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

D6.3 Single criterion AMPERE ecosystem

Version 1.0

Documentation Information

Contract Number	871669
Project Website	www.ampere-euproject.eu
Contratual Deadline	30.03.2021
Dissemination Level	[PU]
Nature	DEM
Author	Thomas Vergnaud (TRT)
Contributors	Olivier Constant (TRT) Alexandre Amory and Tommaso Cucinotta (SSSA) Enkhtuvshin Janchivnyambuu (SYS)
Reviewer	Claudio Scordino (EVI)
Keywords	ecosystem, integration, workflow

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

Change Log

Version	Description Change
V0.1	Initial version
V0.2	Addressed comments after review
V0.3	Updated appendix A to clarify the Capella to Amalthea bridge rules
V0.4	Rephrased explanations in section 6 according to discussions among partners regarding platform configurations
V1.0	Release version

Table of Contents

1. Executive Summary

This deliverable describes the AMPERE ecosystem at milestone MS2.

It explains the general AMPERE workflow, and provides a list of the tools involved in each step of the workflow. For each tool, the document indicates the version, where to get the tool, and a short description of its role in the workflow.

This document is associated with a video that demonstrates the first steps of the ecosystem integration. The video is available at B2DROP repository of BSC through the following link: https://b2drop.bsc.es/index.php/s/yPGEn2wjGrAddeN and by using the password: "RRAXqgZ3" (without quotes). The document provides information to help understand what is shown in the demonstration. It is also a reference document that specifies the version of each tool involved in the demonstration.

In addition, the document makes a specific focus on the description of the concepts involved in the modeling step of the AMPERE workflow. An appendix details the relevant modeling elements in Capella and Amalthea.

2. Introduction

The AMPERE ecosystem consists of a collection of software tools. These tools are independent one from another; they have been developed separately by each partner. For the needs of project AMPERE, the tools are combined in order to create workflows. For some of them, their integration in the AMPERE ecosystem requires either some adaptation or configuration. For others, no specific adaptation is needed.

The AMPERE workflow is made of three main steps:

1. the modeling of the system architecture;
2. the analysis and optimization of the architecture;
3. the execution on a hardware and software platform.

The document first provides an overview of the general workflow. Then the tools involved in each step are briefly described.

3. Overview of the AMPERE ecosystem

The AMPERE ecosystem consists of a collection of tools. These tools are combined to support the AMPERE workflow. The software architecture of the AMPERE tools, as it was designed at the beginning of the project, is illustrated on figure 1.

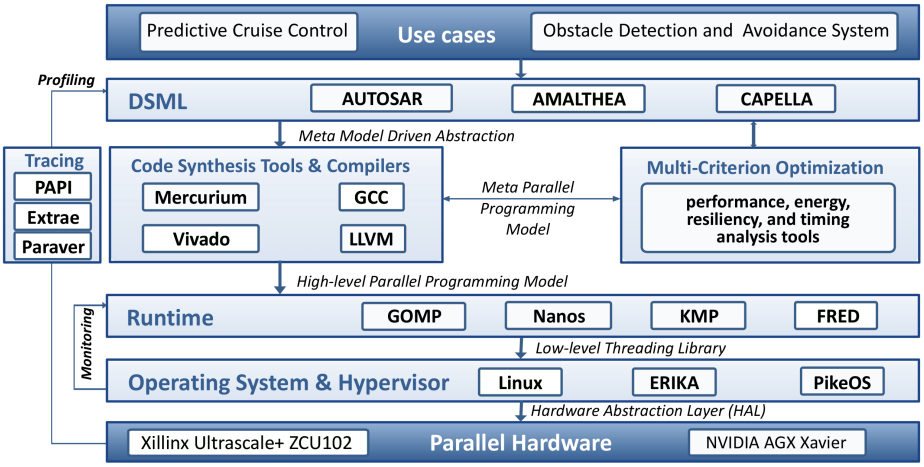

Figure 1: AMPERE software architecture

The integration of the AMPERE tools consists of enabling a workflow between them. This workflow is represented on figure 2. It consists of three main steps:

- system modeling;
- optimization of the model configuration;
- deployment and execution on the target platform.

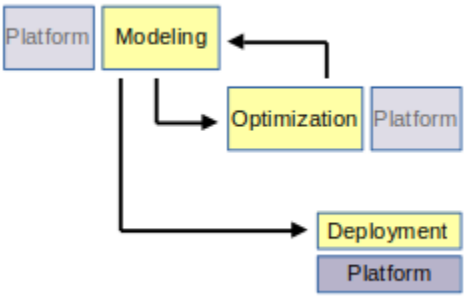

Figure 2: AMPERE workflow

Each step (modeling, optimization, deployment) is related with the execution platform. Models capture the platform characteristics. Optimization may rely on measurements performed on the platform. Algorithms are deployed on the platform.

In the context of AMPERE, system modeling mainly focuses on the allocation of algorithms to the execution resources of the platform (CPU, GPU, FPGA). It also focuses on the fine-grain description of synchronizations and parallelism between computations. The tools involved in this step are listed in section 4.

Several analysis and optimization are applied in order to refine the fine-grain system modeling. The optimization of the fine-grain system modeling is a major step of the AMPERE workflow. Optimization is performed

following the results of various analyses. Project AMPERE aims to combine several optimization criteria in order to reach the best trade-off for the allocation of processings to execution resources and the configuration of the parallelism. At this stage of the project, each optimization criterion is considered separately. The combination of the crtieria will be addressed later in the project. The tools and methods involved in this step are listed in section 5.

Once the best system configuration is identified, the algorithms are actually deployed on the execution platform. Different variations of the execution platform are used in AMPERE, depending on the need for a powerful GPU, an FPGA, safety enforcement, etc. In AMPERE, the execution platform is used for the deployment of the actual system. It may also be used for performance measurements when optimizing the system modeling. The execution platform caracteristics are listed in section 6.

4. Modeling

The AMPERE ecosystem relies on two different modeling tools: Capella and App4MC. Capella is a graphical editor based on the Arcadia methodology; it enables the modeling of high-level system architectures. App4MC is based on the AMALTHEA meta-model; it enables the modeling of fine-grain system architectures.

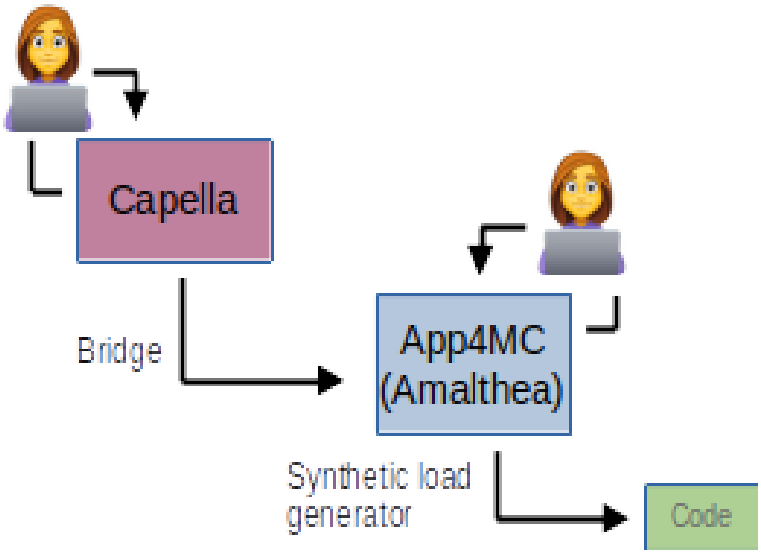

Figure 3: AMPERE modeling steps

4.1. Capella

Capella is a graphical editor for system architecture modeling. It is mainly contributed by Thales. Figure 4 shows a screenshot of Capella.

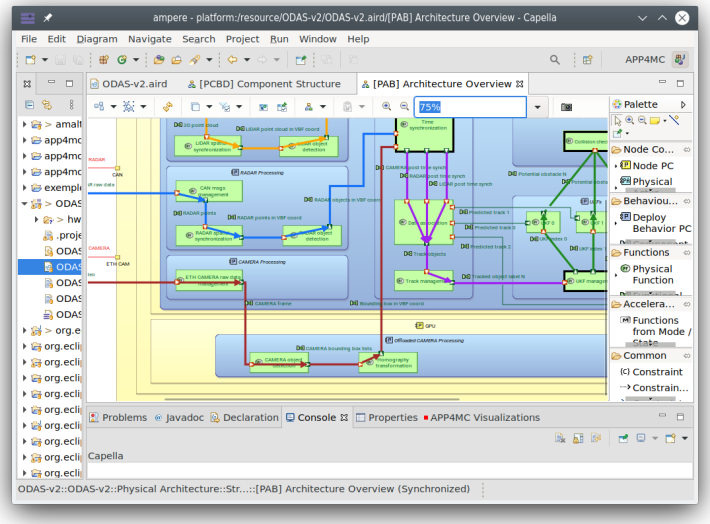

Figure 4: A screenshot of Capella

Capella can be downloaded from https://www.eclipse.org/capella/. The AMPERE ecosystem re-

lies on Capella 5.0.0.

An explanation of the Capella elements that are taken into account for the AMPERE ecosystem is provided in section A.1.

4.2. App4MC/AMALTHEA

App4MC is the implementation of the AMALTHEA project. Therefore, it is sometimes refered to as “Amalthea” in the AMPERE deliverables. It is provided by Bosch. Figure 5 shows a screenshot of App4MC.

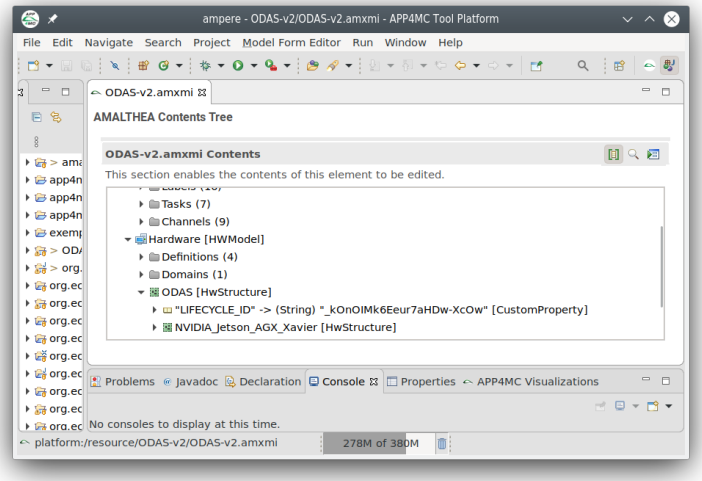

Figure 5: A screenshot of App4MC

App4MC can be fetched from https://www.eclipse.org/app4mc/. The AMPERE ecosystem relies on App4MC 1.0.0.

App4MC/Amalthea is the central system model in project AMPERE. An explanation of the Amalthea elements that are taken into account for the AMPERE ecosystem is provided in section A.2.

4.3. Transition from Capella to App4MC

In order to integrate Capella and App4MC, Thales Research & Technology is developing a transformation tool from Capella to Amalthea. This tools is still under development, and is refered to as the “Capella-Amalthea bridge”. It is provided to every member of the AMPERE consortium. Its source code is currently located in the Thales forge https://gitlab.thalesdigital.io/ with limited access. It is planned to put the source code in the Ampere Git repository once the bridge is released.

An explanation of the transformation rules from Capella to Amalthea is provided in section A.3.

4.4. AMALTHEA to FPGA design flow

Scuola Sant’Anna developed a tool to help design AMALTHEA models to fit FPGA. It allows the designer to easily switch the runnables implementation from software to hardware (or vice versa). This enable the modification of the Amalthea fine-grain system model for a fast design space exploration.

It is combined with DART (Section 5.2.2), FRED (Section 6.2.5), and a specific version of the synthetic code generator (Section 5.2.3).

The FPGA flow repository can be accessed from `https://gitlab.retis.santannapisa.it/a.amory/amalthea4dart_plugin.git`. The AMPERE ecosystem uses FPGA flow version 0.1-alpha.

5. Analysis & Optimization

Architecture optimization is a major aspect of the Ampere workflow. Various optimization methods are applied. At this stage of project AMPERE, each criterion is considered separately. Every optimization chain is under development.

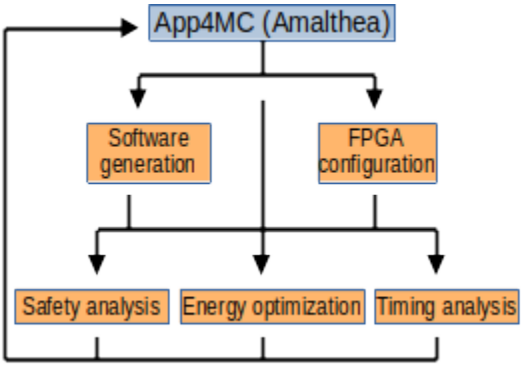

Figure 6: AMPERE optimization criteria

Several optimization criteria are considered for the moment:

- timing analysis;
- safety aspects;
- energy optimization.

The Amalthea model is the central architecture representation in Ampere. Analysis and optimization techniques process the same Amalthea model.

Besides the processing of an Amalthea model, some analysis techniques may require the extraction of data from an actual system execution. Software code generation and FPGA platform configuration are cross-criteria concern. They enable analysis technique to extract data from actual system executions.

Amalthea models are updated following the optimization results.

5.1. Software Code Generation

Some analysis and optimization calculation require measurements made on actual software code execution.

See deliverable D2.2 [1] for details.

5.1.1. Synthetic Load Generator (SLG)

The synthetic load generator (SLG) is a tool developed by Bosch. Its purpose is to generate “dummy” software C++ code from an App4MC model. This generated code simulates the execution of the actual algorithmic code.

The SLG is currently not packaged; it is under development by Bosch. The source code can be fetched from `https://git.eclipse.org/r/app4mc/org.eclipse.app4mc.addon.transformation.git`.

5.1.2. LLVM and GCC

LLVM is a compiler. It turns source code into actual executable applications. LLVM is free software, shipped with Linux distributions.

The web page is https://llvm.org/. The AMPERE ecosystem uses LLVM version 11.1; any newer version should also fit.

Alternatively to LLVM, the AMPERE ecosystem also uses GCC, which is another well-known compiler. GCC is also free software, shipped with Linux distributions. Any version of GCC later than version 7 should fit.

5.1.3. Extrae

Extrae is a software library and tool to trace the processor execution load. It is provided by BSC.

Extrae can be fetched from https://tools.bsc.es/downloads. The AMPERE ecosystem uses Extrae version 3.4.1. Extrae is to be combined with the PAPI library (Performance Application Programming Interface) to get precise metrics. PAPI is provided by Linux distributions. The PAPI website is http://icl.cs.utk.edu/papi.

Extrae can be combined with Paraver to visualize execution measurements. Like Extrae, Paraver can be fetched from https://tools.bsc.es/downloads. The AMPERE ecosystem uses Paraver version 4.9.2.

5.2. FPGA Configuration

See deliverable D3.2 [2] for details.

5.2.1. PARTProfiler

PARTProfiler (or PARTProf) is a collection of software tools to profile application behavior and power consumption on DVFS capable embedded systems. It is provided by SSSA.

PARTProfiler repository can be accessed from https://github.com/gabrieleara/PARTProf. The AMPERE ecosystem uses PARTProfiler version 0.1-alpha.

5.2.2. DART

DART is a tool, developed by Scuola Sant'Anna, to automate the design flow in a real-time dynamic partial reconfiguration based system with both software and hardware components. DART fully automates several steps of the FPGA design process including the partitioning, floorplanning, routing, and bitstream generation phases. DART repository can be accessed from https://repo.retis.sssup.it/pr_tool. The AMPERE ecosystem uses DART version 0.1-alpha.

DART hardware IP repository, or DART IPs for short, is a set of ready to use hardware IPs that, combined with DART and the proposed FPGA flow (Section 4.4), allows the designer to easily switch the runnables design from software to hardware implementation, easing the design space exploration. DART IPs repository can be accessed from https://gitlab.retis.santannapisa.it/a.amory/dart_ips. The AMPERE ecosystem uses DART IPs version 0.1-alpha.

5.2.3. Specific Synthetic Load Generator

A specific declination of the SLG to enable the communication between software and FPGA is developed by Sant'Anna. It is not released yet.

5.3. Analysis and Optimizations

Different analysis and optimization techniques are being developed in project AMPERE. Most of the associated tools are under development. These techniques either need extract information from the actual code (see section 5.1) or the FPGA execution (see section 5.2), or directly from an Amalthea model.

5.3.1. Safety

Safety analysis is directly performed on the Amalthea model by considering safety constraints. See deliverable D1.2 [3] for details.

5.3.2. Timing Analysis for Predictable Execution

See deliverables D3.2 [2] and D4.2 [4] for details.

Techniques are being developed by ISEP for predictable execution. They are based on the analysis of Amalthea models. The associated tools have not been released yet.

5.3.3. Energy

See deliverables D3.2 [2] and D4.2 [4] for details.

A tool is being developed by ETH Zürich to evaluate energy optimization from the modeling of execution tasks in an Amalthea model. The tool is at an early development stage, and has not yet been released.

6. Execution Platform

This chapter describes the execution platform for AMPERE. The execution platform consists of operating systems and software frameworks to execute the algorithms.

The general platform architecture of the AMPERE ecosystem is illustrated on Figure 7. Tasks will be controlled by the Robot Operating System framework (ROS), widely used in the industry.

ROS relies on the Data Distribution Service (DDS) standard to implement communications between software components. Thus, it eases the integration of the software parts that are executed on CPUs. The software parts that are executed on a an Nvidia GPU are not managed by ROS; they are directly managed by the Nvidia CUDA infrastructure. ROS will also manage communications between computation tasks.

At this stage, the integration of ROS is still ongoing. For the moment, project AMPERE targets version “foxy” of ROS. This version of ROS can be fetched from `https://docs.ros.org/en/foxy`.

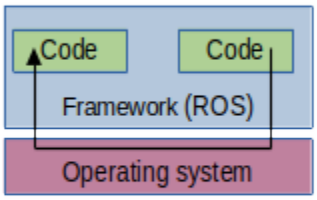

Figure 7: AMPERE generic platform

he general AMPERE platform architecture has several variations, depending on the targeted hardware platform and the use-case requirements.

6.1. Single-OS Configuration

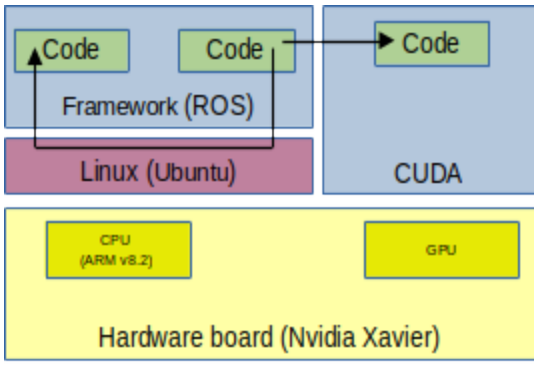

Figure 8: AMPERE platform architecture for Nvidia Xavier

One of the AMPERE platforms consists of the Nvidia Xavier board with a general-purpose Linux distribution.

Figure 8 illustrates the variation of the AMPERE platform for the Nvidia Jetson Xavier board. The board ships a CPU and a GPU.

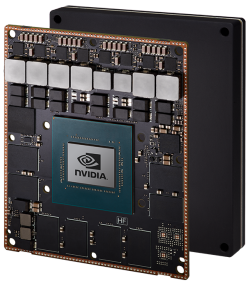

Figure 9: The Nvidia Xavier board

6.1.1. Nvidia Xavier

The Xavier board is available from Nvidia, who is not participating to the project. The specifications can be found at https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-agx-xavier/.

6.1.2. Ubuntu

Ubuntu Linux is a general purpose Linux system. The Ubuntu distribution is directly provided by Nvidia. The AMPERE ecosystem uses Ubuntu version 18.04. The partners plan to evaluate the benefits of adding the PikeOS hypervisor (described in section 6.2.2) and/or choosing a different Linux distribution (e.g. ELinOS, described in section 6.2.4).

6.2. Multi-OS Configuration

The other AMPERE platform consists of a Xilinx ZCU102 board with two operating systems. The board also provides an FPGA. The configuration and the tools are provided by deliverable D5.2 [5].

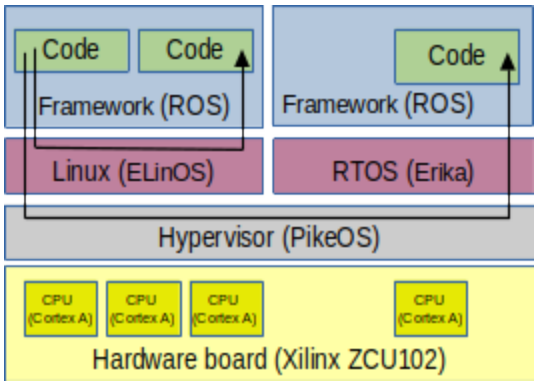

Figure 10: AMPERE platform architecture for a Xilinx board

Figure 10 illustrates the variation of the AMPERE platform for the Xilinx ZCU102 board. The board ships a multicore CPU, driven by an hypervisor.

The envisioned software architecture for the Multi-OS configuration is meant to support different levels of criticality:

- Linux (ELinOS) for the execution of general-purpose or accelerated computations;
- Erika for safety-relevant activities.

Both operating systems are managed by an underlying hypervisor: PikeOS.

6.2.1. Xilinx ZCU102

The ZCU102 board is sold by Xilinx, who is not participating to the project. Information can be found at https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

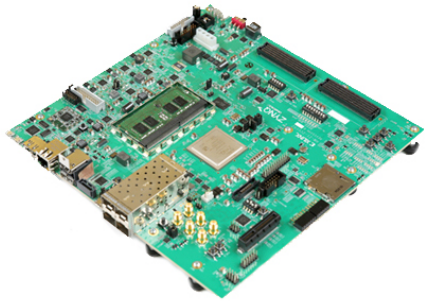

Figure 11: The Xilinx ZCU102 board

6.2.2. PikeOS

PikeOS is an hypervisor developed for safety-critical applications. It is provided by SYSGO.

PikeOS has been developed for safety-critical applications with certification needs in the fields of aerospace and defense, automotive and transportation, industrial automation and medical, network infrastructures and consumer electronics.

To support the various needs of the different markets, different run time environments and guest operating systems can be executed in parallel. In the context of project AMPERE, two guest operating systems are considered: Erika and ELinOS.

PikeOS is available on most of the modern 32 and 64 bit CPU families. Functionality to provide a safe environment for critical applications is a vital part of the PikeOS core components. Safety aspects have been accounted in the PikeOS design.

The PikeOS development environment has its own Cross Development Kit (CDK). This CDK is based on the GNU toolchain; it provides the GNU compiler (GCC) and the GNU binary utilities. In addition, PikeOS specific configuration compiler, ROM image builder, and validation tools are part of the CDK. The web page of PikOS is https://www.sysgo.com/pikeos.

In the scope of project AMPERE, SYSGO works to expand the support of the PikeOS hypervisor for heterogeneous hardware platforms and parallel programming models in order to fit the Ampere requirements. The base version of PikeOS for the AMPERE ecosystem is 5.0.

6.2.3. Erika

Erika Enterprise (http://www.erika-enterprise.com) is a real-time operating system (RTOS) designed by Evidence for the automotive domain. In particular, the RTOS is designed according to the AUTOSAR Classic Platform standard and has been recently certified ISO26262 ASIL-D. The RTOS supports several hardware architectures, including Infineon AURIX, ARM Cortex-A/-M/-R and x86-64.

In the context of project AMPERE, the RTOS has been ported on the Xilinx Cortex-A53 multi-core CPU on top of the PikeOS hypervisor. This version is currently under development by Evidence. Its access is restricted to the members of the AUTOSAR consortium

6.2.4. ELinOS

ELinOS is a Linux distribution for embedded Linux. It is provided by SYSGO. The ELinOS distribution comes with a complete set of tools for the development and configuration of ELinOS-based systems. It is designed to execute on top of the PikeOS hypervisor.

The ELinOS web page is https://www.sysgo.com/elinos. The AMPERE ecosystem relies on version 7 of ELinOS.

6.2.5. FRED

FRED is a framework, developed by Scuola Sant'Anna, to support the design, development, and execution of predictable software on FPGA system-on-chips platforms. It exploits dynamic partial reconfiguration and recurrent execution to virtualize the FPGA fabric, enabling the user to allocate a larger number of hardware accelerators than could otherwise be fit into the physical fabric. It integrates automated floorplanning and a set of runtime mechanisms to enhance predictability by scheduling hardware resources and regulating bus/memory contention. FRED can be downloaded from http://fred.santannapisa.it/. See deliverable D4.2 [4] for details.

7. Conclusions

D6.3 provides a first overview of the tools involved in the AMPERE ecosystem. The complete tool chains are not fully integrated yet: the ecosystem is being built part by part.

For system modeling, Capella and App4MC/Amalthea are integrated – even if a few adjustments will surely be needed in the synchronization bridge.

The execution performance optimization chain is being built around tools developed by BSC; it is therefore consistent. The tool chains for the other criteria are under development.

The work for the coordination between ELinOS and Erika using PikeOS for the Xilinx platform is a work in progress. The Xavier platform already ships with an Ubuntu Linux that can support ROS2; there is no need for specific integration work.

8. Acronyms and abbreviations

ASIL	Automotive Safety Integrity Level
CPU	Central Processing Unit
DDS	Data Distribution Service
GPU	Graphical Processing Unit
FPGA	Field-Programmable Gate Array
ROS	Robot Operating System
RTOS	Real-Time Operating System

9. References

[1] AMPERE, “D2.2. First release of the meta parallel programming abstraction and the single-criterion performance-aware component,” 2021.

[2] ——, “D3.2. Single criterion energy predictability and resiliency,” 2020.

[3] ——, “D1.2. Analysis of safety aspects on single-criterion optimization and 1^{st} release of test suite,” 2021.

[4] ——, “D4.2. Independent runtime energy support, predictability, segregation and resilience mechanisms,” 2021.

[5] ——, “D5.2. Single criterion operating systems and hypervisor software,” 2021.

A. Main Modeling Entities for Capella and Amalthea

Capella and App4MC/Amalthea are modeling tools that cover a wide panel of concepts. Only some of them are considered in AMPERE. This chapter describes the main Capela and Amalthea entities that are taken into account in the scope of AMPERE.

Capella is a high-level system architecture modeling tool. Based on the Arcadia design method, Capella helps elicit operational and system needs, then ensure they are realised by a logical architecture that is refined by a physical architecture. This is done through a graphical syntax that covers data flow, component structure, data and exchanges, state machines, scenarios with sequence diagrams, allocation tables, among others.

It enables the identification of data flows and functional chains within a complete system design process.

App4MC is a fine-grain system architecture modeling tool. Amalthea, the underlying modeling structure, enable the precise specification of the interactions between the system entities. Amalthea carries enough information to support the analysis of the system behavior.

In the AMPERE ecosystem, Capella and App4MC are complementary. The system modeling process in Capella ends up to a system architecture precise enough to be translated into Amalthea for further refinement. In order to integrate the two tool, it is therefore important to clearly identify how to translate a Capella model into an Amalthea model.

A.1. Main Entities in Capella

Capella defines many concepts. Only a few of them are relevant for a transformation to lower-level Amalthea architecture models. This means the patterns to be considered in AMPERE are related with a small number of Capella concepts.

All modelling patterns are parts of the Capella physical architecture. Concepts outside physical architecture are not considered.

As Capella is a graphical editor, it eases the understanding of architectures. Figure 12 illustrates an excerpt of a Capella diagram typically used for AMPERE. IT represents three Kalman filters (algorithms) in a software component that send their outputs to a threshold detector (another algorithm) in another software component.

This section lists the Capella entities to be considered, and provides semantics for each of them. This means we associate a semantics to each entity that fits our needs for architecture analysis and integration. The document considers systems that are meant to be represented using Amalthea. Therefore, we interpret the Capella entities in an Amalthea perspective. That is, we specify the Capella patterns with respect to the Amalthea semantics.

A.1.1. Components

A.1.1.1. Behavioral Physical Components

Behavioral physical components appear as blue rectangles in Capella diagrams. They represent deployment units, also known as “components”.

Behavioral physical components have ports that represent interaction points. Every interaction between the inner parts of the components and the outside shall go through a port. In flow ports and out flow ports should be used. Inout flow ports should not be used. Normal ports may be used, but will not be considered for execution analysis.

A behavioral physical component can contain other behavioral physical components. However, a behavioral physical component shall not contain other behavioral physical component and physical functions. That is, a

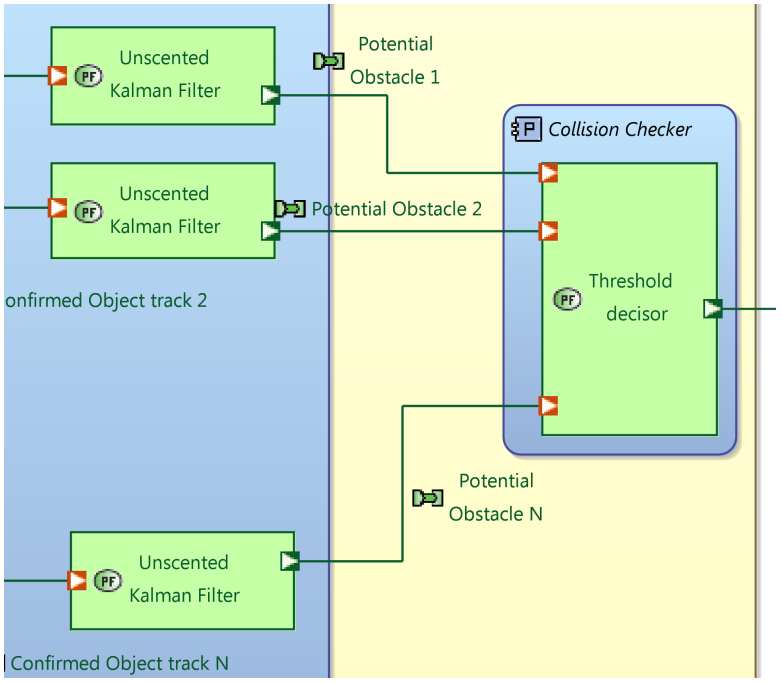

Figure 12: Excerpt of a Capella model

behavioral physical component can either be empty, or contain physical functions, or contain other behavioral physical components.

A.1.1.2. Physical Functions

Physical functions appear as green rectangles in Capella diagrams. They form the functional chains of the system architecture. They are allocated to behavioral physical components. They represent algorithms.

Physical functions have ports that represent algorithm inputs and outputs. The semantics of a physical function shall be the following: all input data are received, then the algorithmic computation is performed, then all output data are produced. This is a very restrictive semantics because Capella itself does not specify when inputs and output occur. Yet, the semantics we specify here corresponds to well-known data flow computations. It is suitable for execution analysis.

The semantics implies that, if some computation time is specified for a physical function, it corresponds to the time necessary to produce all the outputs after having received all the inputs.

Input ports and output ports shall be used to specify data inputs and outputs.

Properties may be associated with physical functions to specify the execution rate of the algorithm. For example, one may specify periodic or sporadic execution.

A.1.1.3. Node Physical Components

Node physical components appears as yellow rectangles in Capella diagrams. They represent hardware elements, on which behavioral physical components are deployed.

A node physical component does not have an exact semantics: it can represent a whole computer or a processing unit. Additional information shall be specified in order to clarify the semantics.

A node physical component can contain another node physical component.

A.1.1.4. Physical Actors

External entities that interact with the system shall be represented by physical actors. Physical actors shall have input or output ports. Thus, the interactions between the behavioral physical components of the system and the outside shall be explicit.

Output ports of physical actors should specify when they produce data. This is useful for architecture analysis. Properties should therefore be associated with output ports of physical actors to specify this information.

A.1.2. Connections

A.1.2.1. Component Exchanges

Component exchanges represent interactions between components. If they connect ports of software behavioral physical components, they represent actual communications handled by software communication libraries.

Every behavioral physical component port shall have one or more component exchanges. A port should preferably have one component exchange. Yet, if a port has several component exchanges, all ports it is connected to through a component exchange shall have only one component exchange. That is, a port shall not have several component exchanges that connect ports that themselves have several component exchanges. This is because the communication semantics is then difficult to implement with communication libraries.

A.1.2.2. Functional Exchanges

A functional exchange represents a data flow between two algorithms. Every functional exchange between two physical functions shall be associated with a component exchange. This is because every functional exchange must be handled by a communication library.

Every functional port shall have one or more functional exchanges. A functional port should preferably have one functional exchange. Yet, if a port has several functional exchanges, all ports it is connected to through a functional exchange shall have only one functional exchange. That is, a port shall not have several functional exchanges that connect ports that themselves have several functional exchanges. This is because the communication semantics is then difficult to implement with communication libraries.

A.1.2.3. Physical Links

Physical links connect node physical components through their physical ports. They are not mandatory for the current bridge.

A.1.3. Functional Chains

Functional chains may be used to gather data flows across physical functions.

A.2. Main Entities in Amalthea

Amalthea addresses a wide range of concepts. In the scope of project AMPERE, only a limited number of concepts is relevant.

An Amalthea model contains several sections, that cover different concerns:

- SWModel contains the software model, which corresponds to the algorithmics computations, synchronizations and tasks;
- HWModel contains the descriptions of the supporting hardware;
- OSModel contains the characterization of the operating systems;
- ComponentsModel contains the structure of the application, which encapsulates the entities of the SWModel;
- StimuliModel contains the activations of the tasks;
- MappingModel contains the allocations of the tasks onto the operating system.

The other sections – PropertyContraintsModel, CommentElements, MeasurementModel, ConfigModel and EventModel – are not relevant for AMPERE.

The SWModel is the main concern for AMPERE, because it contains the elements that are related with optimization. The main entities of SWModel are runnables, tasks, labels and channels.

Runnables represent algorithmic computations. They specify a number of CPU ticks that represents the corresponding the work load. Runnable represent passive pieces of software code. Tasks represent groups of operating system execution threads. They call runnables. Every task and runnable contain an activity graph that specifies the order of communications and calls.

Inter-runnable and inter-tasks communications are modeled by labels and channels. Labels represent variables that are shared between pieces of code (e.g. runnables). Channels represent middleware communications. In the context of AMPERE, channels are likely to correspond to ROS2 communications.

A.3. Transformation Rules Between Capella and Amalthea

The modeling patterns identified in previous section can be transformed into lower-level definitions, targeting Amalthea. Amalthea is a meta-model that can capture interactions between tasks, communication semantics, memory footprint, CPU load and execution time. It is suitable for architecture analysis. However, it is not convenient for software integration.

The transformation of a Capella model into an Amalthea model translates most of the information of the patterns identified in section A.1. While Capella supports the upstream phases of System design, Amalthea allows refining the physical architecture based on lower-level concerns. In a complete engineering workflow, Capella and Amalthea models are thus complementary, but they need to be aligned. To that aim, the bridge supports the initial generation of a partial Amalthea model reflecting the Capella model, and the later realignment of both models based on the detection of discrepancies and the propagation of changes.

The transformation shall create an Amalthea model with a SWModel, an HWModel, an OSMode, a ComponentsModel, a StimuliModel, and a MappingModel. No information for any PropertyContraintsModel, CommentElements, MeasurementModel, ConfigModel or EventModel is created from a Capella model for the moment.

When the bridge is executed, a trace file is generated or updated along with the Amalthea model. Together with dedicated custom properties in the Amalthea model, it helps ensure every element can be properly traced and updated even if it has been renamed or moved. If an Amalthea model already exists, a diff/merge based on the traces is executed between that model and a temporary Amalthea model that is issued from the Capella-Amalthea mapping. Discrepancies between the Capella and Amalthea models can thus be identified and dealt with. In particular, merging changes means propagating architectural changes from Capella to Amalthea or dismissing mis-alignements in Amalthea. The scope of the diff/merge solely focuses on model elements that are concerned with the Capella-Amalthea mapping, which excludes everything that relates to low-level Amalthea concepts for detailed physical concerns. Amalthea users are thus free to carry out this part of the Amalthea modelling activity without any risk of interference, while the bridge will catch every change they make that has an impact on the architecture at Capella level. This is how the Capella-Amalthea bridge plays an essential role

in enabling engineers to implement a workflow where the strengths and complementary aspects of Capella and Amalthea are combined.

A.3.1. Behavioral Physical Components

Every Capella behavioral physical component is transformed into an Amalthea ComponentStructure. Every port is transformed into an Amalthea ComponentPort.

A.3.2. Physical Functions

Every Capella physical function is transformed into an Amalthea runnable. Transformation rules for ports are specified in section A.3.5.

A.3.3. Node Physical Components

Every Capella node physical component is transformed into an Amalthea HwStructure.

A.3.4. Physical Actors

Capella physical actors are optionally translated into the Amalthea model, even though they are by definition external to the system under study.

A.3.5. Functional Exchanges

Capella functional exchanges have different possible equivalence in Amalthea. If a functional exchange connects two physical functions that are transformed into runnables belonging to the same task, then it is transformed into an Amalthea Label. The corresponding ports are transformed into LabelAccess, either read or write. If, on the contrary, a functional exchange connects two physical functions that are transformed into runnables belonging to different tasks, then it is transformed into an Amalthea channel. The corresponding ports are transformed into ChannelReceive and ChannelSend.

A.3.6. Component Exchanges

Every Capella component exchange is transformed into an Amalthea Connector.

A.3.7. Alternatives for the Creation of Amalthea Tasks and Amalthea Mappings

No concept in Capella naturally maps to the concept of Task in Amalthea. Several strategies can be considered. It was thus decided that the bridge would be flexible enough to support several alternatives in order to allow for experiments in the use cases. Which of the alternatives is chosen depends upon a parameter of the bridge specified in the Capella model.

The different strategies for the creation of Amalthea Task are the following:

- consider behavior physical components;
- consider functional chains;

- consider scenarios (Sequence Diagrams), capabilities or composite behavior physical components.

The first strategy consists of considering behavior physical components. Whenever a behavior physical component has allocated physical functions, it is transformed into a task that calls the runnables corresponding to the functions. This mapping is the one illustrated in the demonstration video.

An alternative strategy consists of considering functional chains. Every functional chain is transformed into a task. The runnables called by the task correspond to the functions involved by the functional chain. Similarly, the Amalthea mapping produced by the bridge (mapping of tasks or runnables to a scheduler, mapping of labels to a memory) can be activated and deactivated at fine grain using parameters of the bridge.

Other alternatives, such as scenarios, capabilities or composite behavioral physical components have also been considered but not implemented.

B. Summary of tools and versions

This section recapitulates all the tools listed in the document.

tool	version
Capella	5.0.0
App4MC/AMALTHEA	1.0
Bridge from Capella to Amalthea	developed for AMPERE – no version number
AMALTHEA/FPGA Flow	0.1-alpha

Table 1: Versions of tools for system modeling

tool	version
SLG	not released yet – under development
LLVM	11.1 or newer
GCC	7 or newer
Extrae	3.4.1
PARTProfiler	0.1-alpha
DART	0.1-alpha
DART IPs	0.1-alpha
Offline energy optimizer	developed for AMPERE – no version number

Table 2: Versions of tools for system optimization

tool	version
ROS2	foxy
FRED	?
ELinOS	7
Erika	AMPERE specific
PikeOS	5.0
Ubuntu	18.04

Table 3: Versions of platform elements