



D5.3: MegaM@Rt Integrated Framework - final version

MILESTONE:	31/03/2020 (M36)
STATUS:	FINAL
COORDINATOR:	ATOS
CONTRIBUTORS:	ATOS, INT, SOFT, UCAN, ABO, ARM, UOC, UAQ, CON, SMA, UPAU, MDH, FTS, SSF, VTT
REVIEWERS:	ELIZABETA FOURNERET, VITTORIANO MUTTILLO
DISSEMINATION LEVEL:	PUBLIC
HANDLE:	
LAST EDITED:	31/03/2020

Executive summary

This document describes the software implementation of the last version of the MegaM@Rt Integrated Framework. The first version of this framework was released with [D5.2], according to the integration strategies that were described and selected in [D5.1] for the joint packaging and delivery of the MegaM@Rt framework. This plan described the features to be included in each release of the framework and the release dates.

This last release of the MegaM@Rt Integrated Framework included the features of the first release, with new updates on the integrated toolsets, all them available through the main three integration strategies:

- The MegaM@Rt catalog, a web portal that offers detailed information about all available MegaM@Rt tools, including links to their main artifacts (software downloads, documentation, source code). This catalog facilitates use case providers and external adopters the search and adoption of MegaM@Rt tools that could provide the software features they need to fulfil modeling industrial needs.
- The MegaM@Rt Eclipse IDE, offered both as installable through an update site and as downloadable bundle, which offers an integrated IDE that clusters 90% of the MegaM@Rt open-source tools that runs within the Eclipse IDE. Users just need to download and unzip the bundle to start using the MegaM@Rt features supported in the IDE.
- Docker Container images for few of the remaining MegaM@Rt open-source standalone tools (i.e. executed locally within the user's computer), which can be used by end-users to launch container instances that host the tool and all dependencies. This approach largely simplifies for end-users the installation of the tool, dependencies and examples for getting started.

The MegaM@Rt Eclipse IDE has been published in the Eclipse Marketplace in order to widely promote this toolset into the communities of software developers and modelers.

This document accompanies the MegaM@Rt Integrated Toolset software release, which is made available as a service (e.g. the MegaM@RT catalog) or as downloadable artifacts (e.g. the MegaM@Rt Eclipse IDE and the Docker container images).

MegaM@RT catalog is accessible through the MegaM@RT portal (<https://megamart2-ecsel.eu/>) through the *Downloads/MegaM@RT* Catalog top menu entry.

MegaM@RT Eclipse IDE update site is available in the MegaM@RT Github portal:

<https://github.com/megamart2/integration/raw/master/eu.megamart2.platform.site>

MegaM@RT Docker images are available in the MegaM@RT Github portal at:

<https://github.com/megamart2/integration/tree/master/docker>

MegaM@RT Eclipse IDE is available for Windows/MacOS/Linux as downloadable bundles in the MegaM@RT Github portal at:

<https://github.com/megamart2/integration/releases>

This download link is available for external adopters through the MegaM@RT portal.

Last but not the least, this document also reports technical details and results of the integration hackathon on modelling and traceability, runtime analysis and systems engineering, organized during the last HiPEAC conference in Bologna, open to companies, students, researchers and engineers to experiment together with all kinds of tools for Software Engineering (SE). For companies, this event offered an opportunity to pull expertise and new ideas on interesting practices. For students, the experiments revealed challenges in SE and proposed training opportunities to work in close relation to industry representatives. For researchers, the hackathon was a means to validate their ideas on a case study. For engineers, this was a meeting point to share SE practices in hands-on activities. Different participants in the hackathon decided to work into a unique team, and try to solve a more general problem, common to the hackathon challenges (proposed by THALES, INTECS and University of

L'Aquila), concerning tools transformation and interoperability, being these the reasons for reporting its results in this document.

Table of Contents

Table of Contents	3
1. Introduction	4
1.1. Scope, motivation	4
1.2. Document structure	5
2. MegaM@Rt Toolset Catalog	5
3. MegaM@Rt Eclipse IDE	8
2.1. MegaM@Rt Eclipse Update Site	8
2.2. MegaM@Rt Eclipse IDE Bundle	13
MegaM@Rt inline documentation	14
MegaM@Rt2 Cheat Sheets	15
MegaM@Rt2 IDE published in the Eclipse Marketplace	16
4. Container-based MegaM@Rt standalone tool delivery	17
5. Conclusions	19
6. References	20
Appendix A: Open Hackathons reports at HiPEAC 2020	21
A.1 Bologna (HiPEAC2020) Open Hackathon results	21
A.1.1. THALES Challenge: Synchronize Runtime Aspects with Performance Design and Verification	21
A.1.2. INTECS-UNIVAQ Challenge: Interoperability Pattern and Tool Integration Applying Model-driven Engineering and Development Approach	22

Acronyms

CIF	CERBERO INTEROPERABILITY FRAMEWORK
CSP	COMMUNICATION SEQUENTIAL PROCESS
CPS	CYBER PHYSICAL SYSTEM
CU	COMMUNICATION UNIT
D<X.Y>	DELIVERABLE <X.Y>
HML	HEPSYCODE MODELING LANGUAGE
HW	HARDWARE
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
JSON	JAVASCRIPT OBJECT NOTATION
MARTE	MODELING AND ANALYSIS OF REAL-TIME AND EMBEDDED SYSTEMS
MoC	MODEL OF COMPUTATION
MU	MEMORY UNIT
PU	PROCESSING UNIT
SBM	SYSTEM BEHAVIOR MODEL
SE	SOFTWARE ENGINEERING
SW	SOFTWARE
SysML	SYSTEMS MODELING LANGUAGE
TL	TECHNOLOGY LIBRARY
UC	USE CASE
UML	UNIFIED MODELLING LANGUAGE
XML	EXTENSIBLE MARKUP LANGUAGE

1. Introduction

1.1. Scope, motivation

This document describes the software implementation of the last version of the MegaM@Rt Integrated Framework. In a previous document [D5.2], the first version of the framework was described. In [D5.1] different integration strategies were discussed and some selected for packaging and delivering a joint MegaM@Rt toolset, as well as a plan was scheduled for toolset releases and the main features to be included in each of them, as part of the integration roadmap.

In this document, we describe the features and associated implementations included in the last release of the integrated toolset. This release includes new updates on the integrated MegaM@Rt2 toolsets and the adoption of new strategies to release them to the end-users, including:

- The MegaM@Rt catalog, a web portal that offers detailed information about all available MegaM@Rt tools, including links to their main artifacts (software downloads, documentation, source code) and facilitates searching tools by keyword (i.e. tag cloud).
- The MegaM@Rt Eclipse IDE, offered both as installable through an update site and as downloadable bundle, which offers an integrated IDE that contains 90% of the MegaM@Rt open-source tools that runs within the Eclipse IDE.
- Docker container-based delivery support for some MegaM@Rt standalone tools, that facilitates their adoption by installing the tool and all dependencies in a lightweight container, which can be generated from a public image.

- Public availability of the MegaM@Rt Eclipse IDE in the Eclipse Marketplace, where this IDE can be advertised to the communities of developers, who can be redirected to the MegaM@Rt site for its downloading.

This document accompanies the MegaM@Rt Integrated Toolset software release, which is made available as a service (e.g. the MegaM@RT catalog) or as downloadable artifacts (e.g. the MegaM@Rt Eclipse IDE, container images for some standalone tools). To keep this document self-contained and readable, it evolves from D5.1, keeping its previous content and structure, which is updated to report the new integration features developed during this reporting period.

This document also reports the technical details and the results of the integration hackathon on modelling and traceability, runtime analysis and systems engineering, organized during the last HiPEAC conference in Bologna. It was open to companies, students, researchers and engineers in order to experiment together with all kinds of tools for Software Engineering (SE). For companies, this event offered an opportunity to pull expertise and new ideas on interesting practices. For students, the experiments revealed challenges in SE and proposed training opportunities to work in close relation to industry representatives. For researchers, the hackathon was a means to validate their ideas on a case study. For engineers, it was a meeting point to share SE practices in hands-on activities. During the Hackathon, two different challenges were proposed: one related to an industrial partner (Thales), and another two proposed by one industrial partner (INTECS) and a university (University of L'Aquila). During the Hackathon, the different participants decided to work into a unique team, and to try to solve a more general problem, common to the two proposed challenges, concerning tools transformation and interoperability. These challenges are closely related to the integration of the MegaM@Rt2 toolset.

1.2. Document structure

This document is structured as follows: section 2 describes de MegaM@Rt toolset catalog; section 3 describes the software release of the MegaM@RT Eclipse IDE, which can be installed from its update site (section 3.1) or unzipping the MegaM@RT Eclipse IDE bundle (section 3.2); section 4 describes the Docker container-based delivery of some MegaM@RT open-source standalone tools ;section 5 concludes the document; section 6 lists bibliographical references. The Appendix A reports the technical details and the results of the integration hackathon organized during the last HiPEAC conference.

2. MegaM@Rt Toolset Catalog

In accordance with the release plan described in D5.2, this document describes the release of the last version of the MegaM@Rt catalog, a Web page integrated within the MegaM@Rt Web portal, that maintains the last up-to-date list of available MegaM@Rt tools.

The catalog is accessible at the following location: <https://toolbox.megamart2-ecsel.eu/>



Figure 1 MegaM@Rt catalog

The catalog contains 29 MegaM@Rt tools by the end of March 2020, 6 more than in the first release by January 2019 and new updates on the information about the tools that were already included by that time. Some entries, such as MATERA2, represent a toolset consisting of other related tools. The remaining of this section is taken from D5.2 for self-consistency and reports the updates of the developed features and content of the catalog with respect to that version.

The catalog main page lists all the tools in a grid. This view includes for each tool:

- A tool summary;
- Icons that reference the pages that contain tool information for downloading the software, consulting the documentation and accessing their source code. When a particular information (e.g. download, documentation, source code) is not available, the corresponding icon does not appear.

Tools can be filtered out in the main page based on two main classifications:

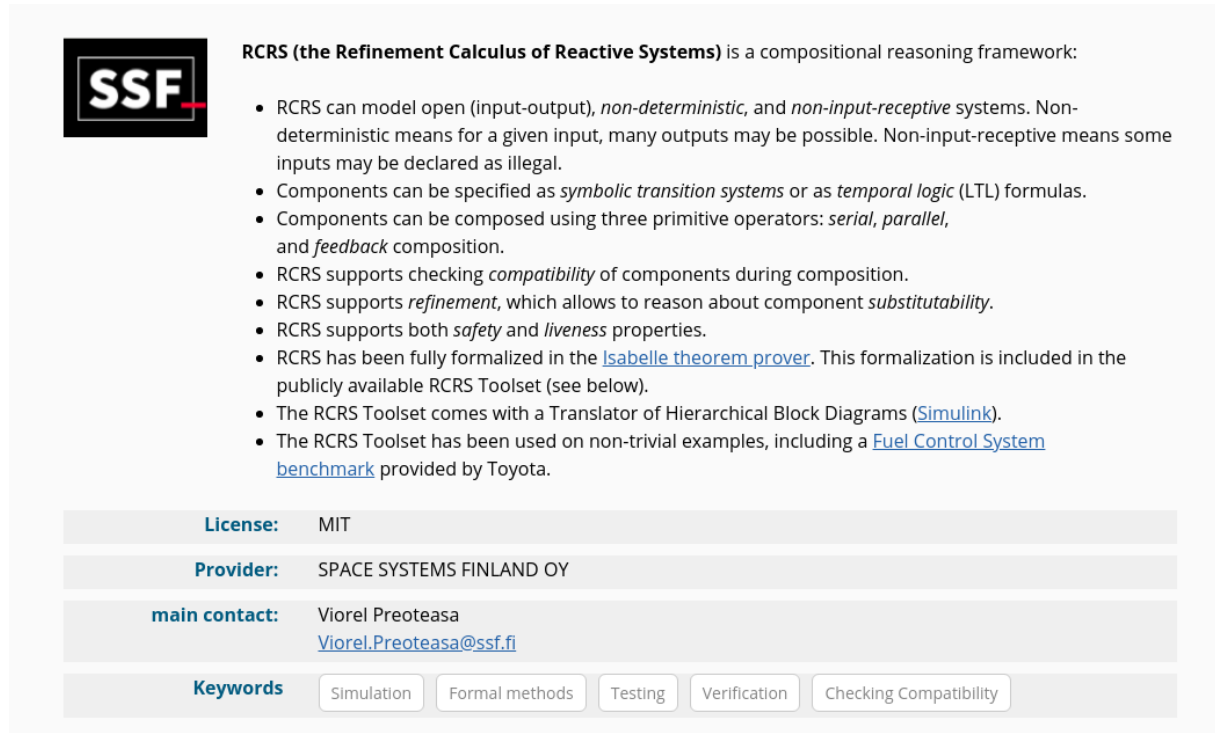
1. Filtering by main MegaM@Rt toolbox category:
 - a) System modeling
 - b) Model and traceability management
 - c) Runtime analysis
2. Filtering by tags (keywords)
3. Filtering by license

Filtering is managed by the left-most panel classifications (Toolbox, Tags). By selecting a concrete entry in both classifications, only the tools classified under such a category are displayed. In case of multiple selections, all the tools that satisfy at least one of the selections are displayed. By deselecting all the categories, the complete list of tools is recovered.

By clicking on one tool title, the corresponding tool page is rendered. This page (see Figure 2) shows a detailed description of the tool (and features), and some meta-data:

- License under what the tool is distributed.
- Tool provider (partner)
- Main contact point
- Associated keywords (used for main page filtering).

By clicking on any of the keywords, the catalog shows a main page with the list of tools that are tagged with that keyword.



SSF

RCRS (the Refinement Calculus of Reactive Systems) is a compositional reasoning framework:

- RCRS can model open (input-output), *non-deterministic*, and *non-input-receptive* systems. Non-deterministic means for a given input, many outputs may be possible. Non-input-receptive means some inputs may be declared as illegal.
- Components can be specified as *symbolic transition systems* or as *temporal logic* (LTL) formulas.
- Components can be composed using three primitive operators: *serial*, *parallel*, and *feedback* composition.
- RCRS supports checking *compatibility* of components during composition.
- RCRS supports *refinement*, which allows to reason about component *substitutability*.
- RCRS supports both *safety* and *liveness* properties.
- RCRS has been fully formalized in the [Isabelle theorem prover](#). This formalization is included in the publicly available RCRS Toolset (see below).
- The RCRS Toolset comes with a Translator of Hierarchical Block Diagrams ([Simulink](#)).
- The RCRS Toolset has been used on non-trivial examples, including a [Fuel Control System benchmark](#) provided by Toyota.

License: MIT

Provider: SPACE SYSTEMS FINLAND OY

main contact: Viorel Preoteasa
Viorel.Preoteasa@ssf.fi

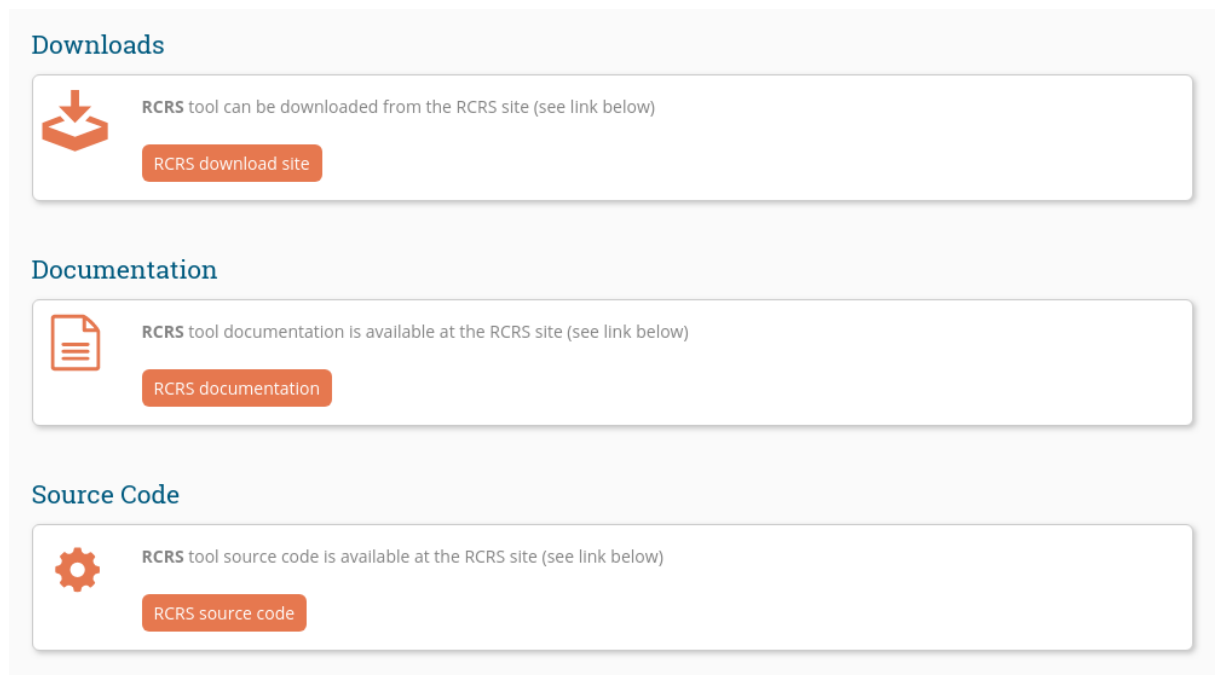
Keywords

Simulation Formal methods Testing Verification Checking Compatibility

Figure 2 Tool detail

The detail tool page may also include a number of subsections specialized on referencing relevant additional sources for:

- Downloading the tool software
- Accessing the tool documentation
- Accessing the tool source code (for those tools released under a open source license)



Downloads

RCRS tool can be downloaded from the RCRS site (see link below)

RCRS download site

Documentation

RCRS tool documentation is available at the RCRS site (see link below)

RCRS documentation

Source Code

RCRS tool source code is available at the RCRS site (see link below)

RCRS source code

Figure 3 Links to relevant sources of the tool

The MegaM@Rt catalog has been implemented in Drupal¹ and the source code is directly available in the catalog server. The Drupal plugin Views² has been adopted (together with additional PHP code) to enable multiple selection in filtering taxonomies (one for main MegaM@Rt classification, another one for keywords/tags).

3. MegaM@Rt Eclipse IDE

Also in accordance to the release plan described in D5.1 [D5.1], this document describes the release of the last version of the MegaM@Rt Integrated Framework, which includes the **MegaM@RT Eclipse IDE**, which is released to the end-users according with the following strategies:

- *The MegaM@Rt Eclipse update site*: this common online update site enables end-users to install individual MegaM@Rt tools (as Eclipse plugins) on their existing compatible Eclipse IDEs.
- *The MegaM@Rt Eclipse bundle*: this downloadable Eclipse bundle includes the complete set of MegaM@Rt Eclipse-based tools integrated within the above update site, and also all required third-party dependencies.
- Eclipse Marketplace: The *MegaM@Rt Eclipse bundle* has been published in the Eclipse Marketplace for broadcasting it within the wide Eclipse community of developers.

The following subsections provide additional details about the implementation of both artifacts.

2.1. MegaM@Rt Eclipse Update Site

The MegaM@Rt Eclipse update site enables end-users to install into their Eclipse instances individual (or the complete set of) MegaM@Rt tools integrated within the common MegaM@Rt Eclipse IDE. This IDE is based on the Eclipse Modeling 2018-09 version, which can be downloaded from:

<https://www.eclipse.org/downloads/packages/release/2018-09/r>

The following MegaM@RT tools have been integrated within the update site:

- Papyrus for AOM extension: <https://github.com/megamart2/tool-papyrus-extensions>
- Moka logging extension: <https://github.com/megamart2/tool-papyrus-extensions>
- NeoEMF: <https://github.com/atlanmod/NeoEMF>
- EMFViews: <https://github.com/atlanmod/emfviews>
- Collaboro: <https://github.com/megamart2/tool-collaboro>
- EMFToCSP: <https://github.com/megamart2/tool-emftocsp>
- PADRE: <https://github.com/SEALABQualityGroup/padre>
- VeriATL: <https://github.com/veriatl/VeriATL/>
- JTL: <https://github.com/MDEGroup/jtl-eclipse>
- HEPSCODE: <https://github.com/megamart2/tool-hepsycode>
- S3D: <https://s3d.unican.es/>

Another MegaM@Rt tool could not be integrated after some upgrade development:

- CHESS: [CHESS source code repository](#)

¹ <https://www.drupal.org/>

² <https://www.drupal.org/project/views>

The development team worked on upgrading CHESS to the Eclipse Modeling 2019-08 version (baseline for MegaM@Rt IDE) during the project lifetime. Unfortunately, there were still some features (e.g. palette, some profiles and other) which did not have migration support and that would require significant effort and time to re-implement them.

Eclipse-based tools whose distribution license is commercial will neither be integrated within this update site nor delivered within the MegaM@Rt Eclipse IDE bundle, since they cannot be freely distributed:

- XPM
- Conformiq Designer

There are other Eclipse-based tools with restricted distribution license that could not be included within the MegaM@Rt Eclipse IDE (update site, bundle) because of such restriction:

- MATERA2 toolset

The MegaM@Rt Eclipse update site is available at:

<https://github.com/megamart2/integration/raw/master/eu.megamart2.platform.site>

This site has been implemented as an Eclipse site project using as target the Eclipse Modelling 2018-09 and other third party required dependencies, whose update sites have been defined in the Eclipse platform target defined at:

<https://github.com/megamart2/integration/tree/master/eu.megamart2.platform.target>

Getting started with MegaM@RT update site

Step 1: Use an existing instance of Eclipse Modeling 2018-09 or download/unzip it from:

<https://www.eclipse.org/downloads/packages/release/2018-09/r>

Step 2: Register the following software sites in the Eclipse IDE:

- Epsilon: <http://download.eclipse.org/epsilon/updates/>
- MARTE Profile: <https://download.eclipse.org/modeling/mdt/papyrus/components/marte/>
- Moka:
- <https://github.com/megamart2/org.eclipse.papyrus-moka/raw/megamart2/releng/org.eclipse.papyrus.moka.p2/target/repository>
- Emfatic: <http://download.eclipse.org/emfatic/update>

These sites are required to install third party dependencies of the MegaM@RT Eclipse IDE.

Register the MegaM@Rt Eclipse IDE site:

MegaM@Rt: <https://github.com/megamart2/integration/raw/master/eu.megamart2.platform.site>

To register each of these software sites, proceed as follows (see Figure 4):

1. Go to the Eclipse main menu *Help/Install New Software*
2. On the Install wizard, click on the *Manage...* button
3. For each site on above list:
 - a) On the Preference wizard, click on the *Add...* button
 - b) Enter the site name and location (the URL listed above for each site)

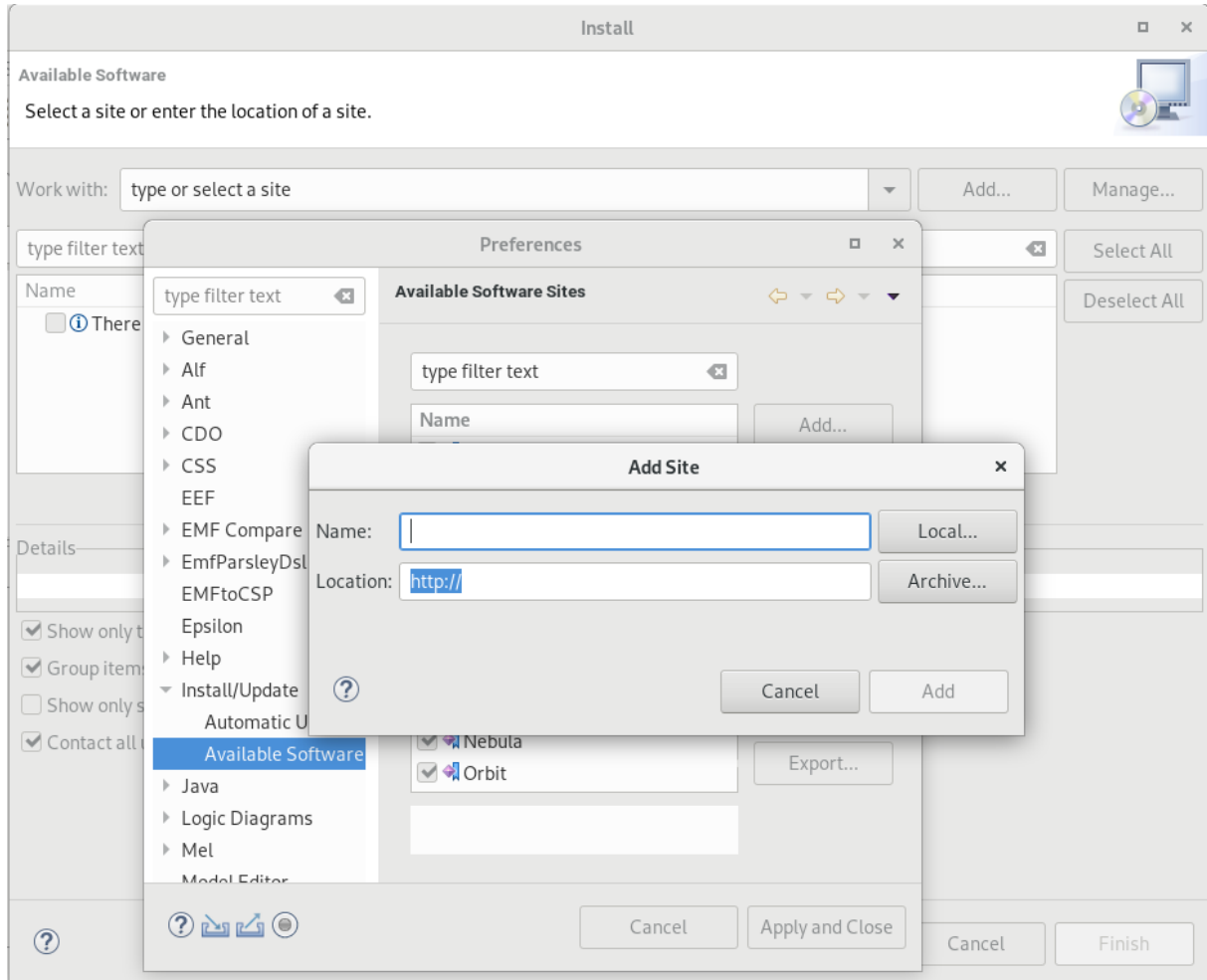


Figure 4 Registering a software site

Once this process is completed, the lists of available software sites should look like shown in Figure 5.

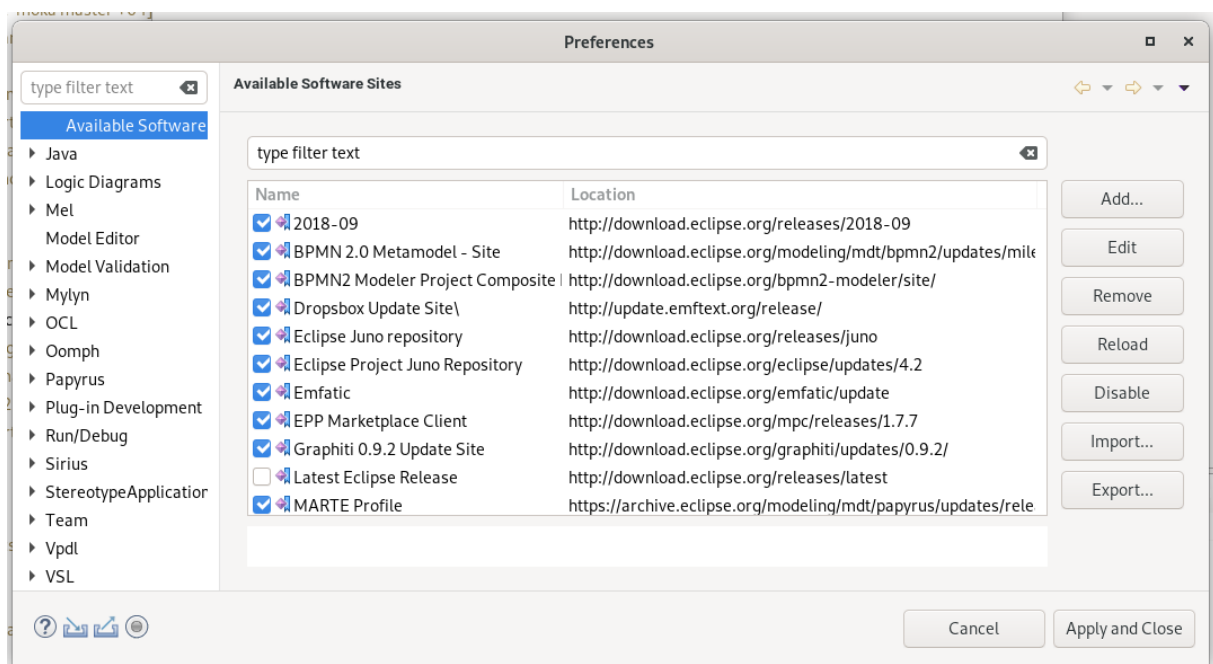


Figure 5 MegaM@Rt Software Sites

Step 3: Install the following dependencies:

- Papyrus for UML 4.1.0, using the 2018-09 software site
- Acceleo 3.7.5, using the 2018-09 software site
- Moka 4.0.0, using the Moka software site
- Epsilon features, using Epsilon software site:
 - Epsilon Core
 - Epsilon Core Development Tools
 - Epsilon Development Tools for EMF
 - Epsilon Development Tools for UML
 - Epsilon GraphML Integration
 - Epsilon UML Integration
 - Epsilon Validation Language EMF Integration
 - Epsilon Wizard Language EMF Integration
 - Epsilon Wizard Language GMF Integration
 - Eugenia
- Emfactic 0.8.0, using Emfactic software site
- ATL SDK 4.0.0, using the 2018-09 software site
- EMFText SDK 1.4.1 from the Eclipse Marketplace
- Papyrus MARTE 1.2.0, using the MARTE software site

Step 4: Install MegaM@Rt tools, using the MegaM@Rt software site. You can install individual tools or the complete set, on demand (see Figure 7).

Step 5: Install tool-specific external dependencies into your OS:

- EMFToCSP requires:
 - Graphviz: <https://www.graphviz.org/>
 - Eclipse constraints Programming System 6.0/6.1: <http://www.eclipseclp.org/>
- EMF Views requires:
 - Sexp2emf: <https://github.com/atlanmod/sexp2emf>

In other to install the dependencies and the MegaM@Rt tools, follow the following procedure:

1. Go to the Eclipse main menu *Help/Install New Software*
2. In the *Work with* combo box, select the target software site. Wait for it to be loaded
3. In case of Papyrus installation, search for it in the *type filter text* (see Figure 6)
4. Select the software to install (see Figure 7)
5. Check the checkbox: *Contact all update sites during install to find required software*
6. Proceed through the following wizard pages accepting when prompted.
7. Restart Eclipse after each software tool is installed.
8. After restart, check that the tool has been installed (see Figure 8)
 - a) Go to the main menu *Help/About Eclipse IDE*
 - b) In this wizard, click on *Installation details*
 - c) Find the tool in the list of *Installed Software* tab

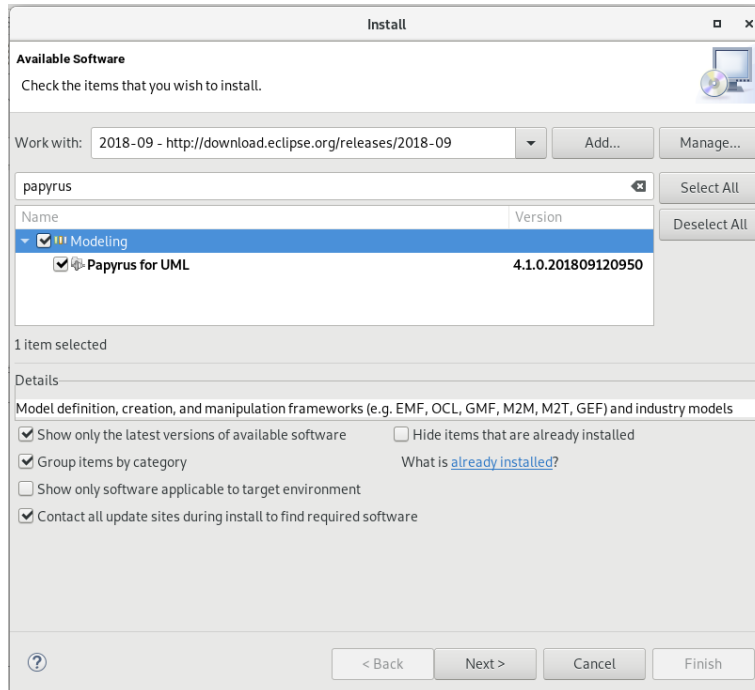


Figure 6 Installing Papyrus

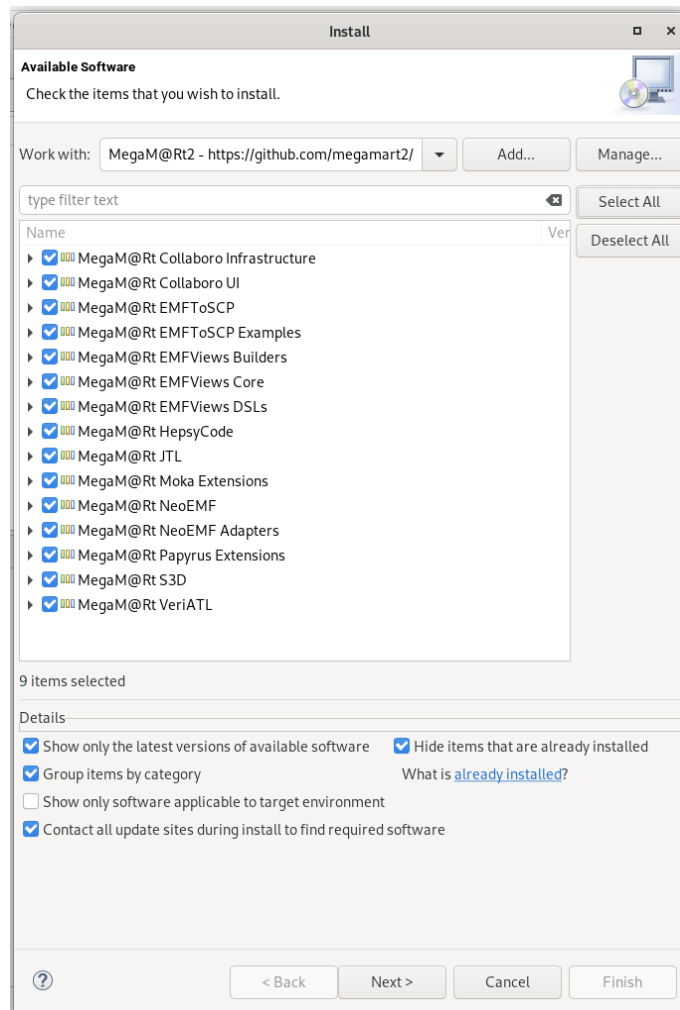


Figure 7 Installing MegaM@Rt tools

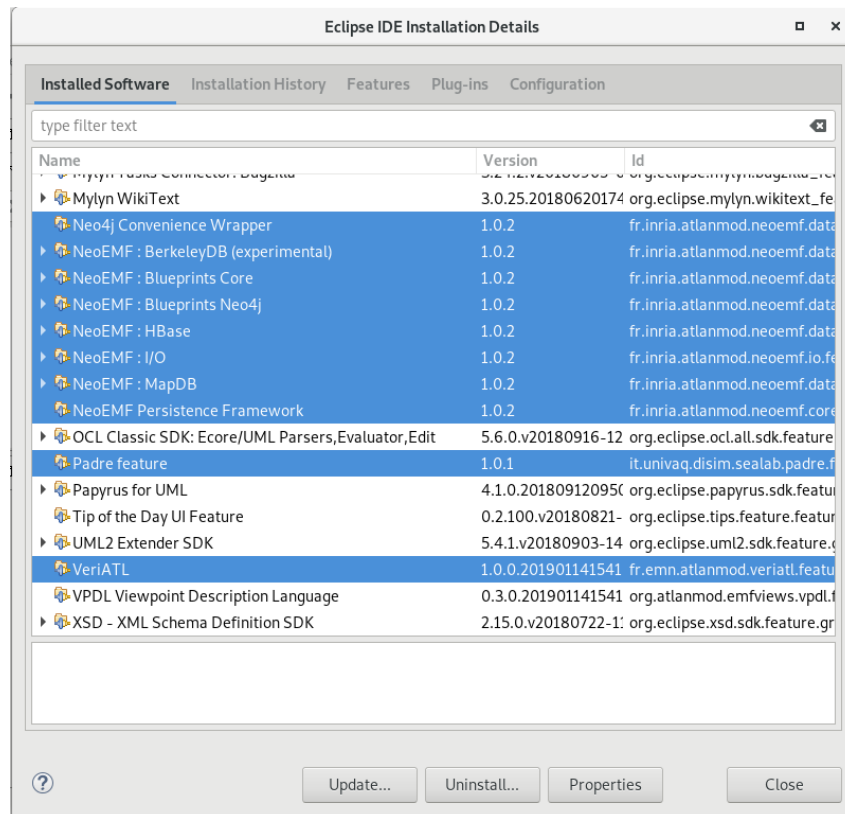


Figure 8 MegaM@Rt tools installed in Eclipse IDE

2.2. MegaM@Rt Eclipse IDE Bundle

In order to facilitate the adoption of the MegaM@Rt Eclipse IDE by end-users, including use case providers and external adopters, instances of the MegaM@Rt Eclipse IDE have been bundled for the supported operating systems (Windows, MacOS and Linux). They are available for download at the MegaM@Rt Web site, under Downloads/MegaM@Rt2 Eclipse IDE, which points to the MegaM@Rt2 GitHub releases page:

<https://github.com/megamart2/integration/releases>

Additionally, they are accessible from the MegaM@Rt GDrive site for all consortium partners:

- [MegaM@Rt Eclipse IDE for Linux](#)
- [MegaM@Rt Eclipse IDE for MacOS](#)
- [MegaM@Rt Eclipse IDE for Windows](#)

These IDE bundles have been created by following the installation procedure described in the previous subsection.

The MegaM@Rt Eclipse IDE bundle is based on Eclipse Modeling 2018-09 and includes all the plugin dependencies required by the integrated MegaM@Rt tools and all the integrated MegaM@Rt tools listed in the previous section.

Next paragraphs described additional features integrated with the MegaM@Rt Eclipse IDE to facilitate to the end users the adoption of these tools.

Eclipse provides an internal inline documentation that ships within every installation, offering a user's manual for the tool plugins installed. MegaM@Rt IDE also provides the documentation of the tools integrated, which are available by launching the Help Contents

The documentation of the following MegaM@Rt tools have been included:

- EMF2CSP
- EMFViews
- HEPSYCODE
- JTL
- Moka logging extension
- NeoEMF
- PADRE
- Papyrus for AOM
- VeriATL
- Collaboro
- S3D

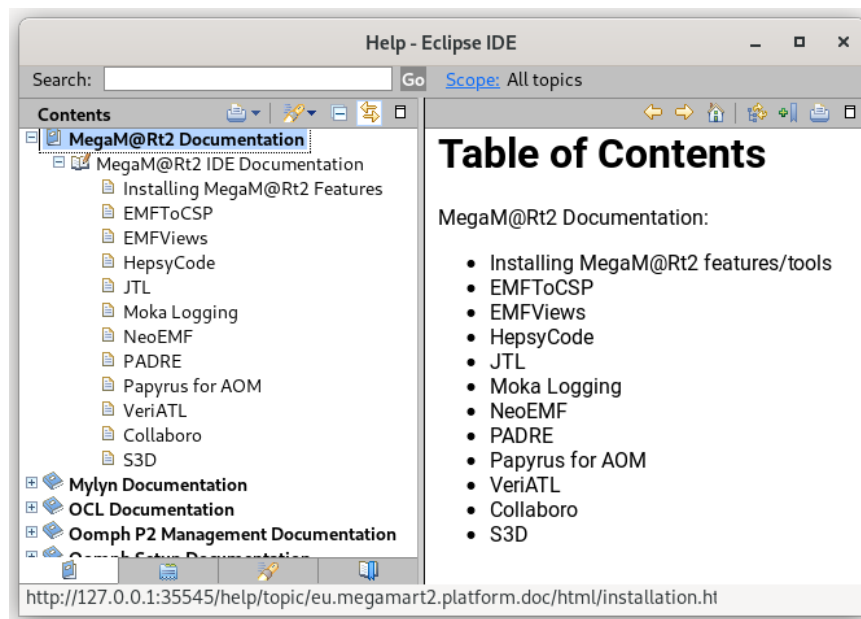


Figure 9 MegaM@Rt inline documentation

Figure 9 shows the MegaM@Rt documentation structure within the Eclipse inline Help contents. Figure 10 shows an example of inline documentation for the EMF2CSP tool.

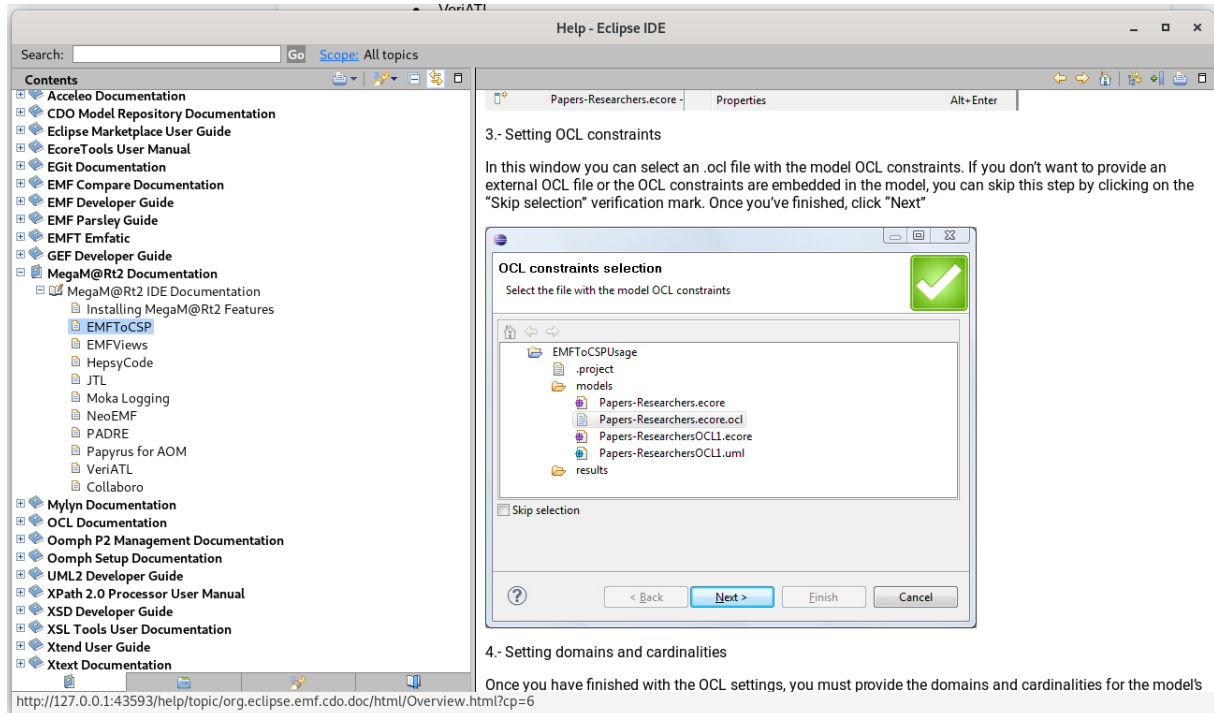


Figure 10 MegaM@Rt inline documentation for EMF2CSP tool

Complementing the inline documentation, and in order to assist end-users in the quick adoption of the MegaM@Rt tools included in the IDE, we have developed cheat sheets for some of these tools (Figure 11).

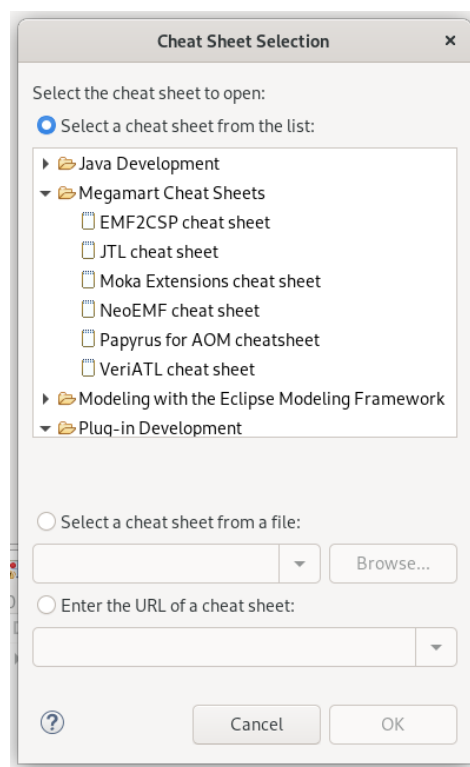


Figure 11 Cheat sheets for MegaM@Rt tools

Cheat sheets provide a walkthrough tutorial that describes in several sequential steps how to use a particular tool. The user is guided through the different steps, even supported to interact with the tool in the Eclipse workbench (Figure 12).

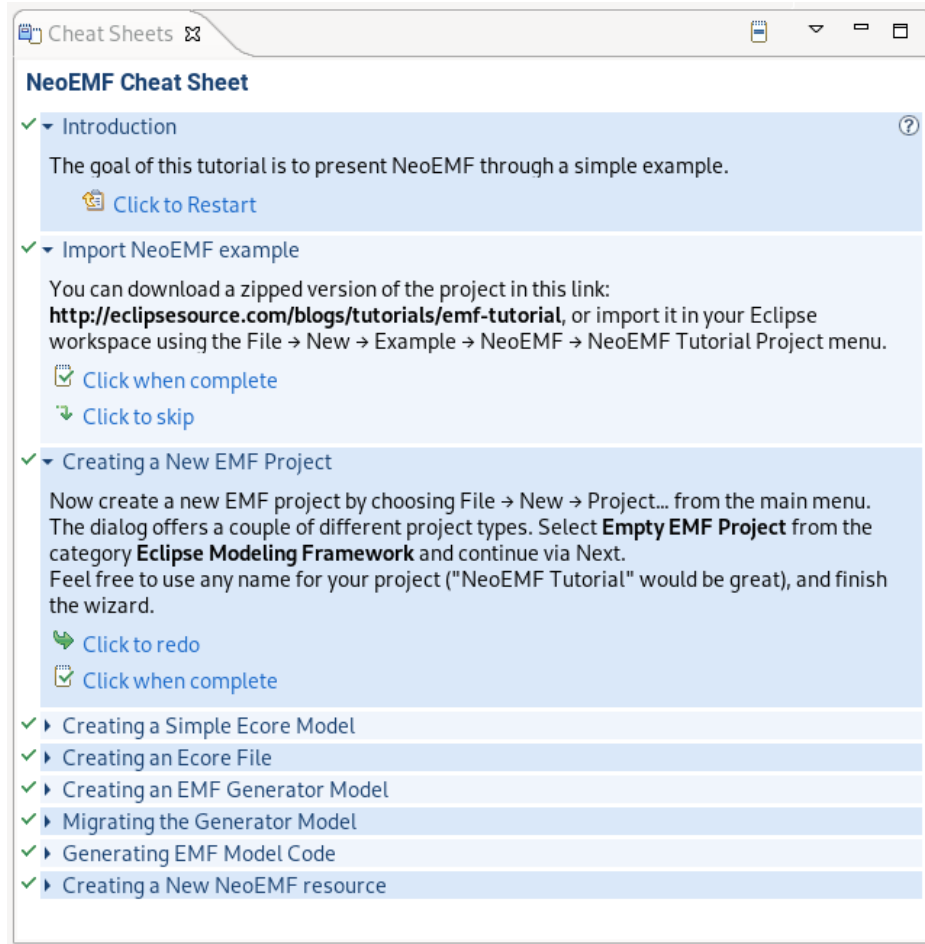


Figure 12 Cheat sheet for NeoEMF tool

The MegaM@rt IDE has been published in the Eclipse Marketplace (Figure 13). This is a popular portal where communities of developers can browse for Eclipse tool plugins and install them. The market is available within the Eclipse workbench and in a browser. Within the Eclipse workbench, users can search for MegaM@Rt IDE in the Marketplace and install the entire toolset or individual tools, without required to setup the MegaM@Rt update site.

MegaM@Rt IDE is available at the Eclipse Marketplace URL:

<https://marketplace.eclipse.org/content/megamart2-ide>

The screenshot shows the Eclipse Marketplace interface for the MegaMart2 IDE. The top navigation bar includes 'Create account', 'Log in', 'My Marketplace', 'Add Content', and 'More'. The breadcrumb trail is 'Home / Marketplace / Tools (1604) / MegaMart2 IDE'. The main content area is titled 'MegaMart2 IDE' and features an 'Install' button, a star rating of 1, and a list of features and tools. The features list includes:

- scalable methods and tools for modelling of functional and non-functional properties such as performance, consumption, security and safety with mechanisms for representation of results of runtime analysis.
- scalable methods and tools for application validation at runtime including scalable methods for `models@runtime`, verification and online testing.
- infrastructure for efficient handling and management of numerous, heterogeneous and large models potentially covering several functional and non-functional domains.
- holistic traceability 1) capable to link and manage models and their elements from different tools as well as 2) suitable for large distributed cross-functional working teams.

The following **MegaM@Rt2** tools have been included in this IDE:

- Papyrus for extension for AOM: <https://github.com/megamart2/tool-papyrus-extensions>
- Moka extension for execution logging: <https://github.com/megamart2/tool-papyrus-extensions>
- NeoEMF: <https://github.com/atlanmod/NeoEMF>
- EMFViews: <https://github.com/atlanmod/emfviews>
- Collaboro: <https://github.com/megamart2/tool-collaboro>
- EMFToCSP: <https://github.com/megamart2/tool-emftocsp>
- PADRE: <https://github.com/SEALABQualityGroup/padre>
- VeriATL: <https://github.com/veriatl/VeriATL/>

Figure 13 The MegaM@Rt IDE published in the Eclipse Marketplace

4. Container-based MegaM@Rt standalone tool delivery

MegaM@rt open-source standalone tools, which are not part of the MegaM@Rt Eclipse IDE require a different delivery strategy. In the clustering of MegaM@rt tools described in [D5.1], there were only few open source standalone tools, namely:

- PauWare API and Engine, XModelling Studio
- AIPHS
- LIME Testbench
- RCRS

For some of these tools, we have adopted a Docker container-based strategy, consisting of creating a Docker image that installs the tools, its dependencies and testing examples within a single lightweight container. End-users can generate the container from the image description and then run the tool within the container. The generation of the image takes place only once. Then, the container is available locally (i.e. within the end-user computer) to run the tool multiple times.

Docker images have been created for **PauWare API and Engine**, **AIPHS** and **LIME Testbench** tools. They are available at the MegaM@rt GitHub portal:

<https://github.com/megamart2/integration/tree/master/docker>

The following paragraphs provide instructions to use these Docker images. To get the images, pull above repository with:

```
git clone https://github.com/megamart2/integration.git
```

This creates a local *integration* repository (\$integration in the following)

PauWare API and Engine

This image installs the PauWare API and Engine library and accompanying examples into an Eclipse IDE. To create the image, go to the \$integration/docker/PauWare directory and execute:

```
./build_pauware_image.sh
```

Then, launch PauWare API (in Linux) by launching *./start_pauware.sh*

An Eclipse IDE with PauWare examples ready to be run appears on the screen (Figure 14)

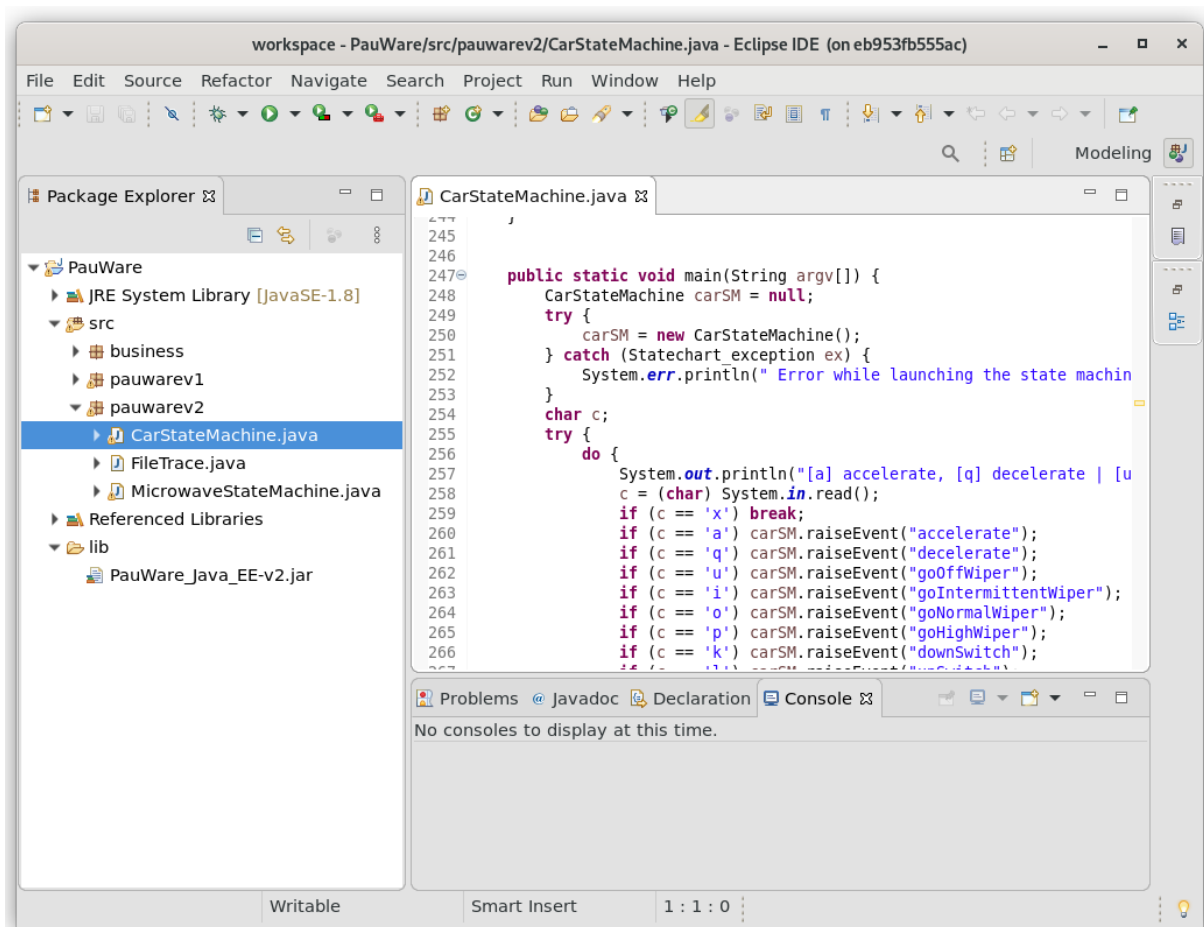


Figure 14 PauWare Tool running on an Eclipse container

The image also integrates the PauWare XModelling Studio³

AIPHS

AIPHS depends on the Vivado 2017.4 proprietary software, which has to be downloaded by the user, after registering in the Xilinx Web site⁴. The *build_aihps_image.sh* script requests the user to download the Linux version of this software and provides the download link. Then, once downloaded, it creates an

³ <https://pauware.univ-pau.fr/xmodeling/index.html>

⁴ www.xilinx.com

AIPHS image that includes this tool and all its dependencies. Next, the user has to execute the following scripts in the following order:

- Build the AIPHS container with *create_aihps_container.sh*
- Install the Vivado software with *install_vivado_in_image.sh*. This process cannot be automated within the Docker image or container, since it requires the user's Xilinx account. During the installation, you will be prompted to get the latest Xilinx Design Tools; ignore the message and continue. In following wizard pages, when prompted, enter your Xilinx account. In the following pages, select to install the WebPack version.
- Then, the user needs to setup manually some aspects of the install software:
 - Access the container with: `docker exec -it aihps bash`
 - Proceed with the user-centric configuration instructions located at: https://github.com/alkalir/aihps_alka/tree/master/getting_started
Skip the instructions to download and install software, since it is already installed within the container.
- Launch AIPHS and Vivado software with *vivado.sh*

LIME Testbench

This image installs the LIME Testbench tool with its dependencies and accompanying examples. To create the image, go to the \$integration/docker/LIME Testbench directory and execute:

```
./build_lime_image.sh
```

Then, open the LIME Testbench tool (in Linux) by launching *./start_lime.sh*. This script gets access to the LIME container. Then, read the README.txt file for instructions to test the tool.

5. Conclusions

This document describes and references the software implementation of the last version of the MegaM@RT Integrated Toolset. This toolset is released with three facets:

- The MegaM@RT catalog: it contains information and useful artifacts that facilitates the adoption of the MegaM@Rt tools by use case providers and external practitioners. It also serves for the dissemination of the MegaM@RT results in order to gain wide visibility.
- The MegaM@Rt Eclipse IDE provides a joint packaging and delivery of the Eclipse-based open source MegaM@RT tools, altogether working on the same IDE and workspace. This IDE largely facilitates the usage of all the MegaM@Rt features supported by the tools included in this toolset.
- A number of Docker based images have been produced for packaging of some MegaM@Rt open source standalone (or not Eclipse-based) tools. A container image packages, in a single container, the tool and its dependencies, enabling a straightforward tool access for users, including exemplary test examples.

The MegaM@Rt IDE has been published into the Eclipse Marketplace, enabling the dissemination of the toolset among the communities of open source software developers aiming for wide adoption.

Both facets of the MegaM@RT Integrated Toolset have been evaluated by the use case providers in project evaluation phases. It is expected that through the dissemination activities, the MegaM@Rt Toolset is adopted by a wide external community of developers and modelers.

6. References

- [D5.1] Atos, D5.1: MegaM@Rt Integration approach. MegaM@Rt technical report. 2018
- [D5.2] MegaM@Rt Integrated Framework - initial version. MegaM@Rt technical report. 2019
- [HiPEAC2020] HiPEAC COnference, “Ultimate CPS Metamodeling and online runtime intelligent analysis Hackathon”, <https://www.hipeac.net/2020/bologna/#/schedule/sessions/7753/>
- [D2.1] MegaM@Rt² “D2.1: Foundation for Model-driven Design Methods”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.1>] [December 2017]
- [D2.2] MegaM@Rt² “D2.2: Design Tool Set Specification”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.2>] [February 2018]
- [D2.3] MegaM@Rt² “Design Tool Set - Initial version”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.3>] [June 2018]
- [D2.4] MegaM@Rt² “MegaM@Rt design tool set – intermediate version”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.4>] [November 2018]
- [D2.5] MegaM@Rt² “MegaM@Rt design tool set – final version”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.5>][November 2019]
- [D2.6] MegaM@Rt² “MegaM@Rt design tool set guidelines – final version”
[<http://hdl.handle.net/20.500.12004/1/P/MMART2/D2.6>][November 2019]
- [D5.1] MegaM@Rt Integration Approach [November 2018]
- [D5.2] MegaM@Rt Integrated Framework – initial version [January 2019]
- [CCM12] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio A. Zovi and T. Vardanega: “CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems”, in proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering, IEEE, 2012.
- [M16] Mazzini, Silvia et al. “CHESS: an Open Source Methodology and Toolset for the Development of Critical Systems.” *EduSymp/OSS4MDE@MoDELS* (2016).
- [Pap18] <https://www.eclipse.org/papyrus/>
- [CHESSML] <https://www.eclipse.org/chess/publis/CHESSMLprofile.pdf>
- [CHESS] <https://www.eclipse.org/chess/start.html>
- [OCRA] <https://ocra.fbk.eu/>
- [MAST] <https://mast.unican.es/>
- [HEPSY] <http://www.hepsycode.com/>
- [Catalog] <http://toolbox.megamart2-ecsel.eu/>

Appendix A: Open Hackathons reports at HiPEAC 2020

This appendix reports technical details and results of the hackathon challenges related to modelling and traceability, runtime analysis and systems engineering, organized during the last HiPEAC conference in Bologna. “*The Ultimate CPS Metamodeling and online runtime intelligent analysis*” Hackathon, presented during the last HiPEAC conference [HiPEAC2020], was open to companies, students, researchers and engineers to experiment together with all kinds of tools for Software Engineering (SE). For companies, this event offered an opportunity to pull expertise and new ideas on interesting practices. For students, the experiments revealed challenges in SE and proposed training opportunities to work in close relation to industry representatives. For researchers, the hackathon was a means to validate their ideas on a case study. For engineers, this was a meeting point to share SE practices in hands-on activities. During the Hackathon, two different challenges have been proposed, one related to an industrial partner (Thales France), and another one proposed by one industrial (INTECS) and one university (University of L’Aquila) Megamart2 partners.

During the Hackathon, the different participants decided to work into a unique team, and try to solve a more general problem, common to the two proposed challenges, concerning tools transformation. The main results have been applied to the INTECS - UNIVAQ challenge, while future work would apply the same found methodology also to the TRT challenge.

The Hackathon core team was composed by the following person:

- Rafik Henia (Thales - TRT)
- Michiel van den Baar (Netherlands Organisation for applied scientific research TNO)
- Julio Oliveira (Netherlands Organisation for applied scientific research TNO)
- Lodiana Beqiri (University of Perugia)
- Vittoriano Muttillio (University of L’Aquila - UAQ)
- Vincenzo Stoico (University of L’Aquila - UAQ)
- Pierluigi Pierini (INTECS)
- Stefano Puri (INTECS)
- Eugenio Villar (University of Cantabria - UCAN)

The remainder of this appendix describes the two challenges, and the results related to the second one.

A.1 Bologna (HiPEAC2020) Open Hackathon results

A.1.1. THALES Challenge: Synchronize Runtime Aspects with Performance Design and Verification

CONTEXT AND GOALS

The objective of this hackathon challenge is to extend the real-time platform Time4Sys with connections to performance design and verification tools. The scenario proposed by Thales consists in synchronizing a Time4Sys model, automatically obtained from runtime traces, with the performance verification tools MAST and/or pyCPA as well as the design tool Papyrus. The connection to Papyrus should be bi-directional. All connections are to be implemented in Java and integrated in Time4Sys.

The system model proposed by Thales consists of an aerial video tracking system model. The main system tasks consist in:

- Displaying high quality video images to the pilots.
- Following the tracked object even when it is temporarily hidden from view (e.g. the vehicle proceeds in and out of a tree obstructed area) through motion prediction.

- Detecting patches of the image that may be moving differently from the background by combining image registration and motion prediction.

The main goal is to synchronize a Time4Sys model, obtained from runtime traces, with the performance verification tools MAST and/or pyCPA and the design tool Papyrus. Communication with Papyrus must be bi-directional. The synchronization with Time4Sys allows filling eventual semantic gaps that may exist between the system model in the design tool and the timing model in performance verification tools. This is done by applying a set of model transformation from the transformation library included in Time4Sys.

PARTICIPANTS

- Proposer: Thales (TRT)
- Implementer: Netherlands Organisation for applied scientific research TNO

RESULTS

As a result of the Hackathon, the team decided to apply a similar solution found for the INTECS - UNIVAQ challenge presented in the next section, but this work will be completed in future activities and projects.

A.1.2. INTECS-UNIVAQ Challenge: Interoperability Pattern and Tool Integration Applying Model-driven Engineering and Development Approach

CONTEXT AND GOALS

The goal of this hackathon challenge inside the Megamart2 Hackathon session at HiPEAC 2020 is to analyze CHESS and HEPHYCODE methodologies, to find similarities and differences, and to propose an integration pattern as-much-as-possible general and applicable to several scenarios (considers model-driven engineering theoretical aspect and model-to-model transformation patterns and frameworks). The main goal is to use the two reference tools (CHESS developed by INTECS and HEPHYCODE developed by University of L'Aquila) as a starting point to propose an interoperability framework between different modeling tools, considering requirement specifications, Model of Computations (MoCs), metamodels and artifacts produced/consumed, modeling activities and possible requirement traceability. The hackathon has been organized in this way:

1. Challenge description: a short presentation given by INTECS and University of L'Aquila regarding their methodology and tool, and the challenge presented.
2. The involved people received documentation regarding:
 - a. HEPHYCODE (papers, installation guide, starter guide)
 - b. CHESS (papers, installation guide, starter guide)
 - c. Megamart2 deliverable D2.1 [D2.1] and D2.2 [D2.2], for a general presentation of the toolset and Megamart approach; D2.3 [D2.3], D2.4 [D2.4], D2.5 [D2.5], describe, in section 4, details of the cooperation between some tools providers with information of integration strategy adopted between respective tools; D2.6 [D2.6], include methodology and guidelines using Megamart toolset; finally D5.1 [D5.1], D5.2 [D5.2] describing the integration activities. These deliverables are available free of charge on the megamart2 official website
 - d. A reference challenge document
3. INTECS and University of L'Aquila gave all the reference links to download, install and try their tools. They have also supported the people involved in the Hackathon to solve problems during the hackathon work.

The next paragraph will present briefly the two considered tools, the detailed activity description, the expected and the actual results

CHESS

CHESS [M16] [CCM12] is a model-driven methodology for the design, verification, and implementation of high-integrity, hard real-time, safety-critical systems adopting the “correctness by construction” concept. CHESS supporting toolset is implemented on top of Eclipse Papyrus UML editor and is made available as an open-source Eclipse-Polarsys project [Pap18].

Metamodel

CHESS comes with its own component model and modeling language, the latter implemented as UML, MARTE and SysML profile, called CHESSML [CHESSML]. CHESSML profile restricts the set of UML entities that can be created, for consistency, avoiding redundancy and fixing semantic variation points, and provides:

1. A set of stereotypes and profiles to support component-based design;
2. Some MARTE stereotypes extensions to allow the specification of computation-independent real-time properties;
3. A new set of stereotypes to support dependability modelling.
4. A new set of stereotypes to support contract-based design and analysis.

Requirement Specification

CHESS comes with a dedicated view/package called “RequirementView”, where the SysML Requirement Diagram is applied to model the requirements. Besides, Papyrus, which is the tool on which CHESS is based, allows importing requirements from external sources like excel, csv files, or by using the ReqIF (requirement interchange format), the latter supported by most of the commonly used requirement management tools.

Component Definition

The definition of the system/software architecture foresees the design of components in isolation, i.e. components designed out of any particular context (to exploit their reuse), or the identification (reuse) of the components from existing libraries/models. The modeling of the component is comprehensive of its boundary identification (in terms of the input/output ports) and of additional information about its possible (functional and extra-functional) behaviors and the expected behaviors of the context. Requirements available in the CHESS/Papyrus model can then be linked to the defined components by using the standard SysML support, e.g. by allocating requirements to components using the “satisfy” relationships.

Requirements Formalization and Contract Definition

This activity has the goal of translating textual requirements allocated to system components into formal properties. Well-formed formal properties are textual expressions specified using a restricted grammar (Linear Temporal Logic) so that their semantics is formally defined in rigorous mathematical terms. Formal properties can be further structured into contracts by assigning them to the assumption and guarantee fields of the contract.

CHESS Guidelines

CHESS is a tool realized on top of Papyrus modeler, thus inheriting the main modeling and editing characteristics. A comprehensive CHESS use and reference guide are available at [CHESS]. Further details can be found on D2.6 [D2.6] MegaM@Rt2 deliverable or on the official website. Several kinds of model-based analysis are supported in CHESS: failure propagation analysis, state-based analysis, contract-based analysis (via integration with OCRA [OCRA] tool), schedulability analysis (via integration with MAST [MAST] tool).

Download

-CHESS Windows 64 bit:

https://drive.google.com/file/d/1pq_JHwqcAhQ5V-Sh0j_WuUDuZq9C8sll/view?usp=sharing

-CHESS Linux 64 bit:

<https://drive.google.com/file/d/1BErdaUCbsqlvzOXQujQy--rPANACFclG/view?usp=sharing>

VIDEO:

<https://drive.google.com/file/d/1eGtTr6TFFrk55en-NvzLjWqwCfD5HIq9/view?usp=sharing>

Documentation

<https://www.eclipse.org/chess/start.html>

HEPSYCODE

HEPSYCODE [HEPSY] (acronym of HW/SW CO-DEsign of HEterogeneous Parallel dedicated SYstems) is a methodology, framework, and tool able to help designers during the whole design flow of embedded parallel dedicated systems. HEPSYCODE defines the different level of abstraction, fully integrated inside the MegaM@Rt2 framework.

HEPSYCODE solves the historical HW/SW Co-Design partitioning problem offering tools features, functionalities and methods able to drive designers from the first input to the final implementation, avoiding system deviation or design errors. All the HEPSYCODE functions have been implemented using Eclipse MDE technologies, the SystemC custom simulator and evolutionary algorithms for partitioning activities, all integrated into an automatic framework.

HEPSYCODE Guidelines

The design activity in HEPSYCODE starts with a high abstraction level modeling language (called HEPSYCODE Modeling Language, HML). The initial HML model is then transformed into an executable SystemC model based on Communicating Sequential Processes (CSP) Model of Computation (MoC). HEPSYCODE defines two system-level models, the System Behavior Model (SBM) and the Technology Library (TL). The former is realized from the initial HML. The latter includes all necessary and low-level hardware architectural details (w.r.t. processing units PUs, memory units, MUs, communication units, CUs). Such models are supported by some C++ libraries that extend the standard SystemC library in order to implement CSP.

The HEPSYCODE System-Level Modeling Language describes the application as a Process Network connected via synchronous channels. Our reference language is SystemC, a C++ class library able to capture and define system specifications. The SBM is implemented by SystemC modules and threads. The Technology Library (TL) defines the basic HW components available to build the final HW platform based on the selected target template architecture. The final HW platform is composed of several basic HW components. These components are collected into the TL, which can be considered as a generic “database” that provides the characterization of the available processor technologies.

Download

HEPSYCODE Eclipse plugins:

<https://bitbucket.org/vittorianomuttillio87/tool-hepsycode/src/7de68d499660e2eb865263059533295921e634aa/?at=master>

DETAILED ACTIVITY DESCRIPTION

In the context of the Megamart project, there were three cooperation initiatives between tool providers (Fentiss/University of Valencia and University of L'Aquila, INTECS and University of Cantabria, and University of Cantabria and University of L'Aquila). These initiatives started during the project and are continuing. Here, a more general interoperability overview is presented trying to catch some methodological and semantical integration pattern between tools providers and UC providers. The [D2.5], [D2.6], [D5.1], and [D5.2] describe the main integration concepts and works. These interoperability and integration guidelines and patterns have been derived from a joint work between University of L'Aquila (UAQ) and University of Cantabria (UCAN).

Starting from these activities, the goal of this hackathon is to complete the cooperation between INTECS and University of L'Aquila, proposing an industrial and research challenge oriented to find patterns and/or guidelines to abstract as-much-as-possible the integration and transformation between the different System Engineering tools present into the Megamart2 catalog [Catalog]. The main megamart2 toolset is divided into different packages, as shown in the [Figure A1.1](#).

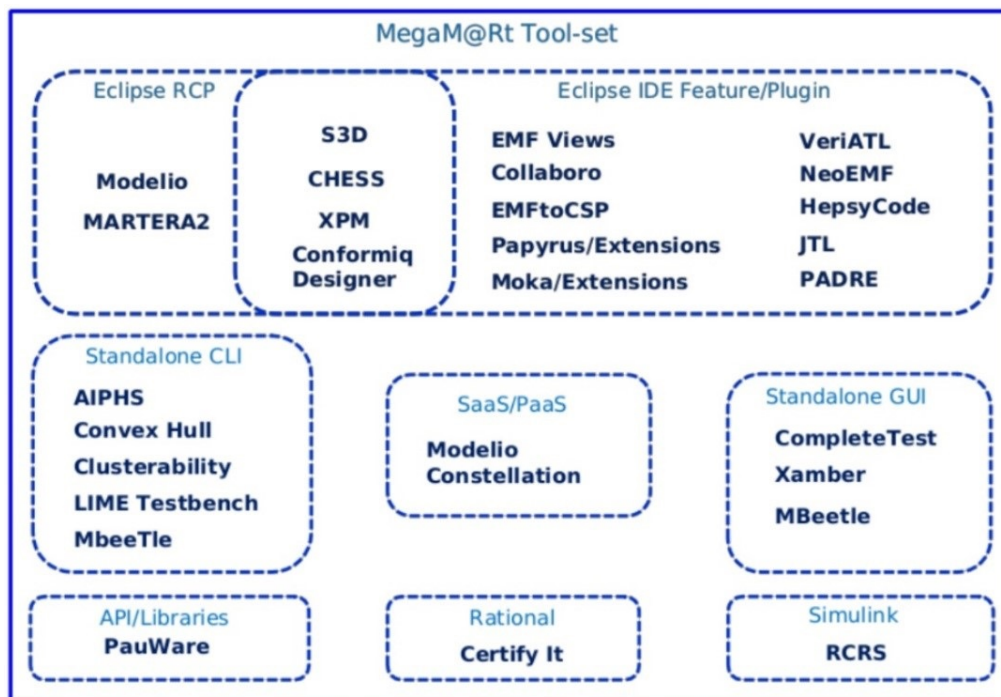


Figure A1.1 Clustering of MegaM@Rt tools by packaging/delivery and interface technology

Starting from this clustering activity, we want to focalize the work only for the Eclipse-based tools (around the EMF framework, a custom RPC application, or an extension of UML/MARTE profile). The actual integration pattern between the considered tools is shown in [Figure A1.2](#) (as reported in D2.3, D2.4, D2.5 and D2.6 deliverables).

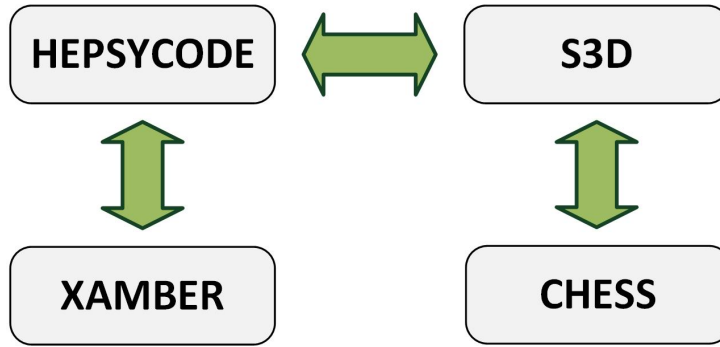


Figure A1.2 Actual Transformation Metamodel

The ideal integration pattern could consider that tools centered on the same modeling technology have a similar (or identical) metamodel at a higher level of abstraction. Therefore, commonalities and differences may be identified between the several tools, offering a possible transformation pattern useful to compare and use tools as an entire common mega-modeling environment. Each tool offers its own approach, and also different possible solutions, but it should be possible to choose or compare at a higher level of abstraction the different modeling activities, as shown in [Figure A1.3](#).

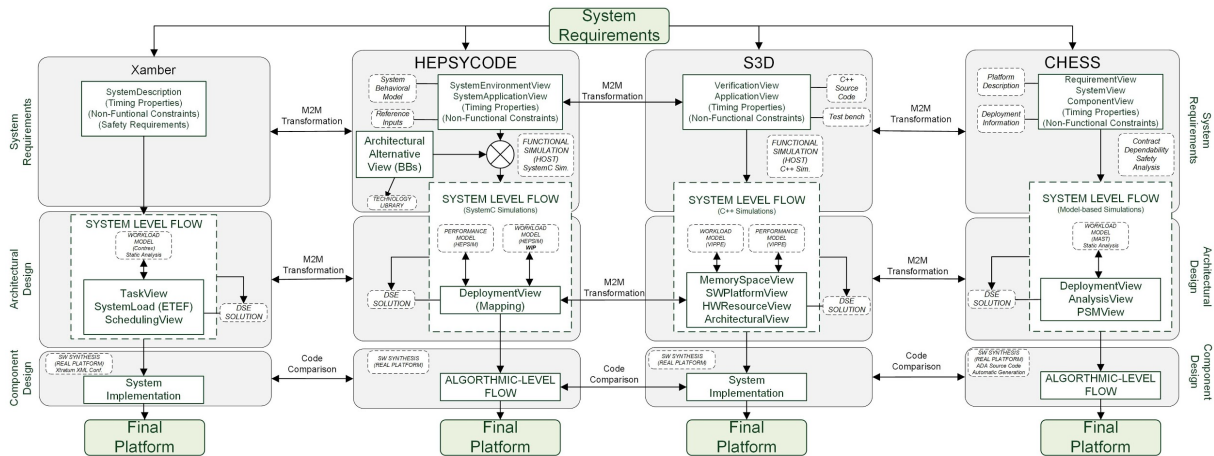


Figure A1.3 Tools Higher Abstraction Layer Transformation Pattern

The main idea is to clustering the tools in several manners, respect to different parameters. One initial idea could be the [Figure A1.4](#), where the considered tools are located in clustering respect to their model of computation

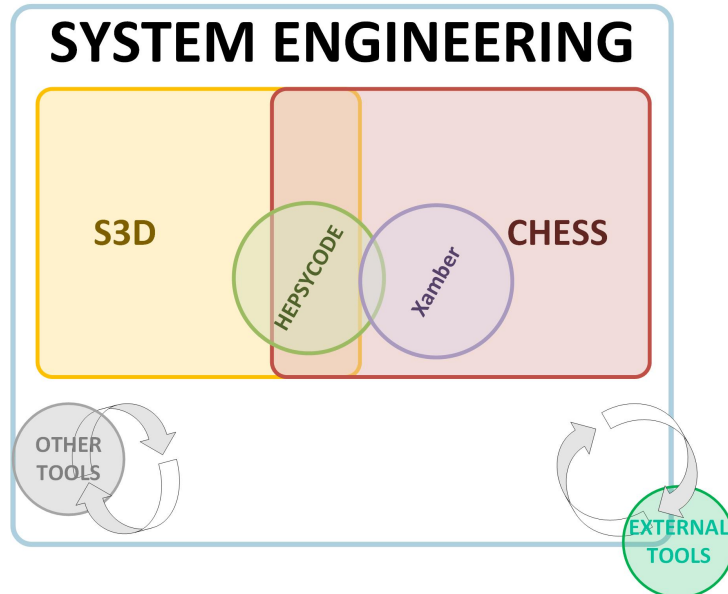


Figure A1.4 MegaM@Rt2 Clustering Approach Considering Model of Computations

Finally, the main integration pattern could arrive to define rules and patterns useful as a model exchange format, and possible implementation of this transformation using Eclipse technologies can validate the results, as shown in [Figure A1.5](#).

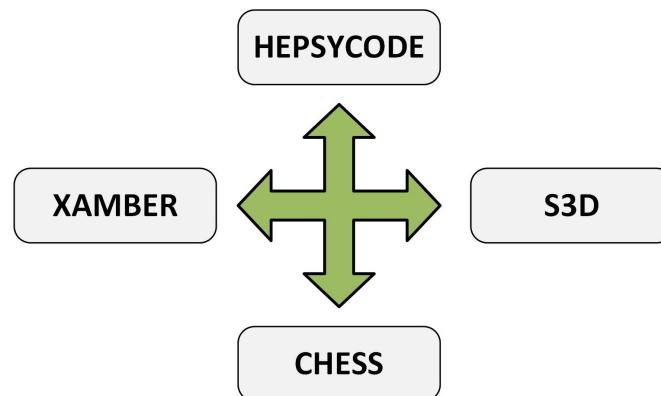


Figure A1.5 Final Transformation Metamodel

The main goal is to center the hackathon on two reference tools (HEPSYCODE and CHESS) and try to exploit these integration activities with a multidisciplinary team able to share, talk and analyze the tools offering their knowledge in several domains, and focus on the main important aspect that it is possible to consider at a higher abstraction layer. A starting point can be the [Figure A1.6](#), that shows an overview of the different modeling and design flow activities made by HEPSYCODE and CHESS.

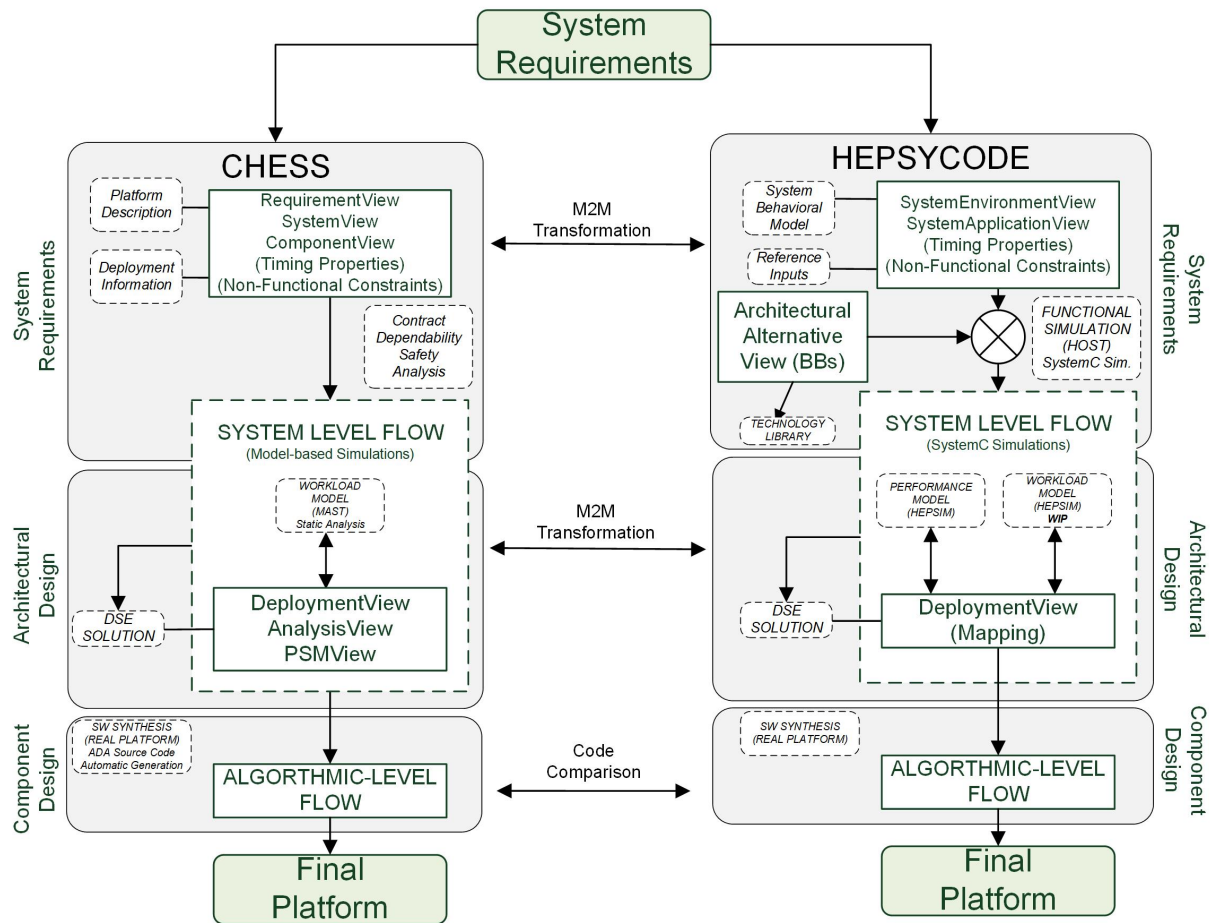


Figure A1.6 CHES - HEPHYCODE Transformation and Integration Pattern

The different Hackathon results can be:

- The produced code (plugins into the eclipse environment) and the related documentation
- A report about commonalities and differences between tools, trying to find as-much-as-possible metrics or parameters useful to compare metamodels, models and design activities
- Possible clustering graphs and/or tables to classify the tools with respect to several aspects (metamodels, MoC, artifacts, inputs, modeling languages, etc.)
- Metamodels or interoperability patterns applicable to the reference tools
- Evolution and/or refinement regarding the presented graphs (the figures above) and/or custom flowchart, activity diagrams, sequence diagrams, general UML diagrams, etc.
- Further and future works, also considering specific industrial or academic domains

PARTICIPANTS

- Proposer: INTECS (INT) and University of L'Aquila (UAQ)
- Implementer: Netherlands Organisation for applied scientific research TNO

HACKATHON OUTLINE

First part: Challenge presentation and tool tutorials. INTECS and UNIVAQ presented the megamart2 hackathon challenge to the participants. INTECS explains CHES methodology and tool, presenting a simple case study and focalize on modeling activity and schedulability analysis. UNIVAQ presented

HEPSYCODE tool, explaining metamodel semantics and syntax and model representation, with an overview of the entire HW/SW Co-Design Flow

Second Part: Preliminary discussion. The hackathon participants share information about their background, knowledge, and related works. Julio and Michiel share information about **CERBERO** project (*Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid enviroNments*), an H2020 European project that would provide a model-based methodology and toolset for design, incremental prototyping, verification and continuous developments of adaptive CPS. The main discussion was focused on:

- Metrics and Parameters useful for automatically generating transformation patterns: since tools can be similar in different design activity, the main idea was to find interoperability patterns at a higher level. This is a huge research question, and an open research problem. We arrive to say that it is not possible to consider (at this research point) a mega-modeling infrastructure able to offer and find automatically model transformation between tools since there is an ontological meaning assigned to the different semantics and syntax. This is a huge work, so at this time it is not possible to generalize these aspects, so the work switches to tool transformation using user experiences and metamodel transformation rules.
- Julio and Michiel propose the use of **CIF (CERBERO Interoperability Framework, <https://www.cerbero-h2020.eu/toolchain/cif/>)** as a solution for the proposed challenge. The main idea is to share the CHESSE and HEPSYCODE metamodel and a simple example for both tools, then they will produce the transformation using CIF.

RESULTS

The MegaMart2 Hackathon in the HiPEAC 2020 put forward the problem of exchanging model files in different formats between tools, and the solution was to convert models from the HEPSYCODE tool to the CHESSE tool. That entails converting ECore models into some form of UML profiles. HEPSYCODE tool uses some form schema defined EMF(EMF), while CHESSE uses some UML dialect, as shown in [Figure A1.7](#) and [Figure A1.8](#).

```
<nodes xsi:type="hml:StructuredNode" name="System">
  <nChannels name="ch2" nFrom="//@nodes.0/@processes.0" nTo="//@nodes.0/@processes.2" queueSize="123">
    <message name="ms2">
      <entry name="mx2" type="sc_uint"/>
    </message>
  </nChannels>
  <nChannels name="ch3" nFrom="//@nodes.0/@processes.2" nTo="//@nodes.0/@processes.1" queueSize="123">
    <message name="ms3">
      <entry name="mx3" type="sc_int"/>
    </message>
  </nChannels>
  <ports name="sy_out" portExtension="//@nodes.0/@processes.0"/>
  <ports name="disp_out"/>
  <processes name="p1" criticality="1"/>
  <processes name="p3" processExtension="//@nodes.0/@ports.1" priority="123" criticality="123"/>
  <processes name="p2" priority="2" criticality="23"/>
</nodes>
<nodes xsi:type="hml:Display" name="Display">...
</nodes>
```

Figure A1.7 HEPSYCODE Ecore Schema.

```

</packagedElement>
<packagedElement xmi:type="uml:Package" xmi:id="_xZBHYDktEeqX7pAFeZus0w" name="modelSystemView">...
</packagedElement>
<packagedElement xmi:type="uml:Package" xmi:id="_xZ3b8DktEeqX7pAFeZus0w" name="modelComponentView">
  <packagedElement xmi:type="uml:Interface" xmi:id="_OGIMEDkuEeqX7pAFeZus0w" name="IProd">...
</packagedElement>
<packagedElement xmi:type="uml:Component" xmi:id="_Stzh0DkuEeqX7pAFeZus0w" name="Producer">
  <ownedAttribute xmi:type="uml:Port" xmi:id="_kRez8DkuEeqX7pAFeZus0w" name="ClientServerPort1" ag
  <ownedAttribute xmi:type="uml:Port" xmi:id="_o2nMADkuEeqX7pAFeZus0w" name="ClientServerPort2" ag
  <ownedOperation xmi:type="uml:Operation" xmi:id="_U4yLoDkuEeqX7pAFeZus0w" name="produce"/>
</packagedElement>
<packagedElement xmi:type="uml:Interface" xmi:id="_XEEXcDkuEeqX7pAFeZus0w" name="ICons">
  <ownedOperation xmi:type="uml:Operation" xmi:id="_cnDe4DkuEeqX7pAFeZus0w" name="consume"/>
</packagedElement>
<packagedElement xmi:type="uml:Component" xmi:id="_Y5BG4DkuEeqX7pAFeZus0w" name="Consumer">
  <ownedAttribute xmi:type="uml:Port" xmi:id="_vBDVADkuEeqX7pAFeZus0w" name="ClientServerPort1" ag
  <ownedOperation xmi:type="uml:Operation" xmi:id="_eqKgDkuEeqX7pAFeZus0w" name="consume"/>
</packagedElement>
<packagedElement xmi:type="uml:Component" xmi:id="_84PHQDkuEeqX7pAFeZus0w" name="SWSsystem">
  <ownedAttribute xmi:type="uml:Property" xmi:id="_JwGKgDkvEeqX7pAFeZus0w" name="producer" type="_
  <ownedAttribute xmi:type="uml:Property" xmi:id="_Qo_U8DkvEeqX7pAFeZus0w" name="consumer" type="_
  <ownedConnector xmi:type="uml:Connector" xmi:id="_VT02gDkvEeqX7pAFeZus0w" name="Connector1">
    <end xmi:type="uml:ConnectorEnd" xmi:id="_VTYngDkvEeqX7pAFeZus0w" partWithPort="_JwGKgDkvEeqX7
    <end xmi:type="uml:ConnectorEnd" xmi:id="_VTZ1oDkvEeqX7pAFeZus0w" partWithPort="_Qo_U8DkvEeqX7
  </ownedConnector>
</packagedElement>

```

Figure A1.8 CHESS Metamodel Schema.

The direct interchange of models via their output files is incompatible and cannot be done. The output files must be interpreted and transformed to express the model language required by the other tool. This leads to major interoperability problems between tools. The goal assignment was to come with possible solutions for this model transformation so to enable the interoperability between HEPSYCODE and CHESS.

As participants, we proposed to achieve (part of) this goal by using the **CERBERO Interoperability Framework tool (CIF)** which TNO is developing.

In this appendix, we demonstrate the use of CIF to transform a small part of the models from one tool format to another. This demo hints on how to proceed in the future when the conversion of modules is required.

The Proposed Solution

Julio de Oliveira Filho and Michiel van den Baar – both from the Dutch company TNO – participated in the Hackathon representing the EU project **CERBERO** (<https://www.cerbero-h2020.eu/>). The project **CERBERO** is developing an interoperability framework called **CIF** or **CERBERO Interoperability Framework**. CIF is targeted in solving the transformation of schema-defined file formats in an intuitive, descriptive way, and without the need of coding model transformation routines.

The proposed solution in this appendix uses CIF to transform part of the models from HEPSYCODE native format to CHESS native format. Concurrently, it **demonstrates the added value of CIF on facilitating tool interoperability and shows its principles**. [Figure A1.9](#) shows how it works.

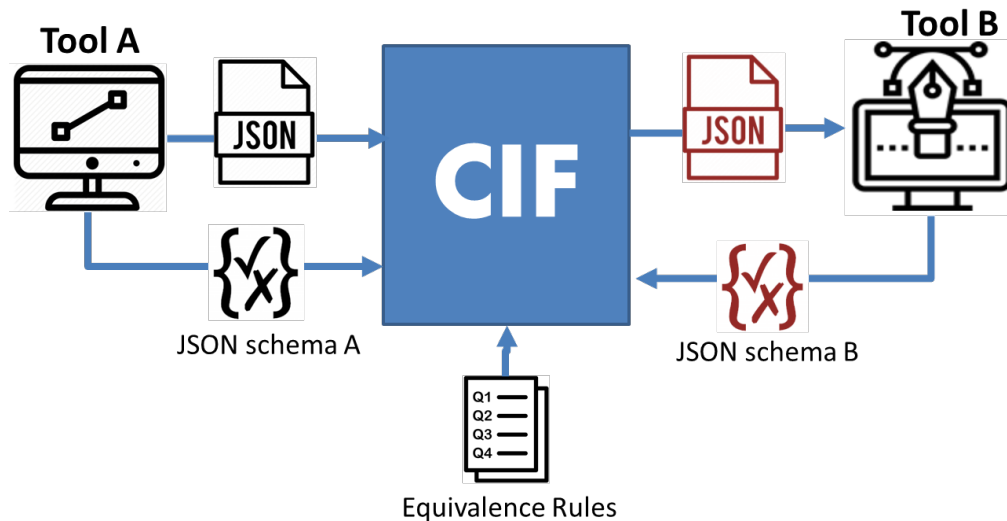


Figure A1.9 CERBERO Interoperability Framework.

For running the model transformation we need to feed CIF with:

1. The output file of the HEPHYCODE tool – conforming to the HEPHYCODE file schema definition.
2. Schema definitions describing each tool file format.
3. A declarative equivalence file written as CIF rules (examples later)

The declarative rule file establishes relationships between the concepts and relationships of the two schema definitions. CIF is then able to transform the model data structures from HEPHYCODE, so that they can be effectively written into the CHESS format.

In the next section we will show examples and discuss each one of these files and the resulting transformed output. All the used files are included as Technical Annex.

As a final remark, in this report we are not concerned in transforming FULL HEPHYCODE models into CHESS format. This is an intensive task that requires extra time understanding both models in detail and some features that are not yet available in the current version of CIF. **This report uses a very small portion of the models – the functional (component) view – to demonstrate the principle.**

FROM HEPHYCODE TO CHESS WITH CIF

Preparing Model and Schema files

Models and schema definition files from both HEPHYCODE and CHESS were provided. They are both formatted in XML but following different schemas: HEPHYCODE is based on the EMF – Ecore metamodel, and CHESS on a UML (Marte/SysML) profile.

These models cannot be read directly by CIF at the current stage. When starting the development of CIF we adopted models and schemas in a JSON format – we had to start somewhere. Consequently, we had to manually translate the model and schema files into a JSON format. Note that this is not a necessary step when CIF is able to read XML files and schemas directly! This feature is planned in our development roadmap, and concrete examples such as the HEPHYCODE - CHESS scenario helps us to understand which formats to support first. So, in the future, this preparation phase will not be necessary.

We reproduced the HEPHYCODE model below in a JSON format. We also produced the required JSON schemas, as shown in [Figure A1.10](#) and [Figure A1.11](#) (see Technical Annex for full files hepsy.ecore and schema_hepsy_*.json files).

```

<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w
  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="hepsy" nsURI="org.univaq.hepsy
<eClassifiers xsi:type="ecore:EClass" name="BehaviorSpecification" eSuperTypes="#//NamedEl
  <eStructuralFeatures xsi:type="ecore:EReference" name="nodes" upperBound="1"
    eType="#//Node" containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Node" abstract="true" eSuperTypes="#//NamedEle
  <eStructuralFeatures xsi:type="ecore:EReference" name="nChannels" upperBound="1"
    eType="#//Channel" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="ports" upperBound="1"
    eType="#//Port" containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Channel" eSuperTypes="#//NamedElement">
  <eStructuralFeatures xsi:type="ecore:EReference" name="nFrom" lowerBound="1" eType="#//N
  <eStructuralFeatures xsi:type="ecore:EReference" name="nTo" lowerBound="1" eType="#//Nod
  <eStructuralFeatures xsi:type="ecore:EReference" name="pFrom" lowerBound="1" eType="#//P
  <eStructuralFeatures xsi:type="ecore:EReference" name="pTo" lowerBound="1" eType="#//Por
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="queueSize" eType="ecore:EDataType
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="timeout" eType="ecore:EDataType
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="rendezVous" eType="ecore:EDataTyp
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="direction" eType="ecore:EDataType
  <eStructuralFeatures xsi:type="ecore:EReference" name="message" eType="#//Message"
    containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="NamedElement">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="StructuredNode" eSuperTypes="#//Node">
  <eStructuralFeatures xsi:type="ecore:EReference" name="processes" upperBound="1"
    eType="#//Process" containment="true"/>
  }
  "namespace": "hepsy",
  "class": "system",
  "schema": {
    "properties": [
      {
        "name": "nodes",
        "optional": false,
        "collection": "set",
        "set": true,
        "value": {
          "type": "object",
          "optional": false,
          "collection": "set",
          "constrains": [],
          "default": null,
          "object": {
            "namespace": "hepsy",
            "class": "node",
            "extensible": false,
            "schema": "schema_hepsy_node",
            "id_type": "object"
          }
        }
      }
    ]
  }
}

```

Figure A1.10 HEPHYCODE schema as ECore file and its JSON equivalent (top part only to see).

```

<?xml version="1.0" encoding="UTF-8"?>
<hml:BehaviorSpecification xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XM
  (nodes xsi:type="hml:StructureNode" name="System")
  <nChannels name="ch2" nFrom="//@nodes.0/@processes.0" nTo="//@nodes.0/@processes.2" queueSize="123" timeout="123"
    <message name="ms2">
      <entry name="mx2" type="sc_uint"/>
    </message>
  </nChannels>
  <nChannels name="ch3" nFrom="//@nodes.0/@processes.2" nTo="//@nodes.0/@processes.1" queueSize="123" timeout="123"
    <message name="ms3">
      <entry name="mx3" type="sc_int"/>
    </message>
  </nChannels>
  <ports name="sy_out" portExtension="//@nodes.0/@processes.0"/>
  <ports name="disp_out"/>
  <processes name="p1" criticality="1"/>
  <processes name="p3" processExtension="//@nodes.0/@ports.1" priority="123" criticality="123"/>
  <processes name="p2" priority="2" criticality="23"/>
</nodes>
<nodes xsi:type="hml:Display" name="Display"/>...
</nodes>
<nodes xsi:type="hml:Stimulus" name="Stimulus"/>...
</nodes>
</hml:BehaviorSpecification>
  {
    "nodes": [
      {
        "name": "System",
        "nChannels": [
          {
            "name": "ch2",
            "nFrom": {
              "name": "p1",
              "criticality": "1"
            },
            "nTo": {
              "name": "p2",
              "priority": "2",
              "criticality": "23"
            },
            "queueSize": "123",
            "timeout": "123",
            "rendezVous": "true",
            "message": {
              "name": "ms2",
              "entry": {
                "name": "mx2",
                "type": "sc_uint"
              }
            }
          },
          {
            "name": "ch3",

```

Figure A1.11 HEPHYCODE model (example) and its JSON format equivalent.

In the re-writing into JSON, we made our best to keep the same data structure and semantic. This transformation is only syntactic and has no severe implications on the semantic organization. In other words, this is just a format re-writing – no changes in the semantics were made.

Equivalently, we also generated a schema for the component view in CHESS as shown in [Figure A1.12](#).

```

<packagedElement xmi:type="uml:Package" xmi:id="_xZ3b80ktEqK7pAFeZus0w" name="modelComponentView">
  <packagedElement xmi:type="uml:Interface" xmi:id="_OG7MEDkuEqK7pAFeZus0w" name="IProd">...
  <packagedElement
    <packagedElement xmi:type="uml:Component" xmi:id="_Stzh80kuEqK7pAFeZus0w" name="Producer">
      <ownedAttribute xmi:type="uml:Port" xmi:id="_kRez80kuEqK7pAFeZus0w" name="ClientServerPort1" aggregator
      <ownedAttribute xmi:type="uml:Port" xmi:id="_o2nVADkuEqK7pAFeZus0w" name="ClientServerPort2" aggregator
      <ownedOperation xmi:type="uml:Operation" xmi:id="_U4yLoDkuEqK7pAFeZus0w" name="produce"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Interface" xmi:id="_XEEXc0kuEqK7pAFeZus0w" name="ICons">...
    </packagedElement>
    <packagedElement xmi:type="uml:Component" xmi:id="_Y5B640kuEqK7pAFeZus0w" name="Consumer">
      <ownedAttribute xmi:type="uml:Port" xmi:id="_v8DVADkuEqK7pAFeZus0w" name="ClientServerPort1" aggregator
      <ownedOperation xmi:type="uml:Operation" xmi:id="_eaqgDkuEqK7pAFeZus0w" name="consume"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Component" xmi:id="_84PH00kuEqK7pAFeZus0w" name="SmsSystem">
      <ownedAttribute xmi:type="uml:Property" xmi:id="_JwGkgDkuEqK7pAFeZus0w" name="producer" type="_Stzh80ku
      <ownedAttribute xmi:type="uml:Property" xmi:id="_Qo_U80kvEqK7pAFeZus0w" name="consumer" type="_Y5B640ku
      <ownedConnector xmi:type="uml:Connector" xmi:id="_VT02gDkuEqK7pAFeZus0w" name="Connector1">
        <end xmi:type="uml:ConnectorEnd" xmi:id="_VTYngDkuEqK7pAFeZus0w" partWithPort="_JwGkgDkuEqK7pAFeZus0
        <end xmi:type="uml:ConnectorEnd" xmi:id="_VTZ1oDkuEqK7pAFeZus0w" partWithPort="_Qo_U80kvEqK7pAFeZus0
      </ownedConnector>
    </packagedElement>
  </profileApplication xmi:type="uml:ProfileApplication" xmi:id="_xaQdgDktEqK7pAFeZus0w">
    <eAnnotations xmi:type="ecore:EAnnotation" xmi:id="_xaSSsDktEqK7pAFeZus0w" source="http://www.eclipse.or
    <references xmi:type="ecore:EPackage" href="http://CHESS#/ComponentModel"/>
  </profileApplication>
</packagedElement>

```

```

{
  "namespace": "chess",
  "class": "package_description",
  "schema": {
    "properties": [
      {
        "name": "xmi:type",
        "optional": false,
        "set": false,
        "value": {
          "type": "str",
          "optional": false,
          "collection": null,
          "constrains": [],
          "default": null,
          "object": null
        }
      },
      {
        "name": "xmi:id",
        "optional": false,
        "set": false,
        "value": {
          "type": "str",
          "optional": false,
          "collection": null,
          "constrains": [],
          "default": null,
          "object": null
        }
      }
    ]
  }
}

```

Figure A1.12 Chess model (example) and extracted schema in JSON to represent it.

Notice that there is no CHESS model translated. This is the goal of the exercise. If all works out, we should have a CHESS model representing the same HEPHYCODE model as an output of the CIF module. From this point on, we follow with the JSON files. XML files are kept only for reference and checks.

After this translation we have three inputs for the CIF:

1. HEPHYCODE schema definition files in JSON: these are in Technical Annex with name *schema_hepsy_XXX.json*
2. One HEPHYCODE model as in the example, but defined in JSON and following the JSON schema: this is the *hepsy_model.json* file in the Technical Annex.
3. CHESS schema definition files in JSON: these are in Technical Annex with name *chess_XXX_schema.json*

We need further an equivalence rule file.

The Equivalence Rules File

In CIF, the Equivalence Rules is a file containing equivalences between two or more schema definitions. It says things such as "...every SYSTEM in HEPHYCODE is a PACKAGE in Chess" or "... the collection pointed by NODE properties in HEPHYCODE is the collection pointed by COMPONENTS in chess". We do that following a concise declarative rules language (proprietary in CIF).

Such language makes declarations on data equivalence, and they are NOT a procedural way to transform the data. In other words, there is no need to code a model transformation routine. Writing the translation rules in CIF is much easier and straightforward in that sense.

In [Figure A1.13](#), you see the picture of the rules we wrote for the transformations in this exercise.

```

hepsy:system == chess:package_description
    IMPLYING
        (hepsy:system.nodes == chess:package_description.components);
hepsy:node == chess:component_description
    IMPLYING
        (hepsy:node.name == chess:component_description.name,
         hepsy:node.ports == chess:component_description.ports);
hepsy:port == chess:port_description
    IMPLYING
        (hepsy:port.name == chess:port_description.name);

```

Figure A1.13 CHESS - Hepsycode Equivalence Roles.

These rules are in file *example.rules* in the Technical Annex.

Note that we do not write rules for all the necessary data transformations in this example. There are two reasons for that:

1. We want to keep it simple as this is just a demonstration of principle
2. Some of the required transformations in this example need rules that are not yet available in the current development stage of CIF. We value this exercise as it points out the concrete need for more advanced rules and it prioritizes our development.
 - a. For example, it is currently not possible to make a proper equivalence description between the channels of HEPHYCODE and the connectors in CHESS. Reason for that is that HEPHYCODE declares its channels inside each component, while CHESS declares the connectors inside a structured component where other components are embedded in as well. This difference in “levels” must be described in rules with different scopes – this feature is planned, but not yet available.

Now we have the last element: a rules file describing (part) of our desired transformation.

FINAL SOLUTION

We now register the schemas in CIF and run the model transformation (script included in Technical Annex file *run-hepsy_to_chess.py*) according to [Figure A1.14](#).

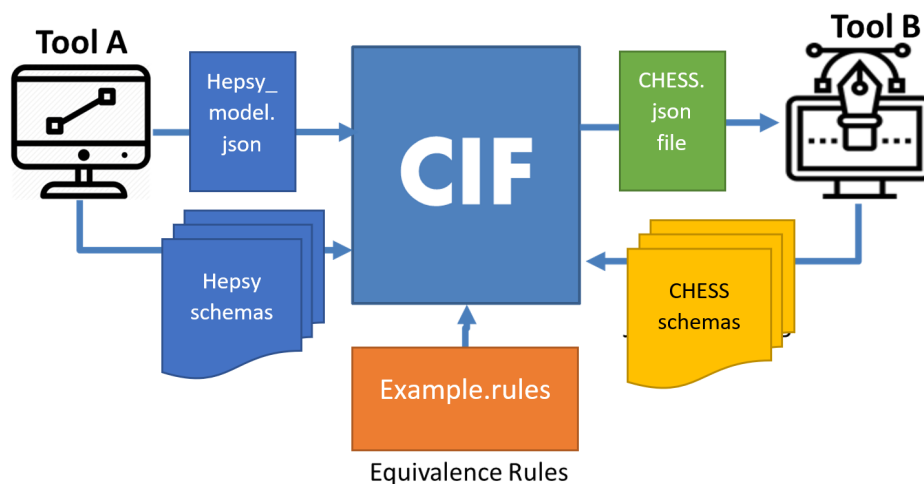


Figure A1.14 Megamart2 Hackathon Final Result using CIF.

The produced file is as [Figure A1.15](#) (in Technical Annex *output_of_cif.json*).

```
"xmi:type": null,  
"xmi:id": null,  
"name": null,  
"components": [  
  {  
    "xmi:type": null,  
    "xmi:id": null,  
    "name": "Display",  
    "ports": [  
      {  
        "xmi:type": null,  
        "xmi:id": null,  
        "name": "disp_in",  
        "aggregation": null  
      }  
    ]  
  },  
  {  
    "xmi:type": null,  
    "xmi:id": null,  
    "name": "Stimulus",  
    "ports": [  
      {  
        "xmi:type": null,  
        "xmi:id": null,  
        "name": "sy_in",  
        "aggregation": null  
      }  
    ]  
  }  
]
```

Figure A1.15: HEPHYCODE - CHESS Transformation using CIF.

The output file produced has the correct structure. That is, it contains all the necessary information fields to be read by CHESS. Two remarks should be made.

First, some field values are still marked as null in value. Reason for that is that this information is not available in the HEPHYCODE model. Example, the *xmi:type* tags do not exist in HEPHYCODE, and are CHESS related. They express in the CHESS schemas the types of nodes and could be automatically filled by attributing default values on the CIF classes. This is done in the CHESS schemas. We did not do that, but it is possible already.

Note that for some of these problems, there is no direct solution. Simply not all information required in CHESS is available in HEPHYCODE. *Null* values that are not tolerated when reading the transformed files in CHESS have then to be filled in by the user in a manual or automated way outside the CIF.

Secondly, some elements necessary to make a full-blown, readable CHESS files are not yet in the output model. The two reasons for that are:

- A. We decided not to include all the modelled elements due to time issues
- B. Some elements would require more powerful rules.

To conclude, these results shown that the transformation and interoperability approach is useful in the context of MegaM@Rt2 project to create a connection among tools based on Eclipse technologies, while future work will exploit better these transformation patterns applying the methodology to the whole MegaM@Rt2 Catalog.