Ref. Ares(2018)4047127 - 31/07/2018

Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105

D5.6: *CERBERO Framework Components*

Lead Beneficiary: UniCA

Workpackage: WP5

Date: 05/04/2018

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.

© 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 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.

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**
Carlo Sau	UniCA
Francesca Palumbo	UniSS
Karol Desnos	INSA
Gasser Ayad	AI
Pablo Muñoz	S&T
Luca Pulina	UniSS
Eduardo Juarez	UPM
Ruben Salvador	UPM
Alfonso Rodriguez	UPM
Julio Oliveira	TNO
Michael Masin	IBM
Evgeny Shindin	IBM

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**
09/01/2018	0.0	UniCA	initial draft
18/01/2018	0.1	UniSS	revision of initial draft
05/02/2018	0.2	UniCA	revised structure
21/02/2018	0.3	UniCA, INSA	populated PREESM, SPIDER, MDC
28/02/2018	0.4	UniCA, AI, S&T	updated MDC, CAPH,MECA
05/03/2018	0.5	UniCA, UNISS, UPM	populated VT, PAPIFY, ARTICo³, JIT HW general revision
08/03/2018	0.6	UniCA, AI, TNO, UPM	populated DynAA updated ARTICo³, HW JIT

15/03/2018	0.7	UniCA, TNO, UniSS, MECA, IBM	updated DynAA, VT, MECA populated AOW document terms uniformity added acronyms appendix
04/04/2018	1.0	UniCA, TNO, IBM	review and integration of AOW
11/04/2018	1.0	UniSS	final review

Table of contents

1. Executive Summary

This document presents the main components/tools of the CERBERO framework, that is basically a design environment for Cyber-Physical Systems (CPSs) based on a cross-layer model based approach and on an advanced adaptivity support. Each framework component/tool will be described separately. In particular, the goal is to provide for each of them the following information:

- state of the art, context and motivation of the addressed problematic;
- main features: what the component/tool does, what the component/tool needs (inputs) and provides (outputs), how the component/tool can be used and how it is available to the users;
- role within CERBERO: what the component/tool brings into the project and how it can be exploited in different use cases;
- strengths and weaknesses with respect to the state of the art and to the use cases needs.

By the description of each component/tool it should be clear on which aspects it will be exploited/modified/extended in order to attain the CERBERO objectives and the use cases needs. Please note that some components/tools were already available before CERBERO. In this document the main motivations that led to the components/tools extensions will be presented before discussing the extension plans.

1.1. Structure of Document

The discussion will focus on the main components/tools that will be part of CERBERO framework and for which modifications, extensions or integrations have been planned. Sections from 2.1 to 2.10 describe in detail the characteristics and the modifications/extensions of each component/tool according to the set of information presented previously. For all the other components/tools that are simply used or interfaced with the components/tools of the CERBERO framework (but that will not be modified or extended within it and that may not formally belong to the partners within the consortium) a dedicated, final section (Section 2.11) is provided to briefly discuss their functionalities and how they are connected to the project. Lastly, Section 3 summarises the features and plans of the CERBERO framework components. Please note that since this document is full of acronyms we have added in this introductory section the most recurrent ones in Section 1.4, in order to make the content of the document more readable.

1.2. Related Documents

The CERBERO deliverables related to this document are:

- D2.7 – CERBERO Technical Requirements
    - The activities behind D5.6 contribute to satisfy the requirements listed in D2.7. Details are given in Section 1.3.
- D3.5 – Models of Computation

  - D5.6 deals with CERBERO framework components which are often based on specific models of computations that are described in D3.5, such as PiSDF for PREESM.
- D3.6 – Cross-layer Modelling Methodology for CPS
  - The cross-layer modelling methodology for CPS that has been described in D3.6 is related to D5.6. Several components/tools apply models on different levels or abstractions or system layers, which need to be connected, exchanging models and the related properties by means of the CERBERO framework integration (see Section **Error! Reference source not found.**).
- D4.3 – Multi-layer Runtime Adaptation Strategies
  - Lot of the tools/components presented in D5.6 already provide or will be extended to provide support for the multi-layer adaptation strategies discussed in D4.3, i.e. hardware reconfiguration offered by ARTICo³, JIT HW and MDC.
- D4.4 – Self-Adaptation Engine
  - As for D4.3, several tools/components presented in D5.6 already provide or will be extended to provide support for CERBERO self-adaptation infrastructure described in D4.4, i.e. runtime monitoring offered by PAPIFY.
- D5.7 – CERBERO Framework Demo
  - In D5.7 the integration of the tools/components presented in D5.6 will be discussed.

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 components described in the current document address 11 CTRs, as described in the following table.

CTR id	**CTR Description**	**Link with the D5.6 document on CERBERO framework components**
0001	CERBERO framework SHOULD increase the level of abstraction at least by one for HW/SW co-design and for System Level Design.	The level of abstraction will be increased by the support of PREESM for HW/SW partitioning purposes, which is raising the abstraction for MDC and ARTICo³ users.
0003	CERBERO framework SHOULD provide incremental prototyping capabilities for HW/SW co-design.	Incremental prototyping capabilities are envisioned at the tools/components level: • MDC has been enhanced with HLS support; • Dynamic Partial Reconfiguration features in ARTICo³ are on the way to be improved thanks to JIT HW implementation and composition tool; • runtime monitoring of ARTICo³, JIT HW and MDC reconfigurable hardware accelerators has been enabled

		thanks to the integration with PAPIFY; • the interconnection of PREESM with ARTICo³, JIT HW and MDC speeds up prototyping by facilitating HW/SW partitioning and co-design.
0004	CERBERO framework SHOULD provide software and system in-the-loop simulation capabilities for HW/SW co-design and System Level Design.	HW/SW co-design will be provided by interconnecting all the lower level tools (from application down to the lower level implementation layer). System-in-the-loop capabilities are a planned extension for DynAA.
0005	CERBERO framework SHOULD provide multi-viewpoint multi-objective correct-by-construction high-level architecture.	Several tools provide or will contribute to the multi-viewpoint multi-objective correct-by-construction high-level architecture: AOW, DynAA, PREESM/SPIDER and MDC (Structural Profiler).
0006	CERBERO framework SHOULD ensure energy efficient and dependable HW/SW co-design using cross-layer runtime adaptation of reconfigurable HW.	Energy efficiency and dependability through cross-layer runtime adaptation is achieved by the interaction among tools/components: • PREESM/SPIDER to perform design time and runtime HW/SW partitioning; • PAPI to provide runtime monitoring; • ARTICo³/JIT HW/MDC to enable HW reconfiguration of different dependable/energy efficient fabrics.
0013	All CERBERO API and most of CERBERO tools SHALL have open source license.	Some CERBERO framework tools/components are already open source (PREESM, SPIDER, PAPI and partially MDC), while open source is a planned feature for most of the remaining ones.
0016	CERBERO tools SHOULD be tested vs state-of-the-art	State-of-the-art comparison is provided for all the tools/components.
0019	CERBERO technology providers SHALL coordinate technical support for their tools with use case engineers.	Source repository, tutorials and documentation of the CERBERO framework components are discussed in D5.6. Tools to use case mapping is provided in this deliverable.
0020	CERBERO framework SHALL provide methodology and tools for development of adaptive applications.	D5.6 describes also components/tools involved in the adaptivity support and how they contribute to this latter.

1.4. List of Acronyms

In this section the most recurrent acronyms of the document are remarked:

- API – Application Programming Interface
- CG – Coarse-Grained
- CPS – Cyber-Physical System
- CPSoS – Cyber-Physical System of Systems
- DSE – Design Space Exploration
- DPR – Dynamic and Partial Reconfiguration
- FG – Fine-Grained
- GUI – Graphic User Interface
- HDL – Hardware Description Language

- HLS – High Level Synthesis
- HW – Hardware
- IR – Intermediate Representation
- KPI – Key Performance Indicator
- MoC – Model of Computation
- MPSoC – Multi-Processor System on Chip
- PiSDF – Parameterized and interfaced Synchronous DataFlow
- PMC – Performance Monitoring Counter
- SW – Software

2. The CERBERO framework components

An overview of the CERBERO framework components is depicted in

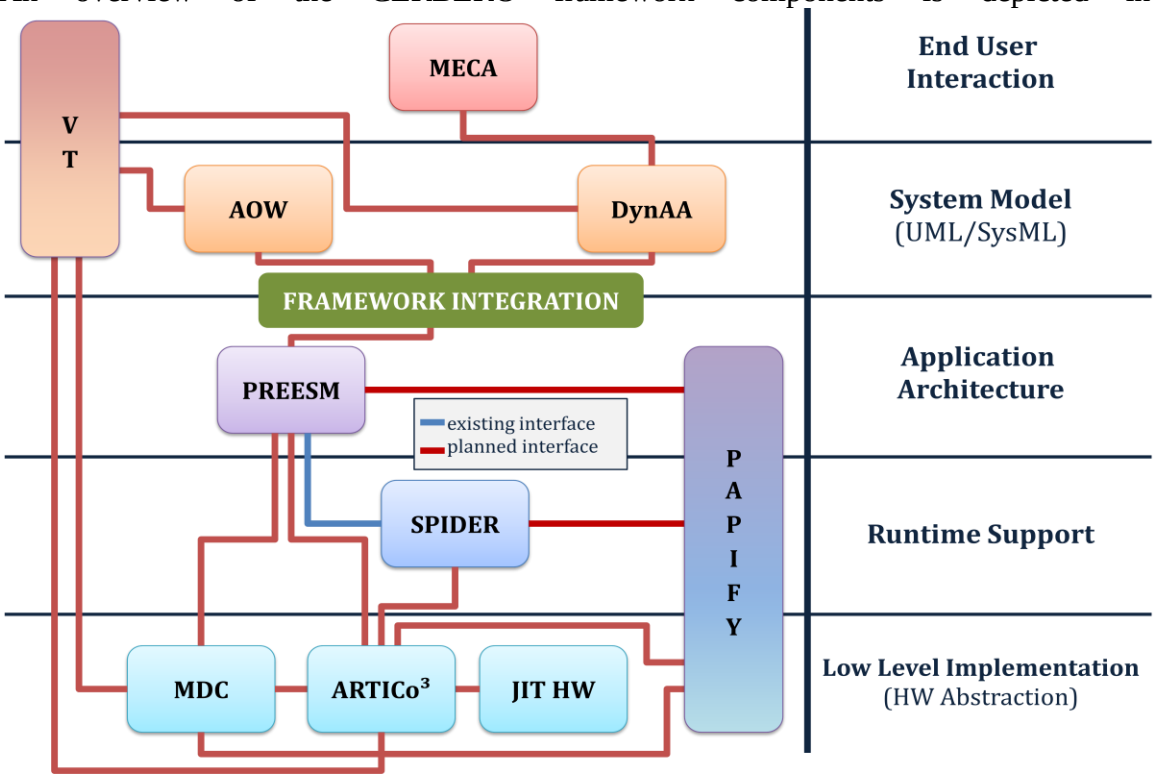

Figure 2-1. They operate at different levels of abstraction, going from end-user interaction level, as the Verification Tool (VT) and Mission Execution Crew Assistant (MECA), to the low level implementation (HardWare (HW) abstraction) one, as the Arquitectura Reconfigurable para el Tratamento Inteligente de Cómputo, Confiabilidad y Consumo de energía (ARTICo³) and the Multi-Dataflow Composer (MDC). Some of the components were already available and are extended within the CERBERO project; others are developed from scratch. Connections among tools have been already identified and the integration process, which can be either direct or based on the usage of an Intermediate Format as it is going to be specified in D5.7, is generally ongoing.

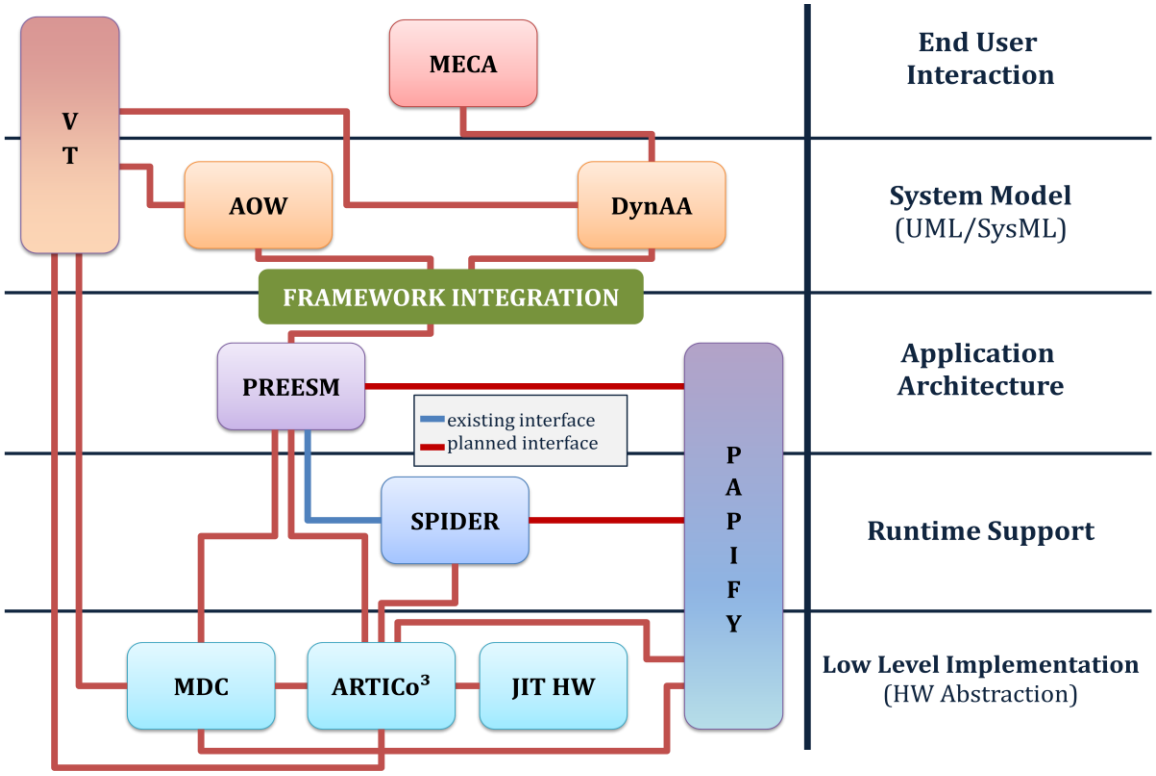

Figure 2-1: Overview of the CERBERO framework components.

2.1. MECA

a) State of the art

The origins of Mission Execution Crew Assistant (MECA) lie in an ESA research project that aimed to develop agent-based crew support for astronauts on deep space missions [Neerincx 2008]. MECA aims at a team composed of astronauts and robots that can perform well without immediate support from ground control, concerning both planned work and anomaly detection, integrating re-planning schemas to achieve the mission objectives. This is done based on the concept of *ePartner* [Neerincx 2010], an agent that has the objective of enabling decision making support to complete tasks (e.g. construction, exploration, etc.) based on monitoring of the relevant tasks' parameters while including the human in the decision loop. In this direction, the ePartner provides relevant information to the user, preventing cognitive overload during the task execution.

Systems that follow the ePartner concept can be found in the literature. For instance, the ABLE toolkit [Bigus 2002] proposes a framework to develop adaptive agents focused on self-optimization. PExA [Myers 2007] is a system developed to monitor and plan activities in office environments following a mixed-initiative planning process, which has slightly different constraints with respect to the scenarios covered by MECA. The work

of Duell [Duell 2008] presents an agent system to monitor user activities based on DESIRE [Brazier 2002] which is similar to MECA, but without considering interaction between multiple agents.

b) Features

MECA is an umbrella name for a collection of ideas, techniques and SoftWare (SW) components that aim to improve the resilience of human-machine teams. Based on these ideas, a *MECA unit* can be seen an implementation that takes the form of a smart digital assistant or ePartner.

Functionally, an ePartner is defined by its application domain, but typically almost every ePartner provides system, environmental and human monitoring and diagnosis, coordination with other ePartners and high level decision support in cases of unforeseen conditions and events. For this last characteristic, the ePartner is an event-driven agent that can behave autonomously to achieve the desired goals (deciding what to do basing on the current state) or providing a set of alternative ways to the human, allowing him/her to take the decisions. This means that a MECA ePartner performs online adaptivity on three distinct levels: system, environmental and human.

A MECA ePartner is built based on the following three key principles:

- *Semantics:* a knowledge-centric approach to enable reasoning about the data, which is stored and shared between ePartners through a shared Knowledge Base.
- *Services:* a loosely coupled architecture of hierarchical knowledge processing services, to ensure that the system is modular and extensible, based on a central toolbox of building blocks with clear interfaces.
- *Agents:* an ePartner capable of reasoning about the current execution status, providing goal-driven behaviour support to its users.

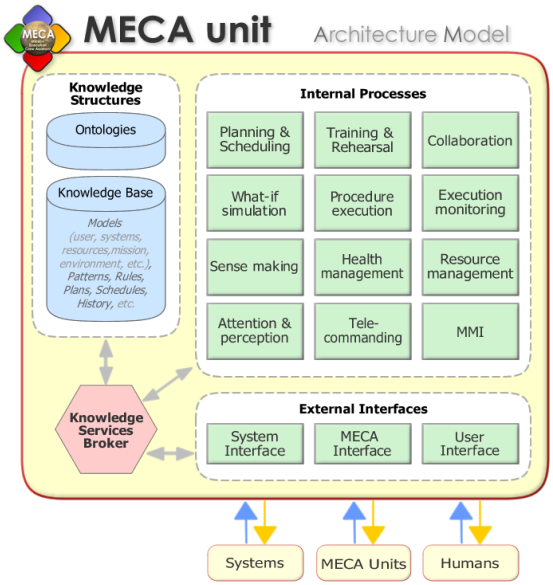

Figure 2-2 MECA ePartner architecture

The SW architecture of a MECA unit that supports these functionalities consists on the following components, depicted in Figure **2-2**:

- *Knowledge Structures:* a shared semantic Knowledge Base, built from ontology to store structured data.
- *Knowledge services broker:* an Application Programming Interface (API) and drivers for data monitoring/triggering and high level read/write with the Knowledge Base.
- *Internal processes:* depending on the application, various internal processes can be deployed. In general, an internal process is a component that extracts information from the Knowledge Base to perform an information enrichment operation to include additional information into the Knowledge Base. For instance, it is possible to perform resource usage prediction, simulation and future state inferencing.
- *External interfaces:* generic interface protocols, methodologies and design patterns for interfacing third party SW and HW systems. This includes Human Machine Interface (HMI) and interaction with other MECA units.

The last implementation of MECA, called MECA-HEART, has been used to demonstrate the MECA functionalities in the context of deep space missions with human-robot cooperation [Bosse 2017]. One of the tested scenarios entails travelling to a remote location and constructing a cache. In such scenario, an astronaut cooperates with a Eurobot robot to complete the different tasks in nominal and off-nominal circumstances. During all the activity, the MECA ePartner monitors the status of all actors and resources. If a certain resource matches a condition, an alarm raises and the ePartner includes the crew in the decision making process, proposing resource reallocations or alternative tasks to solve the current problem. One of the key points of the MECA ePartner is that the solutions presented to the user are properly explained, which improves the decision making process. While the demonstration of MECA relies in that domain, many of the higher level concepts used in MECA are more generally applicable to situations where humans and machines/SW work together as a team.

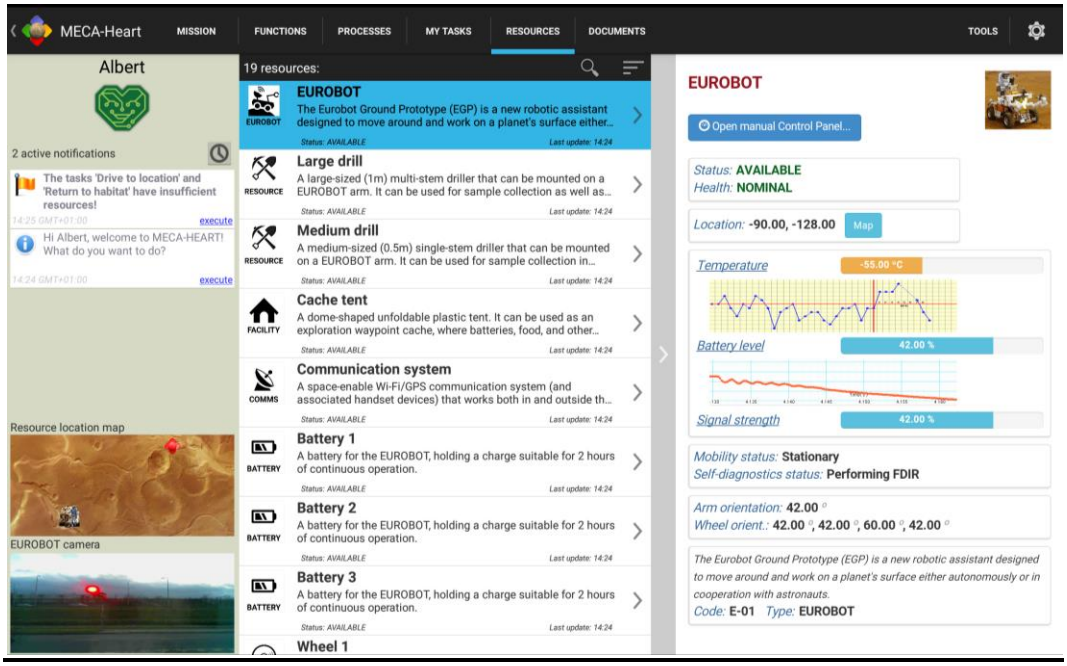

Figure 2-3 MECA-HEART application user interface.

Inputs:

The ontology defining the application scenario, the monitoring rules, procedural task descriptions and the initial data for the monitored parameters.

Outputs:

MECA-HEART is an online tool, so the output is determined by the current execution context. It consists of the parameters' values that are relevant to the current task in execution.

Reference Platform and GUI:

MECA-HEART consists of a server to store the database and an Android user application running in a tablet. The server is implemented in python and uses the Django REST framework [DjangoREST 2018], while the Android application is made in Java. The database information is accessible via web, for both the server and the application. This latter has an interface that is designed to be usable and friendly (see Figure 2-3).

c) Role in the CERBERO framework

MECA is the interfacing component between CPSs and the user, meaning that MECA includes the user as another component of the system for system-in-the-loop simulation. In this direction, the role of MECA in the CERBERO framework is to provide a runtime adaptation manager in those applications that require user-commanded adaptation, allowing the system monitoring and interaction the users and system level tools. The integration with other tools can be done ad-hoc or by means of sharing information through the framework integration.

d) Relevance with respect to use cases

Smart travelling: the smart travelling use case is a complex scenario involving CPSs of Systems (CPSoS) and the user and requiring adaptivity due to system (e.g. charging pole failure), environmental (e.g. weather conditions) and human (e.g. driver is tired) conditions. In this context, the MECA tool enables monitoring of the various levels, providing online adaptivity while including the user in the decision making loop when off-nominal situations occur. MECA will be integrated with other tools belonging to the CERBERO framework (DynAA and AOW) or not (SCANeR, map and information providers) in order to: properly predict and monitor the battery consumption during the trip (DynAA); simulate and monitor the car status (SCANeR); retrieve updated information on charging poles status and weather conditions for instance (map and information providers); calculate optimal solutions according to the available information for route calculation or charging options (AOW).

e) Strengths and gaps

A MECA ePartner serves as a communication intermediary between CPSs at various levels, from low to high level, with the following benefits:

- simplifies the interaction between the human and other systems;
- supports the human in the decision making loop;
- reduces the human cognitive overload providing only relevant information;
- provides event-driven models of computation;
- an online component that has been integrated with external tools;
- enables cooperation for CPSoS.

The points to be addressed to improve the MECA tool within the CERBERO project are:

- rules are currently hand-coded in Python, which makes the tool less reusable;
- the development methodology for different applications is not completely clear, increasing the time required to deploy prototypes;
- there are no decision making modules to cover the smart travelling use case.

f) Tool extension within CERBERO

The main tool extensions envisioned within CERBERO are:

- propose a formal definition of the monitoring rules based on a close to natural language;
- include a general rule monitoring engine for CPSoS;
- implement a route planner that considers the user profile (preferences) for the electric vehicle use case;
- integrate with other tools belonging to the CERBERO framework or not, i.e., DynAA and SCANeR, to provide system-in-the-loop simulation with user interaction;
- implement fuzzy logic support for the monitoring rules;

- include user profile learning components to enhance user profile based route planner.

2.2. VT

a) State of the art

Requirements are informal and semi-formal descriptions of the expected behaviour of a system. They are usually expressed in the form of natural language sentences and checked for errors manually, e.g. by peer reviews. Manual checks are error prone, time consuming and not scalable with the increased complexity of modern systems. In the context of safety- and security-critical CPS, it is therefore essential to automatically or semi-automatically analyze and check the consistency of requirements. The formalization of requirements would enable the application of formal methods techniques in assessing requirements correctness, completeness and consistency. Furthermore, the formalization of requirements can be employed in the subsequent phases of the design process, automatically synthesizing models [Fuxman 2004], tests [Clerissi 2017], verification artifacts, and even whole control algorithms [Liu 2013]. Unfortunately, the formulation of system properties in a purely mathematical fashion requires a high degree of expertise, difficult to find in practice, and it can create a barrier to the adoption of these techniques.

To deal with the problem of the requirements formalization, a common solution is the use of Property Specification Patterns (PSPs), first introduced by [Dwyer 1999]. PSPs are a collection of parameterizable, high level, formalism-independent specification abstractions usually based on a restricted English grammar. They provide an easy way to express properties of a system with an English-like syntax, while preserving a well defined semantic. Since the original work of Dwyer [Dwyer 1999], a considerable number of PSP systems have been proposed, grounding on different logics. PSPs have successfully been applied in many domains, such as automotive [Post 2012], aviation [Esteve 2012] and banking [Bianculli 2012].

b) Features

The Verification Tool (VT) is being developed by scratch during CERBERO. It aims at providing automated consistency checking of requirements expressed as PSPs with both Boolean variables and constrained numerical signals.

Inputs:

Set of requirements in natural (controlled English) language, formulated as PSPs for Linear Temporal Logic (LTL) extended to constrained numerical signals, as described in [Narizzano 2017].

Outputs:

Consistency result (yes/no). In the case of inconsistency, the tool returns the minimal set of requirements that causes the inconsistency.

Reference Platform and GUI:

The VT is being developed in Java and it is at the moment available as a standalone application from command line and via web. It has been planned to provide VT with a proper Graphic User Interface (GUI).

c) Role in the CERBERO framework

The VT is placed at the end-user and at the system model levels. It can be exploited for requirements verification at the early stage of the design process. It can also interact with other abstraction levels besides end-user and system model ones, such as implementation layer.

d) Relevance with respect to use cases

Actually the tool can be exploited for requirements and properties verification in all the three CERBERO use cases.

e) Strengths and gaps

PSPs used in VT have been extended with respect to state of the art LTL PSPs, as shown in [Narizzano 2017]:

- off the shelf LTL model checkers are used as back-engines, while VT is modular with respect to the used model checker;
- VT has been designed to be easily extended to PSPs in more expressive logical languages;
- a Minimum Unsatisfiable Core (MUC) extraction procedure has been implemented in order to provide the minimal set of inconsistent requirements (if the initial set is not consistent);
- a GUI allows its usage to a non expert user;
- VT is open source (https://github.com/SAGE-Lab/snl2fl, https://github.com/SimoV8/ReqV-webapp).

PSP-Wizard [Lumpe 2011] is a literature framework for machine-assisted definition of temporal formulae capturing pattern-based system properties. PSP-Wizard offers a translation into LTL of the patterns encoded in the tool, but it is meant to aid specification, rather than support consistency checking, and it cannot deal with numerical signals.

The work in [Konrad 2005] also provided inspiration to a recent set of works (see, e.g. [Dokhanchi 2018]) about a tool called VI-Spec, to assist the analyst in the elicitation and debugging of formal specifications. VI-Spec lets the user specify requirements through a GUI, translates them to Metric Interval Temporal Logic (MITL) formulae and then supports debugging of the specification using runtime verification techniques. VI-Spec embodies an approach similar to the VT one, to deal with numerical signals by translating inequalities to sets of Boolean variables. However, VI-Spec differs from VT in several aspects; most notably the fact that it performs debugging rather than consistency, so the

behaviour of each signal over time must be known. Also, VI-Spec handles only inequalities and does not deal with sets of requirements written using PSPs.

f) Tool extension within CERBERO

The tool is actually at an early stage of development because its design and implementation began in the context of the Task 5.2 activities. Further extensions will include more expressive PSPs and properties verification with respect to a formal model.

VT will be interfaced with several other components within the CERBERO framework:

- DynAA:
  - dealing with the same input model, VT will be capable of providing a feedback on possible errors appearing on the model, thus facilitating its correction within DynAA;
  - DynAA simulation features could be provided with monitoring capabilities through VT, so that design time verified requirements can be also validated at simulation time;
- AOW, allowing the verification of linear programming models on the bases of the IBM Cplex solver technology [Cplex];
- lower level tools (ARTICo³ and MDC) for automatic test pattern generation.

2.3. DynAA

a) State of the art

DynAA is a modelling and system level analysis tool, built on the top of a discrete event simulation engine. The design of this simulation engine follows established approaches [Banks 2004] [Nutaro 2010], but with special attention for dynamic model reconfiguration. This means that system models may freely change in their structure, composition, and behaviour during simulation. Such changes do not necessarily need to be previously described during the experiment setup phase or simulation design.

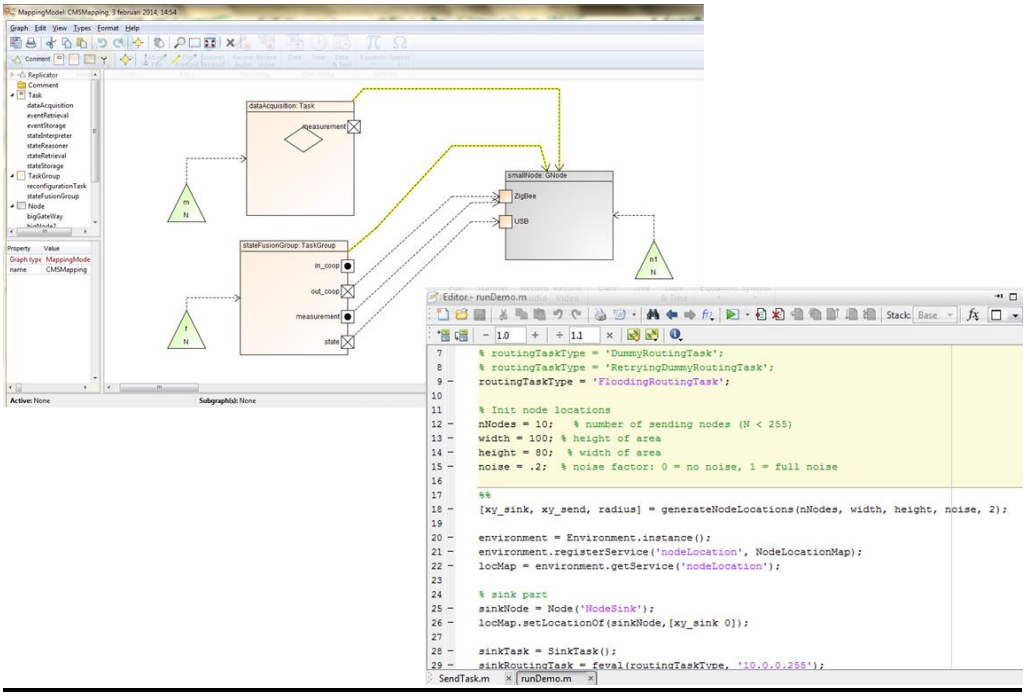

Figure 2-4: Interface of DynAA.

In most of the model-based system-oriented simulation tools, such as Matlab Simulink [Mathworks], and Modelica [Modelica], designers describe their component models, connect them, and start simulation runs. Within a single simulation run, the models and the system structure can only change if modification is previously described within models during experiment setup. The simulation engine in DynAA allows every component, communication link, or environment model to be included, deleted, and/or modified during simulation. Such capability makes DynAA specially interesting for the analysis and simulation of self-adaptive, self-reconfiguring, and self-evolving systems.

DynAA is implemented in Java, supports Matlab and has a graphical modelling interface realized with the commercial tool MetaEdit+. At the moment, this interface is being reshaped to get rid of the commercial dependency (and extra licenses) of MetaEdit+. Figure 2-4 shows a view of the graphical interface and some DynAA code in Matlab.

b) Features

DynAA [Oliveira 2013] [Leeuwen 2014] [Papp 2016] is a modelling and simulation tool originally conceived for designing large, adaptive, and networked (embedded) systems. DynAA enables system designers to design and model individual system components by defining their behaviour, interface, and non functional aspects (as for example energy consumed during operation and storage capacity). System components include physical devices, such as processor units or memory modules, and purely behavioural blocks, such as functions and SW processes (tasks).

Figure 2-5 shows the architecture layers of the tool along with a simplified UML class diagram that represents its main design concepts. The tool was designed along four

abstraction layers: the simulation engine or core, the meta model layer, the model library layer, and the user API layer.

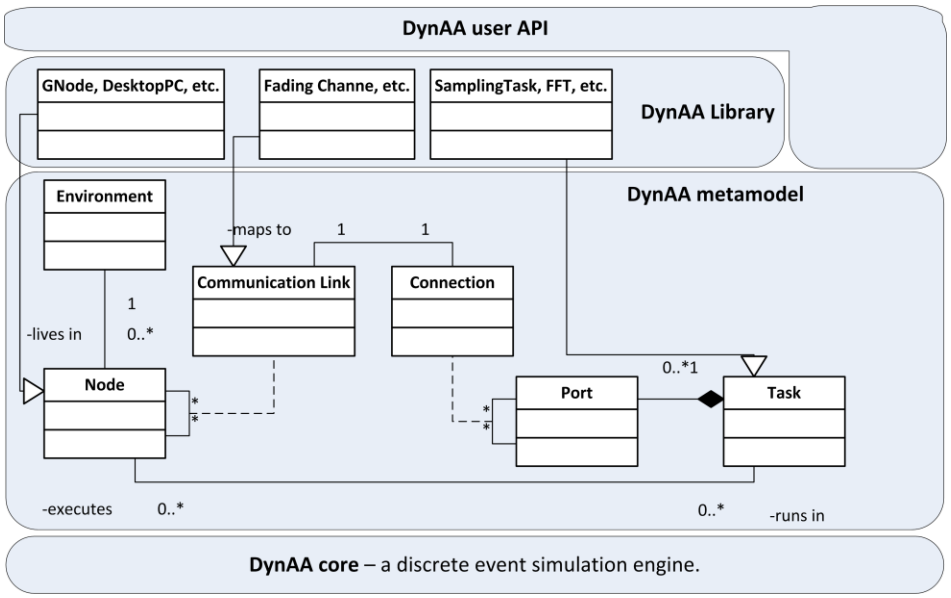

Figure 2-5: DynAA tool layers and basic concepts.

DynAA uses three fundamental models, as depicted in Figure 2-6:

- the task model, which captures the parallelism and the event handling;
- the physical model, which describes the HW configuration of the implementation;
- the function to task allocation (F → T mapping).

The task model (with the associated dataflow model) captures the programmatic properties of the design. Note that no HW and communication related properties are incorporated in the task model. HW related characteristics start playing a role when the task graph is mapped to the physical model: the task network is executed on a physical HW configuration consisting of processing nodes connected by communication links. The task model, the physical model and the F → T mapping jointly determine the system level characteristics, such as response time, throughput, energy consumption, reliability, etc.

Inputs:

DynAA receives as input three models (views) of the system, as depicted in Figure 2-6: the task model, the physical model, and the function to task allocation (F → T mapping model).

Outputs:

A system simulated in DynAA produces a simulation log that can be post processed to extract system Key Performance Indicators (KPIs) and their evolution during the simulation time. Typical plots that can be extracted from DynAA are energy consumption profile, communication latency and throughput for each channel, task activation rate profiles, reliability of the system, etc.

Reference Platform and GUI:

The DynAA simulation environment is a java program that does not have a GUI. Though it can be programmed by the user or integrated in a visualization environment (such as GAZEBO). The DynAA modelling environment is at the moment implemented in the commercial tool MetaEdit++, by MetaCase.

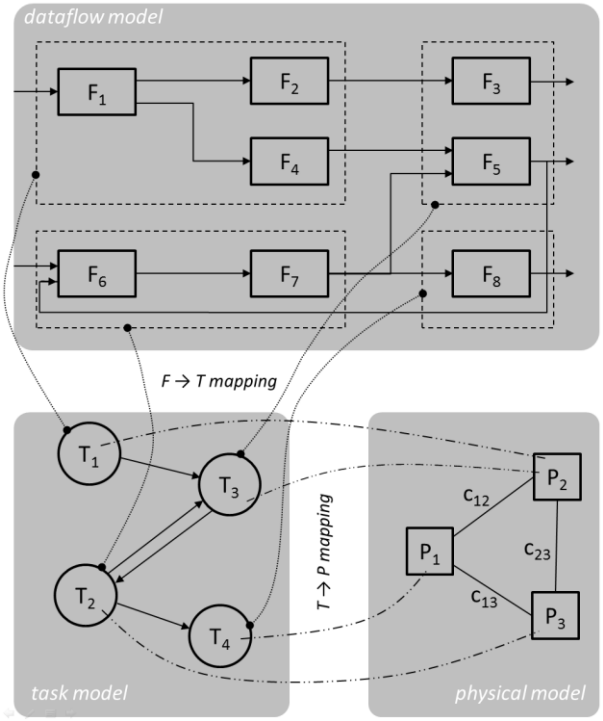

Figure 2-6: Way of modeling in DynAA.

c) Role in the CERBERO framework

Within CERBERO, DynAA is one of the system level design tools. It is used to model, analyze, and simulate aspects of the CPSs under design in early stages of the development. DynAA's added value is the early evaluation (indication) of KPIs, helping the decision making process at the system level design.

d) Relevance with respect to use cases

Smart Travelling: The use of DynAA for the design of CPSs is demonstrated mainly within the context of the Smart Travelling use case. The use case develops a routing planning system tightly coupled with the electric system of the car and with the battery charging grid infrastructure. In this context, two challenges appear where DynAA is the key solving technology:

- **Runtime solution space exploration:** the core of the routing system is a predictive model that allows estimating the consumption of energy that will be necessary to complete a route, and planning re-charging moments at appropriate

places and time spots. The design of this prediction module requires modelling and simulating (1) the physics of the car when driving on different terrains, (2) the electric power train including battery and electric engine, (3) the driver driving style, and (4) the conditions of the road. These elements are modelled in DynAA, which runs simulation to predict energy consumption and battery lifetime KPIs.

- **System in the loop simulations:** it will be also considered the case of developing the electric power train of the car. In this case, models of battery and the electric engine are simulated in DynAA providing a runtime, direct interaction with signals that come from the car (in the use case represented by the simulation environment SCANeR and the cockpit simulator in CRF).

Ocean Monitoring: DynAA usefulness for CPSs design will be also proved within the Ocean Monitoring use case, where it will mainly provide:

- simulation of models for complex cameras/lenses systems in order to optimize, with the aim of AOW, different system KPIs (such as response time, image quality or throughput);
- predictive models to estimate how long the battery will last during mission in order to trigger adaptation accordingly.

e) Strengths and gaps

The major strength of DynAA, as explained in details in the features subsection, is the complete dynamicity of the components in the model. Components may be created or deleted during the simulation as part of the behaviour. This feature makes the description of large, adaptive systems much easier.

As a simulation and analysis tool, DynAA is able to explore a large space of design solutions and system parameters. Such feature is enabled by choosing among an adequate heuristics or different optimization algorithms (based on Nelder–Mead, genetic algorithms, simulated annealing, and Monte Carlo methods) that guide the Design Space Exploration (DSE). The algorithm choice is made based on the type of the problem to be solved. Nevertheless, the process of DSE may take considerable time due to the number of simulations that have to be executed. The DSE under DynAA needs to scale in performance in order to tackle the design of systems with large number of components, such as a network of automobile communications, for example.

Moreover, at the beginning of the CERBERO project, DynAA simulations could only be carried out in isolation, that is, within the environment of DynAA. Such situation is not always desired during the design of CPSs, where simulations are required to work in tight coupling with real parts of the system under development.

f) Tool extension within CERBERO

In CERBERO, it will be explored the possibility of parallelizing the DSE used in DynAA. In this approach, the space to be explored (usually a large set of parameters) is divided in smaller segments and simulations for each segment are executed in parallel. That, combined with smart space exploration methods, will boost the DSE capabilities and allow it to be used also in runtime environments.

In CERBERO, system in the loop capabilities will be developed for DynAA. Such feature will make DynAA a more valuable tool during the design of CPSs, especially when simulations of components under development have to be carried out with parts of the system that already exist.

2.4. AOW

a) State of the art

Architecture Optimization Workbench (AOW) developed by a team from IBM Research lab in Haifa, Israel in order to bestow the power of optimization onto Systems Engineers [Broodney 2012]. AOW uses a unique combination of modeling approach, sound SW engineering and state of the art mixed integer linear optimization technology. Using AOW, engineers have the ability of evaluating hundreds to millions of potential architecture configurations in a matter of hours and to be able to support the architectural decisions with quantifiable benefits in driving cost and performance for the program. While the rest of the existing engineering optimization tools, being an assortment of domain specific solvers or search techniques runs in sequence, are best suited for optimization of design parameters of a known architecture, AOW allows multi-objective optimization of system's architecture topology using the strongest existing solvers, such as Cplex [Cplex].

Currently, AOW is implemented in Java as Rational Rhapsody [Rhapsody] plugin and integrated with MS Excel [MS Excel] and Pacelab Suite [Pacelab]. In current implementation, AOW uses Cplex [Cplex] to perform optimization.

b) Features

AOW is described in [Broodney 2012], [Masin 2013], and [Masin 2014]. In AOW, the system engineer can rapidly create the necessary system architecture, satisfying all functional and technical constraints needed to achieve the specified goals. AOW models the composition rules (also known as architectural patterns, or templates) of the required functional, physical, geometrical, project management and other system structures and relations both inside (data flow, energy flow, etc.) and between them (e.g. potential mapping between functions and physical components). AOW workflow is represented in a Figure 2-7.

For modeling purpose, AOW uses standard SysML with *concise profile* that builds on top of it. Concise profile allows defining different AOW concepts, including template architectures, different viewpoints and mappings, integration capabilities, optimization goals, constraints parameters and decision variables on top of SysML model, converting it to the *concise model*, i.e. template model, suitable for DSE purpose [Broodney 2012]. The potential physical components are imported from a library, along with geometrical data, if relevant for the use case. In current implementation such libraries can be defined using Excel and/or and Pacelab Suite. Optimization goals and constraints are specified as SysML constraints or Parametric Diagrams [Masin 2014], or via pluggable metrics that

are defined using special textual format on top of intermediate model representation known as SEMI. The tool uses all the inputs above in order to automatically generate SEMI representation of the concise model that is further automatically translated into a mathematical optimization program in OPL language [Hentenryck 1999] and solved by the IBM Cplex solver. Usage of SEMI allows performing modular changes and extensions of AOW. For example, AOW can be extended to use different tools and modeling languages for concise model definition or to produce optimization models in other languages, such as AMPL [AMPL], to be used with other solvers. Since there are multiple and usually conflicting goals, the optimization finds diverse Pareto optimal solutions (solutions where no goal can be improved without adversely affecting another goal) using special diversity maximization algorithm [Masin 2008]. This is the maximum effort that can be done automatically before the final human decision. Values of optimization goals from the set of optimal solutions (architectures) found by the optimization are represented to the decision maker using parallel coordinates diagram (Figure 2-8).

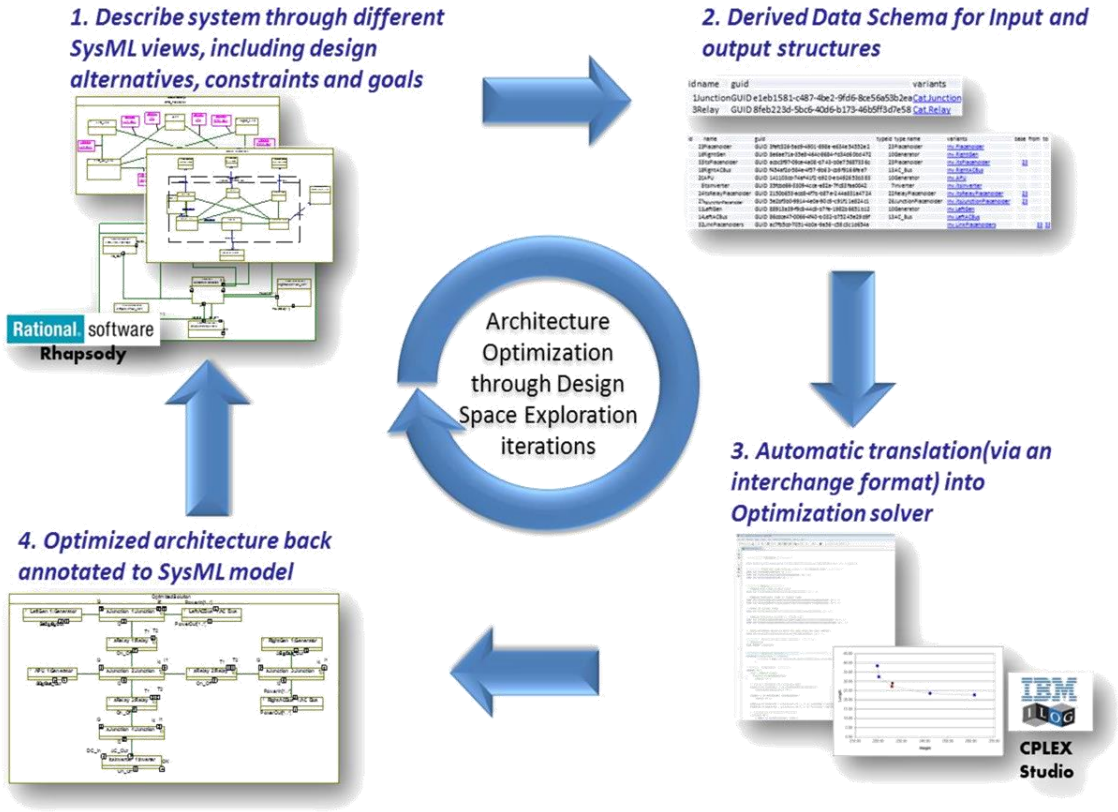

Figure 2-7: AOW workflow.

From this diagram, the system engineer could pick a subset of optimal solutions that will be automatically translated (back annotated) into the regular SysML model in order to be reviewed by the engineer. AOW interface enables: importing and editing data, adding constraints and objectives, and managing the optimization runs, including viewing the results and exporting them to the follow-on processes.

Inputs:

SysML concise modelling of the architecture problem including Rhapsody, Excel models and KPI metrics (algebra) templates.

Outputs:

Pareto frontier of optimal architectures.

Reference Platform and GUI:

AOW is developed in Java as an add-on to Rhapsody. It will be open source at the end of the CERBERO project.

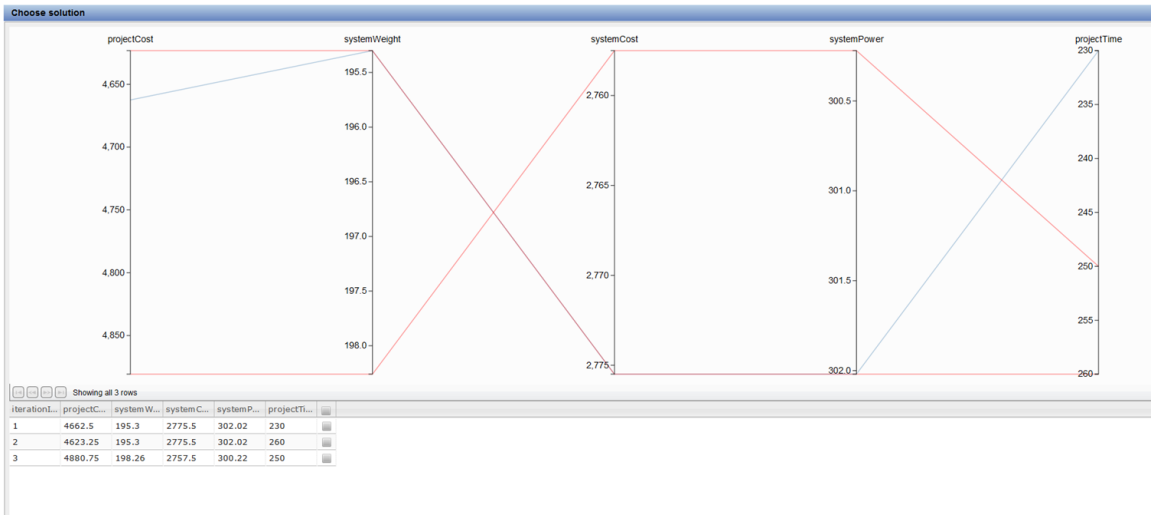

Figure 2-8: Parallel coordinates diagram adopted by AOW to show the optimization results.

c) Role in the CERBERO framework

AOW provides exploration of huge decision/design spaces, both as a standalone system level tool and as an internal optimization engine for HW/SW co-design, in collaboration with tools like PREESM or MDC.

d) Relevance with respect to use cases

Space Exploration: as an internal optimization engine for HW/SW co-design.

Smart Travelling: as an offline optimization engine for optimal routes used during fast online decision making later on.

Ocean Monitoring: as an optimization engine for the application architecture or as an internal optimization engine for HW/SW co-design.

e) Strengths and gaps

The main strength of AOW is in its ability to utilize concise modelling and pluggable viewpoints, making them available to domain specialists with minimal assistance from operations research experts. The main gap is to extend it to optimization of hybrid systems (see D3.6) without discretization.

f) Tool extension within CERBERO

In CERBERO AOW will be extended as follows:

- implementation of Simplex-based algorithm for Separable Continuous-time Linear Programming (SCLP);
- development of HW/SW co-design algebra using SCLP;
- development of Smart Travelling route optimization using SCLP.
- development of theory and proof of concept for Mixed Integer SCLP;
- development of theory and proof of concept for Robust SCLP.

2.5. PREESM

a) State of the art

The ever-increasing performance of embedded systems is driven by the introduction of low-power massively parallel architectures. At the same time, more than 80% of embedded SW is still written using procedural languages such as C/C++. Procedural languages are based on control-dependent sequences of imperative instructions. These characteristics make procedural languages inherently ill-suited for programming heterogeneous Multi-Processor System on Chip (MPSoC) architectures, where hundreds of heterogeneous processing elements communicate through complex on-chip interconnects and distributed memory architectures. Hence, a widening SW gap exists between the developer productivity and the increasing code complexity required to fully exploit parallel computing resources [Ecker 2009].

The main challenges to overcome in order to bridge the SW productivity gap are:

- to exploit enough algorithm parallelism (task, data and pipeline parallelisms) to minimize latency in general;
- to choose the right core for each application subtask;
- to provide data where and when needed so as to avoid stalling cores and under using.

Related tools and techniques used to address the problematic are:

- Parallel Programming APIs: Pthread, OpenMP, MPI, OpenCL, CUDA, GoLang, etc.
- Some Model-Based Design-space exploration tools: SynDEx [Grandpierre 1999], SDF3 [Stuijk 2006], Ptolemy II [Davis 1999], Silexica Studio [Leupers 2017], SpearDE [Torquati 2012].

b) Features

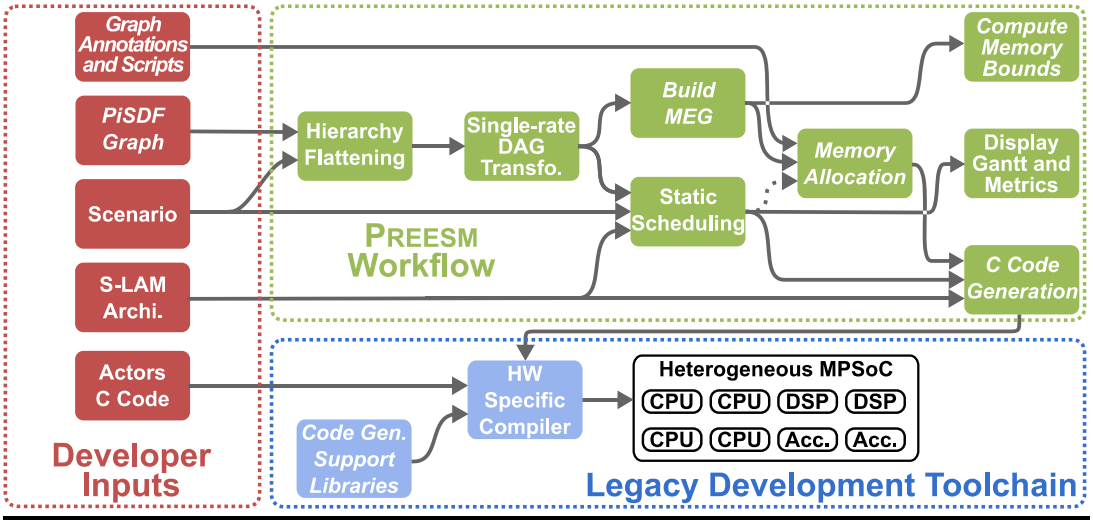

Figure 2-9 - Overview of the PREESM Workflow.

The Parallel and Real-time Embedded Executives Scheduling Method (PREESM) is a rapid prototyping framework that provides methods to study the deployment of dataflow applications onto heterogeneous MPSoCs [Pelcat 2014]. Figure 2-9 shows an overview of the development flow.

Inputs:

The specification of an application in PREESM is based on the following elements:

- **PiSDF Graph:** The behaviour of the application to deploy is specified using the Parameterized & interfaced Synchronous DataFlow (PiSDF) Model of Computation (MoC) [Desnos 2013]. This model fosters parallelism, compositionality and parameterization of the specified application behaviours. Only applications with a statically defined (i.e. not dynamically reconfigurable) behaviour are supported by PREESM. Application graphs are architecture-independent.
- **S-LAM Archi.:** The System-Level Architecture Model (S-LAM) provides a high level description of the platform on which the application has to be deployed [Pelcat 2009]. The objective of S-LAM is to model the characteristics (e.g. communication throughput, computation speed) of the elements composing the architecture with a low complexity and to enable fast simulation in order to reveal the bottlenecks of the system. Architecture graphs are application-independent.
- **Scenario:** The purpose of the scenario is to specify the deployment constraints for a pair of application and architecture.
- **Graph annotations and Scripts:** These optional inputs, specified by the developer, are used by PREESM to optimize the memory allocation of the application during its development on the multi-core target. Graphs annotations specify how dataflow actors read/write data from their input and output buffers,

and scripts give hints on how to minimize the memory footprint of dataflow actors.

- **Actor C code:** Each actor of the PiSDF graph is associated to a C file specifying the function that should be executed when this actor is fired. This input is not used by the PREESM workflow, but it is needed to run the application on the selected target.

The PREESM workflow consists of several modules, each responsible for a specific task or optimization in order to map the application graph on the targeted architecture:

- **Graph transformations:** The *hierarchy flattening* and *single-rate directed acyclic graph (DAG)* transformation are two successive transformations used to customize the granularity of actors and the degree of the application before the mapping and scheduling process.
- **Memory optimizations:** The *build MEG*, *memory allocation*, and *compute memory bound* modules are responsible for the modelling, the minimization and the optimality evaluation of the memory allocation problem, respectively.
- **Mapping/Scheduling:** The *static scheduling* and *display Gantt and metrics* modules are responsible for performing and assessing the distribution of the computations among the different heterogeneous cores of the architecture.

Outputs:

When executed for a given pair of architecture and PiSDF graph, specified in a scenario, the PREESM framework generates the following elements:

- **Log info:** The log info gives valuable information on the different optimization performed by the workflow. It notably informs the developer on the degree of parallelism of the application or on the efficiency of the memory optimization algorithms.
- **Code generation:** The purpose of this module is to translate the prototyping decisions made by PREESM into executable code for the targeted architecture, or into inputs for a Fine-Grained (FG) simulator.

Reference Platform and GUI:

PREESM is developed at the as a set of open source plugins for the Eclipse IDE.

c) Role in the CERBERO framework

As depicted in Figure 2-1, because of its rapid prototyping nature for component level applications (i.e. applications running on a single heterogeneous MPSoCs), PREESM acts as a natural interface between the system level tools and the lower level runtime and implementation tools. As such, PREESM will receive deployment scenarios of applications from the upper level tools, specifying the application graph to deploy, the targeted architecture, and a set of KPIs to optimize or constraints to respect. As a feedback, PREESM will provide an evaluation of the KPIs to the upper level tools, as a contribution to the CPS DSE at the system level. Depending on the deployment decisions made by PREESM, different back-ends will be used to implement and further optimize different parts of an application with different lower level implementation and runtime

tools. To better drive the optimization algorithm of PREESM, lower level tools (like PAPIFY) will also be used to feedback profiling information into PREESM optimization algorithms.

d) Relevance with respect to use cases

Planetary exploration: PREESM can be used to create an implementation of the computational part of the use case. By providing a parallel dataflow description of the robotic arm control algorithm, it will be possible to use PREESM to optimize several key KPIs of the application (time, memory, energy). Leveraging on the new connections of PREESM with PAPIFY, ARTICo³ and MDC, will also make it possible to use PREESM to drive and assess key steps of the DSE for this use case (e.g. HW/SW partitioning, automated profiling of applications).

e) Strengths and gaps

The features that differentiate PREESM from the related works and similar tools are:

- the tool is open source and accessible online;
- the algorithm description is based on a single well known and predictable MoC;
- the scheduling is totally automatic;
- the functional code for heterogeneous multi-core embedded systems is generated automatically;
- rapid prototyping metrics are generated to help the system designer to take decisions;
- the PiSDF algorithm model provides a helpful hierarchical encapsulation and parameterization, thus simplifying the scheduling;
- the S-LAM provides a high level architecture description to study system bottlenecks.

f) Tool extension within CERBERO

The main tool extensions envisioned within CERBERO are:

- direct connections with lower level tools: MDC, PAPIFY, SDSoC [SDSoC], ARTICo³ to provide heterogeneous adaptivity support for CPSs;
- connection with upper level tools through the CERBERO integration framework to provide higher-level of abstraction views of the addressed CPSs;
- implementation of CERBERO modelling contributions on PiSDF model (see D3.5 for more information);
- connection with AOW for the multi-objective optimization of multiple application deployments.

2.6. SPIDER

a) State of the art

The most commonly used runtime management systems found in embedded and CPS are OpenMP [OpenMP 2015], LLVM Runtime [LLVM], OpenCL [Khronos 2017]. The role of these runtime managers is to deploy applications on the fly on the available computational, communication and storage resources, by using greedy strategies. Information given to the runtime is mostly functional, and often transmitted as an IR heavily based on imperative MoCs. As shown in D3.5, imperative MoCs have a poor analyzability, and does not foster application predictability, which makes it very difficult for runtime systems to perform runtime optimization.

There exist very few runtime management systems dealing with applications specified through a well defined MoC and exploiting the analyzability of this MoC to make runtime decisions. SPIDER [Heulot 2014], leveraging on a specific MoC predictability and analyzability, has been able to outperform OpenMP for certain applications.

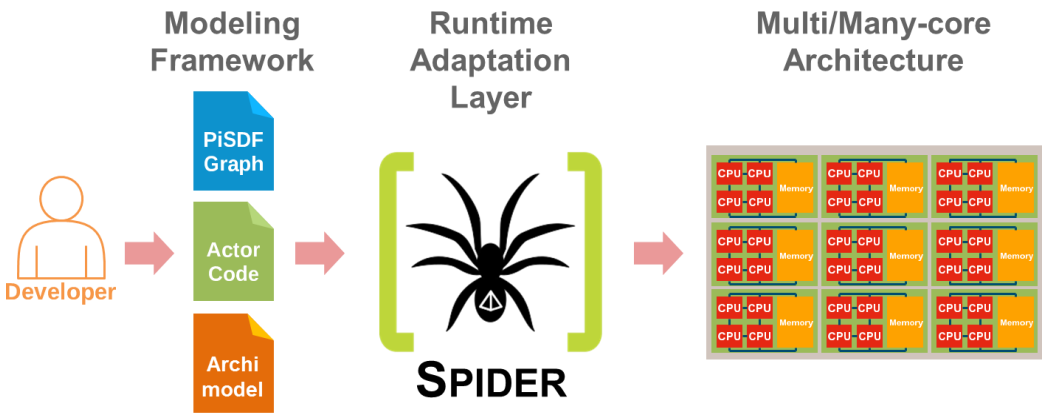

Figure 2-10 - Overview of SPIDER.

b) Features

The Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPIDER) was originally introduced in [Heulot 2014] as a runtime manager for the execution of reconfigurable dataflow graphs on heterogeneous MPSoCs. Figure 2-10 illustrates the simplified workflow of SPIDER.

Inputs:

The inputs used by SPIDER to manage the execution of an application on a multicore target are:

- **PiSDF Graph:** The PiSDF MoC [Desnos 2013] is used to specify the dynamically reconfigurable behaviour of the applications managed by SPIDER. This graph is designed using the PREESM editor, and transmitted to SPIDER as a

set of C++ files generated by PREESM. As in PREESM, PiSDF graphs are architecture-independent.

- **Actor Code:** Each actor of the PiSDF graph is associated to a C file specifying the function that should be executed when this actor is fired.
- **Architecture Model:** As in PREESM, the S-LAM model is used to give SPIDER information on the targeted architecture.

To deploy PiSDF application graphs on multi-core architectures, SPIDER relies on a master/slave structure and a set of communication queues:

- **Global Runtime (GRT):** This master processing element acts as the brain of the runtime. The GRT manages the PiSDF graph topology, performs graph transformations depending on dynamically defined values of parameters, and takes mapping and scheduling decisions. It is usually implemented over a general purpose core.
- **Local Runtimes (LRTs):** These lightweight slave processes are responsible for executing actors assigned to them by the GRT. LRTs can be implemented over heterogeneous types of processing elements: general purpose or specialized processors, accelerators.
- **Queues:** A set of communication First-In, First-Out queues are used to implement communications between the GRT and the LRTs. One *job* queue per LRT is used by the GRT to transmit actor execution commands. A set of *data* queues are used by LRTs to exchange data tokens produced and consumed by dataflow actors. A *parameter* queue is used by LRTs to feedback dynamically resolved parameter values to the GRT. A *timing* queue is used by LRTs to feedback actor profiling information to the GRT. Queues can be implemented in HW or in SW depending on the targeted platform.

Outputs: The SPIDER runtime produces the following output when executing an application on a given architecture:

- **Trace:** SPIDER gives information on the mapping and scheduling decisions it makes when running, and also keeps tracks of the amount of resources (memory, queues, etc.) it uses.
- **Gantt diagram:** On completion of the graph execution, SPIDER can output a multi-core Gantt diagram of the measured start and end execution times of actors.

Reference Platform and GUI:

SPIDER is developed as an open source project and is currently compatible with any architecture supporting the *pthreads* API. Additionally, HW-specific implementations of SPIDER have been developed for Kalray MPPA many-core architectures, and for Texas Instruments Keystone II heterogeneous digital signal processing chips.

c) Role in the CERBERO framework

SPIDER is a key element of the CERBERO adaptive runtime layer. As such, SPIDER will receive deployment scenarios of applications from the upper level tools, mainly via PREESM and the CERBERO framework integration, specifying the application graph

and code to deploy, the targeted architecture, and a set of KPIs to optimize or constraints to respect. The reconfiguration engine of SPIDER will be used to support the dynamic adaptations of the application, notably by managing its deployment on available HW resources (heterogeneous processing elements, memory, etc.).

d) Relevance with respect to use cases

Space exploration & ocean monitoring: SPIDER can be used to support adaptive behaviour of algorithms depending on data captured by the sensors of the system. An example of use case occurs in systems where it can be detected that sensed data is not exploitable by the system (e.g. too noisy), and thus requires low/no computations for its processing. In such a case, if this adaptivity is properly exposed at the dataflow graph level, SPIDER can redeploy the application to minimize its resource overhead, thus lowering the energy footprint of the system and freeing resources for other services executed by the system. Adaptivity can also be used to trigger reconfiguration of the system in cases when faulty HW is detected, thus providing self-healing capacity to the system.

e) Strengths and gaps

The features that differentiate SPIDER from the related works and similar tools are:

- the tool is open source and accessible online;
- the expressiveness of the model supported by the tool is limited, giving its strength to the runtime, but also preventing the representation of some applications;
- optimizations performed at runtime by the tool have an overhead on the application performance, which must be compensated by performance optimization;
- the tool currently support general purpose and embedded heterogeneous targets.

f) Tool extension within CERBERO

The main tool extensions envisioned within CERBERO are:

- support for real-time extension of dataflow MoCs from CERBERO;
- connection with PAPIFY to take energy monitoring information into account for runtime decisions;
- connection with MDC and ARTICo³ for supporting CPU-FPGA heterogeneous targets;
- runtime optimization to reduce the runtime overhead in cases of infrequent system reconfigurations, e.g. for self-healing purposes;
- connection with the CERBERO integration framework for KPI exchanges.

2.7. PAPIFY/PAPIFY VIEWER

a) State of the art

In literature, several tools have been proposed to perform performance monitoring in running systems. They are used to analyze the execution of applications on target platforms, which requires a deep understanding on the underlying architecture. Abstracting away these details and offering a standard API that helps accessing and gathering this HW monitoring information coming from Performance Monitoring Counters (PMCs) [Terpstra 2009] is the objective of the Performance API, PAPI, on which PAPIFY is based and built upon. Even though PAPI can be used as a standalone tool for system and application analysis, it has been widely employed as a middleware component in profiling, tracing and sampling toolkits such as HPCToolkit [Adhianto 2010], Vampir [Knüpfer 2008] and Score-P [Schlütter 2014]. Using PAPI, the PMCs can be transparently accessed to analyze profiling information such as memory usage, code parallelization, workload associated to each PE, I/O utilization, etc... Additionally, some other parameters, such as power or energy [Ren 2013, 2014], can be estimated combining this information. Having these performance indicators transparently extracted would contribute not only to improve designers' productivity, but also to achieve an iterative design flow that can be more easily integrated with tools for rapid prototyping like PREESM and runtime managers like SPIDER.

b) Features

PAPIFY is a tool that implements an event-based performance monitoring in RVC-CAL dataflow applications [Bhattacharyya 2011]. PAPIFY integrates the PAPI into the Open-source RVC-CAL Compiler (ORCC [ORCC]). PAPIFY analyses in detail the performance of an implementation in a processor-based platform. PAPIFY Viewer is a visualization tool to monitor the actions of actors in RVC-CAL specifications. Fired actions can be analysed chronologically from either an actor or a partition point of view. In addition, PAPIFY Viewer can also generate event histograms. PAPIFY employs the annotation syntax defined in ISO/IEC 23001-4 to signal the instrumented actors and actions. Annotations are a common mechanism in the RVC-CAL language to drive the compiler behaviour. In order to profile an actor, annotations of the form *@papify(ListOfEvents)* are employed, where the *ListOfEvents* is a non empty, comma-separated sequence that comprises any of the *preset* events of the PAPI.

PAPIFY Viewer is a tool written in Processing, a programming language for visual applications, that helps in the analysis of the activity file created with PAPIFY. Due to the enormous amount of information typically generated in the activity file, the use of visual tools is recommended to get an insight of the action traces obtained during the execution of an RVC-CAL specification. PAPIFY Viewer can additionally generate per-actor, per-action and per-partition histograms of events.

Inputs:

PAPIFY employs as input a network of RVC-CAL actors. To assess all actions of an actor, an annotation should be included before the actor interface declaration. The format of the annotation is the following:

`@papify([event1], [event2], …, [eventn])`

where each:

`[eventi]`

is a PAPI preset event. At least one event should always be included. With PAPIFY, it is possible to assess specific actions of an actor and remove the rest of them from the evaluation. To do so, the following annotation shall be included above the actions whose performance is required to measure:

`@papify`

Furthermore, to add the events to measure, the same annotation employed for the actor assessment shall also be included:

`@papify([event1], [event2], …, [eventn])`

In this way, the annotated actions can be instrumented with the events indicated at the actor level.

For PAPIFY viewer the input is the activity file created with PAPIFY.

Outputs:

Once the execution of a specification instrumented with PAPIFY has finished, a folder with the name *papi-output* is created. This folder is located in the folder */bin* of the ORCC generated folders. Within the papi-output folder, the output files of each of the instrumented actors are written. These files are the input to the PAPIFY Viewer tool.

PAPIFY viewer generates a chronological view per actor of the activity of a specification. In addition, PAPIFY Viewer can generate per-actor, per-action and per-partition histograms of events.

Reference Platform and GUI:

PAPIFY is integrated within the C back-end of ORCC, that is a set of open source plugins for the Eclipse IDE. It can be used targeting those platforms in which the PAPI can be installed. PAPIFY Viewer is a standalone program that runs independently from PAPI.

c) Role in the CERBERO framework

The main role of PAPIFY within the CERBERO framework is serving as the performance monitoring tool. As such, by a seamless access to the underlying HW and SW computational resources, it will allow developers/users to gather performance information. This is to be provided through a new abstraction layer added to PAPI within CERBERO, a library called *eventLib*, enabling this way a transparent access to standard PMCs in processor cores and specific HW monitoring infrastructure (specifically added in the HW accelerators coming from MDC and ARTICo³). By having access to this information, PAPIFY can be leveraged both at design and at runtime. At design time, it

will assist in rapid prototyping by gathering runtime information that can assist in the DSE done by PREESM. Besides, at runtime, it is a key element to drive system self-adaptation through its combination with the runtime manager SPIDER. The information provided by the PMCs in processor cores and their HW accelerators counterparts, is fed back to the embedded system models, so this way the produced KPIs can be used by the Adaptation Manager to drive the change through the different Adaptation Engines (for more details see D4.3).

d) Relevance with respect to use cases

Space exploration: being the access point to the runtime performance information produced by the different computational elements of the system, PAPIFY is a key element in its adaptivity. Through the embedded system models and the Adaptation Manager, the information obtained by PAPIFY can be used to predict the power consumption (using the prediction models included in the Manager) for the given configuration of the system (itself and the sensed environment). This way it might help in leading the system towards operation points that reduce its energy footprint by deploying a new graph as commanded by SPIDER. This can vary from changing the degree of parallelism, to switching the computational element actually executing the graph (or a part thereof) or even discarding the completion of lower priority tasks in order to extend system lifetime. Very important also, as the data obtained by PAPIFY contains information on faults occurring in the *cyber* part, it is the first step in the chain that triggers system self-healing. Hence, combined with the HW/SW self-reconfiguration capabilities of the system, together with the intelligence included in the Adaptation Manager, PAPIFY provides the self-awareness infrastructure required to fulfill a fully autonomous CPSs self-adaptation.

e) Strengths and gaps

The weaknesses of PAPIFY and PAPIFY Viewer from the related works and similar tools are:

- depending on the granularity of the monitored actor, the introduced overhead can slash the original performance;
- monitoring accuracy depends on the number of monitored events when this number is greater than the number of available PMCs, multiplexing techniques are employed and, consequently, some inaccuracies are incurred);

while the strengths are:

- high level abstraction of the monitoring process in heterogeneous systems;
- graphical selection of KPIs or events to be monitored at the actor level independently from the actual type of resource allocated, HW or SW.

f) Tool extension within CERBERO

The main tool extensions envisioned within CERBERO are:

- PAPIFY integration into PREESM to instrument with selected events the automatically generated SW implementation;
- PAPIFY integration into MDC and ARTICo³ to instrument with selected events the automatically generated HW implementation;
- PAPIFY integration into SPIDER to provide at runtime monitoring information to be used as input of its adaptivity core functionality;
- inclusion of KPI estimators in PAPIFY to directly provide the estimated values from event occurrences when the infrastructure to measure KPIs is not available at the CPS platform;
- Real-time plotting of event occurrences and KPI estimations on heterogeneous platforms with PAPIFY Viewer.

2.8. Just-In-Time HW Composition Implementation Tools

a) State of the art

Dynamic Partial Reconfiguration (DPR) increases the flexibility of FPGAs design enabling the change of the instantiated accelerators in real time. This design flow is based on the definition of a static system that cannot be changed at runtime and one or multiple reconfigurable partitions (RPs), where different accelerators can be allocated in real time.

For each reconfigurable accelerator it is necessary to generate a Partial BitStream (PBS), defining the configuration of the region of the FPGA where the accelerator was initially generated. However, it would be desirable to enable the reallocation of each accelerator in different RPs of the device. Commercial tools, like Vivado from Xilinx, need to create one PBS for each RP, thus if an accelerator must be allocated in three different partitions it is necessary to implement the circuit 3 times and so 3 different PBS will be generated. This affects both the implementation time and the in-system memory requirements.

Another constraint imposed by the commercial tools is that it is not possible to have multiple vertical partitions in the same clock region. Here, the main disadvantage is that it constraints the type of virtual architecture that can be used (understanding that a virtual architecture is mainly the division of the FPGA resources in different RPs)

There are some tools that have been developed from academia to tackle these problems. Most of these tools have been implemented for older FPGAs and design environments [Otero 2012] or they do not allow sub-clock region reconfiguration and waste some resources [Rettkowski 2016]. Apart from the generation of the PBS compatible with module relocation and sub-clock region reconfiguration, it is necessary to count on a reconfiguration engine compatible with these features. For this reason, new tools will be developed within the CERBERO framework.

b) Features

There will be two different tools:

- an implementation tool;

- a runtime reconfiguration engine.

Implementation tool

Inputs:

- VHDL/Verilog files that define the static system;
- VHDL/Verilog files that define the reconfigurable modules;
- information about the virtual architecture;
- interface of the RPs.

Outputs:

- static and reconfigurable bitstreams.

Reference Platform and GUI:

- TCL scripts.

The implementation tool consists of a set of TCL scripts that automatically carry out the process of synthesis and implementation in Vivado in order to obtain re-locatable PBS. In contrast to Vivado reconfiguration flow it is possible to define RPs without knowing in advance the static system and also to define the static system without knowing the content of the RPs. The only thing that is needed is the information of the partition coordinates and the interface that it will have with other partition or the static module. This tool ensures that circuit from the static system are perfectly isolated from RP in such a way that the only connection between them is made through the specified interfaces. This is the main characteristic needed to get re-locatable bitstreams. It will also allow the communication of a reconfigurable module to another reconfigurable module, something that is not allowed with commercial tools.

Runtime Reconfiguration engine

Inputs:

- bitstreams prepared to be reconfigured and coordinates of the region where the bitstreams will be allocated.

Outputs:

- reconfigured FPGA.

Reference Platform and GUI:

- not applicable.

Once we have the PBS generated and stored in a non volatile memory (e.g. SD card), it is necessary to have a reconfiguration engine that, given some coordinates, is able to allocate the bitstreams in the desired position.

In order to achieve sub-clock region reconfiguration, it is necessary firstly to read the configuration memory, and then combine this information with the new PBS. This is necessary because bitstreams need to be allocated for an entire clock region column at a time, thus if the PBS does not occupy the complete column it is necessary to compose the bitstreams to only modify the configuration bytes that need to be changed.

The reconfiguration engine will be defined in two different ways:

- as a C library;
- as a HW peripheral.

c) Role in the CERBERO framework

These tools will be used in conjunction with ARTICo³ to reduce memory usage and implementation times. In addition, they will allow the possibility of creating new tools for Just-In-Time (JIT) HW composition. The goal is to implement 2D mesh type layouts, which have proved their efficacy in dataflow computing algorithms, and to use these layouts to compose HW in real time. At this regard, two possibilities are envisaged:

- Deterministic HW composition from IR: the idea is to be able to design HW from high level programming languages. High level descriptions will be transformed to an IR where it is possible then, on one hand, to compile it to a specific SW core or, on the other hand, to map and route predefined processing elements in the layout to obtain the HW accelerator. In this way, an Adaptation Manager will be able to seamlessly switch tasks between SW and HW (see D4.3 for more details).
- HW composition based on iterative algorithms: this composition is based on the autonomous evolution of HW accelerators (by changing the basic modules in the 2D mesh) to imitate a given functionality. Reinforcement algorithms will be explored in order to solve problems that affect CPSs, for example adaptive controllers.

d) Relevance with respect to use cases

Planetary Exploration: These tools will be used in conjunction with ARTICo³, improving the implementation and memory requirements as it has been explained before. JIT HW composition tools that can be developed from them would improve further adaptability and fault tolerance of this use case application scenario.

e) Strengths and gaps

The main strengths of these tools are:

- reduced implementation time: there is no need to implement an accelerator for compatible RPs every time;
- reduced memory footprint.

The main weakness of these tools is:

- as routing is more constrained, it is more difficult to route the design and it is possible that bigger partitions are needed.

f) Tool extension within CERBERO

These tools will be designed from scratch in CERBERO.

2.9. ARTICo³

a) State of the art

CPSs face increasingly complex and demanding application scenarios where, in some cases, computing performance requirements cannot be met by low-end microprocessors. However, while processing requirements have increased, power/energy consumption is still highly constrained and limited. In this regard, FPGAs can prove beneficial thanks to their runtime reconfiguration capabilities, making it possible to have time multiplexed, high-performance application-specific computing platforms. Nevertheless, DPR techniques need to be coupled with intelligent power-management strategies in order to still meet low-power goals.

Some alternatives to the solutions provided by the Arquitectura Reconfigurable para el Tratamento Inteligente de Cómputo, Confiabilidad y Consumo de energía (ARTICo³) can be found in the literature:

- architecture level: Recobus [Koch 2008], HWThreads [Wang 2012], GUARD [Zhang 2014];
- design/implementation level: Go Ahead [Beckhoff 2012], Dreams [Otero 2012];
- runtime level: ReconOS [Agne 2014].

These solutions make use of DPR to offer reconfigurability as an added value to electronic systems with HW acceleration. Differently, ARTICo³ aims at also providing flexibility in order to trade off three factors: scalability in performance, its associated energy consumption and fault tolerance. The runtime support for this is a solution beyond the state of the art.

b) Features

The ARTICo³ framework provides three components: a HW-based processing architecture, an automated tool chain to build mixed HW/SW systems based on that architecture, and a runtime library to manage their execution.

The mentioned runtime tradeoff between computing performance, energy consumption and fault tolerance is achieved by a combination of module replication using DPR (i.e. HW *copy and paste*) and an optimized, dynamic datapath (called *Data Shuffler*) that changes to meet application requirements at runtime. A Direct Memory Access (DMA) enabled communication infrastructure includes dedicated HW modules to extend the functionality of the architecture (e.g. voter unit to support module redundancy for enhanced fault tolerance). Figure 2-11 shows the block diagram of the architecture, which is further described in D4.3.

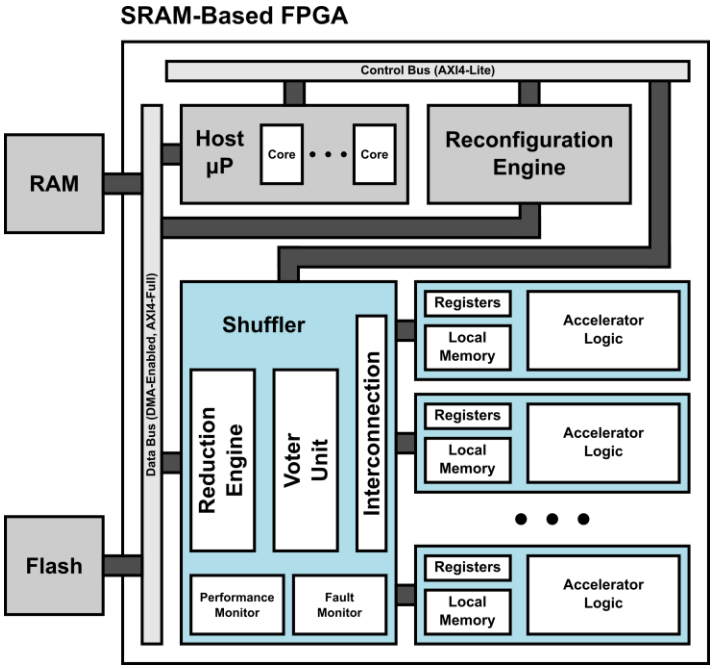

Figure 2-11 - The ARTICo³ architecture.

Dynamically reconfigurable HW designs are usually not accessible to most embedded system designers. The ARTICo³ tool chain provides an automated way to build custom accelerator-based computing systems starting from C/C++ (useful for designers with no prior expertise in low level HW design) or Hardware Description Language (HDL) kernel descriptions. In this regard, a kernel is defined as any program section with both computing-intensive and data-parallel behaviour. Hence, applications are a combination of sequential host code (SW) and a set of computing kernels (HW accelerators). Moreover, the ARTICo³ runtime library acts as an interface between these last components, providing standard function calls to modify the computing fabric (load HW accelerators), allocate shared memory buffers, or start the execution of a given kernel.

In addition, the ARTICo³ framework provides a full self-monitoring stack, from PMCs to SW API calls to read them. Current measurements include execution times per accelerator, accumulated errors and, whenever the dedicated measuring infrastructure is available, device power consumption. Moreover, it includes lightweight estimation models that, when combined with execution profiling using the PMCs, enable dynamic solution space (i.e. all possible combinations of computing performance, energy consumption and fault tolerance) exploration at runtime.

Inputs:

- **Host Application Code:** C/C++ code describing the SW application that runs in a host processor. Requires specific API function calls (included in the framework) to use the HW-based coprocessor infrastructure.
- **Kernel Code:** C/C++ code when using High Level Synthesis (HLS), HDL code otherwise, that describes the computing-intensive, data-parallel functionality to be accelerated using dedicated reconfigurable HW resources.

- **(Optional) Reference Design Template:** system developers can provide the required files to generate systems with a different set of HW IP cores, targeting a new FPGA device, or with a modified FPGA resource floorplan.
- **(Optional) Reference Application Template:** system developers can provide the required files to build SW applications with additional libraries, or for a different OS.

Outputs:

- **FPGA Configuration Files:** binary files containing the information required to program the implemented digital circuits in the target device. The tool chain generates configuration files for both static region (i.e. does not change during circuit operation) and RPs (i.e. each slot, which can be modified at runtime).
- **Application Executable:** binary file that runs in the host processor, offloading the computing-intensive and data-parallel operations to the HW accelerators available in the FPGA.

Reference Platform and GUI:

- **Design Time:** currently, the ARTICo³ tool chain relies on a set of scripts that run in command line mode in a Linux-based operating system. Output products are generated using vendor-specific tools (Xilinx Vivado).
- **Runtime:** currently, ARTICo³-based systems require an embedded Linux-based operating system in the target platform, which is also vendor-specific (Xilinx Zynq-7000).

c) Role in the CERBERO framework

ARTICo³ is located, together with MDC and FG reconfiguration for JIT HW composition, at the lowest level of abstraction in the CERBERO framework. These three elements are the three target fabrics that offer adaptivity at HW level. ARTICo³ provides adaptive and scalable HW acceleration but using a different approach than MDC to HW reconfiguration. In ARTICo³, the computing substrate, i.e. the FPGA, is actively altered to change the available functionality using DPR. As a result, target implementations benefit from both the high performance that HW-based computing provides and a SW-like flexibility that comes from multiplexing the FPGA fabric in time (the same silicon hosts different digital circuits over time). This not only provides higher execution performance in computing-intensive scenarios, but also enables runtime adaptivity in uncertain environments. ARTICo³ can be also conceptualized as a container for MDC or FG fabrics, and hence, these mixed-grained approaches may benefit from the scalability and fault-tolerant support provided by ARTICo³ with the fast switching support provided by MDC or the functional adaptivity provided by JIT HW composition.

d) Relevance with respect to use cases

Planetary Exploration: ARTICo³ can prove beneficial in this scenario for two main reasons. On the one hand, it enables (self-)adaptation when facing changing requirements (e.g. low battery level, faster processing when having better communication link). On the

other hand, the built-in features for enhanced fault tolerance are necessary in aerospace applications, where parts of a chip can malfunction and cause mission-critical errors. ARTICo³ ensures fault-tolerant execution of the functionality in the reconfigurable areas.

e) Strengths and gaps

Strengths of ARTICo³ with respect to the state of the art:

- provides adaptive and scalable HW acceleration at a FG level (individual copies of a HW accelerator can be changed while the rest of the system is still working, even in different accelerators);
- runtime self-characterization, based on a self-monitoring infrastructure and lightweight estimation models;
- runtime tradeoffs between computing performance, energy consumption and fault tolerance.

Weaknesses of ARTICo³ with respect to the state of the art and CERBERO:

- so far, there is no system level entry point (e.g. dataflow graphs) for ARTICo³-based designs, only C/C++ descriptions;
- manual HW/SW partitioning is required;
- This is a target-dependent tool (requires Xilinx FPGAs and development tools).

f) Tool extension within CERBERO

The envisaged extensions of ARTICo³ in CERBERO are:

- support for cross-layer, stream-based dataflow models of computation: high level dataflow descriptions as entry point (PREESM), and low level actor-based processing kernels (MDC).
- unified HW/SW (self-)monitoring approach (the current monitors and models are not generalizable or applicable to heterogeneous processing systems) using a PAPI-compatible infrastructure;
- support for intra-accelerator reconfiguration to enable faster dynamic changes in the datapath inside an ARTICo³ accelerator.

2.10. MDC

a) State of the art

Coarse-Grained (CG) reconfiguration is an interesting solution to face the challenges of modern embedded systems such as flexibility and high performance, without paying the overhead in terms of time and energy of FG approaches (for more details see D4.3). However, mainstream adoption of heterogeneous CG reconfigurable substrates is limited by intrinsic development issues: design and debug of optimal low level processing elements and mapping [Ansaloni 2012]. Automated development flows to speed up the

process and avoid long design and optimization phases have been proposed at the state of the art. In literature, usually, the common approaches are to adopt generic components [Oh 2017] or to select them from a predefined library [Yuan 2017] [Wildermann 2013]. The Multi-Dataflow Composer (MDC), on the contrary, derives components shaped exactly around the requested functionalities by exploiting the modularity of the dataflows adopted as specification format for the input applications. From each actor of the dataflow a different HW component is provided and the combination of the input applications is performed at the same dataflow actors' level.

b) Features

Inputs:

- **Dataflow Specifications**: at the moment RVC-CAL (XDF graph and CAL actors [Bhattacharyya 2011]) dataflow models describing the applications to be combined together;
- **HW Communication Protocol:** defining the handshake between actors in HW;
- **HDL Components Library:** the HDL descriptions corresponding to the involved CAL actors.

Outputs:

- **Multi-Dataflow HDL:** HDL description of the input dataflows combination (*multi-dataflow*);
- **(optional) Multi-Dataflow model:** RVC-CAL dataflow description of the multi-dataflow;
- **(optional) Xilinx IP Wrapper and Drivers:** HDL descriptions and C drivers providing a ready-to-use Xilinx IP around the multi-dataflow HDL description.

Reference Platform and GUI:

MDC at the moment is a plug-in of the Eclipse IDE and it is provided with a GUI in such an environment.

The MDC design suite is a SW framework for the automatic generation and management of CG reconfigurable systems based on the dataflow MoC.

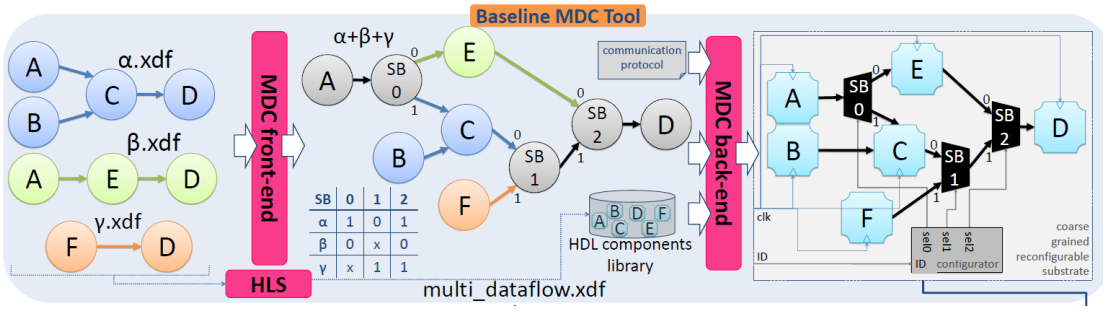

Figure 2-12 Overview of the Baseline MDC Tool.

The main features of the MDC design suite are:

- **Baseline MDC Tool:** the baseline feature of MDC is depicted in Figure 2-12**.** Basically, MDC combines together different input dataflows, each one describing a different application, into a unique reconfigurable multi-dataflow model that shares the common actors/resources (by means of switching boxes, SBs) and that is able to implement all the initial applications, one at a time. The dataflows combination corresponds to an NP-complete problem known as Datapath Merging Problem (DMP) that is solved with a heuristic method proposed by [Moreano 2002]. MDC generates the top Register Transfer Level (RTL) HDL specification of the reconfigurable multi-dataflow, while the RTL description of the dataflow actors (HDL components library, manually or automatically derived from dataflow actors), together with their communication protocol in HW, have to be provided by the user. MDC handle system configuration (how to set each SB selector to implement all the different applications) and encapsulates it into dedicated Look-Up Tables (LUTs) in HW, thus simplifying the configuration phase (performed by simply changing the application ID ideally in a single clock cycle). Figure 2-12 presents a simple example: three different dataflow applications, α, β and γ feed the MDC front-end, that combines them into a multi-dataflow by inserting three different SBs and by sharing two actors, A and D, and three dataflow edges (sharing is highlighted in black). Then, the MDC back-end, from the multi-dataflow, the HDL components library and the corresponding communication protocol, generates the RTL description of the system.
- **Structural Profiler:** it performs a design space topological exploration of all the implementable multi-dataflow systems derivable from the initial dataflow specifications set. This is necessary for two main reasons: (1) combining together all the dataflows is not always the best solution (SBs introduction may lead to higher costs with respect to the decision of non sharing an actor); (2) the feeding order plays a role in the combination process (the underlying algorithm combines two dataflows at a time in an iterative way) and may lead to different SBs chains in the multi-dataflow. This feature relies on an a priori characterization of the initial (non combined) dataflows in terms of area, static power and maximum frequency, which are the KPIs used to explore the design space and identify the optimal topological solution.
- **Dynamic Power Manager:** resource redundancy in reconfigurable architectures can lead to useless consumption due to resources that are not used in the current computation. At this purpose, MDC performs dataflow level logic partitioning of the substrate, to enable the implementation of clock- and power-gating strategies at the HW level. The Dynamic Power Manager keeps trace of which applications involve each actor of the multi-dataflow, so that it can identify common Logic Regions (LRs), which are sets of actors always active/inactive together. Then, when the RTL description is generated, clock- and power-gating strategies are applied by gating together all the actors belonging to the same LR.
- **Co-Processor Generator**: the Baseline MDC tool provides the RTL description of a CG reconfigurable substrate. However, to be effectively adopted in the practice as HW accelerator, its output requires the development of wrappers able to communicate with the external world, primarily with a host processor. To

facilitate this phase and speed up the prototyping, the Co-Processor Generator provides dataflow-to-HW customization of a Xilinx compliant accelerator on the top of the CG reconfigurable substrate generated with the Baseline MDC tool. Loosely coupled (memory-mapped) or tightly coupled (stream-based) accelerators can be provided along with their SW drivers, serving as APIs for a seamless integration into the host code.

c) Role in the CERBERO framework

MDC is able to provide support for CG reconfiguration on heterogeneous platforms. Only MDC in the CERBERO framework offers this kind of support. The other tool operating at the HW level is ARTICo³, which supports a different kind of reconfiguration, where predetermined CG slots are dynamically reconfigured at runtime using partial reconfiguration. Key features and benefits of both kinds of HW reconfiguration supports are clearer in D4.3: partial reconfiguration produces bigger changes in the architecture at the price of bigger overheads in terms of configuration time, power and memory footprint. A lightweight CG reconfiguration, such as the MDC one, will provide the CERBERO framework with a way to refine the system behaviour when small adjustments (i.e. surfing among working points to achieve different quality vs. energy trade-offs) are required to achieve a controlled flexibility with no (or limited) performance penalties.

MDC features also rapid prototyping capabilities, by means of the Co-Processor Generator extension. This feature is particularly useful for continuous deployment purposes and to enable faster time to deployment of custom HW accelerators.

d) Relevance with respect to use cases

Space Exploration: MDC will be exploited within ARTICo³ slots to accelerate and implement self-healing and self-adaptive behaviours in the heterogeneous computing infrastructure that is going to be used in the space use case, by means of a mixed-grained reconfigurable approach.

Ocean Monitoring: Proof of concepts of the benefits of using MDC-compliant computing infrastructure to provide adaptivity support for runtime trade-off management is going to be provided. The idea is proving the concrete usability of custom HW accelerators to accomplish relevant tasks that may run on future (beyond CERBERO) marine robot implementations, i.e. variable encoding/decoding precision to save energy. Adaptivity in this case can be system triggered (the robot is running out of battery) or user triggered (the user remotely changes the encoding quality).

e) Strengths and gaps

Strengths of MDC with respect to the state of the art:

- design/debug and mapping of CG reconfigurable architectures is automated and made it simple by the adoption of dataflow models of computation.

Gaps of MDC with respect to the state of the art:

- a fully automated flow (including the generation of the HDL components library) is not provided or is provided with limitations (target-dependent) [Sau 2016];
- the HW/SW partitioning of the applications has to be manually accomplished and the designer has to specify which type of coupling has to be implemented to connect the CG reconfigurable accelerator with the host processor(s);
- internal state runtime monitoring capabilities (to detect internal activity, faults, energy consumption) are not provided.

f) Tool extension within CERBERO

The extensions planned within the CERBERO project for MDC are:

- Providing a powerful and generic fully automated flow by means of the integration with the dataflow-oriented CAPH HLS engine [CAPH 2017]. This activity is already ongoing, as described in D4.4.
- Including MDC in a HW/SW partitioning flow. To this purpose we intend to leverage on PREESM (see Section 2.5), which requires to extend MDC to support PiSDF models and to create a model of architecture for MDC-compliant CG reconfigurable accelerators.
- Instrumenting the MDC generated CG reconfigurable accelerators with runtime monitors according to the PAPI approach (see Section 2.7). This activity is already ongoing, as described in D4.4.

2.11. Other tools

Besides the components that are actively involved in the CERBERO framework, other tools will be adopted or interfaced with the systems. These other tools will not be extended or modified during the project lifetime, and may not belong to companies or institutions within the CERBERO consortium, but it is useful to briefly describe them to better understand the role and the functionalities of the CERBERO components and of the framework as a whole.

CAPH

CAPH [CAPH] refers either to a domain specific language for streaming applications based on the dataflow MoC (among which Dataflow Process Network, discussed in D3.5) and the related toolset. Figure 2-14 shows an overview of the CAPH toolset. A front-end capable of parsing and checking the typing of CAPH source code (dataflow specifications) is the starting block. The output of such block is an abstract syntax tree, which is processed by the graph visualizer, the reference interpreter and the compiler. These provide respectively a graphical visualization of the dataflows, the golden references to check the final outputs, and SystemC/VHDL code derived from a target-independent IR. With the SystemC back-end, a cycle-accurate source code for simulation and profiling purposes is generated. The VHDL back-end provides generic code for HW synthesis. The SystemC simulation is useful to refine the VHDL design and to predict latency and execution time before synthesis. -

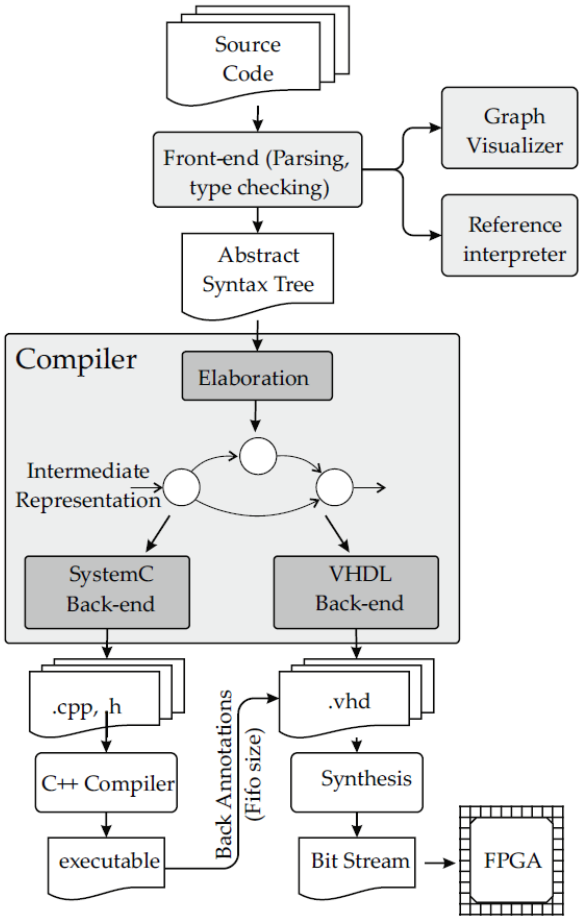

Figure 2-13 CAPH toolset overview.

The CAPH toolset will be connected to the CERBERO framework dealing with the HW adaptation support. In particular, CAPH is meant to be integrated with MDC (as described in D4.4) to speed-up the design of CG reconfigurable accelerators, making the whole flow, from high level specification to RTL description, fully automated.

SCANeR

SCANeR studio [SCANeR] is a SW tool in charge of providing driving simulation. SCANeR is used by CRF to implement and run different test scenarios for the Smart Travelling for Electric Vehicle use case. It is mainly based on *models* of the simulation environment components (vehicle, traffic, pedestrians, etc.), on *acquisitions* (driver, tracking systems, etc.) and on *restitutors* (audio, visual, etc.) from/to the same environment. In CERBERO, SCANeR, DynAA and the end-user will be interfaced through the mediation of MECA in order to provide dynamic adaptation to the user, environment and system itself changing functional and non functional requirements (for more details please refer to D4.4).

3. Conclusions

In this document the main components/tools of the CERBERO framework have been described in details. For each component/tool different aspects have been analyzed in order to highlight the already supported features and the modification/extensions that will be performed (or, in some cases, are being performed) on them aiming at the accomplishment of the CERBERO project objectives and use cases needs.

Table 1 Relevant features already provided (Supported, S) or to be provided (Extension, E) by the CERBERO framework components/tools.

	Modelling	**Optimization**	**HW/SW Design**	**Runtime Support**	**Incremental/ Fast Prototyping**	**In Loop Simulation**	**Open Source**
MECA	S+E			S	S+E		
VT	E						E
DynAA	S+E	S				E	
AOW		S+E					E
PREESM	S+E	S+E	E		S+E		S
SPIDER		S+E	E	S+E			S
PAPIFY				S+E			S+E
JIT HW		E	E	E			E
ARTICo³		S+E	S	S+E	E		E
MDC		E	S	S+E	S+E		E

To summarize the analysis done in the document two different resuming tables are provided. In Table 1 the CERBERO framework components/tools are cross-mapped with the most important features for the CERBERO project objectives (modeling, optimization, HW/SW design, runtime support, rapid prototyping, in loop simulation and open source). Here, already supported features and extensions/modifications to be performed during the project are differentiated. Please note that the VT tool and JIT HW definitions started from scratch within CERBERO.

Table 2 classifies the CERBERO components/tools per layer (end-user interaction, system model, application architecture, runtime support and low level implementation), and cross-link them to the CERBERO use cases (Smart Travelling, Space Exploration and Ocean Monitoring).

From the two resuming tables is possible to appreciate how the CERBERO framework components are going to cover (already or during the CERBERO project lifetime) most of the features related to modelling and implementation aspects, coming from the CERBERO project objectives. Moreover, it is shown how the ensemble of the

CERBERO framework components/tools is able to cover several different system layer, going from user down to implementation. The CERBERO use cases are faced by adopting different framework features: Smart Travelling is more on the high level layers (mostly end-user interaction and system model), Space Exploration spans more on the lower level layers (application architecture, runtime support and low level implementation), while Ocean Monitoring is somehow in the middle involving features provided by both high and low level layers.

Table 2 Level of abstraction on which each tool/component works and related relevance with respect to the CERBERO use cases.

	Level of Abstraction	**Use Case**		
		Smart Travelling	**Space Exploration**	**Ocean Monitoring**
MECA	end-user interaction	X		
VT	end-user interaction system model implementation	X	X	X
DynAA	system model	X		X
AOW	system model application architecture	X	X	X
PREESM	application architecture		X	
SPIDER	runtime support		X	
PAPIFY	application architecture runtime support low level implementation		X	
JIT HW	low level implementation		X	
ARTICo³	low level implementation		X	
MDC	low level implementation		X	

4. References

[Adhianto 2010] L. Adhianto et al., *HPCToolkit: Tools for performance analysis of optimized parallel programs*, Concurrency and Computation: Practice and Experience, 2010.

[Agne 2014] A. Agne et al., *ReconOS: An Operating System Approach for Reconfigurable Computing*, IEEE Micro, 2014.

[AMPL] www.ampl.com/

[Ansaloni 2012] G. Ansaloni et al., *Integrated Kernel Partitioning and Scheduling for Coarse-Grained Reconfigurable Arrays*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012.

[Banks 2004] J. Banks, et al., *Discrete-Event System Simulation*, Prentice Hall, 2004.

[Beckhoff 2012] C. Beckhoff et al., *Go ahead: A partial reconfiguration framework*, International Symposium on Field-Programmable Custom Computing, 2012.

[Bianculli 2012] D. Bianculli et al., *Specification patterns from research to industry: a case study in service-based applications*, International Conference on Software Engineering, 2012.

[Bigus 2002] J.P. Biguset al., *ABLE: a toolkit for building multiagent autonomic systems*., IBM Systems Journal, 2002.

[Bosse 2017] T. Bosse et al., *Developing ePartners for human-robot teams in space based on ontologies and formal abstraction hierarchies*, International Journal on Agent-Oriented Software Engineering, 2017.

[Brazier 2002] F.M.T. Brazier et al., *Principles of Component-Based Design of Intelligent Agents*, Data and Knowledge Engineering, 2002.

[Broodney 2012] H. Broodney et al. *Generic Approach for Systems Design Optimization in MBSE*, International Council in Systems Engineering, 2012.

[CAPH] caph.univ-bpclermont.fr/CAPH/CAPH.html

[Clerissi 2017] D. Clerissi et al., *Towards the Generation of End-to-End Web Test Scripts from Requirements Specifications*, International Requirements Engineering Conference Workshops, 2017.

[Cplex] www.ibm.com/software/commerce/optimization/cplex-optimizer/

[Davis 1999] J. Davis II et al., *Overview of the Ptolemy project*, ERL Technical Report UCB/ERL No.M99/37, 1999.

[Desnos 2013] K. Desnos et al., *PiMM: Parameterized and Interfaced Dataflow*

Meta-Model for MPSoCs Runtime Reconfiguration, International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2013.

[DjangoREST 2018] www.django-rest-framework.org

[Dokhanchi 2018] A. Dokhanchi et al., *Formal requirement debugging for testing and verification of cyber-physical systems,* ACM Transactions on Embedded Computing Systems, 2018.

[Duell 2008] R. Duell et al., *An Ambient Intelligent Agent Model Using Controlled Model-Based Reasoning to Determine Causes and Remedies for Monitored Problems*, IEEE/WIC/ACM International Conference on Web Intelligence, 2008.

[Dwyer 1999] M.B. Dwyer et al., *Patterns in property specifications for finite-state verification*, International Conference on Software Engineering, 1999.

[Ecker 2009] W. Ecker et al., *Hardware-dependent Software*, Springer, 2009.

[Esteve 2012] M.A. Esteve et al., *Formal correctness, safety, dependability, and performance analysis of a satellite*, International Conference on Software Engineering, 2012.

[Fuxman 2004] A. Fuxman et al., *Specifying and analyzing early requirements in Tropos*, Requirements Engineering, 2004.

[Grandpierre 1999] T. Grandpierre et al., *Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors*. International Workshop on Hardware/Software Codesign, 1999.

[Gurobi] www.gurobi.com

[Heulot 2014] J. Heulot et al., *Spider: A Synchronous Parameterized and Interfaced Dataflow-based RTOS for multicore DSPS*, Embedded Design in Education and Research Conference, 2014.

[Hentenryck 1999] P. Van Hentenryck, *The OPL optimization programming language*, MIT Press, 1999.

[Khronos 2017] OpenCL Specification Version 2.2 , online: www.khronos.org/registry/OpenCL/specs/opencl-2.2.pdf

[Knüpfer 2008] A. Knüpfer et al,. *The vampire performance analysis tool-set*, International Conference: Tools for High Performance Computing, 2008.

[Koch 2008] D. Koch et al., *Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs*, IEEE International Conference on Field Programmable Logic and Applications, 2008

[Konard 2005] S. Konrad et al., *Real-time specification patterns*, International conference on Software Engineering, 2005.

[Leeuwen 2014] C. van Leeuwen et al., *Model-based Architecture Optimization for Self-adaptive networked signal processing systems*, International Conference on Self-Adaptive and Self-Organizing systems conference, 2014.

[Leupers 2017] R. Leupers et al., *MAPS: A Software Development Environment for Embedded Multicore Applications*, Handbook of Hardware/Software Codesign, 2017.

[Liu 2013] J. Liu et al., *Synthesis of reactive switching protocols from temporal logic specifications*, IEEE Transactions on Automatic Control, 2013.

[LLVM] compiler-rt.llvm.org

[Lumpe 2011] M. Lumpe et al., *PSPWizard: machine-assisted definition of temporal logical properties with specification patterns*. European conference on Foundations of software engineering, 2011.

[Mathworks] www.mathworks.com

[Masin 2008] Masin, M., and Bukchin, Y., 2008, "Diversity Maximization Approach for Multi-Objective Optimization", *Operations Research*, 56, 411-424.

[Masin 2014] M. Masin et al., *Reusable derivation of operational metrics for architectural optimization*, Conference on System Engineering Research, 2014.

[Masin 2013] M. Masin et al., *Pluggable Analysis Viewpoints for Design Space Exploration*, Conference on System Engineering Research, 2013.

[Modelica] www.modelica.org

[Moreano 2002] N. Moreano et al., *Datapath Merging and Interconnection Sharing for Reconfigurable Architectures*, International Symposium on System Synthesis, 2002.

[MS Excel] products.office.com/en-us/excel

[Myers 2007] K. Myers et al., *An Intelligent Personal Assistant for Task and Time Management*, AI Magazine, 2007.

[Narizzano 2017] M. Narizzano et al., *Consistency of Property Specification Patterns with Boolean and Constrained Numerical Signals*. arXiv preprint arXiv:1712.04162, 2017.

[Neerincx 2008] M.A. Neerincx et al., *The mission execution crew assistant: Improving human- machine team resilience for long duration missions*, International Astronautical Congress, 2008.

[Neerincx 2010] M.A. Neerincx et al., *Evolution of electronic partners: human-automation operations and ePartners during planetary missions*, Journal of Cosmology, 2010.

[Nutaro 2010] J.J. Nutaro, *Building Software for Simulation: Theory and Algorithms with applications in C++*, Wiley Publishing, 2010.

[Oh 2017] S. Oh et al., *Efficient Execution of Stream Graphs on Coarse-Grained Reconfigurable Architectures*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017.

[Oliveira 2013] J. Oliveira et al., *Model-based design of self-adapting networked signal processing systems*, International Conference on Self-Adaptive and Self-Organizing systems, 2013.

[OpenMP 2015] OpenMP 4.5 Specification, Nov. 2015, www.openmp.org/wp-content/uploads/openmp-4.5.pdf

[ORCC] orcc.sourceforge.net/

[Otero 2012] A. Otero et al., *Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems*, International Conference on Reconfigurable Computing and FPGAs, 2012.

[Pacelab] www.pace.de/products/preliminary-design/pacelab-suite.html

[Papp 2016] Z. Papp et al., *Runtime Reconfiguration in Networked Embedded Systems – Design and Testing principles*, Springer, 2016.

[Pelcat 2009] M. Pelcat et al., *A System-Level Architecture Model for Rapid Prototyping of Heterogeneous Multicore Embedded Systems*, Conference on Desing and Architectures for Signal and Image Processing, 2009.

[Pelcat 2014] M. Pelcat et al. *PREESM: A Dataflow-Based Rapid Prototyping Framework for Simplifying Multicore DSP Programming*, European DSP Education and Research Conference, 2014.

[Post 2012] A. Post et al., *Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH*, Requirements Engineering, 2012.

[Ren 2014] R. Ren et al., *Energy estimation models for video decoders: reconfigurable video coding-CAL case-study*, IET Computers & Digital Techniques, 2014.

[Ren 2013] R. Ren et al., *A PMC-driven methodology for energy estimation in RVC-CAL video codec specifications*, Signal Processing: Image Communication, 2013.

[Rettkowski 2016] J. Rettkowski et al., *RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs*, International Conference on ReConFigurable Computing and FPGAs, 2016.

[Rhapsody] www-01.ibm.com/software/awdtools/rhapsody/

[Bhattacharyya 2011] S.S. Bhattacharyya et al., Overview of the MPEG Reconfigurable Video Coding Framework, Journal of Signal Processing Systems, 2011.

[Sau 2016] C. Sau et al., *Automated Design Flow for Multi-Functional*

Dataflow-Based Platforms, Journal of Signal Processing Systems, 2016.

[SCANeR] http://www.oktal.fr//en//automotive//range-of-simulators//software

[SDSoC] www.xilinx.com/products/design-tools/software-zone/sdsoc.html

[Schlütter 2014] M. Schlütter et al., *Profiling Hybrid HMPP Applications with Score-P on Heterogeneous Hardware*, International Conference on Parallel Computing, 2014.

[Stuijk 2006] S. Stuijk et al., *SDF3: SDF For Free*, International Conference on Application of Concurrency to System Design, 2006.

[Terpstra 2009] D. Terpstra et al., *Collecting performance data with PAPI-C*, Tools for High Performance Computing, 2009.

[Torquati 2012] M. Torquati et al., *An innovative compilation tool-chain for embedded multi-core architectures*, Embedded World Conference, 2012.

[Wang 2012] Y. Wang et al., *A partially reconfigurable architecture supporting hardware threads*, IEEE International Conference on Field-Programmable Technology, 2012.

[Wildermann 2013] S. Wildermann et al., *Symbolic system-level design methodology for multi-mode reconfigurable systems*, Design Automation for Embedded Systems, 2013.

[Yuan 2017] Z. Youan et al., *CORAL: Coarse-grained reconfigurable architecture for Convolutional Neural Networks*. International Symposium on Low Power Electronics and Design, 2017.

[Zhang 2014] H. Zhang et al., *GUARD: Guaranteed reliability in dynamically reconfigurable systems*, Design Automation Conference 2014.