, (2) *preprocess* to process the input, (3) *computation* to perform transformations in the image, and (4) *postprocess* to store the image. The model has been tuned to alternate the execution of the pipeline in the host (task *pointcloud2_to_image_host*) and the accelerator (task *pointcloud2_to_image_device*), where the host version is executed periodically, and the device version is triggered when the host version finishes. The figure highlights in yellow how the pipeline to run the host version of the *preprocess* runnable is defined.

3.3 AMALTHEA Extensions for timing metrics

The AMPERE eco-system includes tools able to extract runtime information on program executions or by simulation (e.g. Extrae) and tools that analyse those results into useful metrics. The timing analysis tool, described in deliverable D3.3 [6], extracts metrics from the results provided by Extrae. Once performance traces and estimations have been gathered from these profiling tools, on a given hardware, it is important to feedback that information to the AMALTHEA models. As the notion of timing properties in AMALTHEA is narrow, we have defined an approach such that this additional data is in fact additional information within the model, and does not compromise or change the behavior of the model. For instance, changing the `Ticks` specified in the `ActivityList` of a `Runnable` would change the behavior of the latter.

[1] DAPHNE is an automotive benchmark suite for parallel programming models, which implements automotive workloads using different state-of-the-art programming models for heterogeneous platforms, i.e., OpenMP, CUDA and OpenCL.

Figure 7: Exposing inter-runnable parallelism and code specializations in the DAPHNE Points2Image use-case.

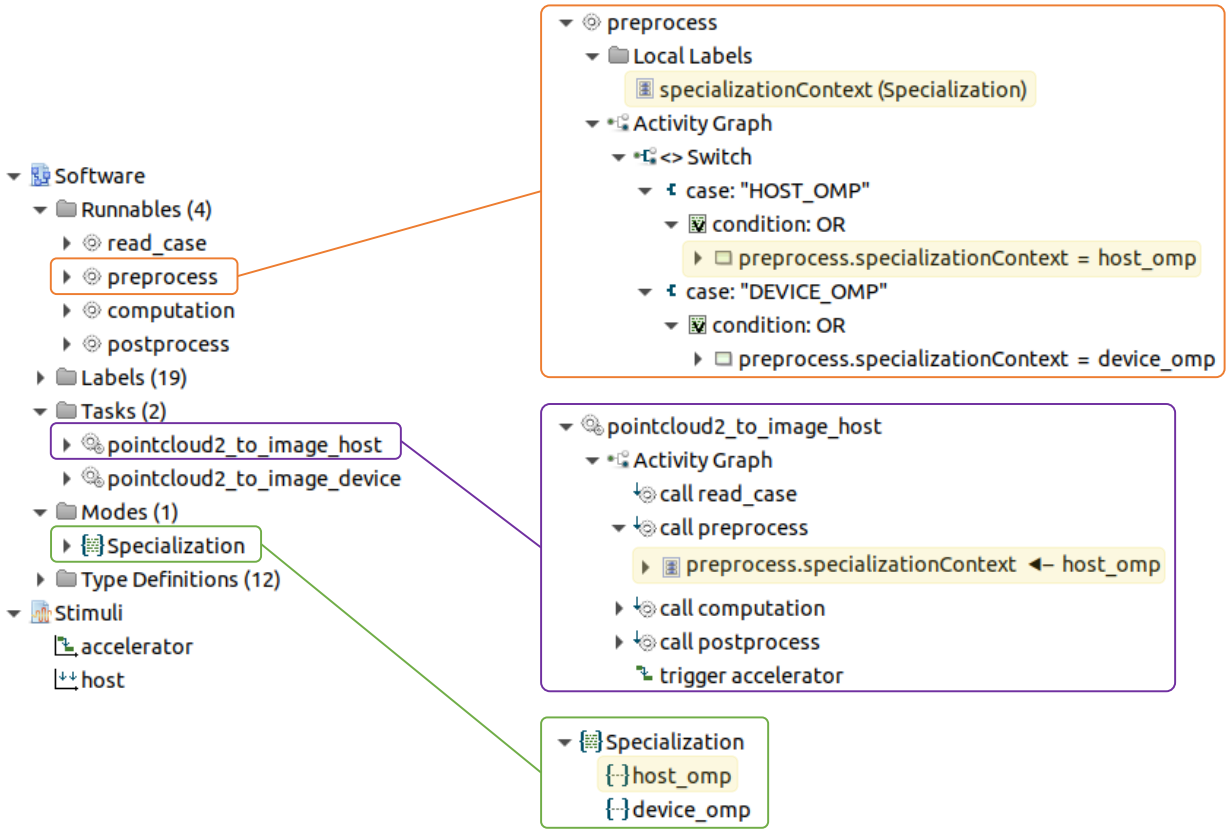


Figure 8 shows an example model extended with performance traces. To add the auxiliary data, custom `Entities` with custom `Properties` are used. By using custom elements, we do not change the designed behavior of the AMALTHEA model and still allow the addition of new features in the subsequent modeling processes (e.g. code generation) able to retrieve and analyze these values. Since results may vary depending on the executing environment, the extracted data is organized by hardware (a `Custom Entity`, as depicted in the first level of Figure 8). Each `Custom Entity` contains a set of custom properties, including a description of the hardware, and a map for each timing metrics organized by `Runnable` (the second level in Figure 8). The map contains the description of the target runnable and the timing analysis metrics, here specified per performance counter (level 3 in Figure 8).

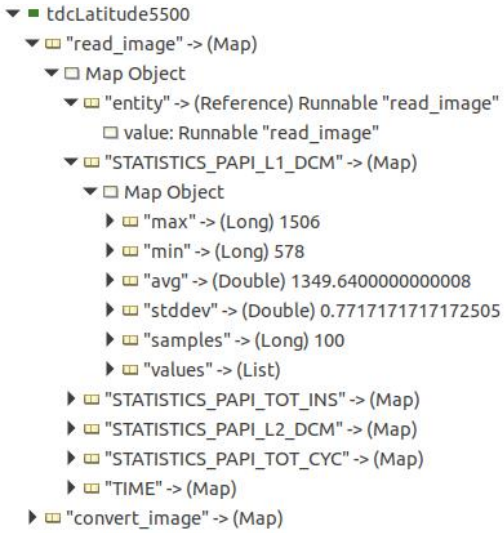
Each `Runnable` has a set of performance counters and execution time, which are defined by the following information:

- `Metric`: the name of the metric (e.g. Time or number of instructions);
- `Entity`: the target Runnable;
- `max`: the maximum value from all the existing executions;
- `min`: the minimum value from all the existing executions;
- `avg`: the average value of the metric;
- `stddev`: the standard deviation;
- `samples`: the total number of samples;
- `values`: a complete list of raw values.

In the case of Figure 8, four performance metrics are shown, besides the execution time of the Runnable. For instance, the example shows the number of instructions executed and the number of L1 cache misses.

Note that the annotation model is very flexible and might contain any metric relevant to the subsequent de-

Figure 8: Example of an annotated `Runnable` in an AMALTHEA model. The annotation was based on custom entities and custom properties.



velopment processes (e.g. model analysis and code generation). Although the approach is not tangled to a specific code generator, the generator could use the data in the model to customize the generated code for a given `Runnable`. Another use for this information is to perform scheduling decisions properly.

3.4 AMALTHEA Extensions for power-aware hardware capabilities

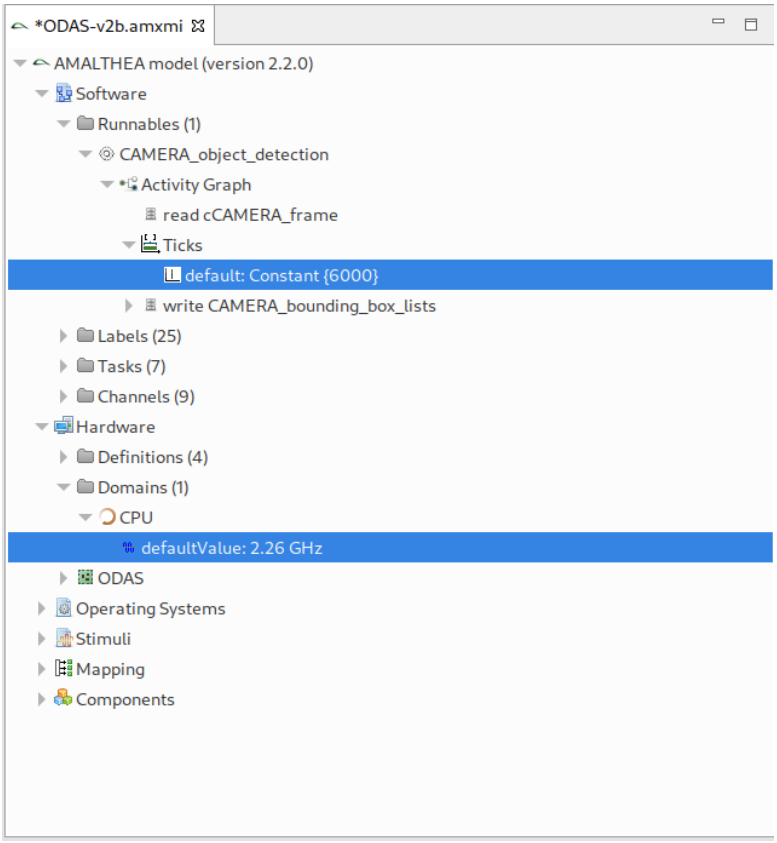
One of the key information provided in an Amalthea model for real-time applications is the specification of (worst-case estimates of) execution times of runnables and tasks. This is done by specifying in the software model the amount of clock cycles (Ticks) required for execution of each Runnable (in the worst-case). This number is meant to be multiplied by the frequency of the CPU where the Runnable will run, in order to obtain the worst-case execution time (WCET) that can be used for real-time analysis purposes. The CPU frequency can be specified in the hardware model, as the default frequency of the power domain associated with the CPU the Runnable is mapped onto (as due to it belonging to a specific Task). An example of the above two bits of information is shown in Figure 9, showing an excerpt of the Amalthea model for the ODAS use-case.

This allows for specifying computational requirements in a frequency-independent fashion. However, the hardware platforms investigated in AMPERE are characterized by a plurality of power operational modes (often called Operating Performance Points – or OPPs)[2]. The typical situation is to have an embedded multi-core hardware platform with power-saving features. Such a platform is characterized typically by one or more CPU islands, each of which can be configured to switch among a discrete number of OPPs. Each OPP corresponds to a different frequency of the CPU, and it results in lower instantaneous power consumption of the CPU both when continuously processing, and also when idle. Lower OPPs are also characterized by a lower voltage in powering the cores associated to the clock domain. Therefore, when switching to a lower-frequency power mode, what happens is that a Runnable execution time expands, while at the same time the power consumption of the CPU lowers.

In order to provide the designer with the ability to catch, albeit at a high level and through possibly approximate figures, this additional dimension of how execution times and power/energy consumption of the platform change, depending on the configuration options of the underlying platform, we propose extensions to the

[2]More information is available at: https://www.kernel.org/doc/Documentation/devicetree/bindings/opp/opp.txt.

Figure 9: WCET modeling in simple AMALTHEA models.



AMALTHEA hardware specification model, making use of Custom Properties. Shortly, we can specify a list of configuration pairs or triplets in terms of frequency of execution, and associated busy power consumption, as well as an optional idle power consumption. This is done by adding to a power Domain a `powerModes` custom property, being a list of map elements, where each map can specify up to 3 key/value pairs: a `frequency`, a `busyPower`, and an `idlePower` (this can be left unspecified and assumed negligible, if preferred).

An example design with this ability is shown in Figure 10, where a simple model excerpt is shown where the `MainClk` power Domain specification attached to the CPU (ECU at the bottom of the model), is characterized by 3 power modes (or OPPs), with different values for the CPU frequency, and associated busy and idle power consumption figures (the numbers in this figure are provided as an example, they do not correspond to any real platform).

The just introduced `powerModes` custom property can also be used to characterize different possible frequencies of operation for GP-GPU (or even FPGA) accelerators.

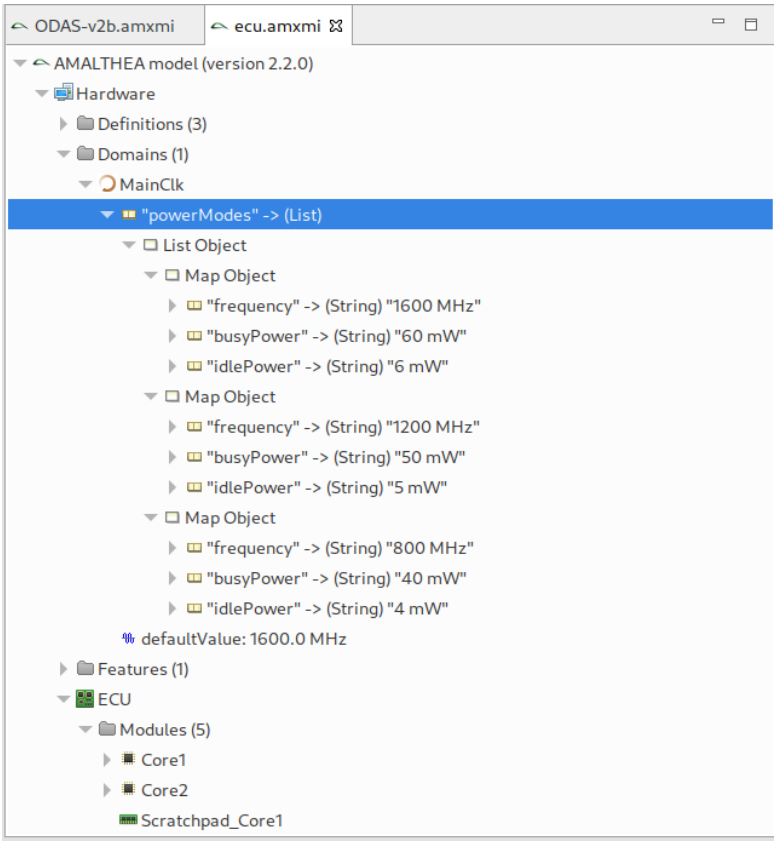
Example hardware designs whose power-saving abilities can be characterized with the above introduced Amalthea extensions, include the two platforms chosen as reference target boards for AMPERE:

- the Xilinx UltraScale+ ZCU102 platform, where there is a quad-core island with Arm Cortex A53 cores, a dual-core island with Arm Cortex R5F cores, and a FPGA fabric;
- the NVIDIA Jetson AGX platform, with an island with 8 CPU cores and a GP-GPU.

But this allows for modeling also generic embedded platforms making use of the Arm big.LITTLE (or DynamIQ) architecture, a very common choice of CPUs for embedded mobile devices, typically manufactured in octa-core configurations, with either 4+4 or 2+6 big and LITTLE CPU islands.

Examples of real island frequencies, as well as busy and idle power consumptions associated to each possible frequency of operations for a number of boards, including the ZCU102 mentioned above, can be found in [7] and the AMPERE deliverable D3.2 [8], Section 4.1.2. The gathered timing rescaling and power consumption

Figure 10: Modeling of different power modes in a simple AMALTHEA model.



figures have also been made available in the open-source PARTSim power-aware real-time system simulator[3] developed at SSSA for simulation-based verification of the correct timing and schedulability of a set of parallel real-time DAGs deployed on heterogeneous platforms, as needed in AMPERE.

3.5 CAPELLA Extensions for safety

Safety constraints imposed by the compliance to safety-related standards and best practices, which are typical of both automotive and railway scenarios, have been considered in the AMPERE project as a basis for a deep analysis of the impact of non-functional constraints on AMPERE ecosystem.

3.5.1 Safety Functions and SIL levels

Before going into details in the DSML extensions required by THALIT use-case, some information must be given about the concepts of “Safety Function” and “Safety Integrity Level” (SIL). Several industrial processes present an inherent risk of accidents that could lead to injuries to people or damages to the environment. To mitigate the impact and the likelihood of these accidents, one or more Safety-Related Systems can be put in place. These systems aim to monitor hazardous processes and intervene whenever something happens that could determine a hazardous event (an example of a Safety-Related System is the system monitoring and controlling the temperature in a nuclear plant). Functions implemented by Safety-Related Systems to monitor and control hazardous processes are called *Safety Functions*.

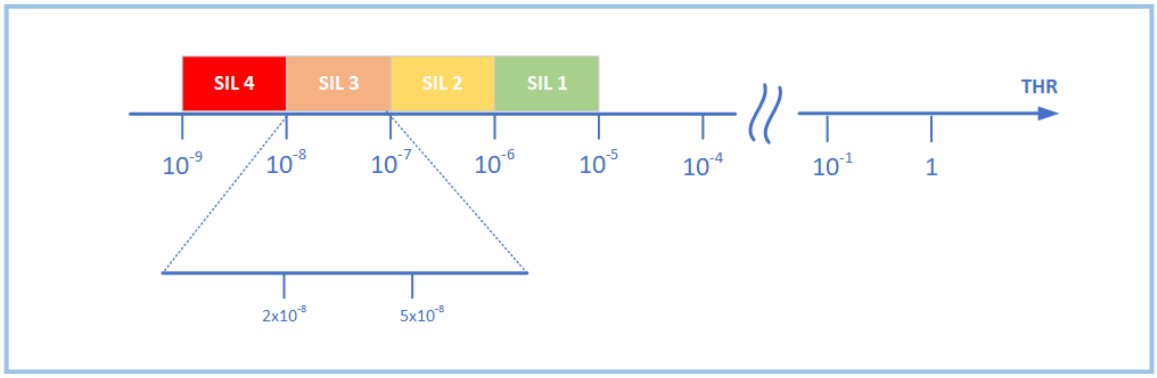
However, since Safety-Related Systems are designed and implemented by human beings, Safety Functions can

[3]More information at: https://github.com/gabrieleara/PARTSim.

also fail. The level of confidence we can have in a given Safety Function is called *Safety Integrity Level* (SIL). SIL ranges from SIL0 to SIL4, the higher the SIL level the safest the referenced Safety Function. By extension, the SIL is also a measure of the confidence level we can have in Safety-Related Systems (as they implement SIL Safety Functions). Rules Safety-Related Systems must adhere to reach a given SIL are defined in international standards (such as EN50126 and EN50128[4]). Deliverable D1.4 [9] contains more information about international safety standards.

The probability of failure in a SIL system is expressed through the Tolerable Hazard Rate (THR) which is measured in failures per hour. The relation between SIL values and THR is shown in Figure 11.

Figure 11: The relation between SIL and Tolerable Hazard rate (THR).



3.5.2 DSMLs extensions

A Preliminary Hazard Analysis has been performed on ODAS systems and two sets of safety constraints have been found. In this deliverable, we focus on one of them, i.e. the safety constraints directly impacting the AMPERE ecosystem. Safety requirements belonging to this set of constraints are reported below (see deliverable D1.4 [9] for more details):

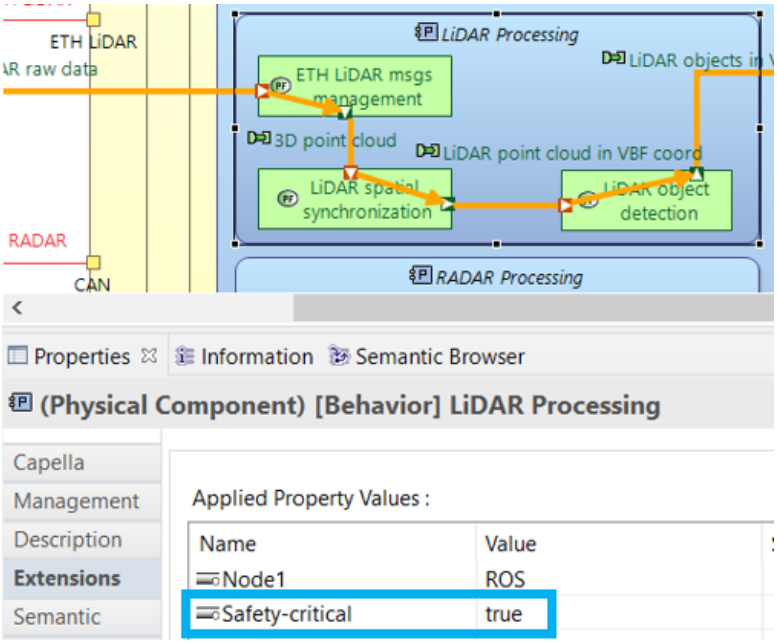
- ***[SYS-ODAS-REQ-200]**: AMPERE items implementing safety-critical aspects must be managed (e.g., designed, implemented, tested, etc.) according to EN50126.*
- ***[SYS-ODAS-REQ-201]**: Code generation tools must consider EN50128 constraints when generating code for safety-related design items.*
- ***[SYS-ODAS-REQ-204]**: A design item attribute "safety-related" must be available to designers to identify design items involved in implementing safety functions (safety related design items).*

As explained above, all these requirements have in common is that they impact on the AMPERE ecosystem as a whole, more than on the particular use-case we are analysing. This is why this set of requirements has been privileged in this deliverable analysis. This set of requirements highlights a weak side of the selected DSML tools (Capella and Amalthea). Namely, these tools don't have a specific way of addressing the design of Safety-Related Systems (which ODAS actually is, for many aspects).

To overcome this weakness, an analysis has been made to find a way to give a designer the possibility of managing safety requirements from the start of the design phase and throughout all project life cycle. At the Capella level, the problem was about how to give an engineer the possibility to specify that a design item was involved in implementing a safety function. To address this problem, a tag has been introduced in the Capella model to allow the designer to mark, right from the beginning of the design phase, modules involved in implementing safety functions. This tag has been labeled "Safety-critical" and can be assigned two values, "true" and "false" (see Figure 12).

[4]Available at https://standards.globalspec.com/std/10262901/EN%2050126-1.

Figure 12: The *Safety-critical* tag in Capella.



Another DSML extension involved the Amalthea tool. In this case, the problem was how to propagate to the Amalthea model the information about a Capella safety-critical design item (i.e. a Capella design item having been labeled with a "Safety-critical" tag set to "true"). Being the Amalthea model automatically generated by the Bridge tool (developed by TRT), this Amalthea extension also triggered a modification to the Bridge tool that had to be extended to propagate the Capella "Safety-critical" information into the Amalthea model. The extension implemented in the Bridge adds a tag to the "Tags" property of an automatically generated Amalthea item according to the following rules:

- if the Amalthea item is derived from a Capella item labeled "Safety-critical" the tag "Tag SIL4" is added by default;
- if the Amalthea item is derived from a Capella item not labeled "Safety-critical" (the default in Capella) the tag "Tag SIL0" is added.

Figure 13 shows how Amalthea design items are assigned a SIL level according to the Safety-critical status of their ancestors in Capella.

Tags added by the Bridge to the Amalthea model must be considered as a kind of *placeholder*. They are only used as a reminder to the designer that a given item is "safety-critical" so that this item requirement can be further managed at the Amalthea level. The designer is free to change the SIL4 tag whenever he/she thinks it's not the right SIL level to be used and add another one in the range SIL1 to SIL3.

This methodological principle is supported by the Bridge. This is required as the Bridge is not a simple translator from Capella to Amalthea but also a model synchronizer. Once an Amalthea model has been generated, it is free to evolve in parallel to the Capella model in order to, in particular, be completed by hand on detailed execution aspects prior to code generation. If the two models diverge on their "common part", the Bridge is able to detect the discrepancies and perform a reconciliation by updating the Amalthea model if and where confirmed.

Thus, a change of SIL level between 1 and 4 must not be considered as a discrepancy, while a change from 0 to 1-4 or the opposite is a significant one. The Bridge can be configured to tolerate or not these "acceptable changes of SIL level" (Figure 14). More information about the integration of the Bridge within the AMPERE ecosystem can be found in D6.4.

To have the AMPERE ecosystem fully compliant with safety standards, some extensions should also be consid-

Figure 13: *SIL* tags in Amalthea.

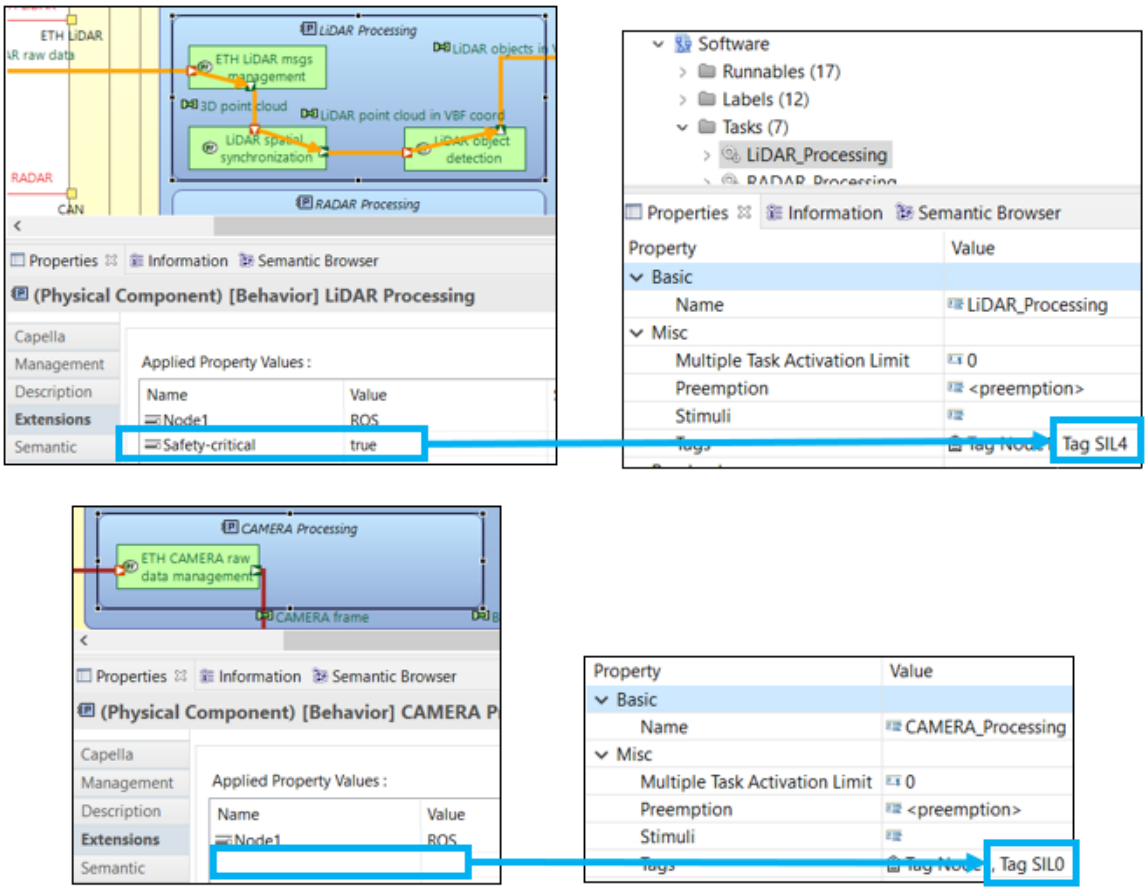
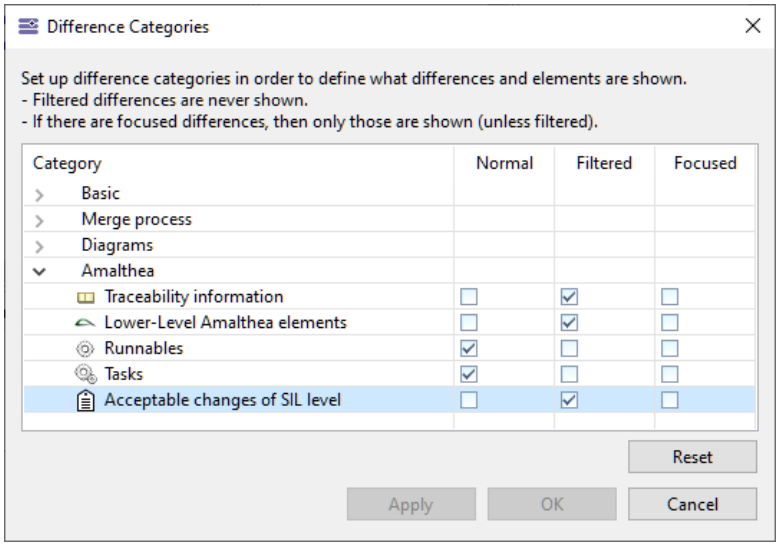


Figure 14: Configuring the Bridge to tolerate (filter out) acceptable SIL changes



ered for both the SLG tool (a code generating tool developed by BOSCH) and the algorithm allocating software modules to hardware components (CPU, GPU, FPGA, etc.). The first should be able to generate code compliant with safety standard EN50128. The latter should be able to manage hardware to software allocation considering that a SILx software module can only be executed on a hardware platform that can satisfy the same SIL requirements. However, these extensions haven't been considered (not falling in the "DSML extensions" category) and have been left for future analysis.

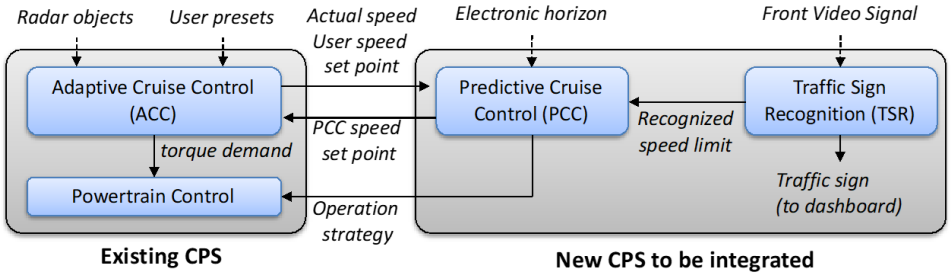
4 Use-Cases

This chapter describes updates for both AMPERE use-cases.

4.1 Automotive Use-Case

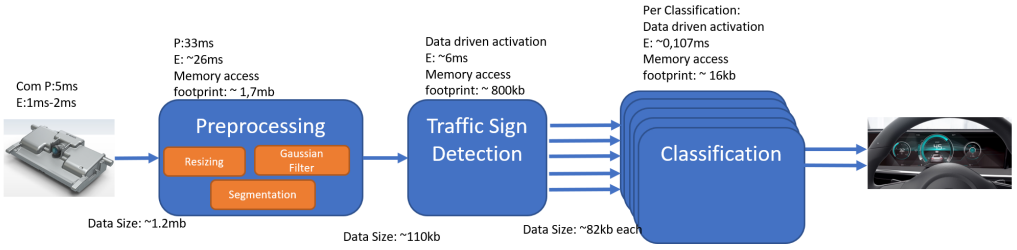
This chapter details the use-case description and explain how we will demonstrate the benefits of methods and technologies developed in AMPERE. The selected Automotive Use-Case is the intelligent *Predictive Cruise Control* that consists of four major components, the *Powertrain Control (PT)*, the *Adaptive Cruise Control (ACC)*, a *Traffic Sign Recognition (TSR) and the actual Predictive Cruise Control (PCC)*. They are connected as depicted in Figure 15. The model can be downloaded in the AMPERE repository [10].

Figure 15: Automotive Use-Case



The power train / engine control management (ECM) is publicly available and described in detail in [11]. The TSR application consists of three main tasks. The first task takes the image provided by the camera and pre-processes the input. The second task identifies traffic sign shapes in the picture and sends these parts of the image to the classification. The classification task is finally identifying the traffic sign and send the result to the dashboard. The classification can be instantiated multiple times and executed in parallel. The TSR is illustrated in Figure 16. In the use-case we define a static upper bound of classification tasks and we assume they execute in every execution. The preprocessing is triggered periodically every 33ms, around 30 frames per second. The subsequent tasks are triggered by an inter process trigger from their predecessor. The complete TSR model consists of 10 runnables and 40 labels.

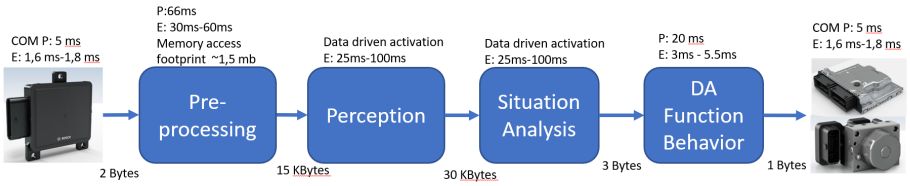
Figure 16: Automotive Use-Case - TSR



The ACC in Figure 17 consists of four main tasks, one for preprocessing the data from the radar system, the perception task to identify objects and correlate them to objects from previous data. The situation analysis predicts the movement of objects and the driver assistant (DA) function behavior to calculate the torque demand to the ECM. The complete ACC model consists of 100 runnables and 450 labels.

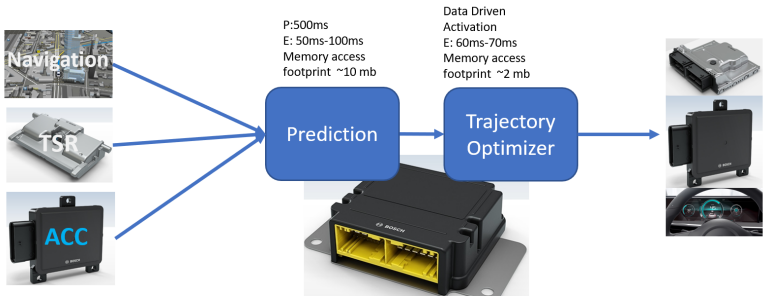
The main PCC functionality is depicted in Figure 18. It consists of two main tasks. The prediction calculates the best driving strategy using data from the navigation system, the current speed limit from the TSR and the user

Figure 17: Automotive Use-Case - ACC



speed set point. The trajectory optimizer calculates the speed value for the ACC and the operation strategy for the ECM. The complete PCC model component consists of 120 runnables and 400 labels.

Figure 18: Automotive Use-Case - PCC



4.1.1 Communication between Applications

In this use-case the communication within an application is modeled with labels while the data exchange between applications is done via channels. The communication is handled via ROS2/mircoROS as explained in Deliverable D1.4 [9], Section 3.1.2.. To bridge the world between the classic AUTOSAR ECM and the other components in the system we introduce a data broker task. Figure 19 shows the data broker from the ECM. The task is reactive to events that indicate the arrival of a new data point at the channels for the ECM. The runnable then copies the data from the channels to the labels within the ECM for a direct access. The ECM task uses microROS as communication infrastructure as it can be seen in the lower part of the screenshot that displays the *Tags* (and other attributes of the selected object).

4.1.2 Requirements

The use-case provides several constraints and requirements that need to be fulfilled as presented in Figure 20. It includes for example response time constraints for real-time tasks and cause-effect chains that impose timing requirements for e.g. the data flow from a sensor to an actuator. Runnable separation constraints are used to specify safety related mapping constraints as described in Deliverable 1.4 [9] in Section 2.2.

To specify the safety level of components we use special Tags in the model. We tag runnables, that need to fulfill certain requirements, with a safety level. Parts of the ECM runnable, mostly the runnables in the 10ms Task, are ASIL-B[1]. They need to be executed on certified hardware. The ACC also has ASIL-B parts. Due to safety decomposition described in Deliverables D1.2 [12] and D1.4 [9] we can implement two diverse algorithms that fulfill the same functionality. The AMPERE tool chain will respect the constraints that are represented by runnable-separation-constraints in the model. A modeling example from the ACC was already presented in Deliverable 1.4 [9].

[1]Defined at https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en.

Figure 19: Automotive Use-Case - ROS Data Broker

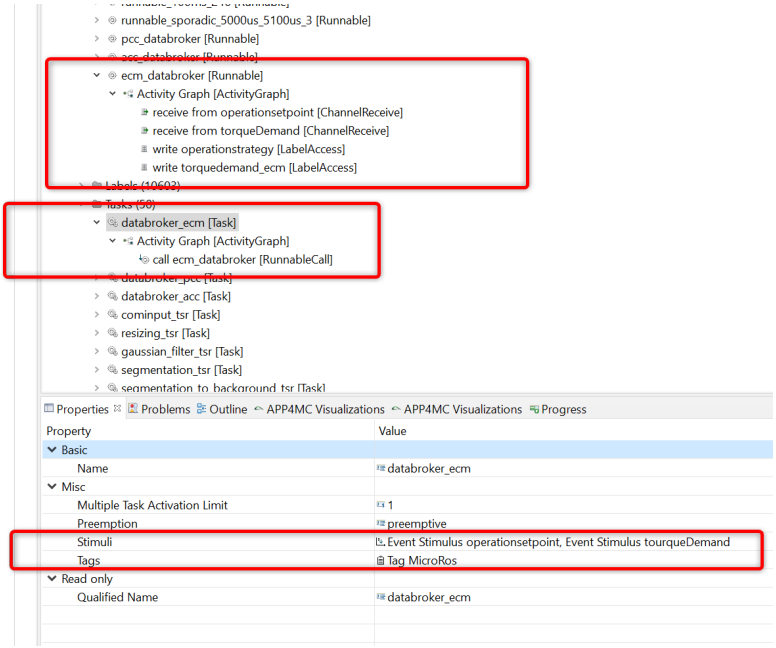
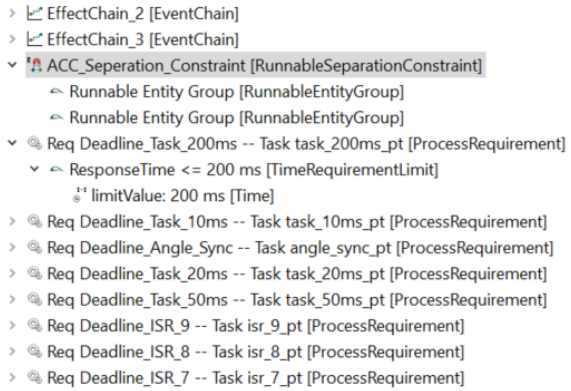


Figure 20: Automotive Use-Case - Constraints and Requirements



4.1.3 Specializations

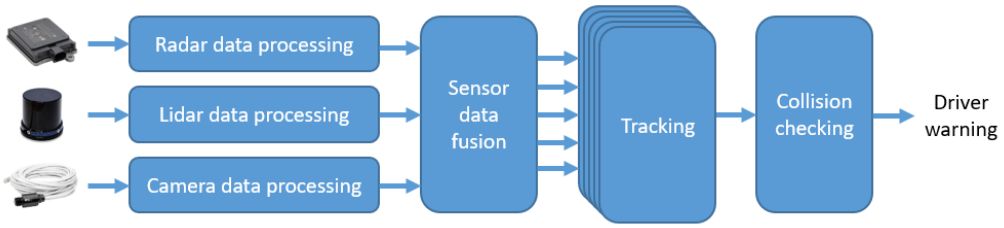
The automotive use-case provides specializations for runnables to exploit the computational power of the accelerators of the heterogeneous platforms. In the use-case the ACC pre-processing and perception contains specializations, that are capable of being offloaded to the FPGA as an alternative to a CPU execution. The TSR applications allows the classification and image pre-processing to be executed on the GPU. This leads to different timing and energy consumption and gives the multi-criteria tool chain options to optimize and improve the system performance and energy consumption. The mechanisms to describe specializations are described in Section 3.2.

4.2 Railway Use-Case

THALIT main contribution to the AMPERE project is the definition of a use-case taken from a real-world railway scenario. The goal of this use-case is to *stress* the AMPERE model by injecting non-functional constraints from a real-world system.

The use-case consists of a system to detect and avoid railway obstacles that could be considerd a hazard to the train movement (Obstacle Detection and Avoidance System, ODAS). The ODAS consists of sensors (LIDAR, camera and RADAR), a data fusion software module used to merge information from all these sources, a tracking module to follow the trajectory of potential obstacles, and a collision checker module to activate a warning in case of a proven obstacle (see Figure 21). The tracking is done by unscented Kalmann filters on mutiple objects at the same time so the filters are executed in parallel. Once transformed into an Amalthea model, the ODAS model consists of 17 runnables and 21 labels.

Figure 21: ODAS System architecture



Safety requirements. As explained in Section 3.5, the safety related functions can be tagged by the "safety-critical" label. In the ODAS, all functions are safety related, except those implementing the image processing. The camera data are used as a complement to the LiDAR and RADAR ones, but they are not mandatory to detect potential obstacles. Therefore, all the inner behavioral components of the Capella ODAS model are labeled as "safety-critical", and all the corresponding tasks in the Amalthea model are tagged SIL4 by default, except the camera processing tasks. As expressed in requirement ***[SYS-ODAS-REQ-208]*** in D1.4 [9], SIL4 software must be executed on a hardware platform with a 2oo2 architecture.

Specializations. In the ODAS use-case, the processing of the camera images may be offloaded to a GPU for better performances. This may lead to optimizations from a performance point of view even if it is not the case from the energy consumption one. The AMPERE tool chain will analyze the different options and produce an optimal combination from both point of view. To express the option among the different implementations, we use the specialization mechanism in Amalthea described in Section 3.2.

4.3 Use-Case Evaluation

The methods and tools developed in AMPERE and their interoperability will be evaluated using both use-cases. The use-case model will be nearly identical which highlights the benefit of a model-based approach that allows industry partners to explore and optimize parallelism and performance on modern high performance embedded platforms while still adhere to (non-)functional requirements. We will highlight the differences in the model for both platforms under evaluation and explain the setup how to evaluate the developed methods.

4.3.1 Automotive Use-case with Xilinx Ultrascale+

The AMPERE workflow for evaluating the use-case on the Xilinx Ultrascale+ is displayed in Figure 22. The platform has four Cortex A cores and one FPGA. The software model will include the redundancy definitions and the specializations for the GPU and the FPGA. Only the FPGA based specializations will be considered in the multi-criteria optimization due to the lack of an available GPU on the board. The Cortex A53 cores of both boards are not certified to ASIL B which would violate the requirements specified in the use-case model[2]. Adhere to mapping requirements is a key part of the multi-criteria toolchain of AMPERE. Fulfilling

[2]Although the Cortex-R5 cores are certified for ASIL C and SIL 3, these are not accessible by PikeOS, thus, they will not be used.

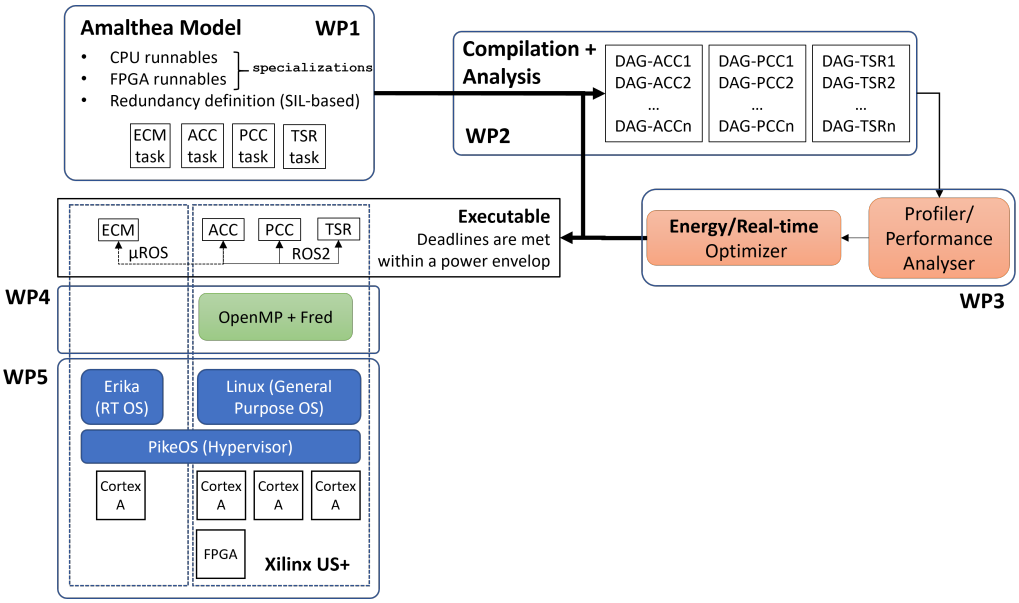
this requirement, if certified core are available is a minor technical extension. That is the reason to relax this requirement in the evaluation. In a real product the availability of certified cores will be ensured.

We deal with the ECM component in the Xilinx evaluation in introducing a mapping constraint, that will bound the ECM application to an isolated core while all other components will be executed on the other cores. On top of that the PikeOS Hypervisor will be deployed on the system for spatial separation the ECM.

The ECM will be deployed on the automotive OSEK/VDX certified Hard Real Time Operating System ERIKA OS while the other components will execute on a Linux system. To bridge the communication between the safe ERIKA OS and the Linux an adapter between microROS and ROS2 was implemented to connect the ECM on ERIKA that uses microROS while the Linux system will use ROS2. OpenMP and FRED[13, 14] will be used to exploit the parallelism of the software and to make use of the FPGA accelerator.

The multi-criteria optimization design flow is explained in detailed in D2.4 [15]. On the Xilinx platform AMPERE will optimize energy consumption and real-time requirements at the same time. That means deadlines will be met while energy consumption will be reduced if possible.

Figure 22: Automotive Use-Case - Xilinx



4.3.2 Automotive Use-case with NVIDIA Jetson AGX

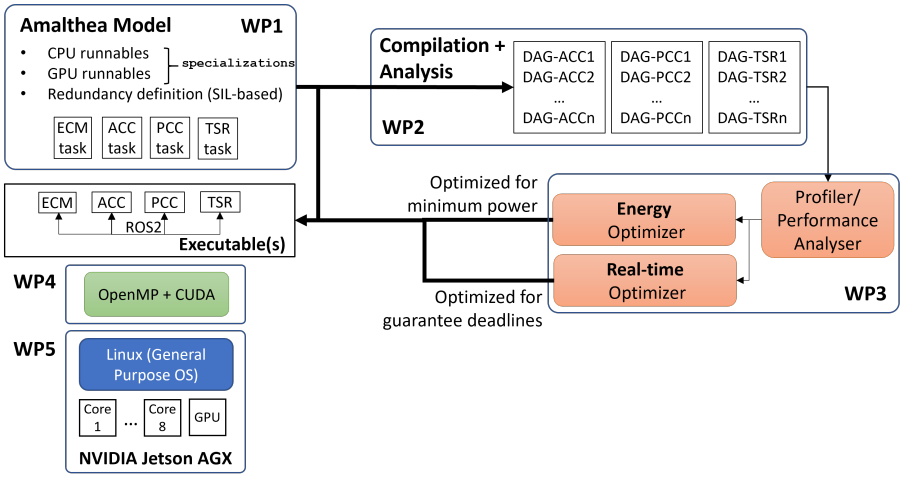
The NVIDIA Jetson AGX platform has eight cores, no FPGA but a GPU. In this evaluation we will use CUDA to offload computation to the GPU instead of FRED for the FPGA offloading. Also only one Linux system will be deployed on this platform since the PikeOS hypervisor cannot be deployed on this platform due to lack of available detailed documentation.

The use-case model has only a few changes. The communication scheme for the ECM application will be changed to ROS2. And we will change the evaluation of the ASIL B requirement on this platform. Again, the platform is not ASIL B certified. In this example we are not using PikeOS and ERIKA OS. Separation of PikeOS and the certifid ErikaOS is therefore not available. In this case we require redundant execution of the ASIL B runnables on separated cores. The AMPERE toolchain will automatically take care of this redundancy in the execution.

In the multi-criteria optimization we follow two different strategies. One strategy is to minimize the energy consumption while fulfilling all timing requirements. The other strategy will optimize the timing requirements. Optimizing timing requirements means to e.g. provide the biggest slack between response time of a task and its deadline. This optimization will still adhere the maximum energy budget requirement.

Having both options is beneficial since depending on the system either timing (hard-real time) or power consumption (soft real-time) with high power consumption in a difficult environment, e.g. in the front wind shield of a car or near a combustion engine is more important. The workflow for evaluating the automotive use-case on the Jetson is displayed in Figure 23.

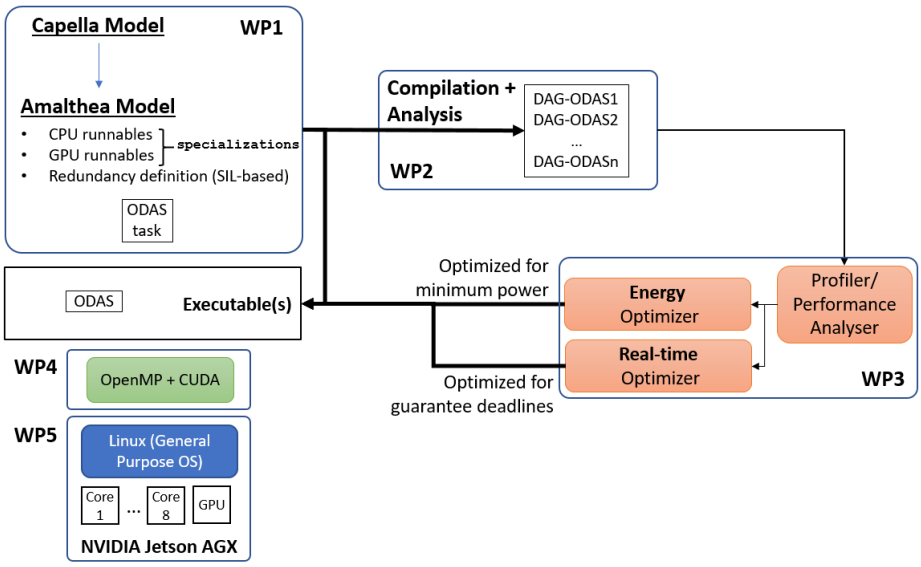
Figure 23: Automotive Use-Case - Jetson



4.3.3 Railway Use-Case with NVIDIA Jetson AGX

The evaluation of the AMPERE tool chain with the NVIDIA board on the railway use-case faces the same challenge as the automotive use-case, since the platform is not compliant with SIL4 software. Therefore, we implement the 2oo2 architecture requirement by requiring redundancy of SIL4 runnables and separation of their executions on different cores. This can be done thanks to the corresponding redundancy and separation constraints mechanisms in Amalthea. The overall multi-criteria strategy is the same as for the automotive use-case. The workflow for evaluating the railway use-case on the Jetson is displayed in Figure 24.

Figure 24: Railway Use-Case - Jetson



5 Conclusions

This document presented the contributions for WP1, related to DSML. These contributions can be summarized as:

1. AMALTHEA/Capella extensions in Chapter 3, proposing extensions for middleware communication (ROS) and execution behavior (Section 3.1), for modeling runnable specializations (Section 3.2), for capturing performance metrics (Section 3.3), for modeling DVFS-enabled platforms, for tagging modeling elements with safety requirements (Section 3.5);
2. Refined description of the Automotive/Railway use-cases, in Chapter 4, and how they will benefit of the toolchain under development in AMPERE, and how they will be deployed and optimized on their respective target platforms.

At this point, all DSML extensions required by the AMPERE ecosystem were proposed. However, the final steps to be performed in AMPERE in the context of WP1, are related to the validation of the proposed extensions and methodology by integration within the two demonstrator use-cases AMPERE is building, and the use of these extensions in an integrated and consistent way throughout the whole design, prototyping/development, optimization and validation process. This concerns, for example, the problem of translating the information provided in the extensions described above, from the DSML modeling language format, to a format compatible with the end-to-end AMPERE workflow described in D2.4 [15] (e.g., TDG file format and/or entries in the accompanying optimization configuration file).

6 Acronyms and Abbreviations

ACC	Adaptive Cruise Control
ATDB	AMALTHEA Trace Database
CPU	Central Processing Unit
D	Deliverable
DAG	Direct Acyclic Graph
DSML	Domain Specific Modeling Language
DVFS	Dynamic Voltage and Frequency Scaling
ECM	Engine Control Management
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
MDE	Model-Driven Engineering
MS	Milestone
ODAS	Obstacle Detection Avoidance System
OPP	Operating Performance Points
OS	Operating System
PCC	Predictive Cruise Control
PT	Powertrain Control
ROS	Robot Operating System
SLG	Synthetic Load Generator
SIL	Safety Integrity Level
T	Task
TDG	Task Dependency Graph
THR	Tolerable Hazard Rate
TSR	Traffic Sign Recognition
UC	Use-Case
WP	Work Package
WCET	Worst-Case Execution Time

7 References

[1] AMPERE, “D1.3. First release of the meta model-driven abstractions,” 2021.

[2] ——, “D1.1. System Models Requirement and Use Case Selection,” 2020.

[3] ——, “D2.3. Programming model extensions and the multi-criteria performance-aware component,” 2022.

[4] ——, “D2.2. First release of the meta parallel programming abstraction and the single-criterion performance-aware optimisations,” 2021.

[5] L. Sommer, F. Stock, L. Solis-Vasquez, and A. Koch, “DAPHNE - An automotive benchmark suite for parallel programming models on embedded heterogeneous platforms: work-in-progress,” in *Proceedings of the International Conference on Embedded Software Companion*, 2019, pp. 1–2.

[6] AMPERE, “D3.3. Energy optimisation framework, predictable execution models and analysis, and software resilient techniques,” September 2022.

[7] G. Ara, T. Cucinotta, and A. Mascitti, “Simulating execution time and power consumption of real-time tasks on embedded platforms,” in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 491–500. [Online]. Available: https://doi.org/10.1145/3477314.3507030

[8] AMPERE, “D3.2. Single-Criterion Energy-Optimization Framework, Predictable Execution Models, and Software Resilient Techniques,” 2022.

[9] ——, “D1.4. Analysis of functional safety aspects on multi-criteria optimization and final release of the test bench suite,” 2021.

[10] “PCC Use-Case Model,” https://gitlab.bsc.es/ampere/ampere-project/-/tree/master/Software/PCC%20model%20in%20Amalthea/PCC_UseCase.

[11] S. Kramer, D. Ziegenbein and A. Hamann, , “"Real world automotive benchmarks for free",” 2015.

[12] AMPERE, “D1.2. Analysis of functional safety aspects on single-criterion optimization and first release of the test bench,” 2022.

[13] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, “A framework for supporting real-time applications on dynamic reconfigurable fpgas,” in *Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)*, December 2016, pp. 1–12.

[14] AMPERE, “D4.3 Integrated run-time energy support, and predictability, segregation and resilience mechanisms,” 2022.

[15] ——, “D2.4 Multi-criteria optimization model transformation,” 2022.