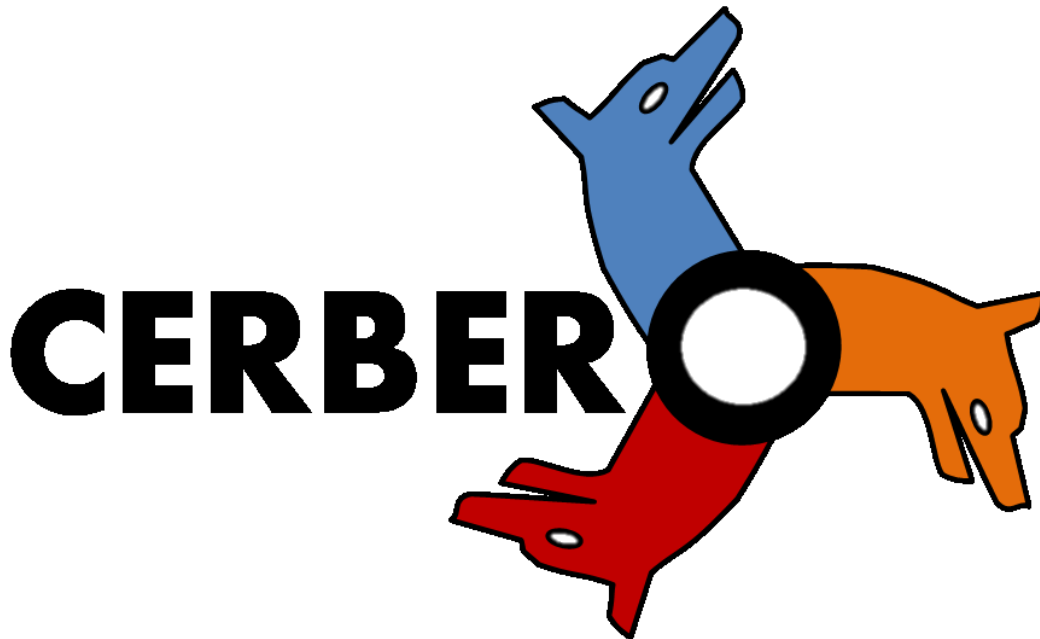


Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105



D5.3: CERBERO Framework Demo (Final Version)

Lead Beneficiary: AI

Workpackage: WP5

Date: 29/08/2019

Distribution - Confidentiality: [Public]

Abstract: This deliverable describes the integrated CERBERO development framework. This is a short explanation and guidelines of the software components that compose CERBERO development framework. The description of the framework characteristics is included in D5.1. The document presents separately all the PoC that use CIF for connecting tools, all the direct connection PoC and a PoC of a tool completely developed during the CERBERO project. This is the final version of the D5.3.

© 2017 CERBERO Consortium, All Rights Reserved.

Disclaimer

This document may contain material that is copyright of certain CERBERO beneficiaries, and may not be reproduced or copied without permission. All CERBERO consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

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 Onderzoek 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.

Except as otherwise expressly provided, the information in this document is provided by CERBERO to members "as is" without warranty of any kind, expressed, implied or statutory, including but not limited to any implied warranties of merchantability, fitness for a particular purpose and non-infringement of third party's rights.

CERBERO shall not be liable for any direct, indirect, incidental, special or consequential damages of any kind or nature whatsoever (including, without limitation, any damages arising from loss of use or lost business, revenue, profits, data or goodwill) arising in connection with any infringement claims by third parties or the specification, whether in an action in contract, tort, strict liability, negligence, or any other theory, even if advised of the possibility of such damages.

The technology disclosed herein may be protected by one or more patents, copyrights, trademarks and/or trade secrets owned by or licensed to CERBERO Partners. The partners reserve all rights with respect to such technology and related materials. Any use

of the protected technology and related material beyond the terms of the License without the prior written consent of CERBERO is prohibited.

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
Luca Pulina	UniSS
Claudio Rubattu	UniSS
Simone Vuotto	UniSS
Alfonso Rodriguez	UPM
Rafael Zamacola	UPM
Daniel Madroñal	UPM
Evgeny Shindin	IBM
Michael Masin	IBM
Karol Desnos	INSA
Julio De Oliveira Filho	TNO

The list of authors does not imply any claim of ownership on the Intellectual Properties described in this document. The authors and the publishers make no expressed or implied warranty of any kind and assume no responsibilities for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information contained in this document.

Document Revision History

Date	Ver.	Contributor (Beneficiary)	Summary of main changes
02/07/2019	0.1	AI	initial draft
15/07/2019	0.2	UNICA	SPiDER PAPIFY MDC
26/07/2019	0.2	IBM and UNICA	AOW-MDC CIF connection
27/07/2019	0.3	UPM	SPiDER-PAPIFY-MDC

31/07/2019	1.0	AI	Release of the complete draft
31/07/2019	1.1	UniSS	Review of the complete draft
01/08/2019	1.2	UPM and AI	Fix and refactoring

Table of contents

1. Executive Summary.....	6
1.1. Structure of the Document.....	6
1.2. Related Documents.....	7
1.3. Related CERBERO Requirements.....	7
2. The CERBERO Framework Integration	9
2.1. Overview of CERBERO tools connections	9
2.2. The Integration processes.....	10
3. Intermediate Format Connections.....	12
3.1. CERBERO Innovative Approach for Semantic Integration.....	12
3.2. Purpose of Integration with CERBERO Intermediate Format	13
3.3. Integration Framework Tool-Flow	13
3.4. PoC CIF Connection PREESM – AOW – DynAA.....	15
3.5. PoC CIF Connection AOW – MDC.....	18
4. Direct Connections	21
4.1. PoC Connection ARTICo3 – MDC – CAPH	21
4.2. PoC Connection PREESM-SPiDER-PAPIFY/PAPIFY-Viewer.....	24
4.3. PoC Connection SPiDER – PAPIFY – MDC	27
4.4. PoC Connection SAGE-ReqV – DynAA	31
4.5. PoC Connection PREESM - APOLLO multiversioning.....	31
5. Standalone PoC	37
5.1. PoC IMPRESS for Just-in-Time HW Composition.....	37
Appendix I: CIF Example	42

1. Executive Summary

This document presents a short description of the main new Proofs of Concept (PoCs) of the integrated CERBERO development framework, which has been largely discussed in all the other WP5 deliverables, including those of the single components (D5.2 and D5.6) and those of the integration methodology (D5.1, D5.4 and D5.5).

It is important to highlight that this document is the second and final version of the D5.7 and they are both supporting documentation of the software deliverables D5.1 and D5.7 respectively. The main scope of this document is to provide the explanation and technical details of the 7 main PoCs that have been developed and tested at M18 and at M27.

In order to cover and test the two connections strategy used within CERBERO, the following PoCs have been developed and reported in this deliverable:

- 2 PoCs using CERBERO Intermediate Format (CIF)
- 3 PoCs using the direct connections among couples or series of CERBERO tools.
- 1 PoCs using the direct connection between CERBERO tools and external tools.
- 1 PoC of a single tool, completely developed during CERBERO project.

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 D5.1 and D5.2.

In order to speed up and ease the reading and review process the text of sections and paragraphs that have NOT been significantly updated and revised are in dark gray.

1.1. Structure of the Document

In Section 2 a general overview of the CERBERO development framework and its integration strategies is provided. In Section 3 a description of the approach for Semantic Integration is presented together with a comprehensive explanation of the PoCs using CIF. Finally, Section 4 is dedicated to present PoCs developing direct connections among

tools. Finally, Section 5 presents a Stand-Alone PoC developed completely during CERBERO project.

1.2. Related Documents

The CERBERO deliverables related to this document are:

- D2.1 – CERBERO Technical Requirements
 - The activities behind D5.3 contribute to satisfy the requirements listed in D2.1.
- D3.3 – Cross-layer Modelling Methodology for CPS
 - D3.3 provides methodological foundation for CERBERO Intermediate Format.
- D5.1 – CERBERO Holistic Methodology and Integration Interfaces
 - D5.1 presents the final version of design framework integration approach and the required interfaces.
- D5.2 – CERBERO Framework component
 - In D5.2 the final version of the technical details of the different components/features of the CERBERO design environment are reported.
- D5.3 – CERBERO Framework Demo (Ver. 1)
 - The PoCs reported in the D5.3 are an integral part of this deliverable.

1.3. Related CERBERO Requirements

Deliverable D2.1 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 and between AOW and MDC 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	The possibility of providing a multi-viewpoint, multi-

	SHOULD provide multi-viewpoint multi-objective correct-by-construction high-level architecture.	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, and among SAGE and DynAA, demonstrated in a dedicated direct connection PoC.
0009	CERBERO SHALL develop integration methodology and framework.	The PoCs presented and developed in this deliverable (and in the previous one) are part of the assessment of the overall CERBERO development framework. Assessment of can be considered still ongoing through the activities carried out within WP6 where the framework is used by the UC providers.

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

A clear explanation of the CERBERO framework has been reported in the D5.1 Section 4.3, but a schema of all the available connections at design time and at run time are also reported respectively in the Figure 2.1 and Figure 2.2 of this document.

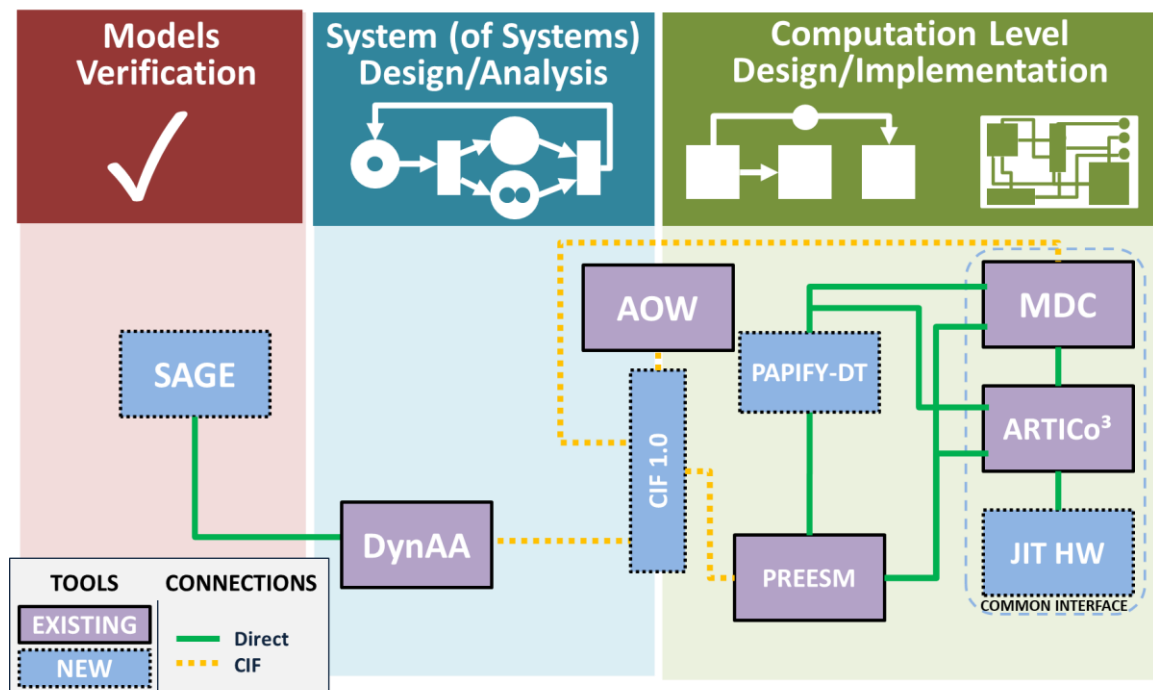


Figure 2.1 - CERBERO toolchain for Design-Time Support

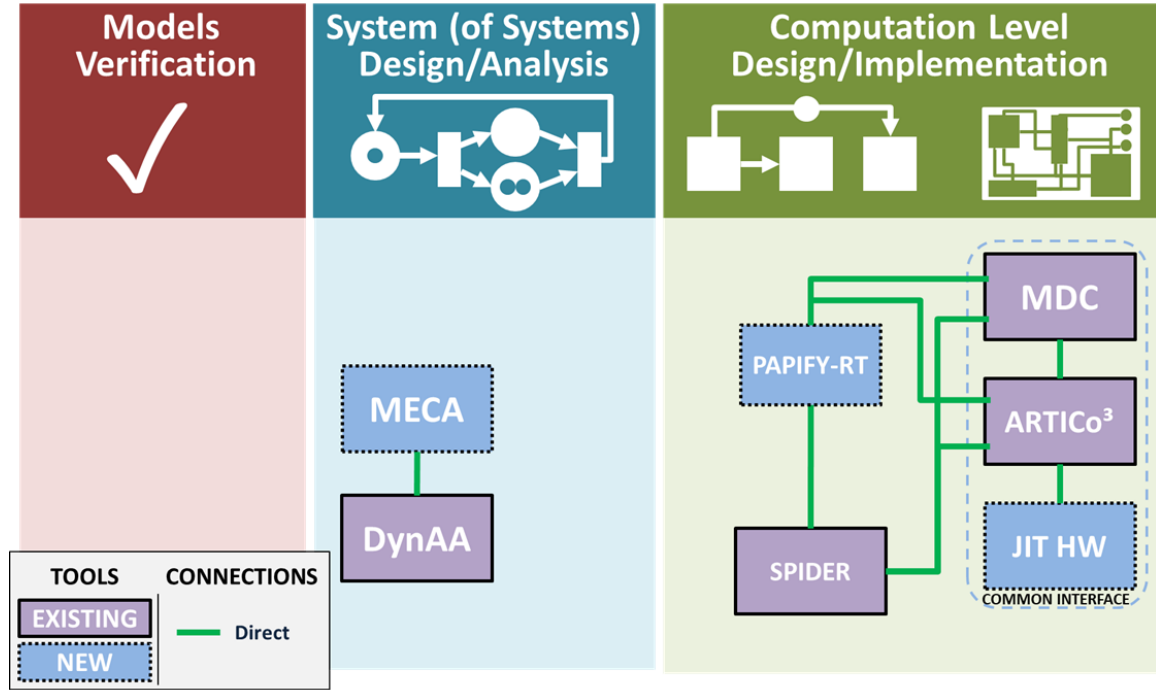


Figure 2.2 - CERBERO toolchain for Run-Time Support

2.2. The Integration processes

D5.1 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).

In the CERBERO project, using a continuous integration methodology 7 PoCs have been developed for testing the connection through CIF and the direct connections. This way, we succeeded in developing and testing:

- connections among tools at the same layer or leveraging on the same Model of Computation. These types of connection are direct ones, such as MDC with Artico3, PREESM-SPiDER with PAPIFY, and SPiDER and PAPIFY with MDC.
- cross layer connections operating at design-time, such as AOW and DynAA with PREESM and AOW with MDC. These connections involve both different semantics and operate at different levels, so that they leverage on CIF.
- cross layer connections operating at run-time, such as MECA with DynAA, (described in D6.10) that leverages on a direct integration methodology.
- the connection of the CERBERO framework with external tools, like MDC with CAPH, PREESM with APOLLO Multi-versioning.

Moreover, in the present deliverable, we also present the standalone PoC of IMPRESS, a tool for JIT HW composition. IMPRESS is a tool that has been developed from scratch in CERBERO and thus, its TRL is low to be included as part of the Use Case demonstrators.

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 becomes a two-step process where in the first step tool describes its output and input object models using class definitions language (schemas), and in the second step tools that requires integration defines mappings between their object models using equivalence rules (see D5.1 for details). Once all this information provided CIF service will care on automatic transformation between models. Class definitions enabling export/import of tool object models into intermediate format, while equivalence rules enabling transformation of intermediate representation of models between connected tools. 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 (CIF service) to facilitate communication between the integrated components. The architecture of CIF service represented on Figure 3.1.

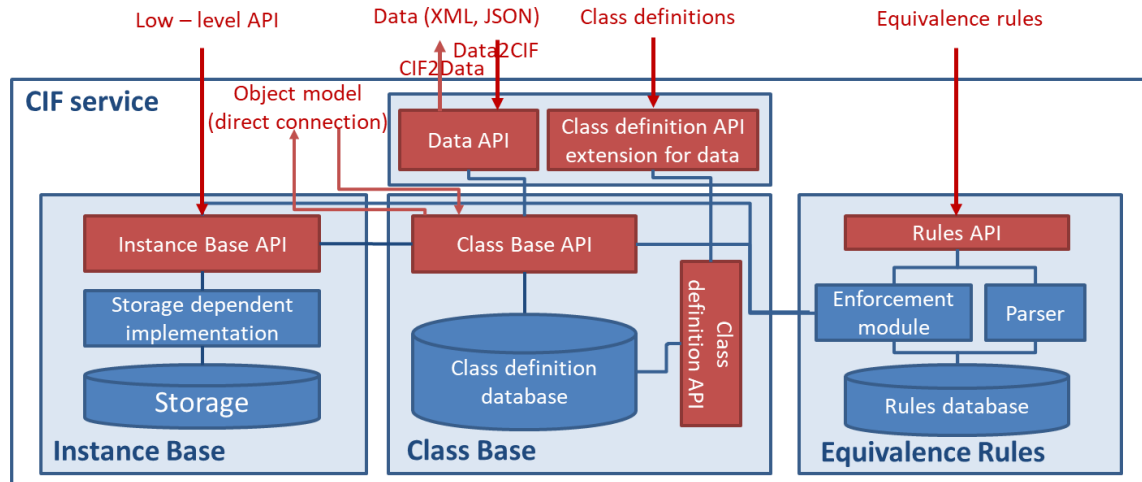


Figure 3.1 - Architecture of CIF service

The building blocks of CIF service are:

- **Instance Base module:** is base module of CIF service that enable storage of information and provides access to this information through Instance Base API. Current implementation of the CIF service built on top TinkerPop Server, but any other graph databases supporting gremlin language can be used instead of it without code modifications. Note, that graph database is not necessary element for CIF service: Instance Base potentially can be implemented also on top wide-column or relational database or even JSON-files or JSON-document database.
- **Class Base module:** provides support of class-related interfaces. Enable storage of class definitions and allow export/import object models into intermediate format. In the current implementation class definition storage utilizes file system, storing each schema as JSON file inside directory structure, where directory names correspond to namespaces and class names.
- **Data extensions of class base module:** built on top of class base module and provide parsing of JSON files into intermediate representation format. Also, current implementation support XML file conversion, as two-step process where standard XML to JSON conversion provided at the first step, and the resulting JSON imported to CIF at the second step.
- **Equivalence rules module:** includes parser that produce abstract syntax tree from equivalence rules, enforcement module that enables equivalence rules enforcement on class base and instance base levels, and rules database that store all provided rules. In the current implementation equivalence rules stored as python serialized objects in the same file system structure that serve as class definitions storage.

Current implementation of CIF service also includes transformation module that will be removed when equivalence rules module development will be finished.

3.4. PoC CIF Connection PREESM – AOW – DynAA

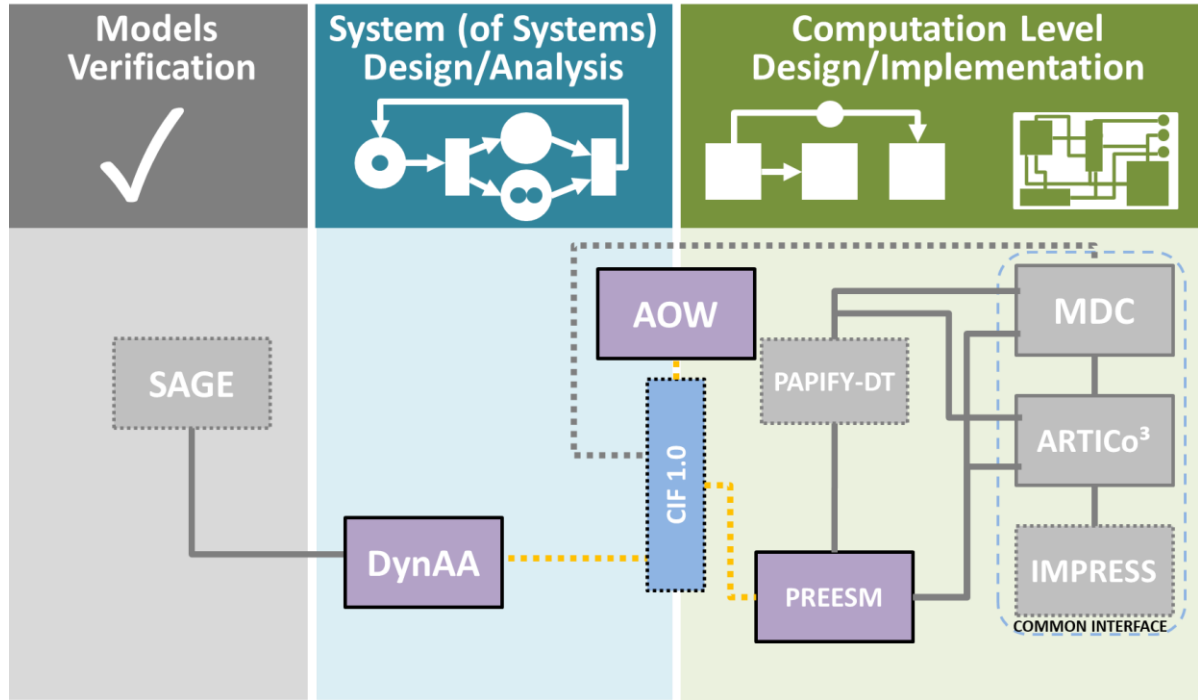


Figure 3.2 – PREESM – AOW DynAA CIF PoC

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.

WP5 – D5.3: CERBERO framework demo

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 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 invoking data transformation according to corresponding schemas. Each data asset receives unique 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 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 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.

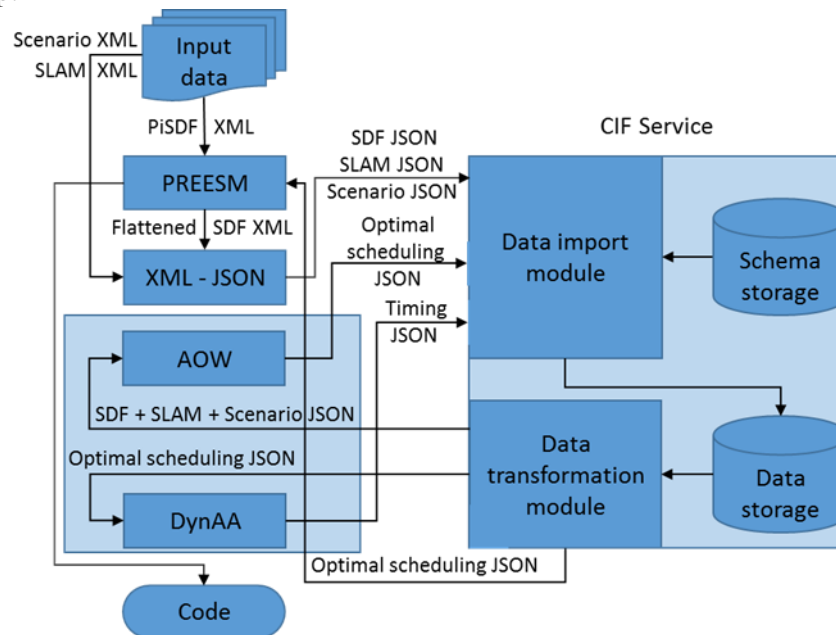


Figure 3.3 – PREESM – AOW DynAA CIF PoC DataFlow

In the final stages of the project the data transformation module of CIF service will be removed, and all transformations will be held by equivalence rules module. That is necessary equivalence rules will be provided instead of transformation scripts.

3.5. PoC CIF Connection AOW – MDC

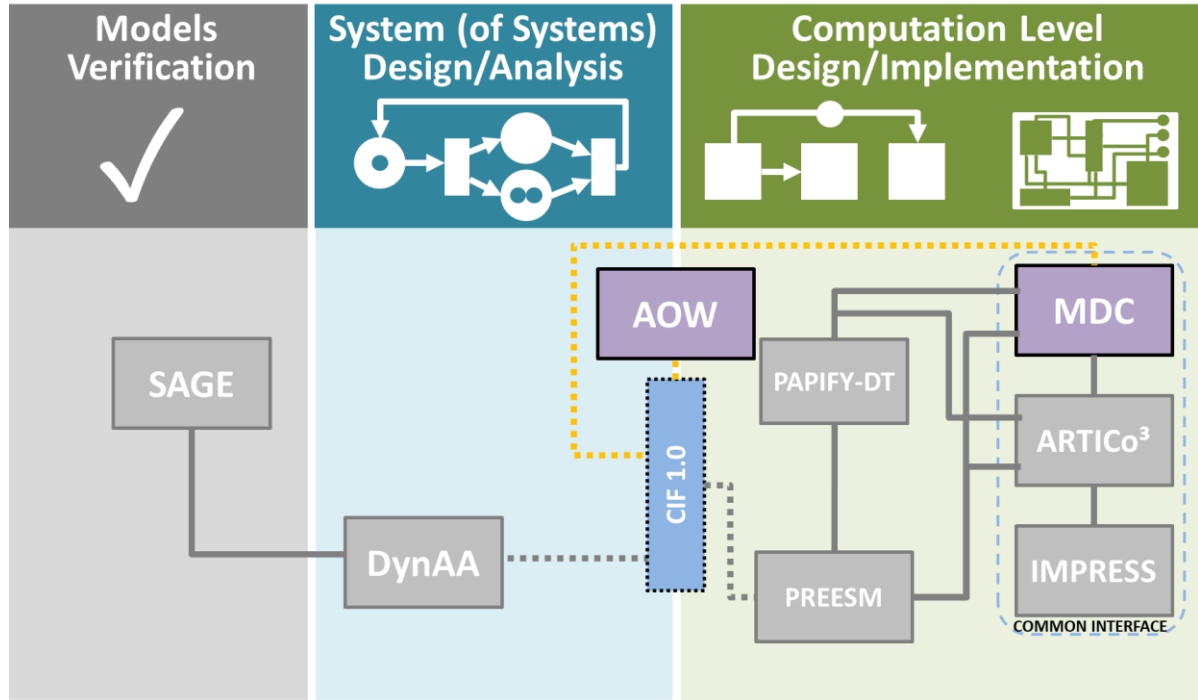


Figure 3.4: AOW – MDC CIF PoC

The purpose of this PoC is to find optimized merged hardware implementation of multiple dataflows, provided as Dataflow Process Network (DPN) graphs, on a hardware, using splitting (1x2) and merging (2x1) switch boxes [Palumbo 2011]. The optimization can be performed with respect to several goals: model metrics (such as minimal number of actors, minimal number of connections, minimal number of switch boxes or shortest switch boxes chain), and implementation metrics (such as minimal area, minimal power consumption or maximal operating frequency). The merging has to be done according to the constraint of keeping the functional correctness of all the considered input dataflows. Moreover, additional constraints can be potentially provided by the user, such as not merging a specific actor or dataflow.

In this scenario MDC is the *service requester*, while AOW is the *service provider*: (i) dataflow models and their parameters described in MDC are passed to AOW, (ii) AOW performs optimization in order to obtain optimized merged design according to the objectives, (iii) the optimal design found by AOW is passed back to MDC.

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

- DPN graphs in XDF (XML dialect) format representing dataflows for (potential) merging;

- (Optional) hardware parameters, such as area and power consumption of actors, or maximum achievable frequency of dataflows, considering their implementation in hardware;
- optimization objectives and (optional) constraints.

In order to fit with CIF and let MDC exploit this latter as a smart bridge to access AOW service, schemas in the agreed JSON format are required on both sides, MDC and AOW. Thus, MDC defines:

- schemas of each kind of input data so that the CIF service will be able to import all the MDC input data and convert them to CIF;
- schemas of result data, i.e. format in which resulting design coming from AOW optimization should be provided to MDC.

From the second integration endpoint, AOW provides two different schemas:

- input format schema, i.e. schema of data required to perform calculation of optimal merged hardware;
- output format schema, i.e. schema of optimal merged hardware.

For the PoC purpose 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 are considered to run on a single Windows (operating system compliant with both of them) machine. The orchestration of execution of the overall toolchain is performed by a 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 and related invocation methods, can be implemented. As in the case of the PREESM – AOW – DynAA connection all transformations at the current phase held by transformation module and will be held by equivalence rules module instead later.

The proposed execution scenario includes the following steps:

1. orchestration script receives three parameters: DPN graphs folder, actors hardware parameters file (XML), optimization criteria;
2. orchestration script invokes XML-to-JSON transformation of all input data files;
3. resulting data in JSON format is sent to the CIF service endpoint that, in turn, invokes data transformation according to corresponding schemas (each data asset receives unique identifier to allow addressing);
4. when data storage is completed, the orchestration script invokes data transformation to AOW format (JSON) performing calls to the corresponding transformation procedures;
5. orchestration script invokes AOW optimization providing files of DPN graphs, hardware parameters and optimization criteria, all in AOW format (JSON);

6. AOW performs optimization and stores resulting data in AOW format (JSON);
7. orchestration script sends resulting data to the CIF service in order to be converted to CIF, again receiving a unique identifier;
8. when data storage is completed, the orchestration script invokes data transformation to MDC format (XML) performing calls to the corresponding transformation procedures, concluding the process.

4. Direct Connections

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

4.1. PoC Connection ARTICo3 – MDC – CAPH

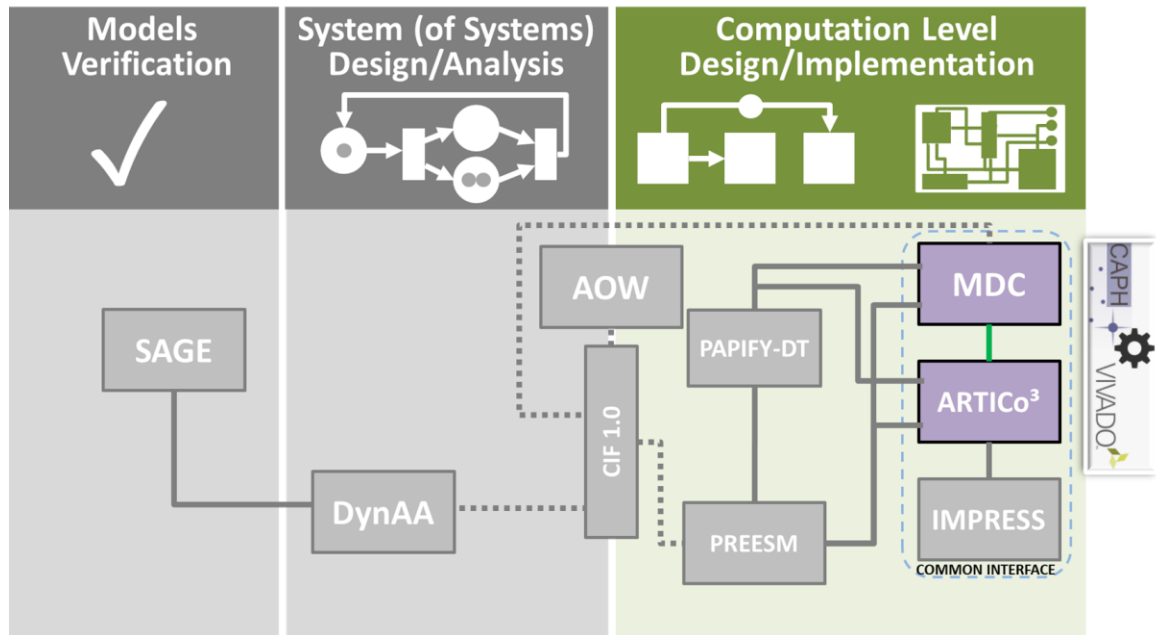


Figure 4.1 - MDC-ARTICo³ PoC

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 ARTICo³ 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 ARTICo³ 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).

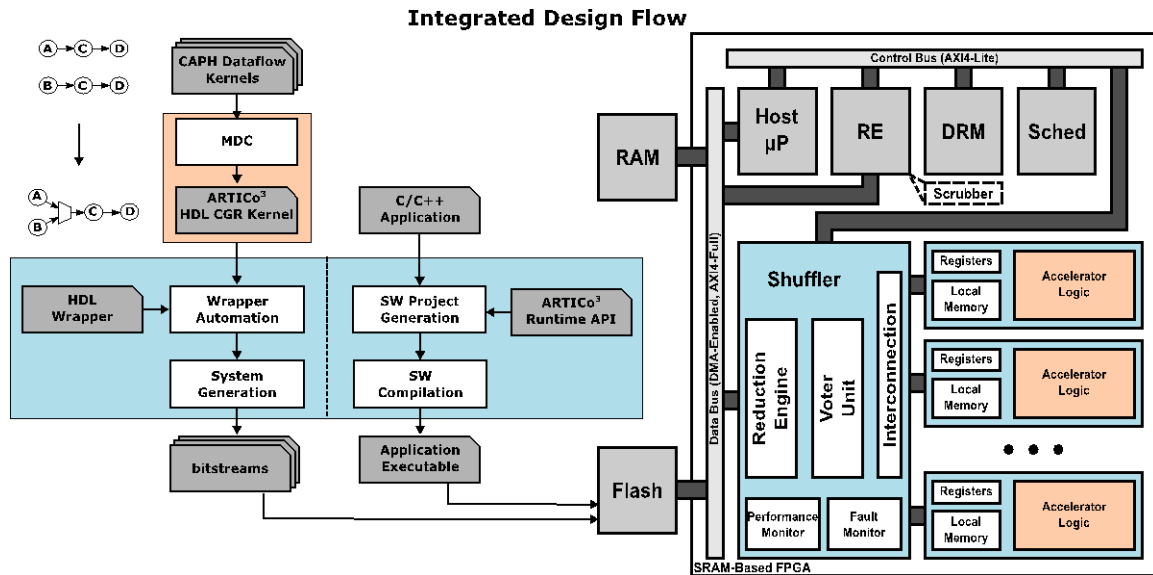


Figure 4.2 - CAPH-MDC-ARTICo3 direct tool-to-tool integration

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 ARTICo³-compliant (properly wrapping it with the glue logic necessary to serve as an ARTICo³ 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 ARTICo³ framework,

WP5 – D5.3: CERBERO framework demo

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-ARTICo³ reconfiguration support are currently shown in an image-processing application scenario. The setup features ARTICo³ 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 ARTICo³ 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](#)

ARTICo³-MDC integration, presented at UPM-CEI: [link](#)

4.2. PoC Connection PREESM-SPiDER-PAPIFY/PAPIFY-Viewer

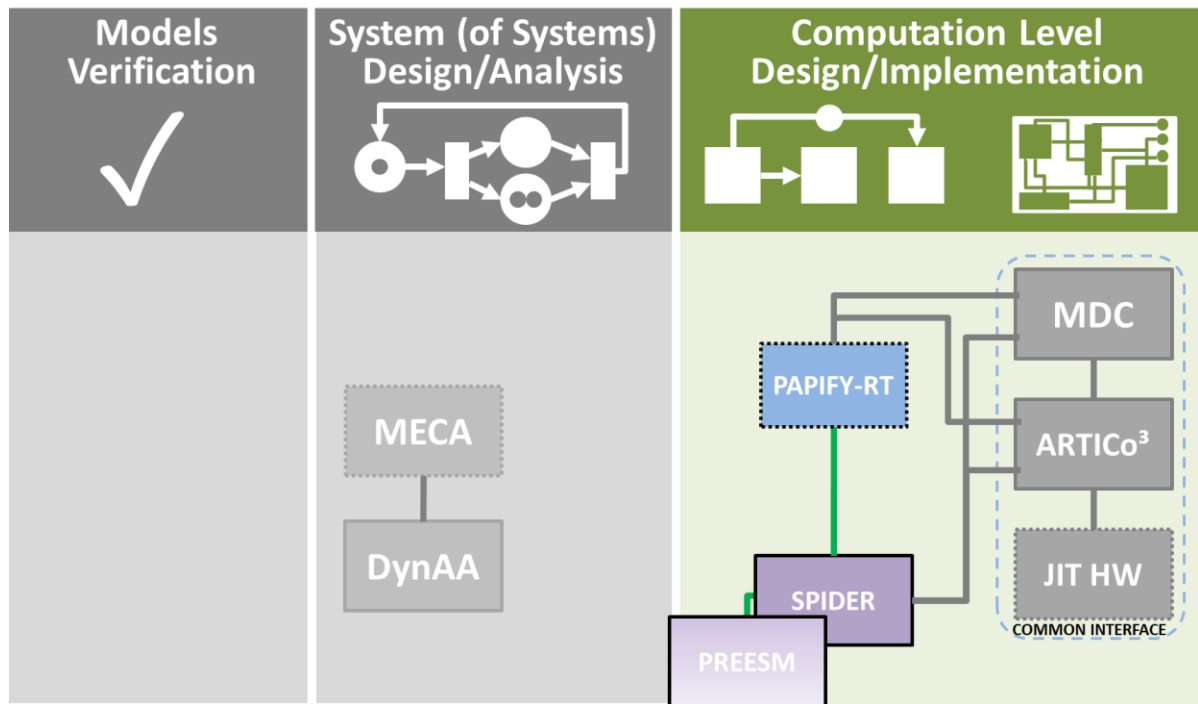


Figure 4.3 - PREESM-SPiDER- PAPIFY PoC

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 Performance 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 independence 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.

Papify

▼ Papify file path

Enter a xml file path that contains the output of the papi_xml_event_info command executed with selection options. Otherwise, the information will be extracted from the command executed for

Edit file

▼ Papify

The events needs to be associated to each core independently

▼

PAPI components	Component type
<input checked="" type="checkbox"/> perf_event	CPU

Event Name	Short Description
<input checked="" type="checkbox"/> Timing	Event to time through PAPI_get_time()
<input checked="" type="checkbox"/> PAPI_L1_DCM	Level 1 data cache misses
<input checked="" type="checkbox"/> PAPI_L1_ICM	Level 1 instruction cache misses
<input type="checkbox"/> PAPI_L2_DCM	Level 2 data cache misses
<input type="checkbox"/> PAPI_L2_ICM	Level 2 instruction cache misses
<input type="checkbox"/> PAPI_L1_TCM	Level 1 cache misses
<input type="checkbox"/> PAPI_L2_TCM	Level 2 cache misses

Figure 4.4 - PREESM- PAPIFY tool-to-tool integration

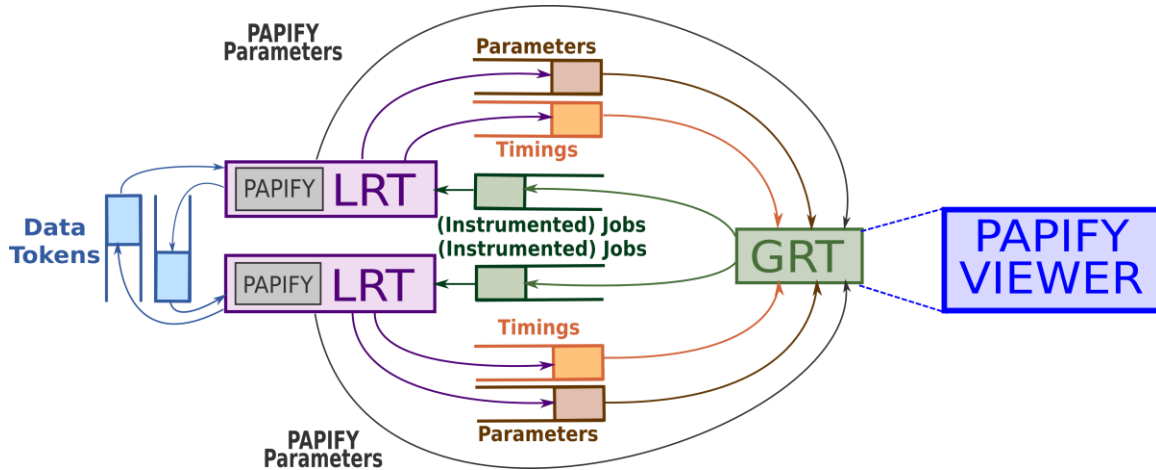


Figure 4.5 - SPiDER-Papify/Papify-Viewer tool-to-tool integration

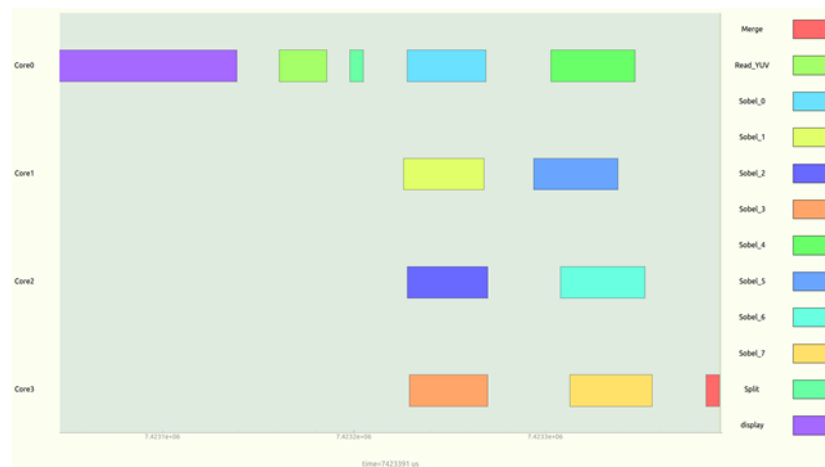


Figure 4.6 - PAPIFY -Viewer display example

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. PoC Connection SPiDER – PAPIFY – MDC

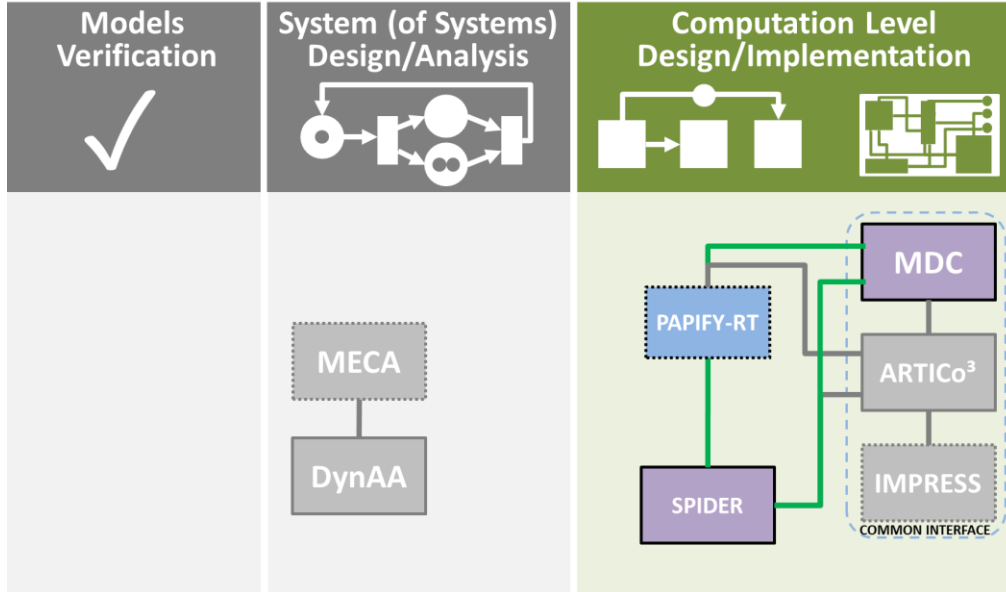


Figure 4.7 - SPiDER - PAPIFY - MDC PoC

Purpose of the Integration: CPSs need to meet several functional and non-functional requirements imposed by the environment, the user and their internal status.

The most diffused software (Sw) approach for enabling self-awareness is based on accessing the existing Performance Monitoring Counters (PMCs) of modern CPUs. In CERBERO, PAPIFY provides a lightweight monitoring infrastructure by means of an event library aimed at generalizing the Performance Application Programming Interface (PAPI) for embedded heterogeneous architectures. PAPIFY has been integrated with PREESM and SPiDER to provide the automatic instrumentation and management of monitored code on multi-processor architectures. If the systems include also Hw acceleration, it may be necessary to instrument it with custom Hw PMCs to provide a proper feedback to trigger reconfiguration.

One of the Hw reconfigurable infrastructures supported in CERBERO is the Coarse-Grain Virtual Reconfigurable Circuits (CG-VRCs) provided by MDC. CG-VRCs offer fast and low power reconfiguration, with a good trade-off between performance and flexibility, being suitable for providing run-time hardware adaptation. In this kind of systems, all the resources belonging to all the configurations are instantiated in the substrate and different configurations are enabled by multiplexing resources in time.

An Hw accelerator can be specialized by the designer to include custom monitors. This solution is not suitable for Sw developers who may have limited knowledge of the Hw design flow. Furthermore, if these solutions rely on custom methods to read the monitors, the process of reading the monitors in the Hw accelerators and the PMCs already available on the CPU could not be the same, and a heterogeneity of solutions, complex to be implemented, may be required. This integration relies on the idea of offering to Sw developers the support to design and implement run-time reconfigurable systems as the

CG-VRCs and, at the same time, to monitor both the processors and the Hw accelerators using a unified methodology based on PAPIFY.

Exchanged Data:

The Hw accelerator is modelled starting from a set of dataflow networks described in CAPH, that are parsed by MDC to generate a Xilinx-compliant IP able to execute all the different functionalities described by the input dataflow specifications, one at a time. The code incorporates the accelerator-level monitors as described in D5.5. Together with the accelerator, MDC generates the APIs to mask the communication with the accelerator. Furthermore, thanks to the developed configurable PAPI-compliant MDC-component, PAPIFY can transparently access the above cited Hw monitors.

This Hw accelerator is used by a Sw application, which is modelled as a Parameterized Interfaced Synchronous Dataflow (PiSDF) specification, using the design-time tool PREESM and the run-time manager SPiDER, that automatically integrate the PAPIFY necessary monitoring code. In this context, actors exchange tokens through edges depending on the feasible working points of the application scenario. With respect to the mapping strategy, SPiDER handles all Sw tasks taking into account the constraints given as input by the application designer. The Sw actors delegated to communicate with the Hw accelerator embed the code to talk to the accelerator, that exploits the APIs provided by MDC.

PoC: The monitoring capabilities of the combined SPiDER-PAPIFY-MDC design and management support are shown in an image-processing application scenario, involving a multi-functional accelerator for edge detection, able to compute two different algorithms: Sobel and Roberts. The setup features the MDC accelerator on a Pynq board running Linux, in which are installed SPiDER, PAPIFY and PAPI.

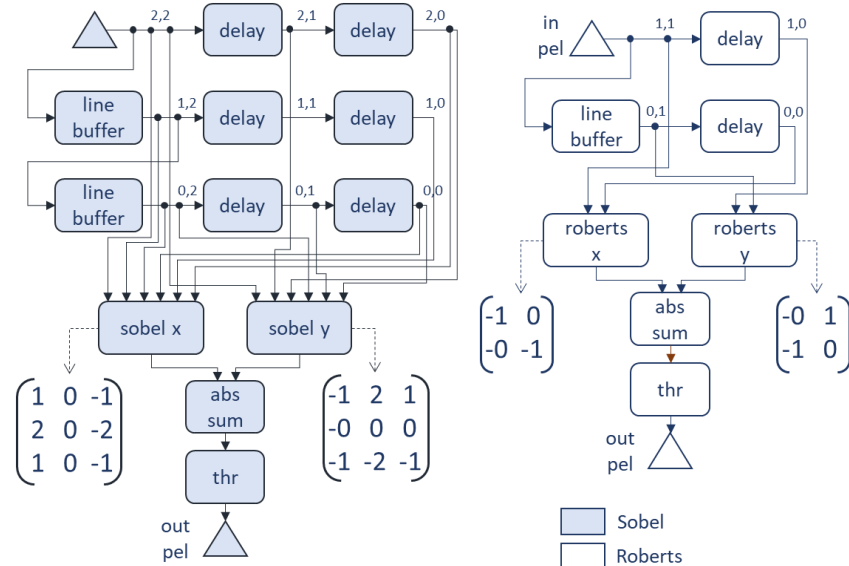


Figure 4.8 - Schematic graphs representation of the Sobel and the Roberts edge detectors.

The dataflows processed by MDC have been described in CAPH language. Figure 4.8 depicts a schematic graph representation of the Sobel and Roberts kernels. The *line buffer*

actors are adopted to store previous rows of the image, while *delay* actors are in charge of memorizing one previous pixel within a row. Once the actors are filled with the proper numbers of rows and pixels, the convolution actors can compute the horizontal and vertical gradients. Actor *abs sum* sums up the absolute values of the horizontal and vertical gradients and right-shifts the result for a given scaling factor n . Lastly, the thresholding actor *thr* sets to 255 all the magnitudes that are above a certain threshold (in this case it has been fixed to 80), while setting to 0 the others. The generated Xilinx-compliant IP is instrumented with 4 monitors at accelerator-level to keep trace of standard dataflow metrics during execution, such as the execution time, the number of input tokens and the number of output tokens.

This accelerator is adopted in a Sw application, modelled using PiSDF, in which both Sw and Hw monitoring are automatically inserted using the PREESM-SPiDER-PAPIFY design flow as described above.

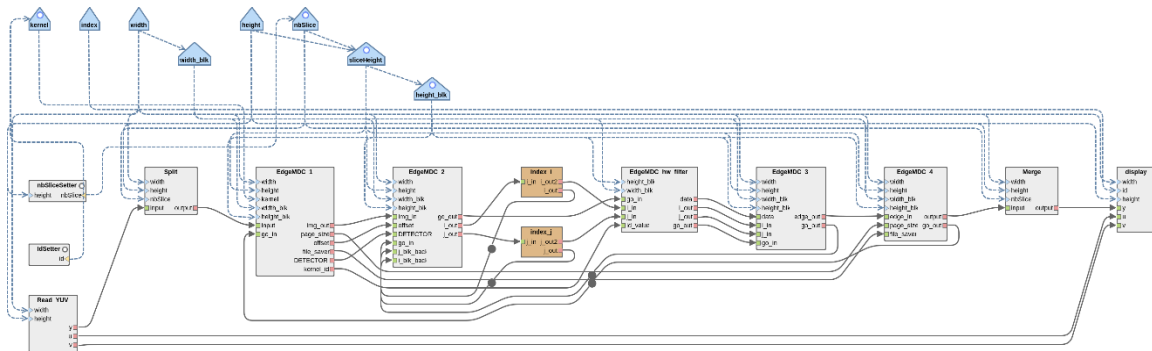


Figure 4.9 - Dataflow description of the Sw application, modelled using PREESM.

Figure 4.9 illustrates the developed Sw application.

- Given as input to the actor *Read_YUV*, a YUV video is read frame by frame, where the number of rows and columns correspond to height and width parameters respectively. Edge detection is applied only to the Y component, while the other ones are directly sent to be displayed.
- Before the edge detection, the block *Split* divides the image in slices depending on the degree of exploitable parallelism. In this PoC, having available one single Hw accelerator, no adaptation has been considered in this sense.
- At this point, verified the on-the-fly selected kernel (set by *IdSetter*) among Sobel and Roberts, an initialization phase is performed in *EdgeMDC_1*. In this phase, the processing data and the communication with the accelerator (through the APIs provided by MDC) are handled.
- Then, processing occurs by blocks of pixels of a size suitable for the accelerator specifications (in the assessed example, 32). *EdgeMDC_2* sends a number of blocks, to the *EdgeMDC_hw_filter*, which forwards the data to the accelerator. Therefore, *EdgeMDC_3* receives the result of each iteration, which is collected in *EdgeMDC_4*.

- Finally, the filtered frame is merged and displayed with the applied type of kernel and the execution time expressed in Frames per Second (FpS).

With respect to the mapping strategy, SPiDER handles all Sw tasks taking into account the constraints given as input by the application designer. In the evaluated case, the actors performing splitting and merging have to be executed onto the same core. Moreover, SPiDER has managed 305 instances of the single-rate graph. Indeed, 8 out of 11 actors are executed 1 time per firing, and 99 times per firing the other 3 ones (*EdgeMDC_2*, *EdgeMDC_hw_filter*, and *EdgeMDC_3*), since 99 blocks are present in the frame size (352x288 pixels).

The described Sw application has been mapped onto two cores. Specifically, *Display* and *Read_YUV* actors are mapped onto the *Core 0* while the others are mapped onto the *Core 1* of the adopted board. Among the actors mapped onto *Core 1*, three actors are repeated more than one time per firing: *EdgeMDC_2*, *EdgeMDC_hw_filter*, and *EdgeMDC_3*. Actors *Display* and *Read_YUV* are selected for the monitoring of the clock cycles and number of instructions events, while in the Hw accelerator the monitored events are the execution time (monitoring the clock cycles) and the throughput (monitoring the number of output tokens).

During the execution the monitored events are written in csv files to 1) analyse the application and 2) locate possible bottlenecks using PAPIFY-Viewer. Figure 4.10 illustrates the monitored timing for every actor and, as it can be noticed, *EdgeMDC_1*, *EdgeMDC_4* and *Read_YUV* are the actors taking longer. This is coherent with the reality because these three actors are the ones managing the whole frame. On the contrary, the actors being executed 99 times per iteration (*EdgeMDC_2*, *EdgeMDC_hw_filter*, and *EdgeMDC_3*), are among the fastest actors in the specification.

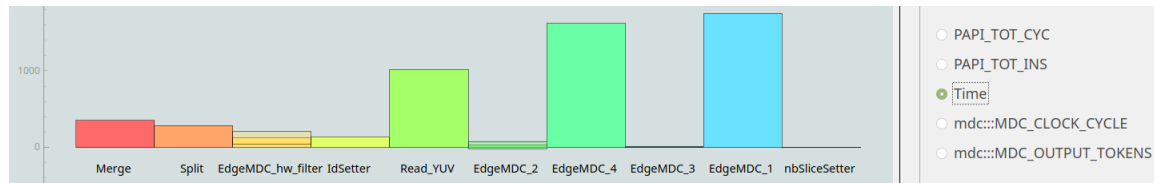


Figure 4.10 - Timing event

In Figure 4.11 and Figure 4.12, events associated to *perf_event* and *MDC* PAPI components are shown, respectively. Here, it can be observed that the events associated to the real execution of the Hw accelerator (*EdgeMDC_hw_filter*) are properly measured for the only actor associated to real Hw accelerator execution.

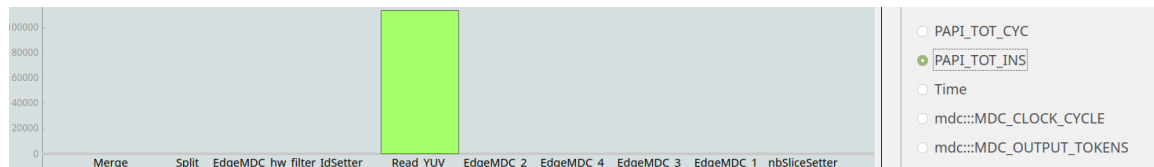


Figure 4.11 - PAPI_TOT_INS Sw event



Figure 4.12 - MDC_CLOCK_CYCLE Hw event

Useful material/links:

SPIDER-PAPIFY integration, presented at CF 2018: [link](#)

PAPIFY-MDC integration, presented at CF 2019: [link](#)

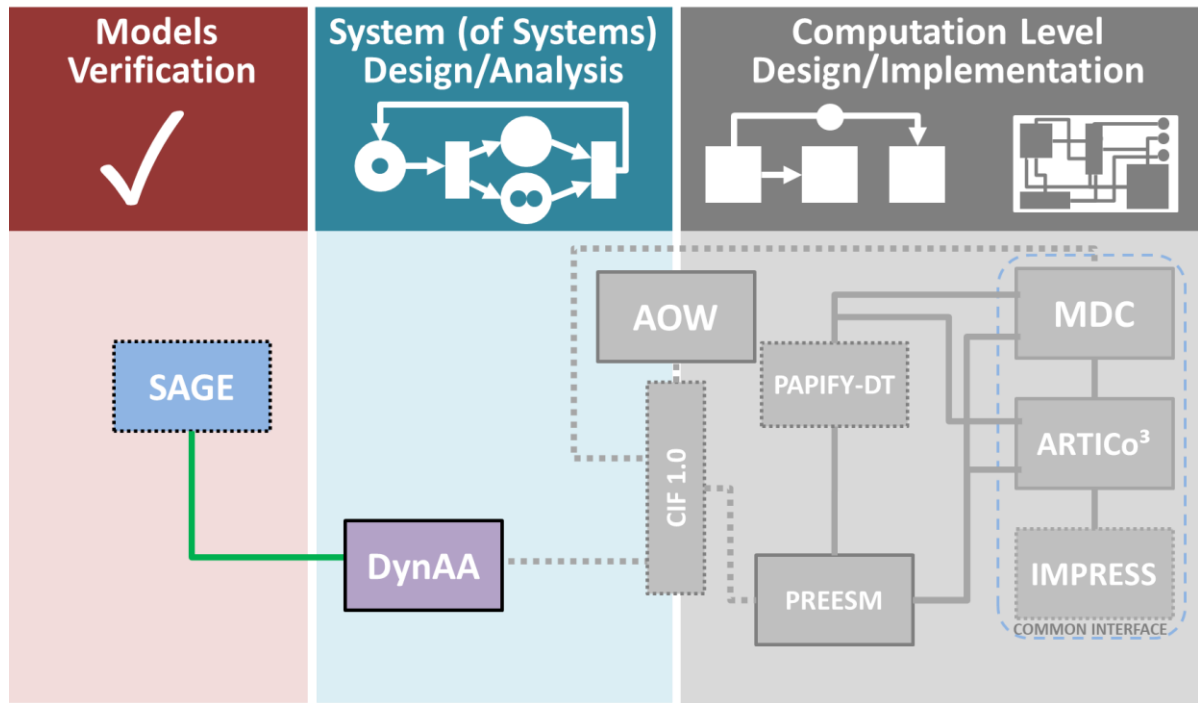
4.4. PoC Connection SAGE-ReqV – DynAA

Figure 4.13 – SAGE -ReqV - DynAA

Purpose of the Integration: Formal verification of the discrete part of a DynAA model with respect to a set of consistent specification formally checked with ReqV.

Exchanged Data:

DynaaGen is the tool designed to integrate ReqV and DynAA. It takes in input two text files: a Promela model, containing the discrete description of the system, and a property

file, containing the LTL requirements that the model should satisfy. The first file is provided by the designer, while the second one is produced by ReqV.

The provided output is a set of Java classes implementing the input model in DynAA.

PoC:

The integration between ReqV and DynAA is obtained with the pipeline depicted in Figure 4.14. The requirements are first collected and checked in ReqV. When this process is ended, and the whole specification is consistent, the requirements are translated into LTL formulae and passed to DynaaGen, along with the Promela model. DynaaGen first check every LTL property against the model with the Spin Model Checker. It reports a countexample if a property is violated. If the model satisfies all the requirements, DynaaGen automatically generates the Java Source Code with the Dynaa classes

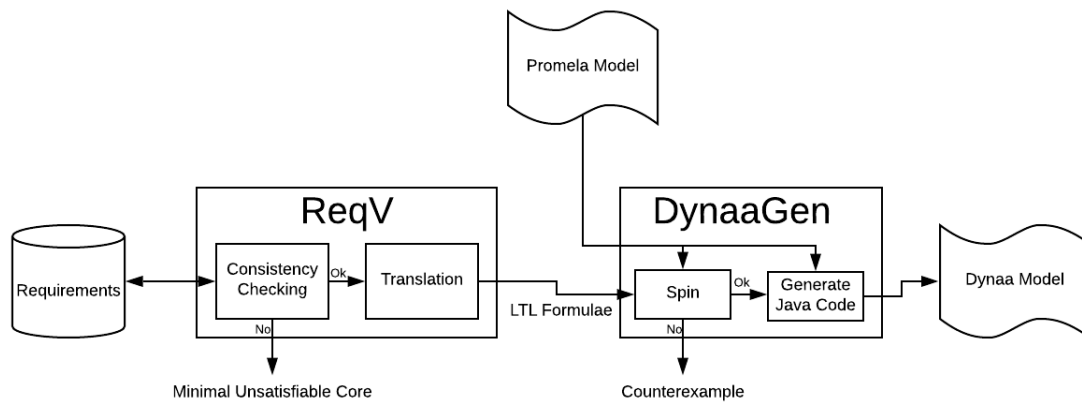


Figure 4.14: ReqV- Dynaa Integration Pipeline

corresponding to the same modelled system.

Figure 4.15 shows the definition of a client-server model written in Promela. In this example, two client processes try to enter a critical session at the same time, and a server process grant access to the critical session to only one client at a time. Figure 4.16 shows some requirements inserted and checked in ReqV. One of such requirements, for instance, is that only 0 or 1 processes can stay in the critical session in every time steps. DynaaGen read the Promela process and the requirements passed by ReqV, in LTL format, and check with the Spin model checker that such requirements are in fact satisfied by the model. Finally, DynaaGen generates the Java source code representing the same model in a Dynaa model. Since the generated Java code is quite verbose, only a fragment of it is shown in Figure 4.17.


```
byte cnt, request, respond

active proctype server()
{
    do
        :: (request != 0) ->
            respond = request
            (respond == 0)
            request = 0
    od
}

active [2] proctype client()
{
    assert(_pid > 0)
    do
        :: respond != _pid ->
            request = _pid
        :: else ->
            cnt++
            assert(cnt == 1) /* critical section */
            cnt--
            respond = 0
    od
}
```

Figure 4.15 - Client-Server Promela Model

Client-Server

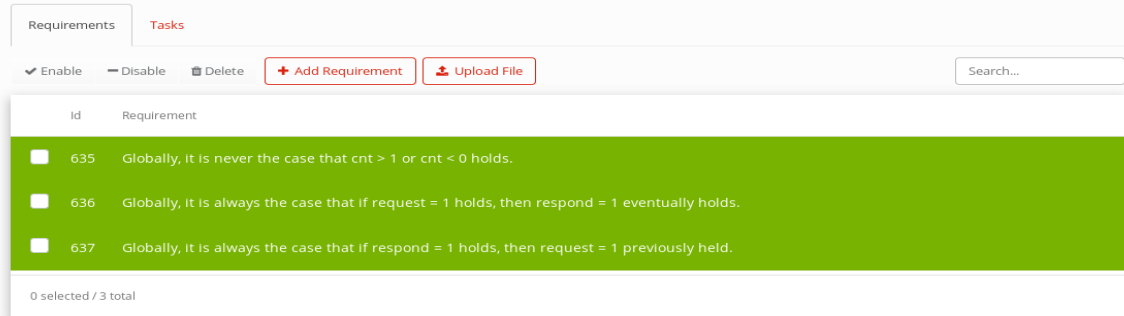


Figure 4.16 - Client-Server requirements in ReqV

```

public static Task build(int _pid, DataContainer globalVars) {
    final TransitionBehaviourSpecification behaviour = new TransitionBehaviourSpecification();
    final Task task = new Task(behaviour);
    task.setProperty("GLOBAL_VARS", globalVars);
    task.setProperty("_pid", _pid);

    final TaskSegment doSegment0 = ServerTask.createDoSegment0(task);
    final TaskSegment assingSegment1 = ServerTask.createAssingSegment1(task);
    final TaskSegment exprSegment2 = ServerTask.createExprSegment2(task);
    final TaskSegment assingSegment3 = ServerTask.createAssingSegment3(task);

    behaviour.setInitialSegment(doSegment0);
    behaviour.addTransition(doSegment0, SEGMENT_BRANCH1, assingSegment1);
    behaviour.addTransition(doSegment0, SEGMENT_BLOCK, doSegment0);
    behaviour.addTransition(assingSegment1, TaskSegment.SEGMENT_SUCCESS, exprSegment2);
    behaviour.addTransition(exprSegment2, TaskSegment.SEGMENT_SUCCESS, assingSegment3);
    behaviour.addTransition(exprSegment2, SEGMENT_BLOCK, exprSegment2);
    behaviour.addTransition(assingSegment3, TaskSegment.SEGMENT_SUCCESS, doSegment0);

    return task;
}

```

Global variables and process Id (`_pid`) passed as task properties

Segments Initialization

Task behavior definition as a set of transitions

Figure 4.17 - Fragment of the code generated by DynaaGen

Useful material/links:

Git repository of DynaaGen (<https://gitlab.sagelab.it/sage/dynaagen>)

4.5. PoC Connection PREESM - APOLLO multiversioning

The PoC explained below demonstrates the possibility and potentiality to connect CERBERO tools with external tools.

Purpose of the Integration: The combination of a dataflow framework such as PREESM with an external tool like APOLLO multiversioning reveals the possibility to exploit intra-actor optimizations which are initially hidden since actors are considered as primitives of the model. These new potential optimizations can improve the usage of (1)

available resources or (2) memory, either creating more threads or applying the right optimization. As a result, significant speed-ups can be achieved for all software parts of CPS systems modelled with dataflow models of computation without changing the mapping between actors and resources performed by PREESM. In particular, massively parallel computation with a fine granularity of parallelizable actors, such as the computation used in matrix operations, are particularly well suited to benefit from polyhedral optimizations.

Exchanged Data: An Application Programming Interface (API) is defined to embed APOLLO multiversioning run-time into the binary generated by PREESM

PoC: A PREESM application to multiply two matrices has been implemented. The actual PREESM implementation can be seen in Figure 4.18. This application contains four actors: two for data generation, one to perform matrix multiplication, and one to collect the result. Two different configurations will be demonstrated: (1) sequential configuration, with one thread, and (2) multithreaded configuration, with several threads to compute the matrix multiplication in parallel.

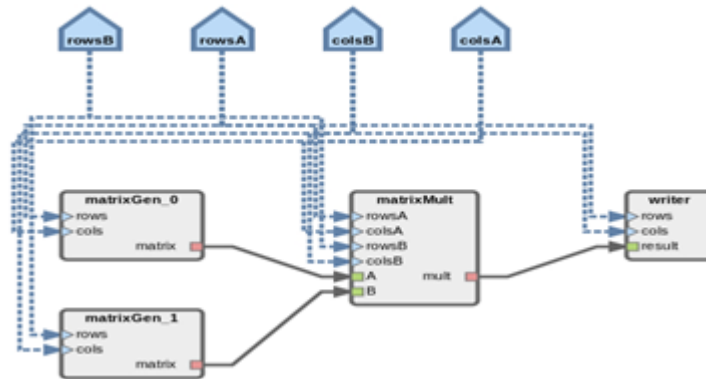


Figure 4.18 - PREESM application to demonstrate CERBERO polyhedral multiversioning mechanism

Speed-up results for both configurations and three different scenarios will be shown in the PoC: (1) when compiled with gcc, (2) when compiled with APOLLO and (3) when compiled with APOLLO multi-versioning. As can be seen in section 5.4 of D3.2: Models of Computation, speed-ups up to 20 are achieved when APOLLO multiversioning is employed.

Useful material/links:

APOLLO site: <http://apollo.gforge.inria.fr/about>

5. Standalone PoC

The PoC reported below is not a not a connection of tools but a standalone proof of concept of how IMPRESS can be used to implement just-in-time HW composition applications.

5.1. PoC IMPRESS for Just-in-Time HW Composition

Purpose of the Integration: As IMPRESS and Just-in-time HW composition applications have been developed from scratch in CERBERO, they have a lower TRL and are not as mature as other tools of the project. Thus, they are not going to be integrated in the use cases but are going to be validated using a PoC.

Exchanged Data: the just-in-time HW composition applications use the API provided by IMPRESS to build and reconfigure the overlays at run-time.

PoC: This PoC is divided in two different applications.

Deterministic JIT HW composition

The deterministic JIT HW composition consists in generating a custom accelerator from C code without using vendor specific implementation tools.

In this PoC the accelerator is described in a C function. In order to generate the accelerator, it is necessary to extract the Data Flow Graph (DFG) of the function. To do that, the C code is converted to LLVM intermediate representation (IR) [Lattner 2004] and from there the DFG is extracted. The dataflow graph generated should be a direct acyclic graph, which implies that only IR basic blocks (i.e. blocks without branches) can be transformed to valid DFGs. It is possible to transform the IR code using LLVM passes to convert a block with control statements into a basic block. For example, an if statement can be transformed into a comparison & multiplexer operation, and there are some loops that can be unrolled at compilation time. Therefore, the C function can contain some control flow statements.

In order to achieve JIT HW composition, the FPGA contains an overlay that implements a coarse-grain reconfigurable array. Each cell contains multiplexers in each cardinal direction so that it can be connected with its neighbor cells. Moreover, each cell contains a functional unit that can implement a set of operations. Therefore, the overlay can be configured to implement a functionality described in a data flow graph.

At design time it is possible to obtain the configuration needed to map the DFG into the FPGA overlay. This process has two steps, the first one is to place the nodes of the DFG into the overlay cells. Different cells of the overlay can only implement a subset of functions and therefore the nodes cannot be placed in every cell. The second step is to

route the cells of the overlay as described in the DFG. The configuration is then converted into a custom bitstream.

At run-time it is possible to use the previous bitstream to compose the accelerator. This placement of the cells is done by using IMPRESS partial reconfiguration to change the functional units of the cells. The routing is done using IMPRESS LUT-based partial reconfiguration to change the routing of the overlay.

All these steps are represented in Figure 5.1. In this PoC a simple C function is used to implement an accelerator.

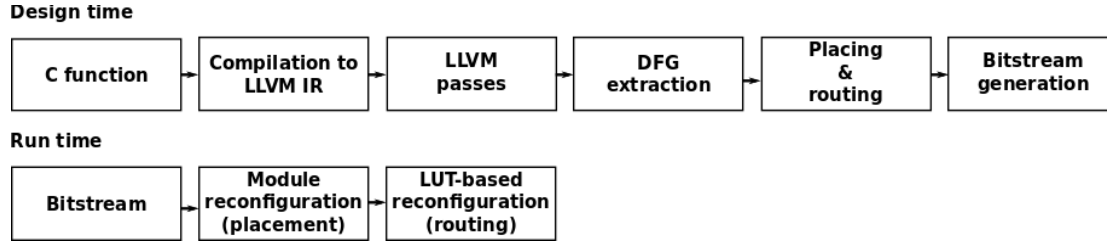


Figure 5.1 - Deterministic JIT HW composition flow

Evolutionary JIT HW composition

The evolutionary JIT HW composition approach uses a Block-based Neural Network (BbNN) to compose HW accelerators. A BbNN is a neural network that is composed by blocks that can implement 16 different configurations. Each configuration has different number of neurons (i.e., 1 to 3) and different interconnections. Figure 5.2 shows three different valid configurations.

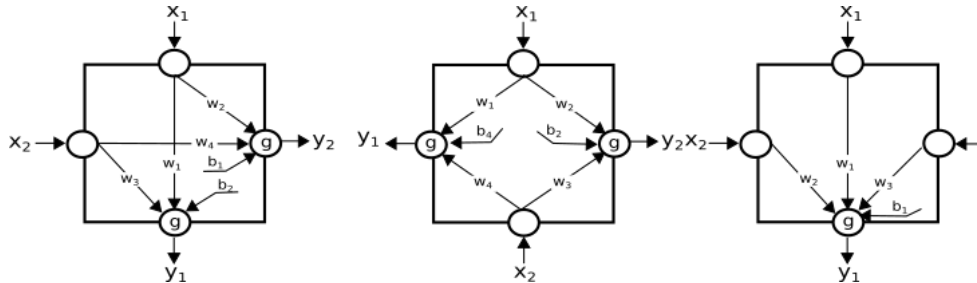


Figure 5.2 - Different block configuration

The BbNN is a 2D architecture composed with modular blocks that are connected to each other. Therefore, the BbNN is the perfect candidate to be implemented leveraging IMPRESS reconfiguration features. IMPRESS allows to reconfigure each block of the BbNN using sub-module reconfiguration, direct reconfigurable-to-reconfigurable communication and 2D grid composition, thus allowing to add blocks to scale the size of the BbNN at run-time. Moreover, the configuration of each block, the weights and the bias can be modified using the LUT-based reconfiguration capabilities provided by IMPRESS.

The block of the BbNN has been implemented with a reduced footprint to minimize the number of DSP needed. Using a finite state machine, it is possible to use just one DSP to perform the whole block calculation in 7 clock cycles. The block implementation is

shown in Figure 5.3. As it can be seen the weights, bias and configuration are implemented with reconfigurable LUT-based constants.

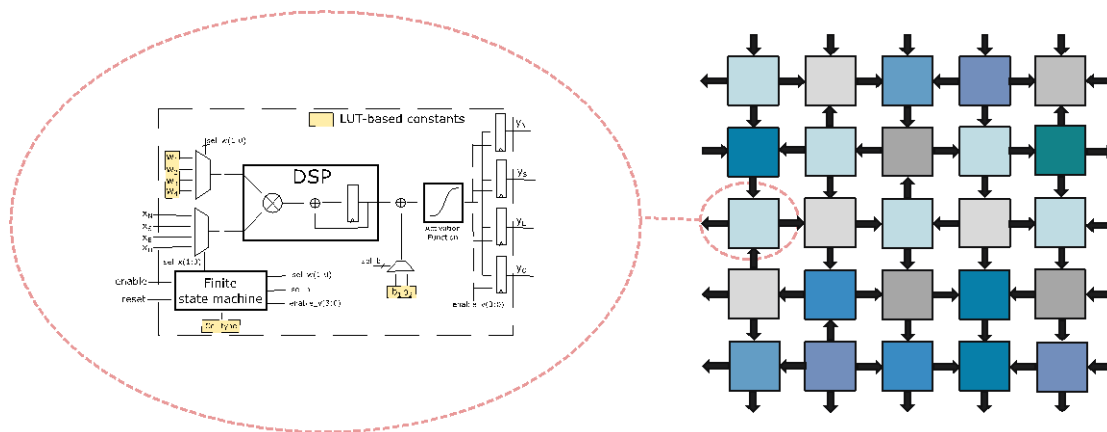


Figure 5.3 - Block implementation

In contrast to deep neural networks the BbNN is trained using an evolutionary algorithm which needs to find the weights, bias and configuration of each block. The main benefit of using an evolutionary algorithm is that the training and inference operations are the same and therefore the training stage can be done in the physical device. The algorithm that is used in this PoC is shown in Figure 5.4.

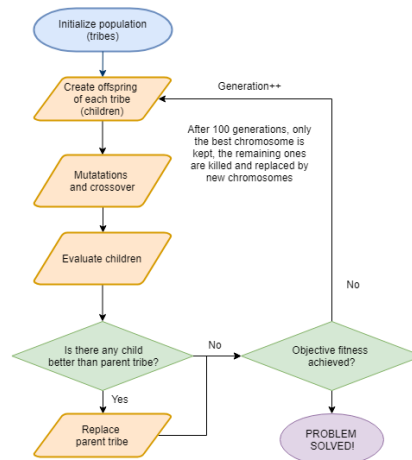


Figure 5.4 - Evolutionary algorithm

In this PoC the BbNN is used to generate an adaptable controller of a CPS system. The selected CPS is a discrete simulated cart pole obtained from the open AI gym toolkit [Gym]. The open AI gym toolkit considers this problem solved when the controller is able to control the cart pole for at least 200 steps. In this PoC we consider that a controller is valid when it can maintain the cart pole in an upright position for at least 1000 steps.

The PoC runs on a Zynq Z-7020 that combines a hard ARM processor and an FPGA fabric. The evolutionary algorithm run on the processor and the BbNN is implemented in

the FPGA fabric. The Zynq device is connected through an UART to a PC that runs the python simulation of the cart pole as shown in Figure 5.5.

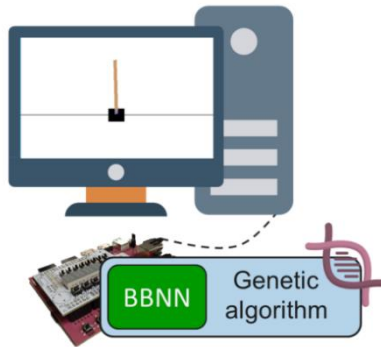


Figure 5.5 - BbNN and cartpole

The first step of the demo is to build the BbNN with a 3x3 size using partial reconfiguration and then use the evolutionary algorithm to find a valid controller. Once a valid controller has been found it is possible to change the cart pole parameters (i.e., varying the pole size). If the controller is no longer valid for the new cart pole, the BbNN is trained again to find a new valid controller. Therefore, this PoC shows how this approach can be used in lifelong learning applications.

Useful material/links:

[Zamacola 2018]

[Zamacola 2019]

References

- [SEMI] E. Shindin et al., *SEmantic Middleware presentation*, IBM Research - Haifa, Israel, 2018.
- [Parsons 2000] J. Parsons et al., *Emancipating instances from the tyranny of classes in information modelling*, ACM Transactions on Database Systems, 2000.
- [Palumbo 2011] F. Palumbo et al., *The Multi-Dataflow Composer Tool: a Runtime Reconfigurable HDL Platform Composer*, Conference on Design and Architectures for Signal and Image Processing, 2011.
- [JanusGraph] www.janusgraph.org
- [ThinkerPop/ThinkerGraph] <http://tinkerpop.apache.org/javadocs/3.2.2/full/org/apache/tinkerpop/gremlin/tinkergraph/structure/TinkerGraph.html>
- [Cassandra] www.cassandra.apache.org
- [ElasticSearch] www.elastic.co
- [Gym] www.gym.openai.com/
- [Lattner 2004] C. Lattner et al., *Llvm: A compilation framework for lifelong program analysis & transformation*, International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization, 2004.
- [Zamacola 2018] R. Zamacola, et al., *IMPRESS: Automated Tool for the Implementation of Highly Flexible Partial Reconfigurable Systems with Xilinx Vivado*, International Conference on ReConFigurable Computing and FPGAs, 2018.
- [Zamacola 2019] R. Zamacola et al., *Automated Tool and Runtime Support for Fine-Grain Reconfiguration in Highly Flexible Reconfigurable Systems*, IEEE International Symposium on Field-Programmable Custom Computing, 2019.

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",
```

WP5 – D5.3: CERBERO framework demo

```
"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)

WP5 – D5.3: CERBERO framework demo

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).