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

D6.4 First release of the AMPERE ecosystem

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	DEM
Authors	Thomas Vergnaud (TRT), Olivier Constant (TRT)
Contributors	Tiago Carvalho (ISEP) Ida Maria Savino (EVI)
Reviewer	Arne Hamann (BOS)
Keywords	Ecosystem, Integration, Workflow

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 creation using Microsoft Word
V0.2	Switch to Latex
V0.3	Added details in all sections
V0.4	Took Bosch coments into account
V1.0	Final release of the document

Table of Contents

1 Executive Summary

Deliverable D6.4 is a demonstrator that is the continuation of D6.3 [1]. D6.4 consists of the actual AMPERE ecosystem and the associated continuous integration chain hosted in BSC and TRT.

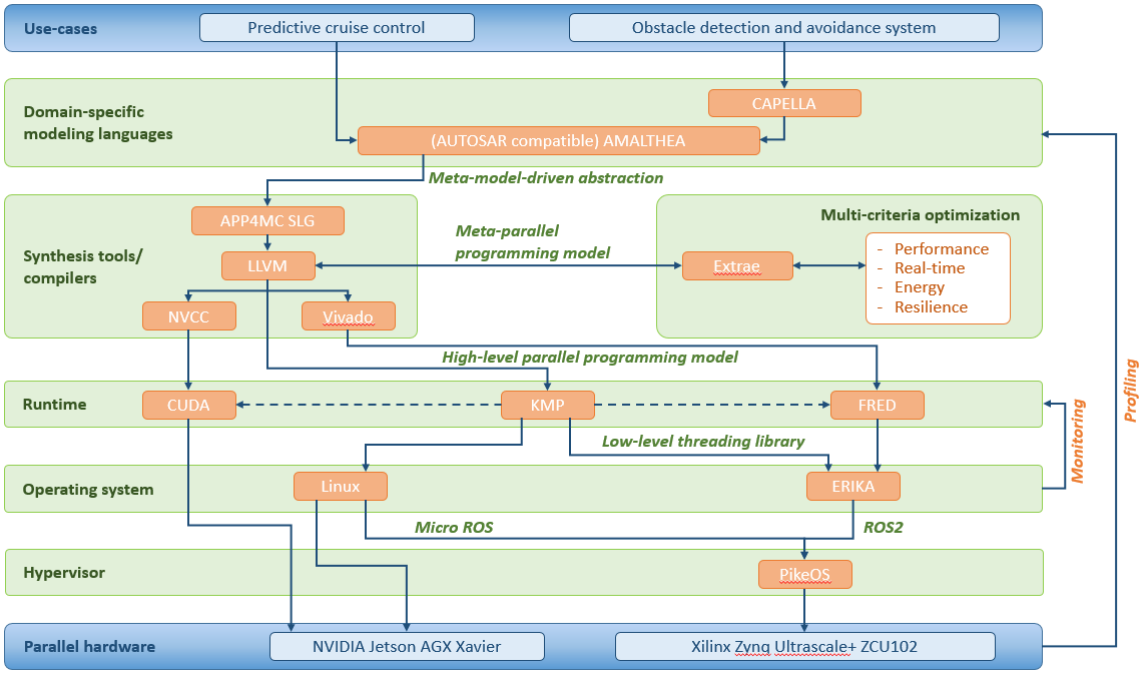
This document focuses on the description of the general workflow and the continuous integration chain of the AMPERE ecosystem, based on Gitlab and Docker.

The details of the tool developments are described in other deliverables for WP2, WP3, WP4 and WP5. Hence the document provides overall information and refers to the deliverables of the other work packages for details.

2 Introduction

The AMPERE ecosystem consists of a collection of system design tools, system analysis tools and execution platforms (including operating systems and runtime mechanisms). The tools support the AMPERE workflow while the execution platforms are the targets for the execution of the systems designed using the AMPERE workflow. Figure 1 illustrates how the ecosystem is structured.

Figure 1: AMPERE workflow



In order to assist the users of the AMPERE workflow in their design and analysis work, we aim at gathering the AMPERE tools in a central repository and perform automatic checks on the system under design whenever possible. To do so, we set up a continuous integration chain based on Gitlab[1] and Docker [2]. This chain automates the execution of generation and analysis tools to enable systematic verifications of the models the users provide.

The document is structured as follows. Section 3 describes the tools of the AMPERE workflow. This section makes references to other deliverables, where the tools are described in details. Section 4 explains how the source codes for the use cases and the tools are stored to make them available to all partners. Section 5 explains how we automate the integration chain and apply it to the use cases.

[1] https://gitlab.bsc.es/

3 Overview of the AMPERE workflow

The AMPERE ecosystem relies on three main aspects, which are addressed by different work packages: system design, system analysis and execution platform. They are arranged to support the AMPERE workflow represented on figure 2.

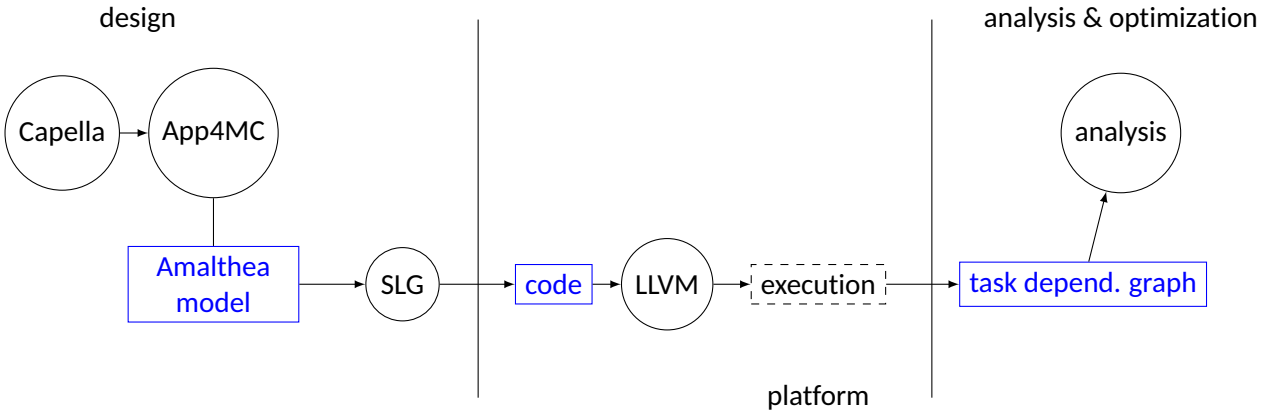
System design consists of system modeling and code generation. The Amalthea model managed by App4MC is at the core of the system modeling process: Amalthea models may either be directly created using App4MC (in the PCC use case) or generated from a Capella model (in the ODAS use case). In any case, synthetic code is generated from the Amalthea model.

The system source code is compiled with LLVM with Extrae libraries.

System analysis consists of executing the system software, extracting performance measurements and applying analysis techniques. The task dependency graph (TDG) is at the core of the system analysis process: this structure is produced by Extrae by monitoring the system execution and then processed by the analysis and optimization tools.

The execution platform supports the execution of the system software.

Figure 2: Key structures of the AMPERE workflow



All AMPERE tools have their own development lifecycles, but they all collaborate through three reference data structures: the Amalthea model, the source code and the task dependency graph (TDG). That is, all the tools involved in the system design phase work with the same version of the Amalthea model definition; all the analysis and optimization tools work with the same version of the TDG. The Amalthea model definition is driven by BOS while the TDG definition is driven by BSC. The syntax of the source code is conformant to ISO C or ISO C++ standards and therefore does not change.

3.1 System design and generation tools

The system design involves Capella and App4MC. These two modeling tools are not developed by AMPERE – they have their own development cycles and are used by the project. Most of the work (in particular, modeling) is addressed by WP2; WP6 provides the bridge between Capella and App4MC. WP1 provides the synthetic load generator (SLG).

3.1.1 APP4MC

APP4MC is an open source platform for engineering embedded multi- and many-core software systems based on the Amalthea modeling language. APP4MC is a project of the Eclipse Foundation under the Eclipse Public

License (EPL) v2.0. More information is available here: https://www.eclipse.org/app4mc/.

In AMPERE, APP4MC is used to model the detailed software and hardware structures as well as the behavior of the systems.

This release of the AMPERE ecosystem contains APP4MC v1.2.0. It is available on the links below.

- Windows 64 bits: https://archive.eclipse.org/app4mc/products/releases/1.2.0/org.eclipse.app4mc.platform-1.2.0-20210730-134656-win32.win32.x86_64.zip
- Linux 64 bits: https://archive.eclipse.org/app4mc/products/releases/1.2.0/org.eclipse.app4mc.platform-1.2.0-20210730-134656-linux.gtk.x86_64.tar.gz

Installation simply consists in extracting the archive. The tool can be started by running executable `app4mc.exe` (Windows) or `app4mc` (Linux).

3.1.2 Capella

Capella is an open source system architecture modeling tool. It provides a method and a graphical syntax to elicit needs and define a solution architecture. Capella is a project of the Eclipse Foundation under the Eclipse Public License (EPL) v2.0. More information is available here: https://www.eclipse.org/capella/.

In AMPERE, Capella is used to model the functional aspects and high-level architecture of a system.

This release of the AMPERE ecosystem contains Capella v5.2.0. It is available on the links below.

- Windows 64 bits: https://www.eclipse.org/downloads/download.php?file=/capella/core/products/releases/5.2.0-R20211130-125709/capella-5.2.0.202111301257-win32-win32-x86_64.zip
- Linux 64 bits: https://www.eclipse.org/downloads/download.php?file=/capella/core/products/releases/5.2.0-R20211130-125709/capella-5.2.0.202111301257-linux-gtk-x86_64.tar.gz

Installation simply consists in extracting the archive. The tool can be started by running executable `capella/capella.exe` (Windows) or `capella/capella` (Linux).

3.1.3 Capella-Amalthea bridge

The Capella-Amalthea Bridge is an AMPERE tool that supports the integration of Capella and Amalthea modeling work in an iterative workflow. It is developed in the context of WP6 to provide the required level of tool integration for the use cases.

The need comes from concurrent engineering concerns. Although Capella and Amalthea target complementary perspectives and levels of abstraction in system design as represented in figure 3, these different levels of abstraction must remain consistent with each other. Concretely, the two modeling languages have a non-empty conceptual intersection illustrated in figure 4. Therefore, when a Capella model and an Amalthea model coexist, their respective subsets on this conceptual intersection must be consistent. It must be possible to check this consistency by detecting discrepancies and apply re-alignment changes as soon as the two models evolve.

This is what the Bridge does. Technically, it is an interactive model synchronizer that relies on about 30 semantic equivalence rules between certain Capella and Amalthea modeling concepts, and a synchronization policy based on a diff-merge and automatically-managed traceability links. Additional information on its usage can be found in D1.5 [3] (§4.2).

This release of the AMPERE ecosystem contains version ‘MS3’ of the Bridge.

Figure 3: Bridge positioning - Engineering workflow

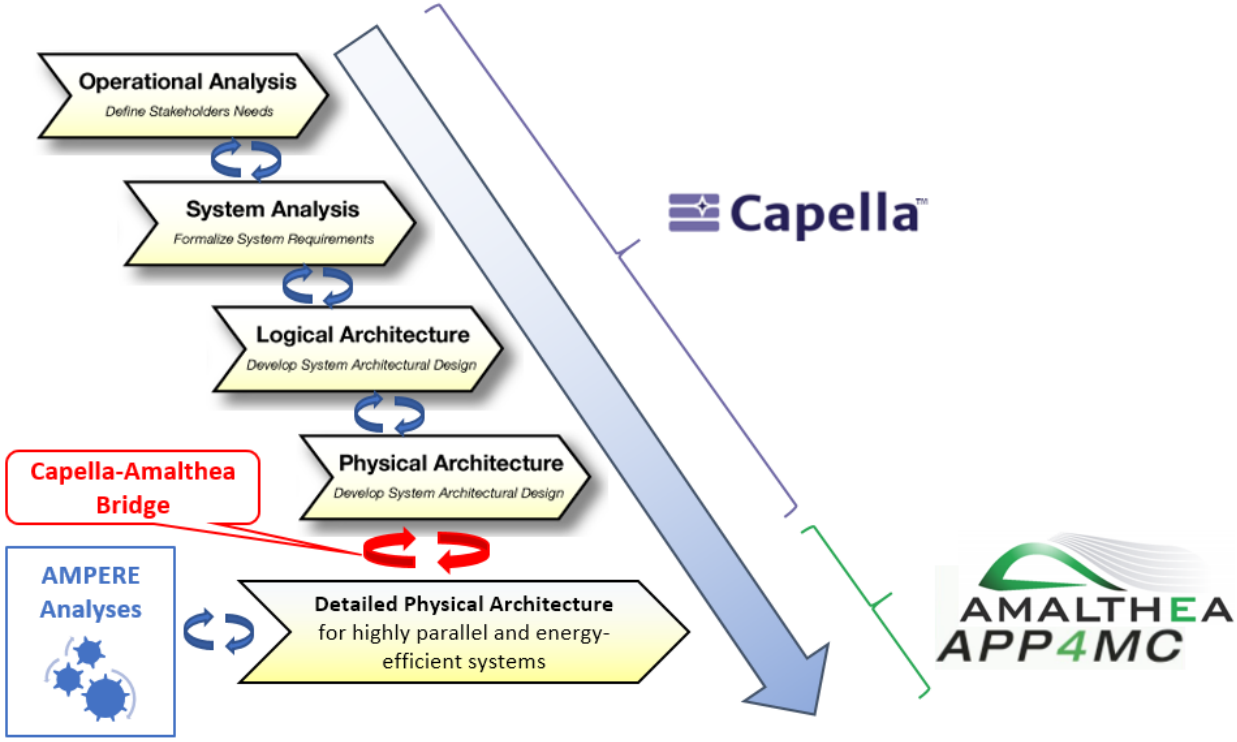
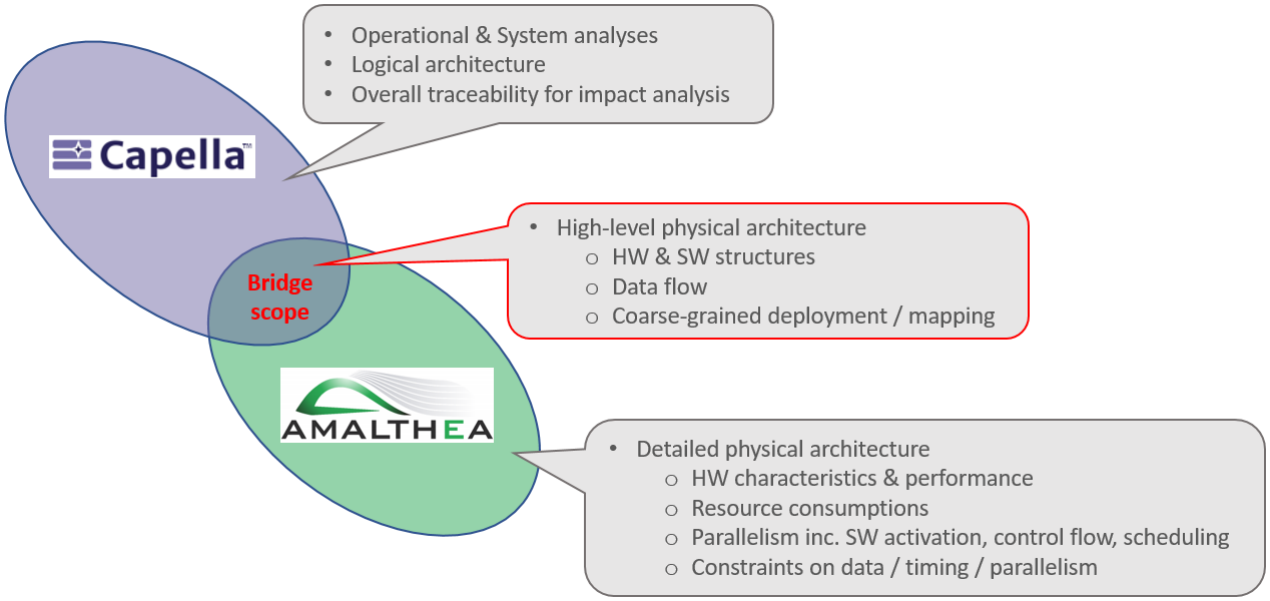


Figure 4: Bridge positioning - Modeling languages



- The source code is located here: https://gitlab.bsc.es/ampere-sw/wp6/capella-bridge
- Version 'MS3' is available here: https://gitlab.bsc.es/ampere-sw/wp6/capella-bridge/-/tags
- The corresponding built artifact for Linux and Windows is available here: https://gitlab.bsc.es/ampere-sw/wp6/capella-bridge/-/jobs/457443/artifacts/download

The installation and usage instructions are available here: https://gitlab.bsc.es/ampere-sw/wp6/capella-bridge/-/blob/main/README.md

Even though the Bridge is interactive, it can also be run as a command line in automatic mode. This capability is essential to integrate it into a continuous integration workflow such as the one described in section 5.

3.1.4 Synthetic load generator (SLG)

The synthetic load generator (SLG) produces code from an Amalthea model. The generated code simulates computation load, communication load and memory accesses according to what the Amalthea model specifies, without the need to model and execute real algorithms.

In addition to the main contribution of the tool as an element of the AMPERE ecosystem, the SLG is subject to a contribution in the scope of D2.3 [4].

The SLG is central in the AMPERE ecosystem since it initiates the transition from the modelling realm to the analysis realm, the latter relying on the execution of code.

- The binary and documentation of the SLG can be found here: https://gitlab.bsc.es/ampere/ampere-project/-/tree/master/Software/SLG.
- The commit that contributes the concerned version of the SLG is visible here: https://gitlab.bsc.es/ampere/ampere-project/-/commit/e50a798211201016008811e4e10043ffb8beb2fc

3.2 System analysis tools

System analysis tools are divided in two essential criteria: energy optimization and time predictability. The analysis tools for energy optimization are described in deliverable D4.3 [5].

The time predictability is divided in three tools: time analysis, mapping simulation and mapping exploration. All the time predictability tools work essentially at the TDG level. These tools are described in deliverable D3.3 [6] and the source code is located in the ampere main repository, more specifically: https://gitlab.bsc.es/ampere-sw/wp3/time-predictability. Each tool can be used via two methods: by command line or as a python module. The repository contains instructions to install, compile and run the tools as necessary, via the "README" file: https://gitlab.bsc.es/ampere-sw/WP3/time-predictability/-/blob/main/README.md

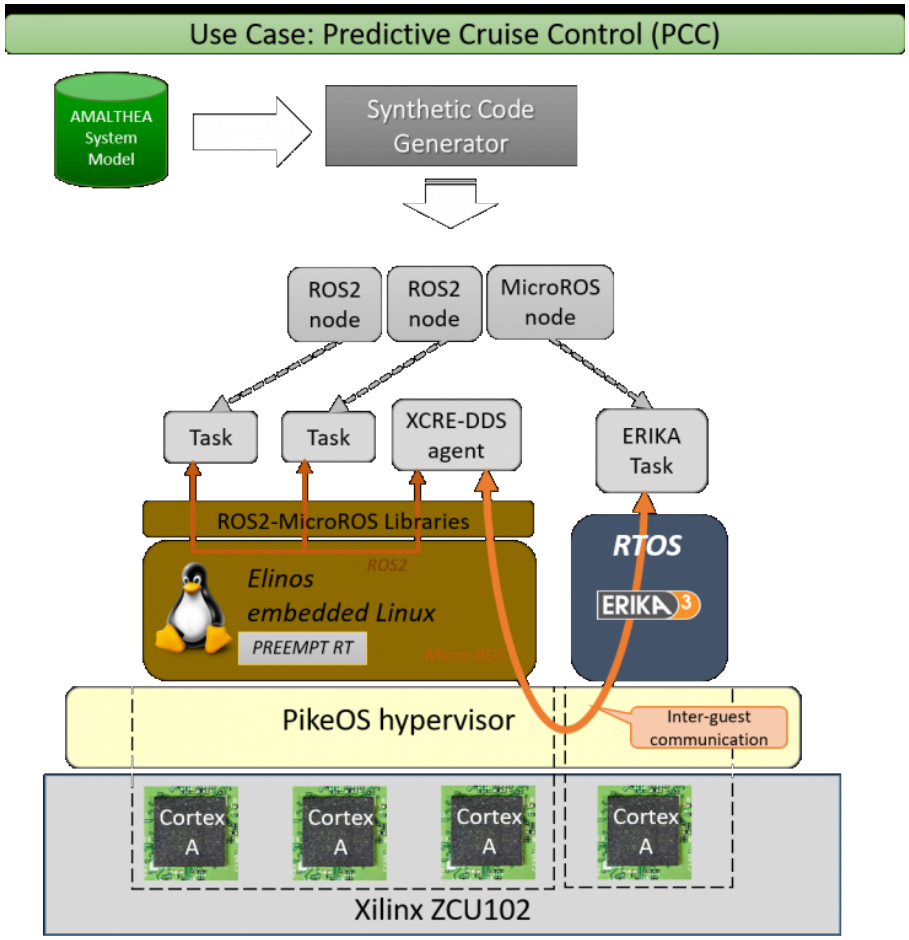
3.3 Execution platforms

The execution run-time platform, defined by WP5, consists of a multi-OS architecture built on top of the PikeOS hypervisor provided by SYSGO. The hypervisor supports guests OSes such as ElinOS, the Linux guest OS specifically developed by SYSGO to run on PikeOS, and ERIKA Enterprise RTOS, a real-time kernel based on the AUTOSAR Classic automotive standard developed by EVI.

The integration work of the AMPERE run-time platform is detailed in deliverable D5.3 [7]. Figure 5 illustrates this.

With reference to the PCC use case made by BOSCH, EVI is going to deploy the synthetic code generated by the SLG provided by BOSCH in the run-time platform on the Xilinx ZCU102 board. As shown in previous figure, the AMALTHEA SLG generates synthetic code that can be deployed as application tasks. Then, the ELinOS guest is going to execute ROS2 node applications generated by the AMALTHEA SLG, whereas the Erika RTOS guest is going to execute the generated Micro-ROS node application that can publish/subscribe on PCC topics. In order to guarantee the communication between Micro-ROS client application on Erika RTOS and ROS2 client applications on ElinOS, the run-time platform includes a XCRE-DDS agent, that relies on the inter-guest communication mechanisms provided by PikeOS.

Figure 5: PikeOS, Linux and Erika interacting through ROS2



Deliverable D5.3 [7] also details how to execute the ROS2/microROS applications on different guests in the AMPERE run-time platform.

4 Data repository infrastructure

In order to help users design their systems, the AMPERE workflow relies on Gitlab to store the developments. The AMPERE Gitlab contains both the software developments for the tools and the use case data (models, code, etc.). It is hosted by BSC[1].

4.1 Tool developments

Within the AMPERE Gitlab, a project is dedicated to the tool development. Thus we have a reference location for the developments of the tools involved in the AMPERE ecosystem. This enables use case providers (THALIT and BOS) to work with well-identified versions of the tools. The project is named “ampere-sw”. It contains subprojects, one for each work package. Each work package subproject contains Git repositories dedicated to a given tool.

The Gitlab only contains the developments directly related with AMPERE. External tools, such as Capella or App4MC are not included because their development cycles are independent from AMPERE.

The AMPERE Gitlab currently hosts reference versions of the SLG (WP1), the Capella-Amalthea bridge (WP6), Extrae, the TDG instrumentation and a modified version of LLVM (WP2). The analysis tools of WP3 and WP4 are currently being integrated in the AMPERE Gitlab. The WP5 developments are mainly related with operating systems (PikeOS and Erika) and external libraries (ROS2, MicroROS); they are considered as external tools.

Figure 6 is a screenshot of the AMPERE Gitlab project for the tool developments.

4.2 Use case developments

Another Gitlab project stores the use case developments; it is named “ampere”. It contains a Git repository for each use case: one for ODAS use case made by THALIT, one for the PCC use case made by BOS. It also contains a Git repository dedicated to the storage of the deliverables, papers, etc.

Figure 7 is a screenshot of the AMPERE Gitlab project for the use case development.

[1] https://gitlab.bsc.es

Figure 6: Gitlab project for the AMPERE tools

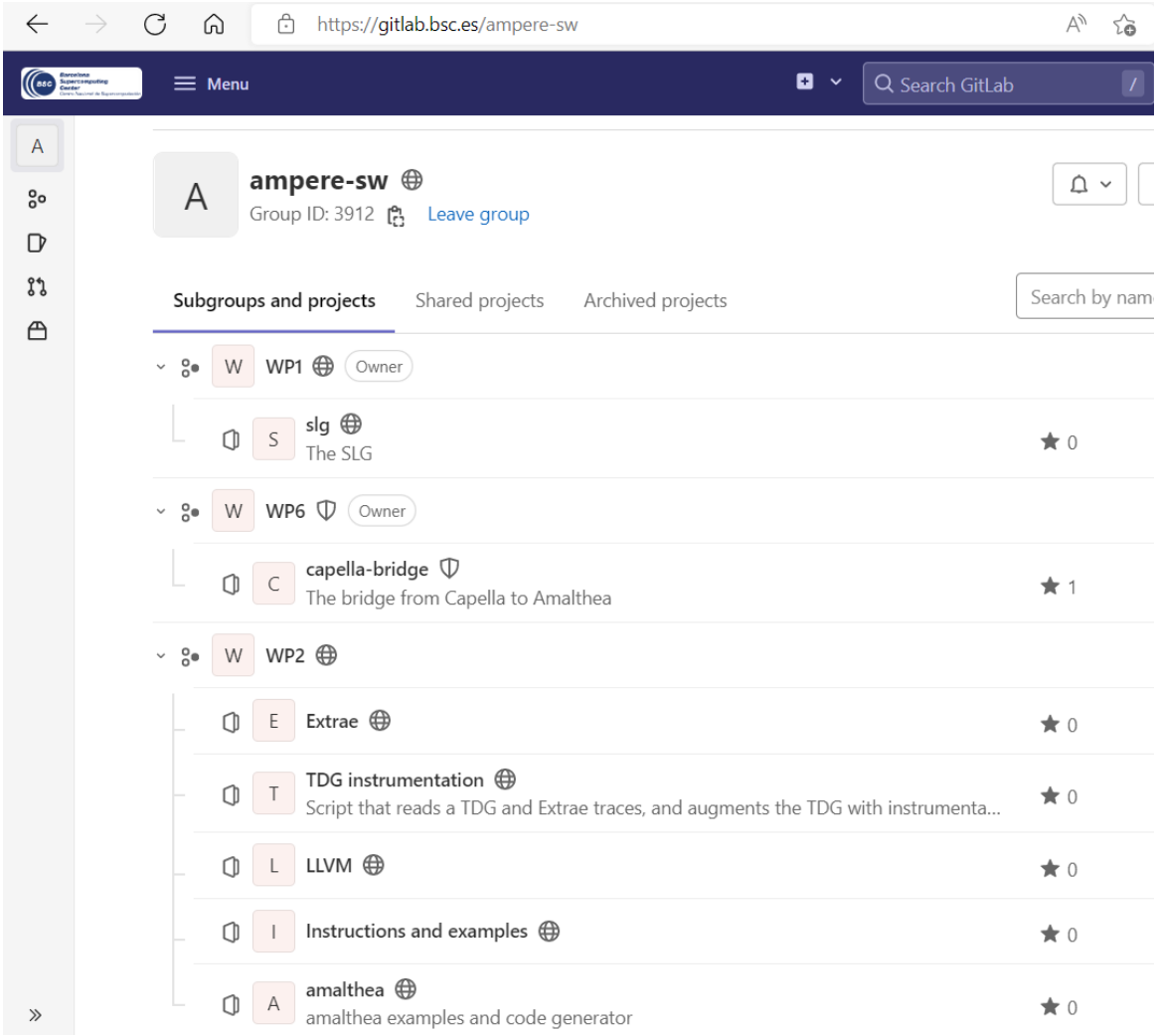
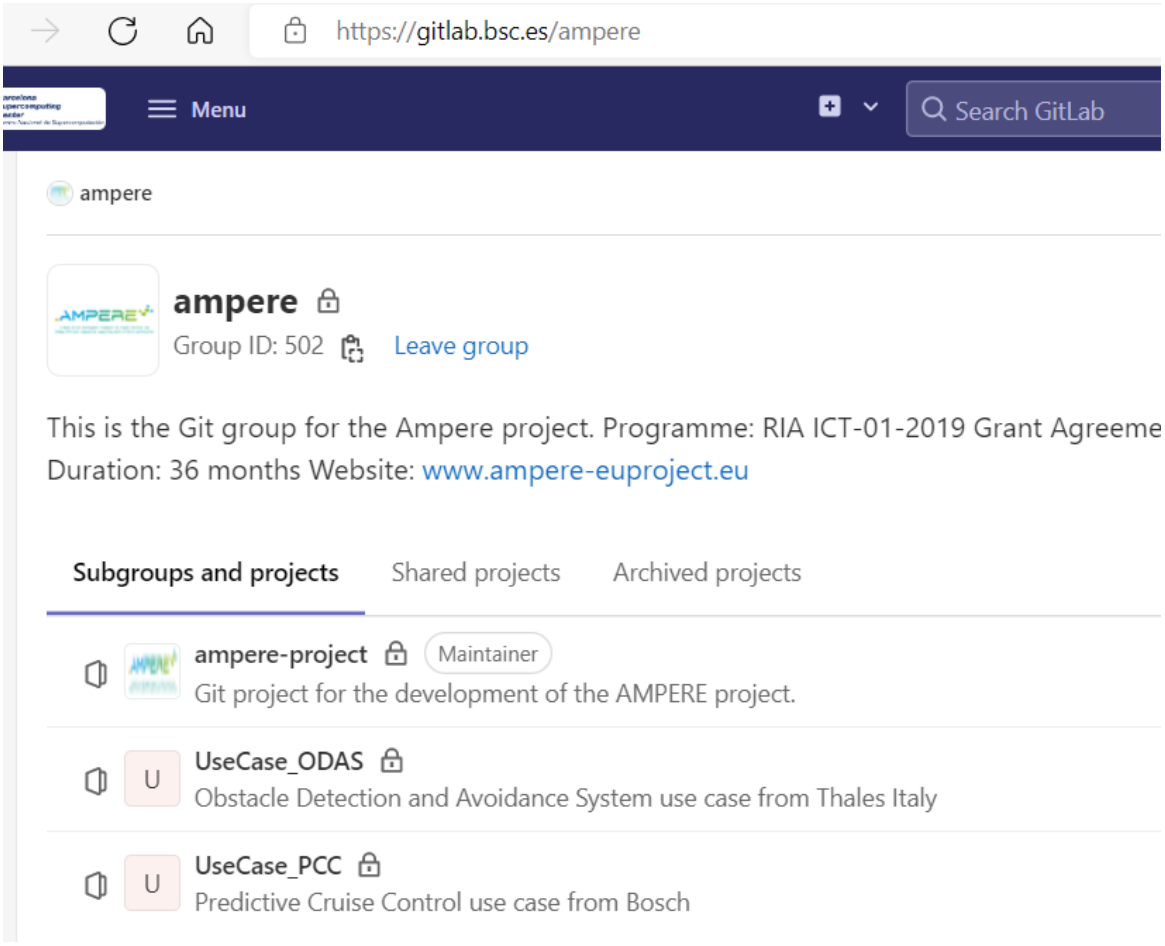


Figure 7: Gitlab project for the use cases



5 Continuous integration chain

In order to help use case and tool developers and provide guarantees on the integration of the AMPERE ecosystem, we set up a continuous integration (CI) chain based on the Gitlab hosted by BSC. Elements of the AMPERE ecosystem have different levels of maturity and different development processes and lifecycles, so we had to adopt flexible principles and different integration strategies. At this time, only a subset of the AMPERE ecosystem is covered by the continuous integration workflow but all principles and main technological assets are in place.

This first subset was driven by the core needs of the two use cases.

5.1 Technological background

Gitlab, through Gitlab CI, provides the capability to trigger actions whenever new data is pushed into a given Git repository. The actions to perform are described in a file named ".gitlab-ci.yml", stored at the root of each Git repository. These actions typically consist in reading the files stored in the Git repository and performing validation steps on them. A workload executed by a CI execution is called a *job*. A job is executed either on the same infrastructure as the Gitlab instance or on another infrastructure, in which case communications between the two infrastructures can occur on the internet using HTTPS.

A program is in charge of checking whether a job should be executed and to trigger it. Its name is gitlab-runner. It executes on the infrastructure where the job is executed. Consequently, gitlab-runner knows how to connect to the Gitlab instance while the opposite is not true and not needed. This is important for security reasons since it makes it easy to have job executions happening behind a corporate proxy.

Note that CI actions are not supposed to write data in the Git repository; they only read files. Consequently, the continuous integration workflow never interferes with the work of tool or use case providers.

5.2 AMPERE CI infrastructure

Given the context of the AMPERE ecosystem, we have set up a CI infrastructure as described in Figure 8.

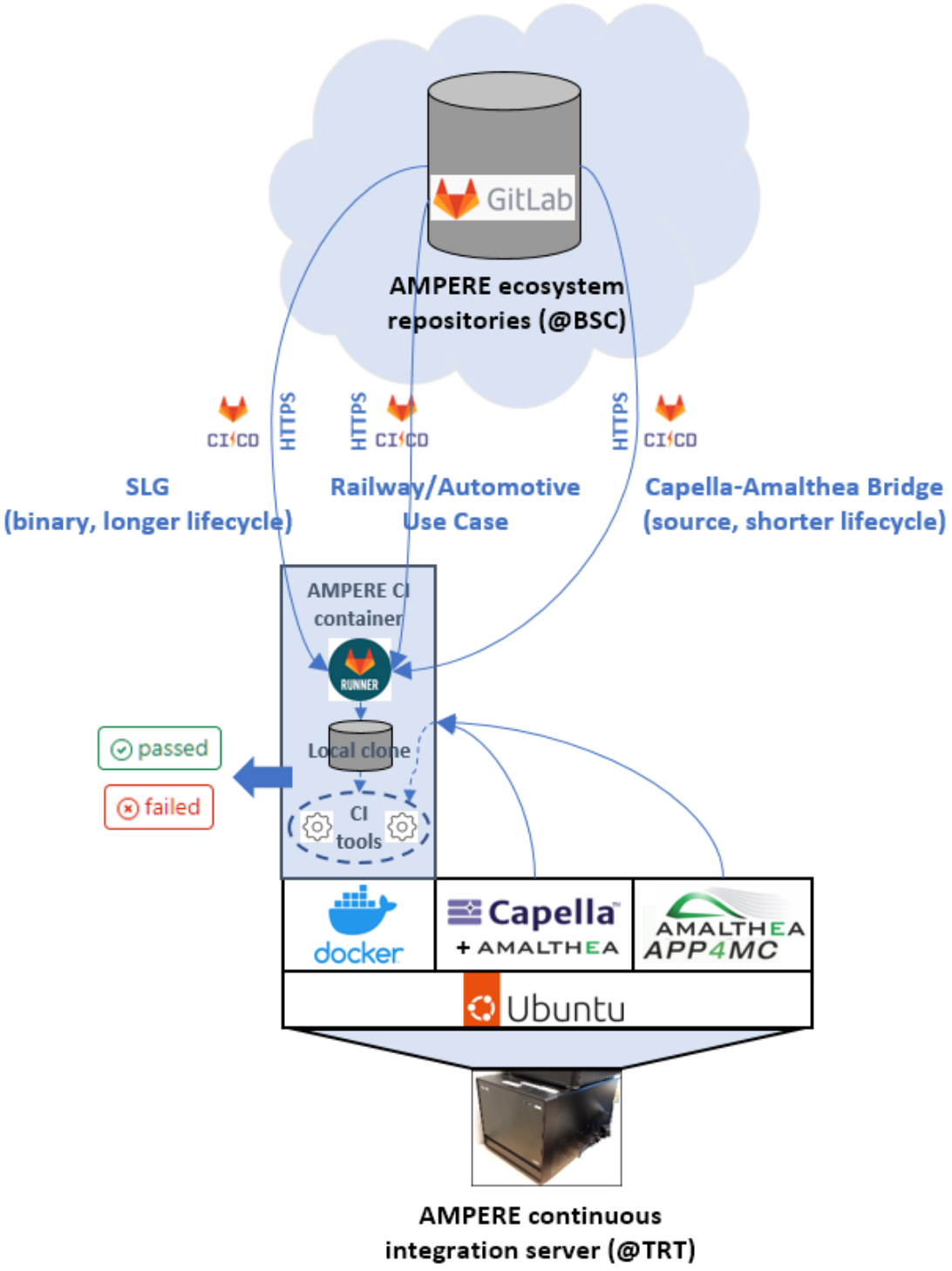
As mentioned in section 4, the data repositories of AMPERE, which contain tool development and use case artifacts, are hosted by BSC. The CI environment and supporting integration server are hosted by TRT, which ensures easy and reliable access for extension and maintenance operations by TRT staff. The operating system is classically Ubuntu 20.0.4 LTS (Long-Time Support).

5.2.1 AMPERE CI container

For security and reproducibility reasons, it is important that all CI executions happen within a container. We rely on Docker because Gitlab provides good support for it. Container technologies in general, and Docker in particular, are part of de facto standard practices in the field of Continuous Integration. A container contains all software that is needed for a specific purpose and executes on the host operating system. It also has an isolated filesystem except for the parts of the host filesystem it is explicitly given access to, if any. Similarly, communications from the container to the outside world are tightly managed. These principles are fundamental to protect the host machine and local network against undesirable, unexpected or malicious effects of CI execution.

We have chosen to use a Docker container that is "stateful" in the sense that it is not restarted each time a CI execution occurs. Thus, changes in its filesystem done by a CI execution still hold for the next CI execution. Although it opens the door to non-reproducibility issues in theory, it also simplifies certain aspects of the CI

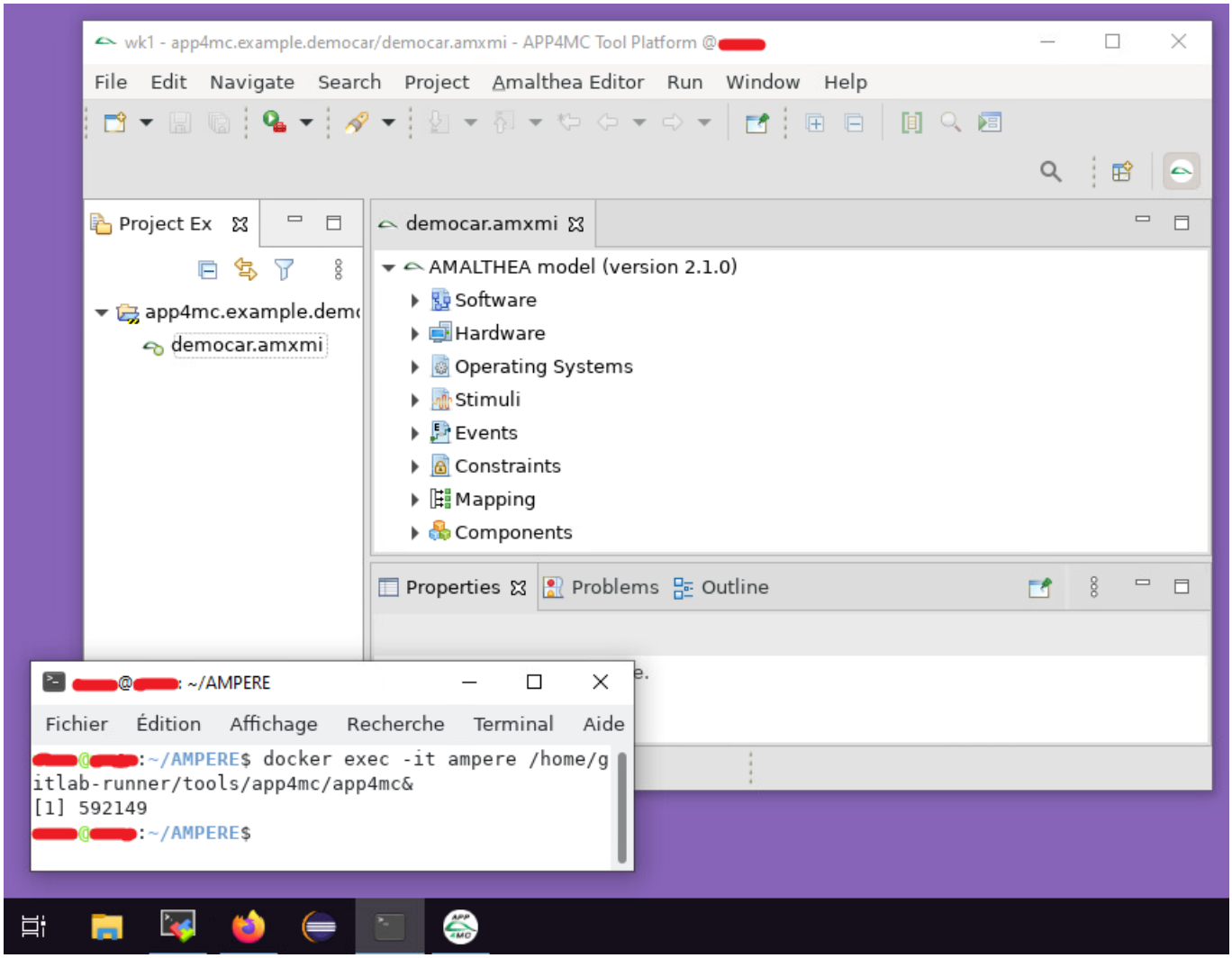
Figure 8: AMPERE CI infrastructure



and allows for better performance. In the context of AMPERE, we considered that this was the better choice in the tradeoff. Also, for simplicity reasons we opted for a single container, and thus a unique container image, covering all CI needs of the different AMPERE repositories, be them for tools or use cases.

A last design choice was to add in the container all software libraries that are needed to interactively run graphical tools. This is not required to execute CI workflows, which must be fully automatic and thus require no GUI. However, it significantly eases the debugging and the maintenance of the container when graphical tools such are APP4MC or Capella are involved (see illustration in figure 9).

Figure 9: APP4MC executed interactively and graphically from within the CI container



5.2.2 Tools in the container

Given the choices explained above, the single AMPERE CI container has to contain all the tools needed for the CI of all AMPERE tools and use cases. Tools in the AMPERE ecosystem in the large, i.e., all tools that are needed for the overall AMPERE CI, can be divided into different categories according to the constraints they bring and the way they can be handled to set up the CI process.

We can put aside common third-party tools and libraries, for example for C compilation or Latex document production: they do not bring any particular challenge since they are subject to standard integration mechanisms (Linux packages available on trusted web sites). We can also put aside AMPERE tools that are not integrated at this time, such as analysis tools: whenever they are integrated they will, together with their software dependencies, hopefully fall into one of the categories described hereafter.

The remaining categories are the ones that require explicit handling. At this time each of them only covers one tool but this may evolve when more AMPERE tools are integrated.

1. Third-party tools that can be found on the internet and can be installed as-is. The one tool in this category is APP4MC, the application for Amalthea modeling.
2. Third-party tools that can be found on the internet and require a custom configuration after installation for AMPERE needs. The one tool in this category is Capella. In order to ensure interoperability and concurrent design with Amalthea through the Capella-Amalthea Bridge, certain Amalthea artifacts that can

be found in APP4MC must be deployed in Capella together with their dependencies. This configuration step can be automated in principle, at a cost. Given that it must be redone only when the version of Capella or APP4MC changes, which only happened a couple of times since the beginning of AMPERE, we have chosen to keep this step manual.

3. AMPERE tools whose development is loosely coupled with the AMPERE collaboration infrastructure because they have their own one. The one tool in this category is the SLG, which is part of an open source project at the Eclipse Foundation, relying on the infrastructure of that foundation. Given this context, as agreed with the main contributor of the SLG (BOS), BOS delivers stable versions of the SLG that have been validated on AMPERE needs. These versions are made available as binaries on a dedicated AMPERE Gitlab repository.
4. AMPERE tools whose development is tightly bound to the AMPERE collaboration infrastructure. The one tool in this category is the Capella-Amalthea Bridge: it is not built outside the AMPERE collaboration infrastructure. Besides, it is a front-end tool with a non-trivial GUI: its development typically requires a fast experiment and feedback loop with its users, i.e., people working on the AMPERE use cases. Deploying even its unstable versions is thus relevant.

Given these different cases and associated constraints, we have opted for the following approach.

- Tools of categories 1 and 2 (Capella and APP4MC) are installed and configured by hand on the local filesystem, then automatically copied into the container every time it is restarted. This simple principle makes it possible to perform any kind of manual configuration or customisation of the tools, which is especially useful to customise Capella for the needs of the Bridge.
- Tools of category 3 (SLG) are installed in the container through a dedicated job every time their Gitlab repository receives a change. In other words, every time a new version of the tool is delivered, it automatically replaces the older version in the container. Consequently, whenever the repository of a use case receives a change the CI of the use case will use the new version of the tool. At this time there is no immediate check that the introduction of the new version does not bring regressions w.r.t. the use cases, but this feature could be added later.
- Tools of category 4 (Bridge) are handled similarly, except that the job is also in charge of compiling and packaging them before they are installed or deployed. As an illustration, figure 10 shows the build of the Bridge and its deployment into the custom Capella installation whithin the container.
- Additionally, every time the container is restarted, tools of category 3 and 4 are automatically downloaded in their latest version and imported into the container. Thus, the container is immediately ready to handle all AMPERE CI needs.

The Docker container configuration files are stored in a specific Gitlab project located at `https://gitlab.bsc.es/ampere-sw/wp6/integration-chain`, so that every AMPERE partner can access the configuration and install an AMPERE environment within its premises.

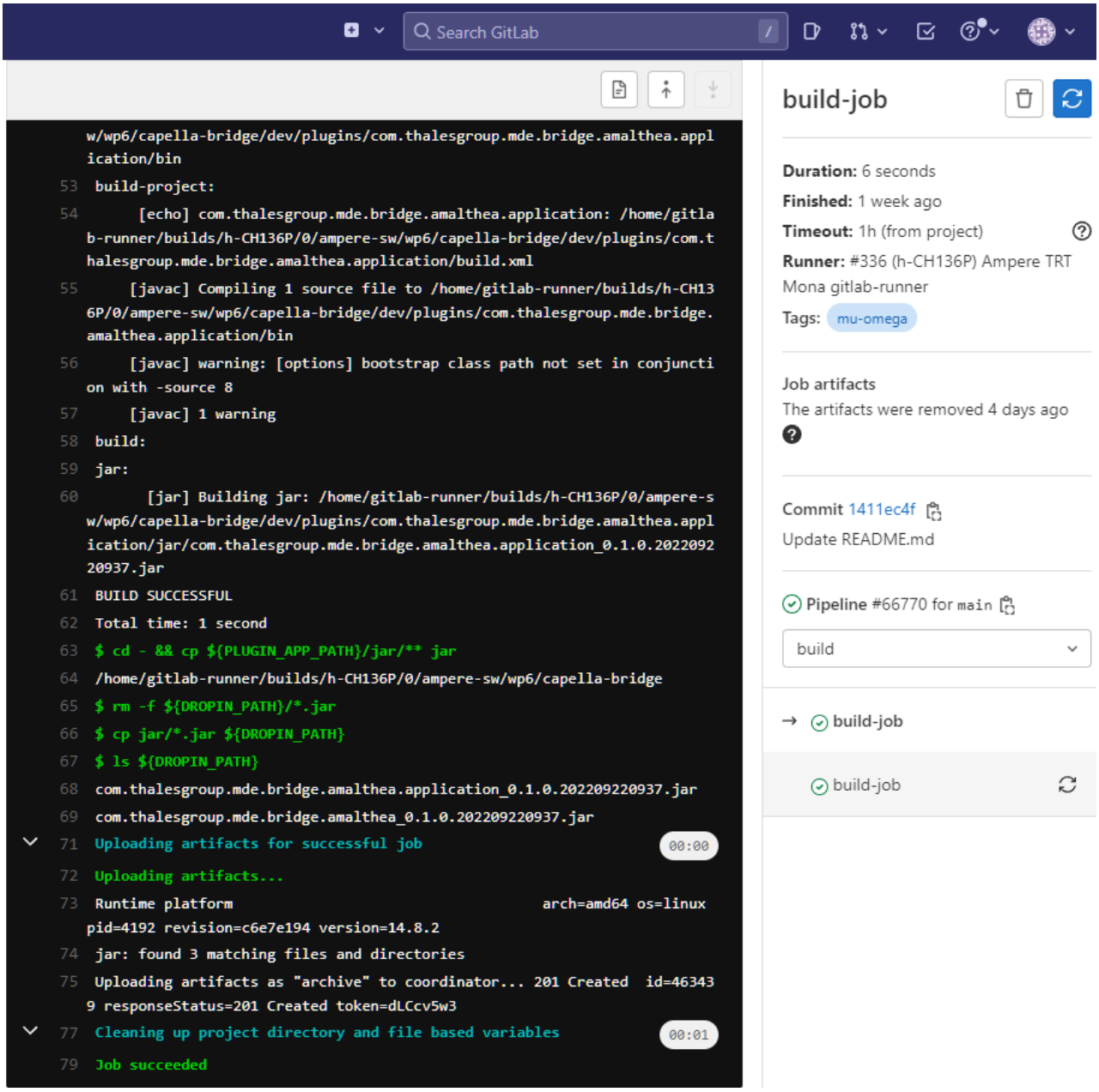
5.2.3 Use case handling by the container

Whenever a change is made on a use case repository, the corresponding artifacts are imported in the container through a local clone of the repository. The mechanism is the same for tool repositories. The container is assumed to contain all tools that are necessary to validate or test the artifacts of the use case: typically, exploit use case models by transformation, code generation or direct analysis, compile code, execute and test binaries.

5.3 Current integration workflows

We configured a set of actions for each use case repository in order to validate all the modeling process steps. The process for the Railway (ODAS) use case consists of the following steps:

Figure 10: The Capella-Amalthea Bridge CI execution, covering build and deployment



1. high-level modeling using Capella
2. production of an Amalthea model from the Capella model, using the Capella to Amalthea bridge
3. refinement of the Amalthea model using APP4MC
4. production of a synthetic code from the Amalthea model, using the SLG
5. compilation of the synthetic code

Steps 1 and 3 correspond to human activity and cannot be automated. Steps 2, 4 and 5 can be automated in the continuous integration chain to check the models in steps 1 and 3 do not contain errors that would prevent steps 2, 4 and 5 from executing. Therefore, the continuous integration chain for the ODAS use case

- triggers steps 2, 4 and 5 whenever a new version of the Capella model is pushed into the repository, the Amalthea model produced by step 2 is directly processed “as is” by step 4;
- triggers steps 4 and 5 whenever a new version of the Amalhea model is pushed into the repository.

The modeling process of the PCC use case consists of the following steps:

1. production of an Amalthea model using APP4MC
2. production of a synthetic code from the Amalthea model, using the SLG
3. compilation of the synthetic code

Steps 2 and 3 can be automated to check Amalthea models do not contain errors that would prevent the production of a synthetic code that would compile.

5.4 Summary: execution of the continuous integration chain

The Gitlab instance hosted at BSC drives the continuous integration chain by executing the instructions described in the .gitlab-ci.yaml files upon each push into the git repositories. The execution is managed by the gitlab-runner installed in a Docker container that runs on a computer hosted at TRT

Upon execution, the gitlab-runner fetches the last versions of the Git repository files and execute the continuous integration steps. The whole process is illustrated on figure 8.

6 Conclusions

We are progressively setting up the AMPERE ecosystem by integrating all the tools into AMPERE Gitlab. The Gitlab also stores the use case data. We configured Gitlab to execute the AMPERE workflow each time use case data is updated. The workflow is actually executed within a Docker container; thus, we can control the configuration of the workflow environment. As Docker allows the packaging of container images, this also allows the migration of the AMPERE ecosystem from a computer to another.

The continuous integration chain is limited to the AMPERE software tools. That is, the system design and analysis tools. The actual execution of the applications produced with the AMPERE workflow must run on the target platform in order to perform reliable measurements. The platforms (operating systems and hardware boards) obviously cannot be manipulated within a Docker container.

The integration of the AMPERE tools is still on-going. The continuous integration chain already works but is not complete yet. In particular, we mainly focused on the system design; in the remaining of the project, we will focus on the integration of the analysis tools.

7 Acronyms and Abbreviations

CI	Continuous Integration
CPU	Central Processing Unit
D	Deliverable
DSML	Domain Specific Modeling Language
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HPC	High-Performance Computing
MDE	Model-Driven Engineering
MS	Milestone
ODAS	Obstacle Detection Avoidance System
PCC	Predictive Cruise Control
OS	Operating System
SLG	Synthetic Load Generator
T	Task
TDG	Task Dependency Graph
WP	Work Package

8 References

[1] AMPERE, “D6.3. Single criterion AMPERE ecosystem,” June 2021.

[2] “Docker documentation,” Sep. 2022, https://docs.docker.com/.

[3] AMPERE, “D1.5, Metamodel-driven abstraction and model-driven extensions and use case enhancements,” September 2022.

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

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

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

[7] ——, “D5.3, Final multi-criteria operating systems and hypervisor,” September 2022.