Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105

D5.7: CERBERO Framework Demo (Ver. 1)

Lead Beneficiary: AI

Workpackage: WP5

Date: 31/07/2018

Distribution - Confidentiality: [Public]

Abstract: This deliverable describes all the CERBERO framework components, identifying all the parts composing the cross-layer model-based structure for design, optimization, verification and deployment of complex cyber-physical systems and systems of systems. The document presents separately all the components/tools, starting from their motivations and already provided features, going to the extensions envisioned in order to attain CERBERO objectives and to accomplish use case requirements.

Disclaimer

The CERBERO Consortium is the following:

Num.	**Beneficiary name**	**Acronym**	**Country**
1 (Coord.)	IBM Israel – Science and Technology LTD	IBM	IL
2	Università degli Studi di Sassari	UniSS	IT
3	Thales Alenia Space Espana, SA	TASE	ES
4	Università degli Studi di Cagliari	UniCA	IT
5	Institut National des Sciences Appliquees de Rennes	INSA	FR
6	Universidad Politecnica de Madrid	UPM	ES
7	Università della Svizzera italiana	USI	CH
8	Abinsula SRL	AI	IT
9	Ambiesense LTD	AS	UK
10	Nederlandse Organisatie Voor Toegepast Natuurwetenschappelijk Ondeerzoek TNO	TNO	NL
11	Science and Technology	S&T	NL
12	Centro Ricerche FIAT	CRF	IT

For the CERBERO Consortium, please see the http://cerbero-h2020.eu web-site.

Document Authors

The following list of authors reflects the major contribution to the writing of the document.

Name(s)	**Organization Acronym**
Antonio Solinas	AI
Giuseppe Meloni	AI
Maria Katiuscia Zedda	AI
Tiziana Fanni	UniCA
Carlo Sau	UniCA
Francesca Palumbo	UniSS
Claudio Rubattu	UniSS
Alfonso Rodriguez	UPM
Daniel Madroñal	UPM
Evgeny Shindin	IBM
Michael Masin	IBM
Karol Desnos	INSA
Julio De Oliveira Filho	TNO

Document Revision History

Date	**Ver.**	**Contributor (Beneficiary)**	**Summary of main changes**
06/06/2018	0.0	AI	initial draft
22/06/2018	0.0	UNISS, UNICA, UPM	description of integration and demonstration activities related to the lower part of the CERBERO framework
29/06/2018	0.5	IBM	Integration of UNISS, UNICA, UPM, and IBM contributions
19/07/2018	1.0	AI	AI contribution
19/07/2018	1.0	UNISS	Review
23/07/2018	1.0	UPM	Review
24/07/2018	1.0	UniSS	Review

24/07/2018	2.0	AI	Implementation of Review Comments
31/07/2018	3.0	IBM	Review of Section 3, Appendix I

Table of contents

1. Executive Summary

This document presents a short description of the main Proofs of Concept (PoCs) of the integrated CERBERO development framework, which is basically a design environment for Cyber-Physical Systems (CPSs) based on a cross-layer model-based approach and on an advanced adaptivity support.

To develop the framework an incremental methodology has been followed, and it will be used for developing the final cross-layer exploration, design and optimisation platform.

It is important to highlight that this document is the supporting documentation of the software deliverable D5.7. The main scope of this document is to provide the explanation and technical details of the 3 main PoCs that have been developed and tested at M18.

In order to cover and test the two connections strategy that will be used in the final demonstrator, this document provide:

- 1 PoC using CERBERO Intermediate Format (CIF)
- 2 PoCs using the direct connections among couples or series of tools.

Each PoC will be described separately. The main goal of the description is to provide the following information for each of them:

- Purpose of the integration
- Explanation of the technical features of the connection
- Exchanged data
- Explanation of the example that will be used for testing the PoC
- Link to video or any other material considered relevant for emphasising the main PoC achievements.

Therefore, the mission of this document is neither to describe the components/tools that are integrated, nor their standalone use; for that information please refer to D5.6.

1.1. Structure of the Document

In Section 2 a general overview of the CERBERO development framework and its current state of development is given with a short explanation of the integration process that will follow to develop its final version. In Section 3 a description of the approach for Semantic Integration is presented together with a comprehensive explanation of the PoC using CIF. Finally, Section 4 is dedicated to present two PoCs developing direct connections among tools.

1.2. Related Documents

The CERBERO deliverables related to this document are:

- D2.7 – CERBERO Technical Requirements

  - The activities behind D5.7 contribute to satisfy the requirements listed in D2.7. Details are given in Section 1.3.
- D3.6 – Cross-layer Modelling Methodology for CPS
  - D3.6 provides methodological foundation for CERBERO Intermediate Format.
- D5.4 – CERBERO Holistic Methodology and Integration Interfaces
  - D5.4 presents the design framework integration approach and the required Interfaces.
- D5.6 – CERBERO Framework component
  - In D5.6 the technical details of the different components/features of the CERBERO design environment are reported.

1.3. Related CERBERO Requirements

Deliverable D2.7 of the CERBERO project defines a list of CERBERO Technical Requirements (CTRs) the project should achieve. Each of them is referenced with a unique identifier ranging from 0001 to 0020. The CERBERO framework Demo described in the current document address 4 CTRs, as described in the following table. It is important to note that most of the requirements related to the framework are covered by the tools integrated in the framework and are not reported in the following table.

CTR id	**CTR Description**	**Link with the D5.7 document on CERBERO framework components**
0002	CERBERO framework SHOULD provide interoperability between cross-layer tools and semantics at the same level of abstraction.	The semantic integration at the same level of abstraction and the interoperability between cross-layer tool is demonstrated and tested with the PoC that connects AOW, DynAA and PREESM using the CIF.
0004	CERBERO framework SHOULD provide software and system in-the-loop simulation capabilities for HW/SW co-design and System Level Design.	System in-the-loop simulation capabilities have been achieved by the integration of DynAA with MECA with the SCANeR simulator. Extensive description is provided in D6.10 since it has already been used in the use case demonstrator of the Electric Vehicle.
0005	CERBERO framework SHOULD provide multi-viewpoint multi-objective correct-by-construction high-level architecture.	The possibility of providing a multi-viewpoint, multi-objective and correct-by-construction high-level architecture has been guaranteed by the interconnection of AOW, DynAA, PREESM, demonstrated in the PoC of the CIF.
0009	CERBERO SHALL develop integration methodology and framework.	The three PoCs presented and developed in this deliverable are the base for the final development of the framework.

2. The CERBERO Framework Integration

A design environment for CPSs, in general, should be an integrated platform or tool chain that can be broken down into various interacting components serving the needs of the different physical and computational elements or subsystems across different layers. Appropriate software components (*a.k.a*. the design environment or framework components) are required to be inter-linked to form a holistic operational framework following design requirements and seeking a new foundation for CPS design, integration and operation. One of the goals of CERBERO is to deliver a semantic integration framework that is customizable per application scenario or use case, yet generalizable enough to a broad range of application domains.

Integration aims at interconnecting the components together, in a layered fashion, to facilitate exchange of information and control data between these components or subsystems and assuring that the integrated system meets performance and behavioural expectations.

2.1. Overview of CERBERO tools connections

An overview of the CERBERO framework is depicted in Figure 2-1. The infrastructure has evolved with respect to previous deliverables. Based on current developments, it seems that the connections among SAGE VS (originally VT tool), PREESM, SPiDER and PAPIFY to the lower-level components could be similar. To this regard please refer to D4.4, which provides an example related to PAPIFY, MDC and ARTICo^3 (still ongoing so it will not be part of the present assessment).

As it can be noted, the connections among tools have already been identified as the integration methodology, which can be either direct or based on the usage of the CIF.

Furthermore, in Figure 2-1 the connections and their current state are depicted.

1. In blue: already existing connections.
2. In green: implemented connections already assessed at M18.
3. In yellow: connections currently under development. Please note that among these, PREESM, AOW and DynAA connection is still indicated. Section 3 describes how this connection is about to turn green.
4. In red: planned connections.

In the following sections we present a brief explanation of three PoCs that have already been developed.

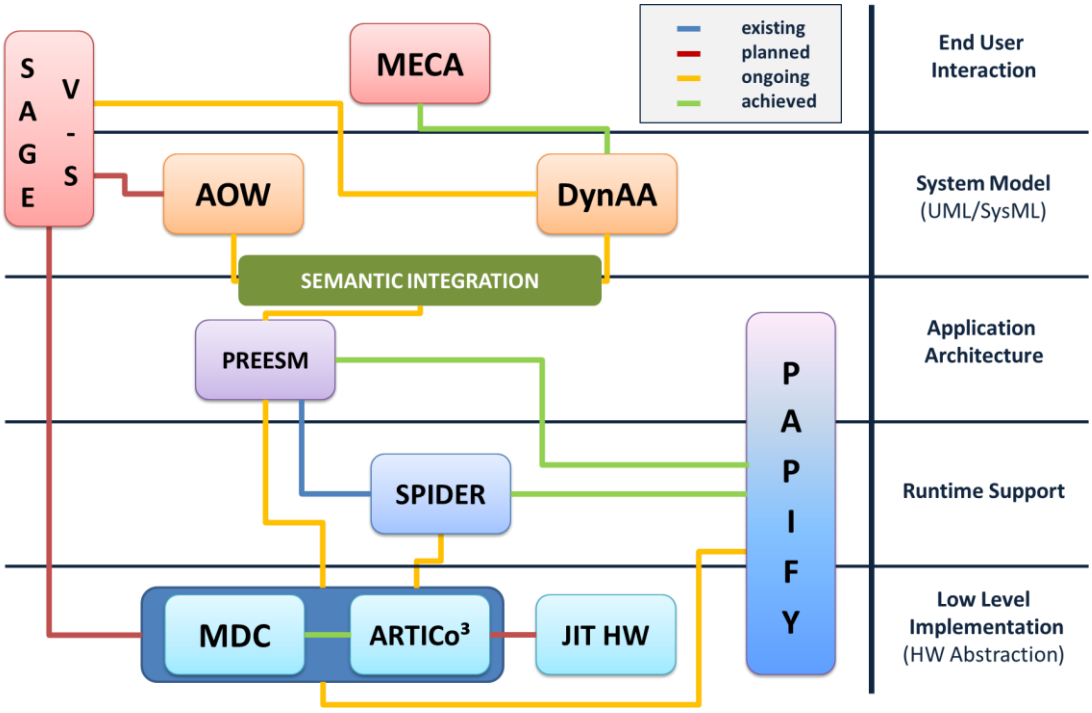

Figure 2-1: Overview of the CERBERO framework components.

2.2. The Integration processes

D5.4 has already fully explained the CERBERO design framework integration approach, with required interfaces among all the CERBERO tools across all layers of the toolchain (model, application, runtime, and hardware layers).

For the first phase of the CERBERO project, using a continuous integration methodology three PoCs have been developed for testing a first version of the CIF and the direct connection. This way, we succeeded in developing and testing:

- connections among tools at the same layer, such as MDC with Artico3 and AOW with DynAA
- cross layer connections as MECA with DynAA (described in D6,10), AOW and DynAA with PREESM (through the semantic integration) and PREESM and SPiDER with PAPIFY (though a direct connection)
- the connection of the CERBERO framework with external tools, like MDC with CAPH.

3. Intermediate Format Connections

3.1. CERBERO Innovative Approach for Semantic Integration

Information modelling underlies representing or formatting information in a certain way to guarantee its uniformity and consistency. A *meta-model* defines (i) the concepts or information that can be present in a model and that can be accessed and manipulated by different tools, and (ii) the rules that regulate accesses to the information. However, sticking with a single meta-model for the entire information model does not come without a problem, such as: multi-view interoperability, multi-tool interoperability, and model maintenance (information models compliance to the meta-model).

Hence, strict coupling of information model and meta-model poses interoperability and maintenance concerns. CERBERO consortium attempts to improve the state-of-the-art of information modelling and semantic integration, particularly when dealing with multi-view cross-layer designs. In this sense, CERBERO proposes an approach to decouple the model information from the meta-model by model's intermediate format (a.k.a. intermediate representation) meeting the following requirements:

1. Can be used efficiently for sharing information across different levels of abstraction and different modelling aspects (views). In other words, an intermediate format should fully exploit the idea of one-model-with-multiple-views representation of the system.
   Rationale: The modelling of CPS is intrinsically multi-disciplinary, multi-aspect, and involves different abstraction layers. Any unique model representation for the system that cannot cope with these intrinsic characteristics is doomed to fail. The model information should be equally adequate and accessible to the different tools manipulating the model for the representation of several aspects (modelling, analysis, code-generation, runtime management, validation), and for manipulation at different abstraction levels.
2. Allows different tools to access information about a system model with minimally incorporating details of the meta-models used in other tools.
   Rationale: Tools should be able to read, understand, and manipulate the model information without or minimal knowledge on how this information is organized in other tools since it both can be changed without notice and is irrelevant to modelled system.

CERBERO consortium considers that such points are not yet covered coherently and well enough by state-of-the-art work proposed so far in the literature or readily available, see D3.6 – Cross-layer Modelling Methodology for CPS for more discussion. In the following sections we describe our proposal and corresponding Proof of Concept (PoC) study.

3.2. Purpose of Integration with CERBERO Intermediate Format

Integration with CERBERO Intermediate Format (CIF) allows achieving easy exchange of relevant information between all connected tools. Unlike tool-to-tool integration, CIF provides a unified platform for model and data transformation that allows implementing automatic transformation capabilities. Within the CIF framework, meta-models (or schemas) of all input and output data are defined in a declarative way allowing to define transformation process as mapping between corresponding schemas. Such unification allows achieving easy integration of multiple tools having multiple views and/or providing multiple functionality. The integration of new tools become a three-step process where in the first step tool describes its output and input data schemas, in the second step defines mapping of this schemas onto flat CIF representation, and in the third step data transformations are made, when needed, using CIF middleware API. Such construction provides additional benefits for tool developers and integrators: it is not necessary to describe whole data provided by the tool in a case when this data is too complex and full description requires big effort; instead, one can define only schema of data that is necessary for other tools in a scope of an integration goal. Thus, integration with CIF allows achieving data interchange between connected tools without additional software development process and without effort of complex ontological description of whole data.

3.3. Integration Framework Tool-Flow

As CERBERO consortium components/tools and technologies undergo continuous development, CERBERO adopts an iterative integration approach, i.e., continuous and constantly evolving rather than static or fixed. To facilitate components/tools interconnection, interfaces are defined and created as points of interaction between communicating components. Interfacing means using a common message format or intermediate representation to provide a unified communication paradigm across the system, entirely or partially. Translation is required from the interface of one component to the intermediate format and vice versa for bilateral or duplex communication.

CERBERO integration approach considers underlying systems as black boxes, thus creating a middleware to facilitate communication between the integrated components. This CERBERO integration middleware itself is considered a component. The main building blocks of CERBERO framework integration are:

- **Semantic Interface:** as opposed to a programming interface, a semantic interface is a middleware customized for integration of tools across two layers, such as user/model layer and the application/runtime layer. Semantic interfaces are defined using SEMI approach [D3.6], for which the implementation will be provided in Python for the framework demonstrator.
- **IR:** the intermediate representation, also known as intermediate format or middleware, is the semantic format that serves the purpose of structuring system data or information to facilitate its storage and exchange (sharing) for tool

interoperability across two or more layers of the tool chain. Tools span user and modelling, application and runtime, implementation and validation layers. CERBERO consortium has agreed to use IBM SEMI as IR of choice as discussed in D3.6.

- **CIF Models:** high-level models are most naturally represented by graphs. For this purpose, [JanusGraph] or [ThinkerPop/ThinkerGraph] technology will be evaluated, but relational high-performance databases could be used as well. It allows distributed processing of big graphs as well as real-time graph traversal and analytics through efficient queries.
- **Persistence Format:** relates to information storage on disk or database where a model graph or intermediate format of functional model is saved, e.g., as a file representation, such as JSON, wide column or regular relational database. Parsing the persistence format file directly yields the model graph (in intermediate format). CERBERO consortium evaluates Apache [Cassandra] as the data store of choice for its performance and scalability in addition to JSON support.

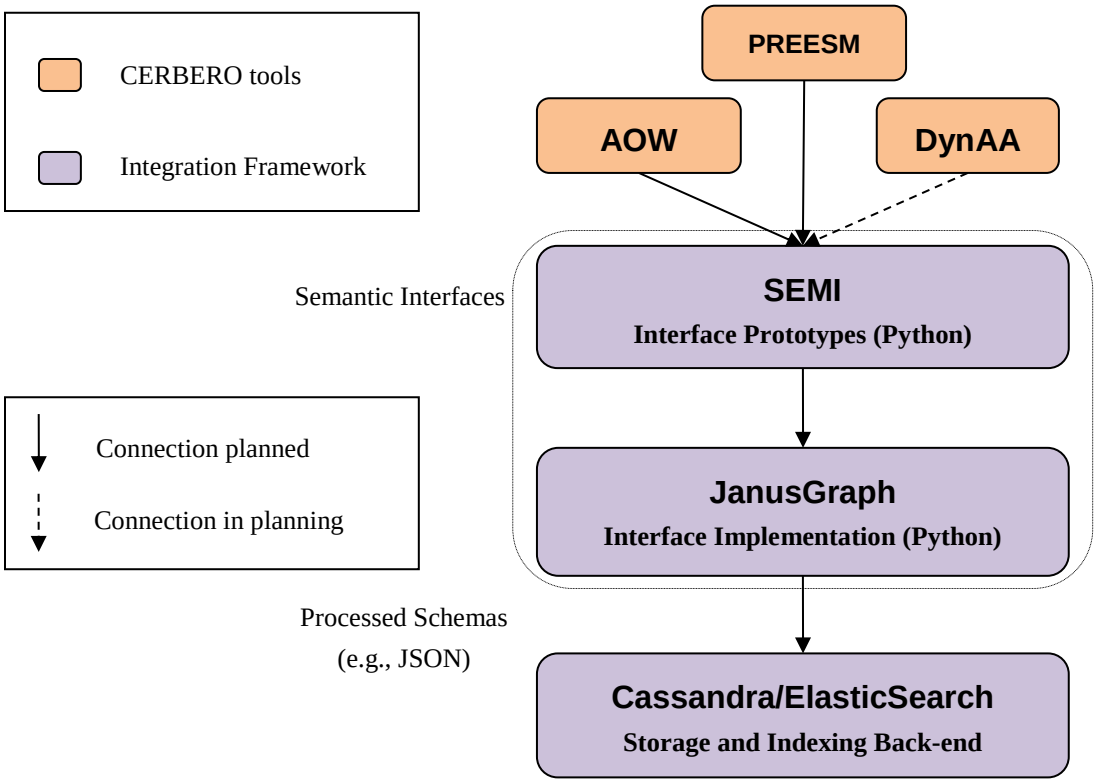

Figure 3-1: Semantic Integration Tool-Flow

Figure 3-1 demonstrations the current CIF architecture within CERBERO tool chain.

3.4. POCs CIF Connection PREESM – AOW – DynAA

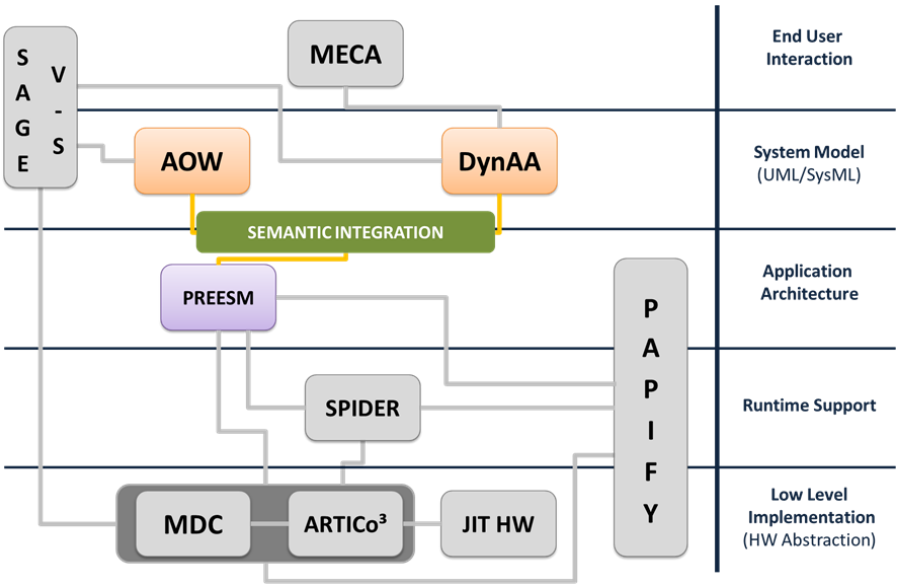

Figure 3-2: CIF PoC

The purpose of the PoC is to calculate optimized scheduling of a software, provided as an SDF graph, on a hardware, provided as a hardware architecture description. The optimization can be performed with respect of several goals, such as minimal latency, maximum throughput, and minimum energy and subject to different constraints, such as computation and memory capacity. In this scenario PREESM takes a role of the service requester while AOW and DynAA take a role of the service providers: (i) software and hardware models described in PREESM passing to AOW, (ii) AOW performs optimization in order to obtain optimized scheduling, which is passed to DynAA, (iii) DynAA performs simulation of the proposed scheduling, updates run-times of software components on the hardware architecture according to the simulation results and pass them back to AOW (in order to perform another optimization run) or back to PREESM (if maximum numbers of iterations achieved or if there are no further updates required).

In order to achieve desired integration PREESM provides following types of data:

- SDF graph in XML format representing software architecture, that also includes additional parameter indicating maximal number of iterations between AOW and DynAA
- Hardware architecture description in XML format
- Possible mapping scenario between software and hardware including estimated execution times of different software actors in different processing units in XML format.

PREESM also defines schemas of each kind of data in agreed JSON format. Once defined, these schemas allow the CIF service to import all these data and convert it to CIF. Finally, PREESM also defines schema of its input data, i.e. format in which resulting scheduling should be provided to PREESM.

From the second integration endpoint AOW provide two different schemas:

- Scheduling analytic schema, i.e. schema of data required to perform calculation of optimal scheduling
- Output format schema, i.e. schema of scheduling data produced by optimization, including current optimization run number and maximal number of iterations obtained from PREESM.

Finally, DynAA endpoint provides:

- Input schema for scheduling
- Output schema of simulation results.

In scope of the PoC, communication between different tools, as well as communication between tools and CIF service, are performed in a straightforward way where results (output) produced by one tool serve as input for another tool. To reduce network communication overhead all tools considered to run on a single Windows machine. The orchestration of execution of overall toolchain performed by Windows batch script allowing verification and demonstration of the integration capabilities without big development/adaptation overhead of corresponding tools. More complex communication procedures requiring adaptation of tools invocation methods are postponed to final stages of the project.

The proposed execution scenario includes the following steps (more details are provided in Appendix I: CIF Example and the PoC data flow is shown in Figure 3-3).

1. Orchestration script receives three parameters: PiSDF graph folder, target HW architecture file in XML format and possible mapping scenario between software actors from PiSDF graph to processing elements in HW architecture file in XML format.
2. Orchestration script invoke PREESM execution that generates a flattened SDF graph from the PiSDF input.
3. When the flattened SDF graph is ready, the orchestration script invokes XML-to-JSON transformation of all input data files.
4. Resulting data in JSON format is sent to the CIF service endpoint invocating data transformation according to corresponding schemas. Each data asset receives unique namespace ID to allow addressing.
5. When data storage completed orchestration script, the script invokes data transformation to AOW format performing call of corresponding transformation procedures. This produce JSON files required for AOW.
6. The orchestration script invokes AOW optimization start providing JSON files of software architecture model, hardware architecture model and possible mappings data in AOW format.
7. AOW performs optimization process and store resulting data as JSON in AOW format.
8. Orchestration script send data to CIF service. Resulting data is converted to CIF and receiving unique namespace ID.

9. When data storage completed orchestration script, the script invokes data transformation to DynAA format performing call of corresponding transformation procedures. This produce JSON files that required for DynAA.
10. Orchestration script invokes DynAA execution providing optimal scheduling results obtained from AOW in DynAA format.
11. DynAA performs simulation of obtained scheduling results and stores resulting data in its JSON format.
12. Orchestration script send resulting data to CIF service. Resulting data asset converted to CIF and receiving unique namespace ID.
13. Orchestration script checks difference between simulation results and optimization results. If this difference is below provided threshold, or maximum number of iterations achieved, the Orchestration script invokes data transformation to PREESM format and executes PREESM passing as parameters both simulation results and optimization results. Otherwise, the orchestration script invokes transformation of simulation results to AOW format and calls AOW providing these results as well as converted PREESM data obtained at Step 5.
14. If the orchestration script executes AOW in the previous step, go to Steps 7. If the orchestration script executes PREESM, PREESM generates runtime code and Stop.

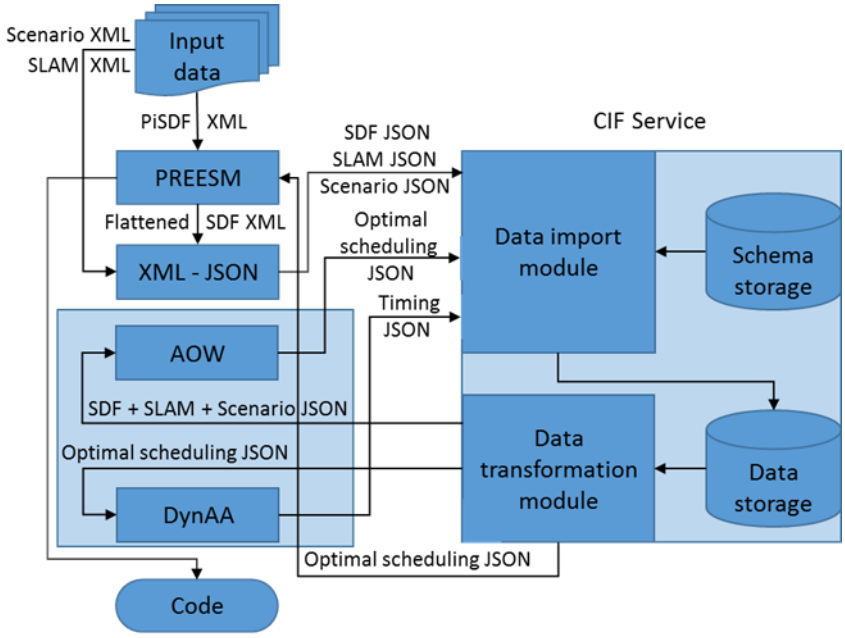

Figure 3-3. CIF PoC data flow

4. Direct Connections

This section is dedicated to direct tool-to-tool connections.

4.1. PoC Connection ARTICo3 – MDC – CAPH

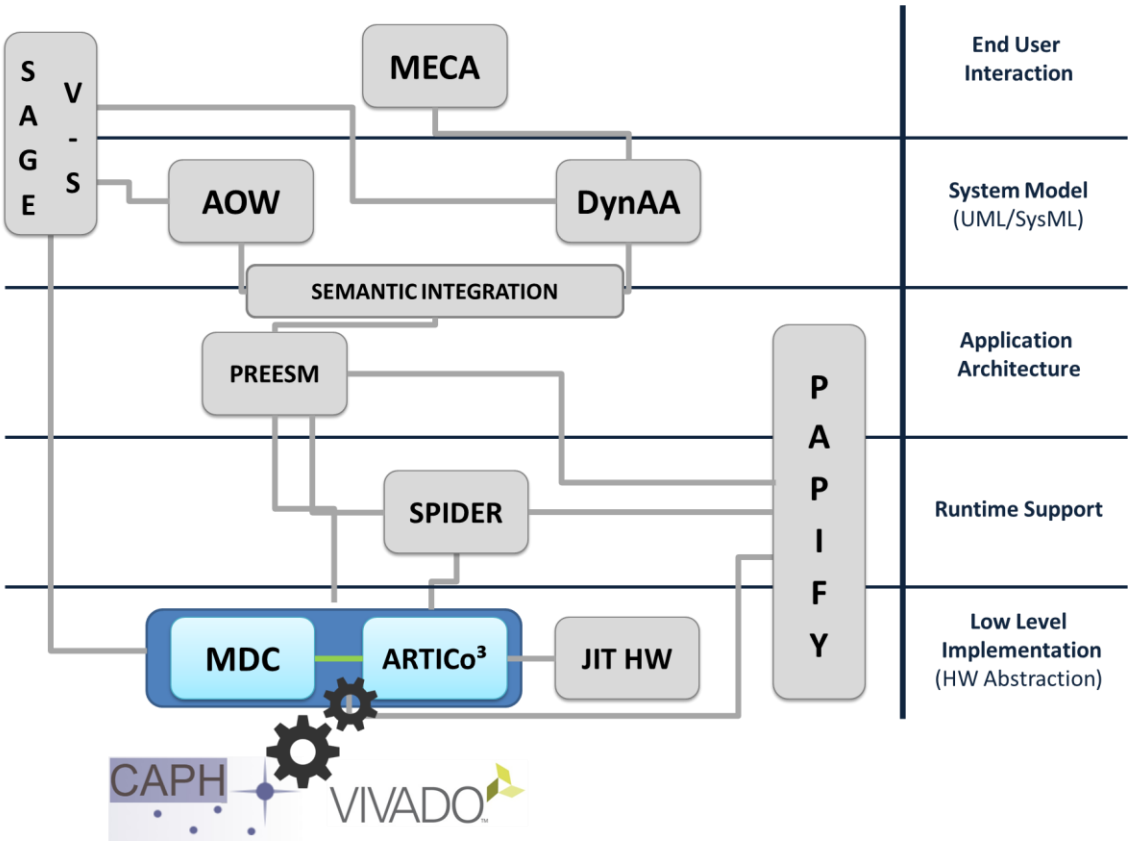

Figure 4-1: MDC-ARTICo³ PoC

Purpose of the Integration: CPS need to meet several functional and non-functional requirements imposed by the environment, the user and their internal status. The presence of different, concurrent requirements influencing the system during operation introduces the need for an advanced adaptivity support. FPGA-based reconfigurable systems provide a valuable solution to this problem: lying in the middle between general purpose computing platforms and application specific circuits, they offer a trade-off between software-like flexibility and hardware-based execution performance. The point is that there are many kind of reconfigurable systems and that their design is not straightforward. It requires detailed knowledge of both the application and the hardware infrastructure and the flow is highly variable, depending on the chosen reconfigurability strategy. As explained in D4.3, reconfigurable systems can be divided, according to their granularity, in: Fine-Grain Reconfigurable (FGR, changes at bit level) and Coarse-Grain reconfigurable (CGR, changes at word level) systems.

In CERBERO two tools offer support for hardware reconfiguration: (1) The ARTICo^3 framework provides adaptive and scalable hardware acceleration, actively altering the computing substrate to change the available functionality using DPR (see D5.6), while (2) the MDC tool delivers automatic generation and management of CGR systems based on the dataflow model of computation (see D5.6). Their integration brings together all the benefits from both DPR and CGR, leading to more flexible solutions that can cope with the changing of functional and non-functional requirements affecting CPS operating contexts. The integration of ARTICo^3 and the MDC Tool offers a unique toolchain capable of automatically implementing and managing multi-grain reconfigurable systems, offering support for advanced adaptivity.

To raise the level of abstraction and make hardware reconfigurable platforms usable by programmers with little to none hardware design skills, we also integrated in this flow the CAPH tool, an open source HLS engine external to the CERBERO partnership (see D4.4). With the MDC & CAPH integration it is possible to automatically generate generic CGR accelerators for the CERBERO adaptivity support (see D4.4).

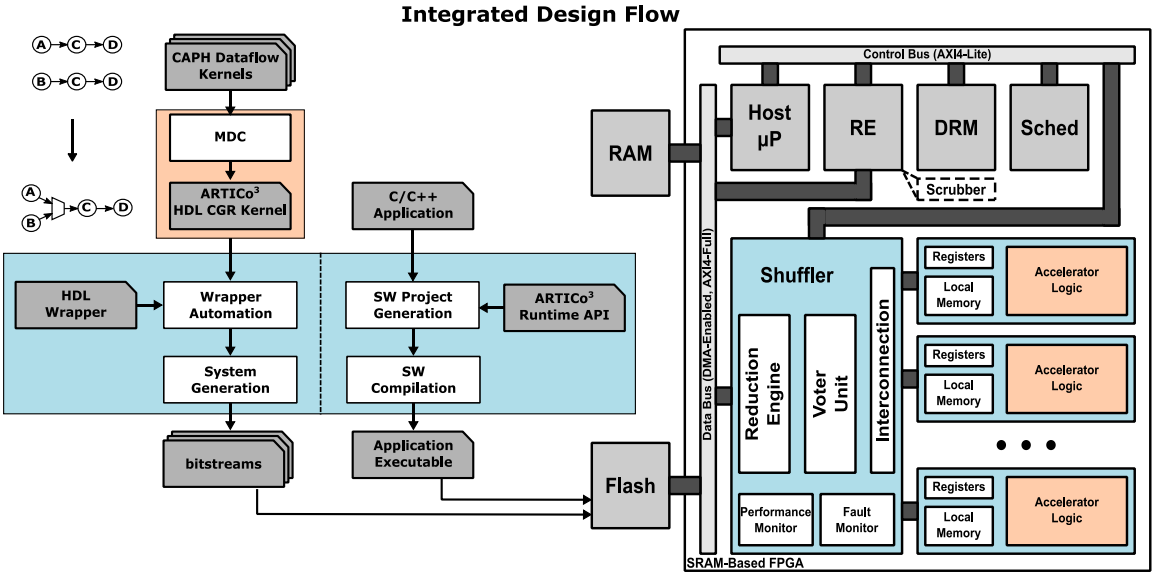

Figure 4-2: CAPH-MDC-ARTICo3 direct tool-to-tool integration

Exchanged Data: Figure **4-2** shows the integrated design flow and the runtime setup. The hardware generation flow (on the left hand side) starts from high-level dataflow descriptions of the behaviours to be implemented in the configurable logic. Such descriptions are compliant with CAPH dataflow specifications. CAPH is an open source HLS engine supporting dataflow models as specification format (similar to the MDC one) that generates target independent code (it generates generic RTL descriptions for any kind of FPGA vendor or for ASIC flows) (see D4.4). CAPH forwards to MDC the SDF models of the networks to be accelerated and the HDL descriptions of the actors composing them. MDC merges the SDF models to create the HDL description of the CGR accelerator, which is post-processed by an ad-hoc MDC back end that derives the corresponding CGR HDL (Verilog) computational kernel, making it ARTICo^3-compliant (properly wrapping it with the glue logic necessary to serve as an ARTICo^3 DPR reconfigurable partition). Finally,

the toolchain generates the bitstreams related to the system (static part) and to the hardware accelerators (reconfigurable partitions). On the software side, the toolchain keeps the capability, inherited from the ARTICo^3 framework, of generating the application executable that manages operation execution and computation offloading to the hardware accelerators also when these latter are MDC-generated CGR accelerators. Both (DPR and CGR) reconfiguration mechanisms are transparently managed from the user code running in the host processor.

PoC: The multi-grain reconfiguration capabilities of the combined CAPH-MDC-ARTICo^3 reconfiguration support are currently shown in an image-processing application scenario. The setup features ARTICo^3 on a Zynq board running Linux and a camera that acquires live video. The input images are sent to a configurable number of hardware accelerators where two edge detection kernels have been implemented (Sobel and Roberts). In order to switch from one kernel to another, the user can decide to use the FGR approach of ARTICo^3 to completely change the logic instantiated in each slot, or to use the CGR approach of the MDC-generated accelerators to multiplex the internal datapath of the accelerators. As a result, it is possible to see, in real time, the runtime overheads of each type of reconfiguration mechanism. Additional adaptivity evaluation can be performed by changing the working point of the application, which is based on several parameters: input image size, number of hardware accelerators used to exploit data-level parallelism, and hardware redundancy level (simplex, DMR, TMR) for fault-tolerant execution.

Useful material/links:

CAPH-MDC integration, presented at SIE 2018: link

ARTICo^3-MDC integration, presented at UPM-CEI: link

4.2. PoC Connection PREESM-Spider-Papify/Papify-Viewer

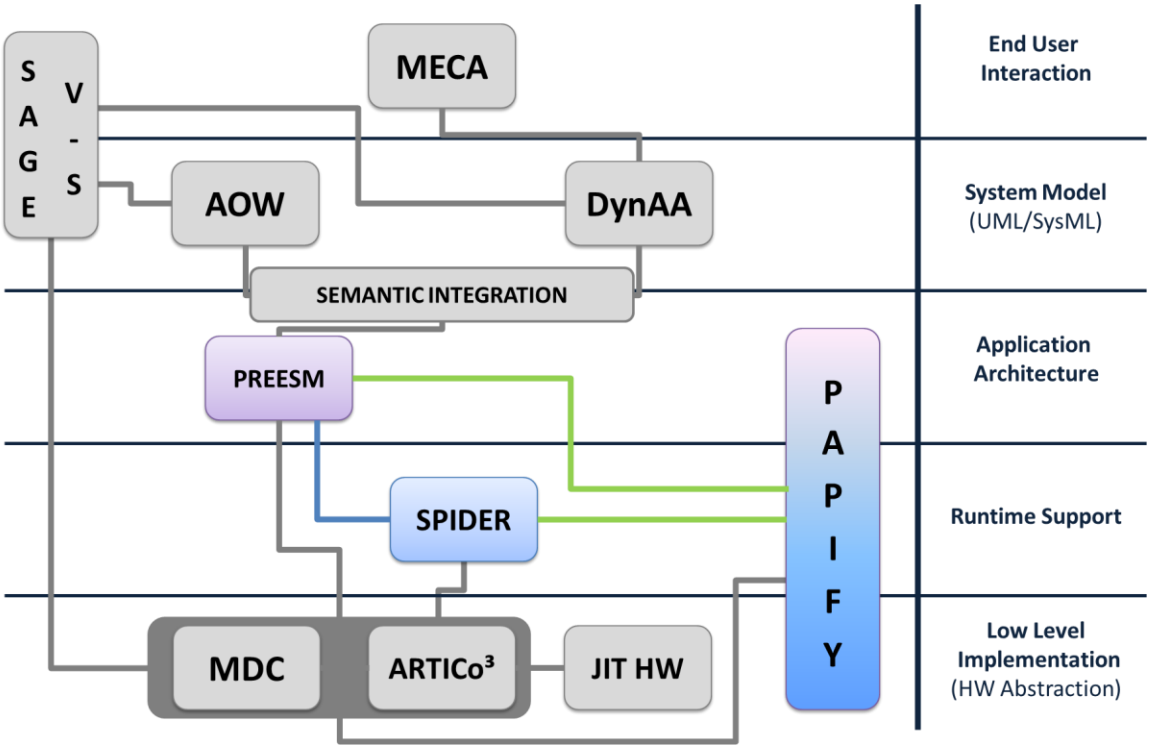

Figure 4-3: PREESM-SPIDER-Papify PoC

Purpose of the Integration: In the context of CPS, the productivity gap between platform complexity and application productivity is widening. To cope with this aspect, current Y-chart design flows isolate the platform and the algorithm development and, automatically, generate a generic solution for the problem. However, these solutions are usually generated following a predefined methodology for any application and, in consequence, they can be easily improved by a trained developer.

In order to improve the quality of these automatic deployments, Design Space Exploration (DSE) techniques need to be included within the generation procedure and, additionally, to assess execution performance can be used to refine the work distribution and improve the final system performance.

In CERBERO three tools can be combined to fulfil this requirement: (1) The PREESM rapid prototyping framework provides a Y-chart design flow tool; (2) SPiDER is able to manage the information of the system execution and make changes on the system workload distribution; (3) finally, Papify tool retrieves the system performance information by accessing Perfomance Monitoring Counters through the open-source PAPI library. The integration of PREESM, SPiDER and Papify offers the capability of refining the design time proposed solutions, while increasing the decision criteria managed by SPiDER.

Finally, the platform independencee supported by every tool increases the level of abstraction reachable by the developer, who can easily obtain real-time system

performance information and visualize in real-time the behaviour of the system thanks to Papify-Viewer.

Exchanged Data: Figure **4-4** and Figure **4-5** show the resulting integration of (1) Papify into PREESM framework and the (2) SPiDER execution block diagram with Papify and Papify-Viewer tools included, respectively. Additionally, Figure **4-6** shows an example of Papify-Viewer displaying execution time information. In Figure **4-4**, the monitoring configuration of the application is set up employing a new user interface. After that, PREESM automatically generates instrumented code that is compliant with either PREESM backend or the SPiDER run-time manager. Secondly, as can be seen in Figure **4-5**, Papify performance monitoring has been included within the Local Run-Time (LRT) of SPiDER, which means that the monitoring happens in each Processing Element (PE) independently. Additionally, this information is sent to the Global Run-Time (GRT), which can analyse this information so as to make changes in the system behaviour to increase the application performance. Finally, Papify-Viewer, which is an independent application, can display the information in real-time providing the user with a graphical representation of the current system behaviour, as shown in Figure **4-6**.

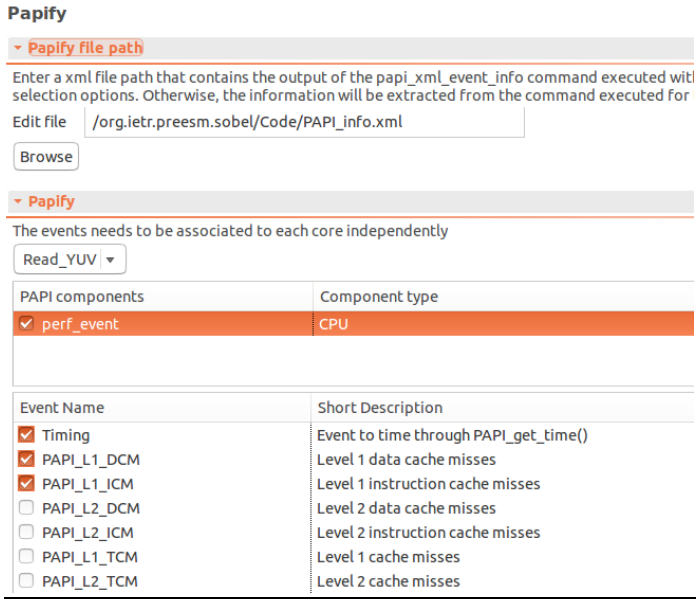

Figure 4-4: PREESM-Papify tool-to-tool integration

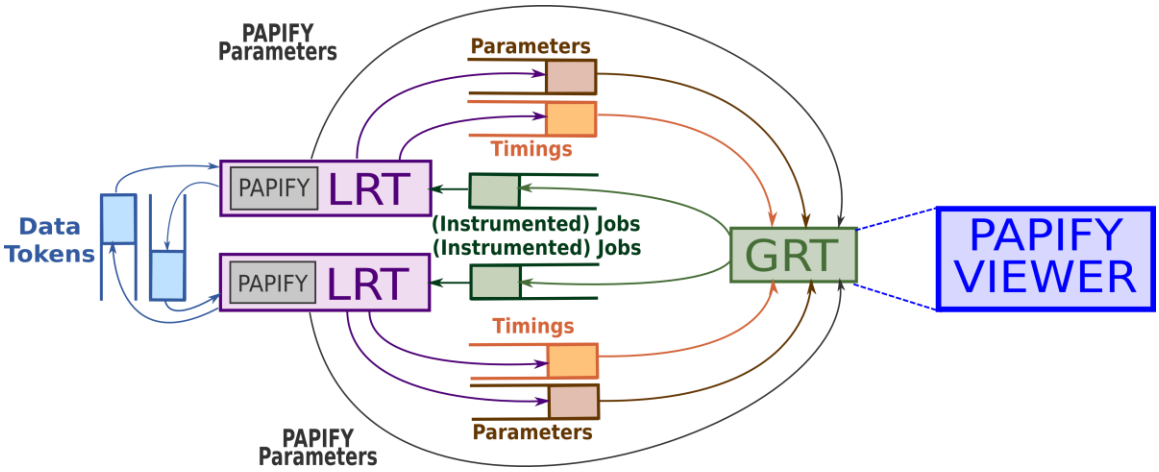

Figure 4-5: SPiDER-Papify/Papify-Viewer tool-to-tool integration

Figure 4-6: Papify-Viewer display example

PoC: The system performance monitoring capabilities of the combined PREESM-SPiDER-Papify/Papify-Viewer is currently shown using an image-processing application scenario, a sobel-morpho image filter. The application monitoring is configured using the PREESM framework and generationcode compliant with the SPiDER run-time manager. In this case, the user is able to decide how many CPU cores the system will use. Likewise, during the system execution, Papify-Viewer displays the workload distribution, the timing and the events that the user has selected to be monitored. As a result, it is possible to see how the system is affected by the redistribution of the workload together with a real-time application profiling.

Useful material/links:

PREESM-Papify integration, presented at CF 2018: link

Spider-Papify integration, presented at COWOMO 2018: link

4.3. Other Direct Connections

The direct connection between DynAA and MECA is not discussed in this deliverable since it has been deeply addressed in D6.10. Other connections like PAPIFY, SPiDER and PREESM with the low-level tools are not mature enough yet to be presented in this deliverable. The same applies for the connection of SAGE VS (originally VT tool) with DynAA.

5. References

[SEMI]	E. Shindin et al., *SEmantic Middleware presentation*, IBM Research - Haifa, Israel, 2018.
[JanusGraph]	janusgraph.org
[ThinkerPop/ThinkerGraph]	http://tinkerpop.apache.org/javadocs/3.2.2/full/org/apache/tinkerpop/gremlin/tinkergraph/structure/TinkerGraph.html
[Cassandra]	cassandra.apache.org
[ElasticSearch]	www.elastic.co

Appendix I: CIF Example

CIF meta-meta model

Initial CIF meta-meta model defines schema of tool's input and output files. The following schema syntax provides an example in JSON format:

```
{
 "view" : {
  "name": "str",
  "classes": [
   "class_def"
  ]
 },

 "class_def" : {
  "name": "str",
        "representation" : "repr_def",
  "schema": "class_schema"
 },

 "repr_def" : {
  "type": "repr_type_def", //one of repr_type_defs
        "property_base" : "property_base_def",
        "key_value_base": "key_value_base_def"
 },

 "repr_type_def" : ["mixed", "key_value_base", "property_base"],
 //key_value_base representation: key: value - key property name value property value
 //property_base representation main_key: [{name_key: name_value, value_key: value_value}]

 "property_base_def" : {
                "base_key": "str", //key under which we have list of property base representation
                "property_name_key": "str", //key of property name
                "property_value_key": "str" //key of property value
 },

 "key_value_base_def": {
                "key_prefix": "str" //prefix of keys in key value represebtation
 },

 "class_schema" : {
  "extensible" : "bool",
```

```
  "properties" : ["property_schema"],
  "keys": {"key_name": "key"},
  "id": "str" //property name (property that gives unique id of objects of this class)
 },

 "property_schema": {
  "name": "str",
  "type": "type_def", // one of type defs
  "value": "value_schema",
  "optional": "bool",
  "set": "bool"
 },

 "key": [
  "str" //names of properties that form unique key
 ],


 "type_defs": [
  "bool","float","int","str","object"
 ],

 "value_schema": {
  "optional": "bool",
  "default": null,
  "constraints": [],
  "object": "object_schema" // in case if value is object, otherwise null
 },

 "object_schema" : {
  "domain": "str",
  "class": "str",
  "extensible": "bool",
  "id_type": "id_type_def" // one of id type defs
 },

"id_type_defs": [
  "object_id", {"key":"key_name"}, "object"]

}
```

CIF example for PREESM-AOW semantic integration

The example starts from three files native in PREESM:

- 03-parallel_sobel.graphml – flattened sdf graph (XML),
- 4CoreX86.slam – architecture (XML),
- sobel_scenario.xml – timing (XML) [not presents in the example].

XML files converted to JSON using an XML-2-JSON converter.

After conversion, three JSON files are consequently produced:

- sobel_sdf.json,
- 4CoreX86.json,
- sobel_timing.json [directly defined].

PREESM meta models of the files are given according to CIF meta-meta model in directory "schemas". All files converted to CIF according to corresponding schemas. Schemas are processed recursively. Top-level schema for flattened sdf graph represented in sdf.json file, top-level schema for architecture represented in slam.json file, top-level schema for architecture represented in timing.json file. Intermediate representation after conversion described by following JSON files:

- sobel_sdf_cif.json
- slam_cif.json
- sobel_timing_cif.json
- preesm_classes_cif.json

Note, that representation divided to several JSON files for convenience only. Actually, there is a single database containing connected objects.

From CIF, data transformation to AOW format is performed. There are various property name transformations as well as more complex architecture transformation. In the PoC all these transformations are performed by a script containing sequence of CIF API calls. Transformation add to CIF representation additional set of objects that are represented in sobel_aow_cif.json and aow_classes.json files.

Flattened sdf graph represented by object of "sdf" class in "preesm" namespace converted to object of "scheduling_application" class in "aow" namespace. The transformation mainly converts objects from classes defined in "preesm" namespace to objects from classes defined in "aow" namespace by renaming various property names.

Architecture represented by object of "slam" class in "preesm" namespace converted to object of "scheduling_architecture" class in "aow" namespace. The transformation converts objects from classes defined in "preesm" namespace to objects from classes defined in "aow" namespace by renaming various property names and perform more complex aggregative transformations (for example objects of "componentInstance" class are aggregated by "componentRef" property values and if "componentDescription" object having property "componentRef" with same value have also "componentType" property with value "Operator" it is transformed to "processingElement" object)

Timing data represented by object of “timing” class in “preesm” namespace converted to aggregation of objects of "scheduling_execution" class in “aow” namespace, where each object transformed from corresponding “timingEntry” object in “preesm” namespace, by changing property names.

Then data to JSON file according to AOW input data schema (top-level schema represented in aow.json file) are converted. After these steps, a single JSON file in AOW format (sobel_aow.json) is obtained.

AOW performs optimization and store optimal scheduling result in JSON file in AOW format (aow_result.json). Next, JSON is converted to CIF and then transformed to PREESM format using CIF middleware API, enriching existing PREESM SDF representation by scheduling results. Finally, SDF together with scheduling results converted to JSON according to PREESM schema (sobel_sdf_result.json).