Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105

D4.4: Self-adaptation Manager

Lead Beneficiary: INSA
Workpackage: WP4
Date: March 30 2018
Distribution - Confidentiality: [Public/Confidential]

Abstract: The CERBERO self-adaptation manager is composed of external, CERBERO-enhanced and CERBERO-developed tools. This document presents the tools integration activities, conducted for building the CERBERO self-adaptation management.

Disclaimer

The CERBERO Consortium is the following:

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

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

Document Authors

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

Name(s)	Organization Acronym
Maxime Pelcat	INSA
Florian Arrestier	INSA
Claudio Rubattu	INSA / UNISS
Francesca Palumbo	UNISS
Eduardo Juarez	UPM
Eduardo de la Torre	UPM
Alfonso Rodriguez	UPM
Leonardo Suriano	UPM
Tiziana Fanni	UNICA
Pablo Muñoz	S&T
Edo Loenen	S&T
Gasser Ayad	AI
Carlo Sau	UNICA
Hans Myrhaug	AS
Stuart Watt	AS
Ayse Goker	AS
Joost Adriaanse	TNO
Evgeny Shindin	IBM
Michael Masin	IBM
Katiuscia Zedda	AI

Document Revision History

Date	Ver.	Contributor (Beneficiary)	Summary of main changes
2017/12/22	0.1	INSA	TOC
2018/01/10	0.2	INSA	TOC Update

2018/01/19	0.3	INSA	Integrating data from T4.4.2, T4.4.3 and T4.4.4. Connecting to D5.6.
2018/01/22	0.4	INSA	Added details on P2P integration section.
2018/01/24	0.5	INSA	Added leaders for each section.
2018/02/19	0.6	INSA	Written on adaptivity.
2018/02/26	0.7	INSA	Written on CERBERO adaptivity strategy.
2018/02/27	0.7.4	UNICA, UNISS, UPM	MDC & ARTICo3.
2018/03/01	0.7.5	INSA, UPM	New deliverable name, 4.4, 4.6.
2018/03/02	0.7.6	INSA, S&T, TNO, AI, AS, USI	4.7, 5.3, 4.2, 5.2, 2.4.
2018/03/02	0.7.7	INSA	Linking sections together.
2018/03/26	0.8.1	INSA, AS, USI, IBM, UPM, TASE	Linking sections together.
2018/03/27	0.8.2	INSA, AI	Corrections and integration.
2018/03/29	0.8.5	INSA	Added tools table.
2018/03/30	1.0.0	INSA	Corrections.
2018/04/04	2.0	UNISS	Final Version
2018/05/24	2.1	INSA	Inserted corrections from IBM.

Table of contents

1. Executive Summary

The CERBERO self-adaptation manager integration aims at building the first self-adaptive management support capable of driving heterogeneous and embedded Cyber-Physical Systems (CPS). From models of the system and its environment, and functional hardware and software representations, the CERBERO self-adaptation support will provide features such as:

1. Real-time system and environment monitoring gathering Key Performance Indicators (KPIs) retrieved from different sensors,
2. KPI analysis through models,
3. Self-scheduled distributed processing,
4. Unified self-adaptation to system, environment, and human triggers,
5. Software and hardware reconfiguration, including Dynamic Partial Reconfiguration (DPR) and Coarse Grain Reconfiguration (CGR).

This variety of capabilities, augmenting a system with awareness and reconfiguration features, comes from a set of external, CERBERO-enhanced and CERBERO-developed tools. This D4.4 document covers the tool integration activities planned within the project for advancing the field of CPS self-adaptation.

The objective of this document is to provide a plan for the integration of the CERBERO self-adaptation manager. This integration effort has started on M13 within the tasks T4.4 "Path towards full heterogeneous system self-adaptation" and will be fully demonstrated on M30 in "D4.2 - CERBERO self-adaptation manager (Final Version)". As a plan of integration, this document focuses on tool-to-tool integration activities that aim at bounding a cross-layer runtime support.

1.1. Structure of the Document

The document is organized as follows. Section 2 explains the CERBERO self-adaptation strategy that motivates for connecting the different tools, and locates it in state-of-the-art. Section 3 lists the tools involved in the CERBERO self-adaptation support. Section 4 details each point-to-point tool integration activity. Section 5 discusses the applicability of the developed self-adaptation strategy and management to the CERBERO use cases. Finally, Section 6 concludes on the intended advances of the CERBERO self-adaptation management with respect to the State-of-the-Art and on the plans of integration to be implemented.

1.2. Main Related Documents

The most related CERBERO Deliverables to D4.4 are:

- D2.7 - CERBERO Technical Requirements
  - D4.4 contributes to satisfy D2.7 requirements. Details are given in Section 1.3.
- D4.2 - CERBERO self-adaptation engine (Final Version)

  - The D4.4 deliverable (M15) focuses on point-to-point integration activities while D4.2 (M30) will discuss the capabilities of the constituted CERBERO runtime management.
- D4.3 - CERBERO Multi-Layer Runtime Adaptation Strategies (Ver 1)
  - D4.3 explains the runtime adaptation strategy and its elements while D4.4 integration activities serve in building this strategy.
- D5.6 - CERBERO Framework Components (Ver 1)
  - The D4.4 deliverable does not enter into the details of the CERBERO tools, but rather focuses on the point-to-point integration activities.
  - The reader interested to get more details on individual tools will find them in D5.6.

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 self-adaptation manager integration activities described in the current document address 6 CTRs, as described in Table 1-1.

Table 1-1: CERBERO Technical Requirements driving self-adaptation manager integration activities.

CTR id	**CTR Description**	**Link with the D4.4 document on *Self-Adaptation Manager***
0001	CERBERO framework SHOULD increase the level of abstraction at least by one for HW/SW co-design and for System Level Design.	The integration of the CERBERO self-adaptation tool chain increases the design level of abstraction by automating tasks that, in state-of-the-art systems, are manually conducted, including e.g. HW/SW co-design, coordination of environment, system and human, and reconfigurability.
0003	CERBERO framework SHOULD provide incremental prototyping capabilities for HW/SW co-design.	The CERBERO self-adaptation managing framework aims at helping the designer to build fast HW/SW hybrid and heterogeneous prototypes with adaptation capabilities.
0006	CERBERO framework SHOULD ensure energy efficient and dependable HW/SW co-design using cross-layer runtime adaptation of reconfigurable HW.	Through system and environment monitoring, and self-adaptation, combined with SW and HW reconfiguration, the CERBERO self-adaptation manager provides a framework for raising energy efficiency and dependability.
0016	CERBERO tools SHOULD be tested vs. state-of-the-art.	The CERBERO integrated tools are constantly tested vs. state-of-the-art solutions. The built self-adaptation manager brings unique design automation features, as explained in Section 2.5.
0019	CERBERO technology providers SHALL coordinate technical support for their tools with use case engineers.	As shown in Section 5, use cases are aligned with the CERBERO proposed technology. Live and online tutorials are proposed to synchronize partners.
0020	CERBERO framework SHALL provide methodology and tools for development of adaptive applications.	This document develops the tooling part of CERBERO adaptive systems development.

2. Prepared CERBERO Self-Adaptation Support

A precise definition of self-adaptation is a necessary starting point to this document. Within the CERBERO project, self-adaptation is a runtime action that consists in changing structure, functionality and/or parameters of a system, based on information from environment, user or self-sensing. This definition is not specific to CPS and is compatible with the general definitions proposed in [MACIAS 2013]. However, by entangling physical and cyber constraints, CPS create a new world of challenges, especially targeting new levels of resource usage efficiency.

System self-adaptation refers to a combination of 1) awareness and 2) system reconfiguration. For the system to be self-adaptive, a reconfiguration is decided *inside* the system itself, which presupposes that the self-adaptation manager has some degrees of freedom when deciding which modifications to apply.

The objectives for self-adaptations are numerous and include:

- **Fault tolerance and recovering after a system fault**, as especially required by the CERBERO Planetary Exploration use case,
- **Adapting system resources to timely requirements** so as to raise system efficiency. This objective is particularly important in distributed and networked systems and the related constraints are among the requirements of the CERBERO Planetary Exploration and Ocean Monitoring use cases,
- **Modifying system functional behavior to match modifications in the environment**. This type of adaptation is common to the three CERBERO Use Cases.

Self-adaptation necessarily involves a feedback loop from sensors to a decision entity (or entities). Many publications have focused on defining the loop structure of self-adaptive systems, decomposing them into phases such as Collect-Analyze-Decide-Act [BRUN 2009] or Monitoring-Analyzing-Planning-Executing [SALEHIE 2012]. These cycles are at the heart of self-adaptiveness, making systems reacting to sensor information as an individual or as a group. The self-adaptation activities of the CERBERO project rely on this type of decomposition, namely:

1. Monitoring: retrieving data from system and environment through sensors, and model-based pre-analysis. This step is handled by *monitors*.
2. Management: planning/deciding the execution. This step is handled by self-adaptation *managers*.
3. Execution: application functional execution. This step is handled by execution *engines*.

This document is named “self-adaptation manager”, but it gathers information on the integration of tools targeting the three types of elements: {monitor, manager, engine}.

The CERBERO project aims at providing model-based methods and tools to implement the self-adaptation cycle in a concrete, efficient CPS.

In current developments of self-adaptive systems, software is the main element that introduces self-adaptation [BETTY 2009]. **A strong objective of the CERBERO project is to add to current software-based approaches of self-adaptation the novel capabilities of System-on-Chip and hardware adaptation**, crossing the design layer boundary between software adaptation and hardware adaptation.

2.1. CERBERO Methods for Awareness

In the CERBERO cyber-physical context, awareness has three main modes: self-, user- and environment-awareness. The concept of self-awareness covers many ideas in the domains of control, artificial intelligence, autonomic computing and self-adaptive systems [CAMARA 2017]. It mostly consists in observing the system state through sensors and acting automatically upon state modifications. In the cyber-physical context, self-awareness is broadened to general awareness by extending the information provided by sensors to environment and user. Awareness can apply at different levels of system design. In the CERBERO project, awareness is experimented both at the platform level within the SPIDER, MDC, CAPH, ARTICo3 and Papify tool combinations (potentially with hardware/software heterogeneous computing) and at the application level within the DynAA, SCANeR and MECA tool combinations (Section 4).

For the targeted systems to be self-aware, the CERBERO runtime manager voluntarily monitors a set of KPIs constantly made available by sensors. The sensor information is either directly sent to the planning/decision manager or it goes through a model that analyzes and extracts higher level information prior planning.

2.2. CERBERO Runtime Levels of Reconfiguration

On the actuation side, the CERBERO project considers reconfiguration at four different levels:

1. At the ***system and system-of-systems processing levels***, self-scheduled distributed software computing is considered where computation, potentially consisting of several applications, is spread over a heterogeneous set of processing resources while optimizing selected KPIs.
2. At the ***applicative level***, the Smart Travelling use case considers reconfigurations of a Cyber Physical System of System (CPSoS) providing drivers with smart mobility services.
3. At a ***coarse hardware adaptation granularity***, Coarse Grain Reconfigurable (CGR) substrates are supported. They provide high-speed reconfiguration between a limited set of pre-computed configurations and make it possible to exploit the extreme efficiency of FPGA and ASIC custom hardware accelerators.
4. At the ***finest hardware adaptation granularity***, Dynamic Partial Reconfiguration (DPR) is employed to replace a hardware task by another hardware task on the same FPGA hardware resources. This reconfiguration costs time and energy when reconfiguring, but it allows the designer to tune its architecture at runtime, reusing hardware resources in completely different contexts and protecting hardware against failures.

2.3. Assessing CERBERO Adaptivity at Design Time through Mathematical Programming

Adaptivity capabilities of CPS should be considered at design time in order to provide CPS architectures capable to adapt themselves to possible environmental changes. As CERBERO methodology for design space exploration considers the modeling of system and environment uncertainty at design time (please see D3.6. for more details), these models can be used also to provide self-adaptation policies together with CPS architecture. The basis for this technology is already included in the domain of Robust and Stochastic optimization methods.

Stochastic optimization provides two-stage and multi-stage models that are capable to model adaptive policies. In two-stage models, variables of first stage are design variables that are non-adaptive, while variables of the second stage depend on uncertainty realizations that can be modeled by the set of possible scenarios. Thus, the optimization result here is two-fold: on one hand, we obtain an optimal CPS architecture described by variables from first stage and on the other hand we obtain optimal policies for each modeled scenario. A multi-stage model is even more expressive because it considers changes in uncertainty realizations, and therefore policies obtained from multi-stage models can be used in order to adopt CPS for uncertainties that changes over time.

Robust optimization provides Affinely Adjustable Robust Counterpart (AARC) that also defines two kinds of decision variables: adjustable and non-adjustable. Adjustable variables here represent affine functions of uncertainty realizations. Unlike policies in scenario-based optimization, AARC provides policies that are suitable for all possible uncertainty realizations, modeled by a number of uncertainty sources that defines ranges of possible values for uncertain parameters. Moreover, robust optimization can combine scenario-based uncertainties together with AARC in order to provide adaptation policies that are suitable for many different real-life problems.

Thus, appropriate modeling adaptation at design time within the CERBERO project will lead to adaptation policies, which will be implemented at runtime, and will provide optimal CPS adaptation to uncertain and changing environment.

2.4. Enhancing CERBERO Adaptive Runtime Security and Reliability

Reliability and security are non-functional requirements that are extremely critical for today CPS. Currently, the level of security and the level of reliability of one system are statically decided at design time and none or little capability is offered of adapting or modifying it during the lifetime of a system. However, this is not the best approach, especially for CPS that are designed based on variables which are uncertain at design time or evolving during the life of the device.

The problem of adaptive level of security was addressed in the past, especially for sensor node, where there have been proposals of adapting security parameters to the level of existing threat [VENKATASUBRAMANIAN 2014] or adapting the functionalities offered by protocols to the resourced of a device [WEN 2010]. Similarly, the problem of adaptive reliability was addressed in the past, for instance for sensor nodes [ALIREZAEYAN 2015] adapting the number of active path, or for multiprocessor

system systems modeling and adapting the reliability requirement to the task needs [ALOUANI 2017]. However, these past works and proposals towards the direction of adapting security and reliability have not been integrated into a framework capable of continuous monitoring the evolution of a system in its globality. Furthermore, they are designed to be extremely specific, for the case of study and they cannot be immediately applied to other security and reliability problems.

In CERBERO, we will propose a framework that is sufficiently generic to model security and reliability requirements of the CPS and to dynamically adapt to them, allowing to globally optimize the behaviors and performance.

Following the CERBERO approach, we will develop a way to model security and reliability requirements, sufficiently general to capture all possible requirements (thus not only limited to a specific subset of them, as currently in literature). Similarly, we will provide a generic way to specify a metric to evaluate security and reliability, allowing the CERBERO framework to trade security and reliability exactly as other design variables. Thanks to this, security and reliability will be then adapted at runtime, also considering other resources available in the system. Run time adaptation for security can be used to adapt the level of security to the situation or while maintain the level of security reducing the energy used to provide it (or to increase the performance of the compute algorithm). An example of the first adaptation is the change of the type and amount of security checks to the situation. This is for instance the case when several security policies are required in the system, but not all of them have to be active at the same time. An example of the second adaptation is the dynamic selection of different implementations of the same cryptographic algorithm (one having high throughput but high energy consumption, another being slower but more energy efficient). Preliminary ideas in this direction have been explored [PALUMBO 2017]. In the following month of the project we will concentrate in developing a more advance approach to support adaptation of security algorithms and policies. We will use a similar approach to address reliability. We expect that the probability of failure of the system evolves and changes with the context (and so does the eventual consequence of an error). We plan to develop a support for adapt the level of redundancy to the available resources and to the failure of the system.

2.5. Related Work on Self-Adaptation Tools

This D4.4 deliverable puts the focus on the CERBERO self-adaptation tools and their integration. In order to overview current work on this subject, Table 2-1 references recent state-of-the-art related tools that aim at managing system adaptation.

Table 2-1: State-of-the-art of Adaptive Runtime Tools

Adaptivity Tool/Method	Available tooling?	Supported app.s	Supported arch.s	Supported adaptivity	Main keywords
Adaptive MapReduce [ZHANG 2015]	A generic framework is available, but published version is	Streaming scientific applications	Heterogeneous cloud platforms	Software, daemon-based, dynamic work redistribution	Task-level adaptivity, adaptive mapping

	proprietary				
Flextream Framework [HORMATI 2014]	No	Streaming applications	Heterogeneous multicore systems	Software, dynamic work redistribution	Online adaptation, partition refinement, flexible compilation
HoneyComb design flow [THOMAS 2004]	No	Data flow based applications and control flow based applications	HoneyComb processor arrays	Adaptive routing configuration (support in hardware)	Runtime routing topology, multigrain hardware links
Improved Dellacherie algorithm [ASSAYAD 2017]	No	Task parallel applications	Homogeneous network-on-chip mesh architectures	Software, schedule-level, DVFS-level, and topology-level adaptivity	Adaptive mapping, multi-objective optimization
SHARA [QUAN 2016]	No	Multimedia applications (streaming applications)	Large-scale heterogeneous MPSoC systems	Software, scenario-based, different QoS requirement, system fault adaptivity	Runtime task mapping, hierarchical resource manager
Models at RunTime (M@RT) [BENNACEUR 2014]	No	Different application contexts	It can consist of middleware, a language runtime environment, an operating system, a virtualization system, and hardware resources.	Software, Model-based	Runtime software adaptation
Draco [VANDEWOUDE 2003]	No	Streaming static applications	Not specified. Theoretical paper.	Not detailed	Middleware platform
SEEC: SElf-awarE Computing [HOFFMANN 2011]	No	Streaming applications	Not specified.	Uses Heartbeat API to measure performance and specify application goals. User specifies	Self-adaptive, machine learning

				tradeoffs at application / system-level	
Heterogeneity-Aware Runtime System (HARS) [YUN 2015]	No	See PARSEC benchmark	Architectures similar to ARM big.LITTLE (the ODROID platform is used for tests)	Metrics based (uses heartbeat performance measurement)	Heterogeneous multiprocessing, self-adaptive computing
Criticality-and Heterogeneity-aware Runtime system for Task-parallel (CHRT) [HAN 2017]	No	Task parallel applications	Two types of core cluster with higher and lower performance (ARM big.LITTLE)	Software, dynamically adjusts core frequency to optimize energy	Task-parallel, energy efficiency, heterogeneous multiprocessing
SPADE [SCHNEIDER 2009]	Proprietary tool	Streaming applications	Multicore processors	Software, dynamic threads parallelization	Dynamic adaptation, computational elasticity
Sambamba [STREIT 2013]	Website but no link to download the framework	Not streaming application	Not clear but seems to be homogeneous CPU-based platforms	Software, compile time analysis with automatic parallelization as well as runtime decisions	Program transformation, just-in-time compilation, adaptation, automatic parallelization
Invariant Refinement Method for Self-Adaptation (IRM-SA) [GEROSTA 2016]	Yes	Safety-critical applications	Software-intensive cyber–physical systems (siCPS), distributed systems	Software, sensor-based	Self-adaptivity, Dependability
XKaapi: local work-stealing [GAUTIER 2013]	Yes. Available and Open Source (Cecill-C license)	Data-flow based streaming applications	Heterogeneous Architectures multi-CPU/multi-GPU (Evaluated on 4CPUs + 8 GPUs)	Heterogeneous platforms, locality-aware work-stealing based on heuristics, asynchronous task repartition for GPUs	High-performance computing, data-flow, heterogeneous architectures, locality aware work stealing

Charm++ & AMPI [ROBSON 2017]	Yes. Charm++ and AMPI libraries	Irregular and dynamic applications	Heterogeneous with CPU and GPU	Heterogeneous platforms, object remapping for load balancing	Accelerator architectures, parallel programming, high performance computing

Most state-of-the-art publications listed in Table 2-1 refer to non-available tools. Among these, most approaches concentrate on software management and rely on existing software frameworks. An exception is [THOMAS 2004] that routes data in a specific HoneyComb processor array hardware.

Compared to all these approaches, the CERBERO framework is the first to provide an open-source self-adaptive management system, portable over several families of off-the-shelf heterogeneous embedded hardware and software systems.

Among the three listed self-adaptive management systems for which code is available, XKaapi and Charm++ are High Performance Computing (HPC) management systems that place themselves over large-scale facilities composed of multiple CPUs and GPUs. In contrast, the CERBERO runtime system is targeting Cyber Physical embedded systems where lightweight, predictable and efficient runtime management is crucial. The closest work to the CERBERO runtime system is [GEROSTA 2016]. However, [GEROSTA 2016] does not consider hardware acceleration, which is today compulsory in most High Performance Embedded Computing (HPEC) systems, such as video processing, deep learning, telecommunication and computer vision systems [WOLF 2014]. As a consequence, the CERBERO self-adaptation support is a radically new approach of system management.

3. Integrated Self-Adaptation Tools

Table 3-1 overviews the CERBERO tools, as detailed in Deliverable D5.6, adding CAPH and SCANeR as integrated external tools. Some properties of the tools are recalled, either currently provided (Supported, S), or to be provided (Extension, E).

Table 3-1: CERBERO Integrated tools for self-adaptation management.

	CERBERO Internal	Modelling	Optimization	HW/SW Design	Runtime Support	In Loop Simulation	Open Source
VT	Yes	E					E
AOW	Yes	S+E	S+E	E			E
PREESM	Yes	S+E	S+E	E			S
DynAA	Yes	S+E	S			E	
SCANeR	No					S	
MECA	Yes	S+E			S		
SPIDER	Yes		S+E	E	S+E		S
PAPIFY	Yes				S+E		S+E
JIT HW	Yes		E	E	E		E
ARTICo³	Yes		S+E	S	S+E		E
MDC	Yes		E	S	S+E		E
CAPH	No	S		S			

This document concentrates on the last 8 (gray) tools of the table, excluding JIT HW. These tools have self-adaptation as a strong objective. The document deals with the integration of supported (S) properties from the different tools. As a consequence, JIT HW is not evoked because under study (E).

The case of the CAPH [1] compiler, integrated as an external tool, will be detailed in Section 4.5. It is integrated for its capacity to generate hardware from a high level description (HLS for High Level Synthesis) and for its complementarity with CERBERO internal tools.

The SCANeR [2] external tool is a driving simulation software tool integrated specifically for the Smart Travelling use case. The next section details the on-going CERBERO integration activities.

[1] http://caph.univ-bpclermont.fr
[2] http://www.oktal.fr/en/automotive/range-of-simulators/software

4. Point-to-Point Tool Integration Activities

4.1. Overview of the Tool Point-to-Point Integrations

The next sections detail activities on tool-point-to-point integrations for building the CERBERO self-adaptive runtime support. Figure 1 overviews the main on-going point-to-point tool integration activities (which timeline is provided in Section 6) and refers to their related sections in this document.

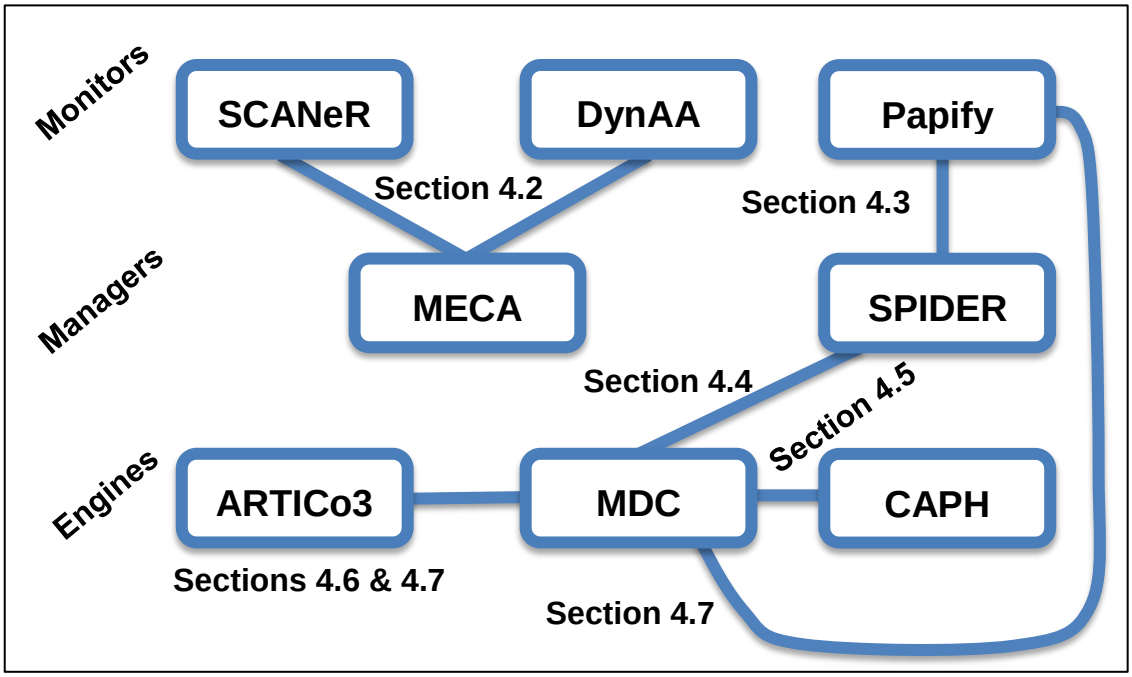

Figure 1: Overview of the main on-going point-to-point tool integration activities.

Figure 1 concentrates on runtime tools and roughly categorizes them into {Monitor, Manager, Engine}, but some tools overlap these categories. The current integration activities target the application-level approach of SCANeR, DynAA and MECA, and the HW-SW co-design approach of Papify, SPIDER, ARTICo3, MDC and CAPH. Design time tools VT, AOW and PREESM are covered in Deliverable D5.6 - CERBERO Framework Components. They connect to Figure 1 displayed tools to help designing predictable systems with adaptation management.

4.2. DynAA, SCANeR & MECA Integration

This integration activity brings together two simulation tools, DynAA and SCANeR, and a decision tool, MECA, to adapt at application level the system to a large set of triggers.

In Figure 2, a schematic overview is given of the DynAA, SCANeR and MECA tools enhanced within the CERBERO project. As an application-level tool chain, the links between DynAA, SCANeR and MECA are tailored to the Smart Travelling use case.

MECA will receive monitoring data from a vehicle (via the SCANeR simulator), sensor data from the system environment (via DynAA) or user input from a driver (indicating for example a new destination). Based on the data received, MECA will trigger impact processing functionality which will determine if adaptation is required. MECA will initiate adaptation based on different types of triggers, such as:

- Environment (environment-awareness);
- System (self-awareness);
- Human (user-commanded).

The adaptation itself can for example be the initiation of an investigation of alternative routes (in case planned charging poles are found to be out of service (=environment)), the proposal of advised routes (based on impact analysis performed on DynAA, triggered by route request from driver (=human)) or advice to user to reduce energy consumption by reducing or switching off the airco (in case for example battery charge is found to become critically low (=system)).

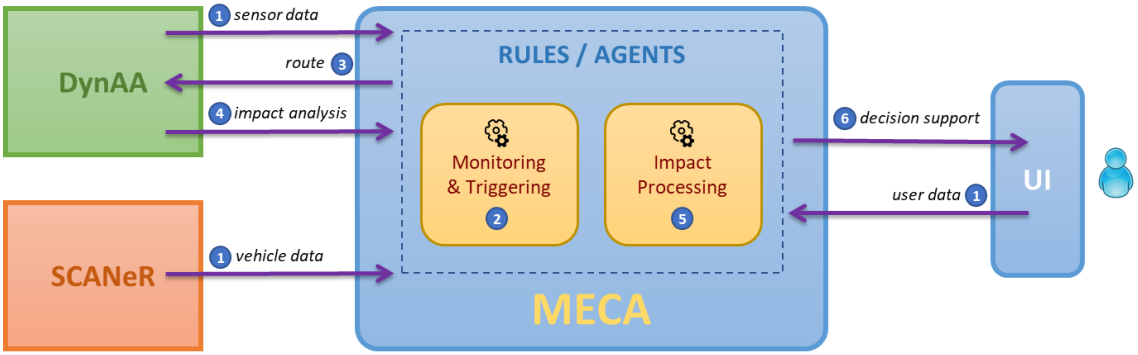

Figure 2: Schematic overview of MECA, DynAA and SCANeR interworking.

The adaptivity is controlled by functions developed on the MECA tool, which already possesses basic functionality for storing knowledge, monitor, and trigger adaptation on received data, as well as generation or adaptation of travelling plans.

4.3. Papify & SPIDER Integration

This integration activity brings together the monitoring capabilities of Papify with the software adaptivity management capabilities of SPIDER.

The SPIDER runtime manager focuses on executing reconfigurable dataflow graphs. This reconfiguration capability is mostly based on timing information and on an amount of used hardware resources.

As depicted in Figure 3 (left), SPIDER feedback loop provides timing and resource utilization information to a Global Runtime (GRT) with global knowledge of the system. As can be seen in Figure 3 (right), in case Papify is included within the SPIDER Local Runtime (LRT), managing one sub-system, and the jobs store their own instrumentation configuration, KPI information is conveyed to the GRT through additional feedback connections.

The PAPI library is commonly employed as a middleware for HPC tools that are focused on profiling, sampling and tracing but not on real-time reconfiguration [ADHIANTO 2010] [KNÜPFER 2008] [SCHLÜTTER 2014]. Papify-SPIDER integration will support these capabilities together with a real-time reconfiguration based on KPI values, e.g. energy consumption data. These KPI values can be either measured or estimated from the performance event occurrences provided by Papify. In this context, the combination of both tools will extend the reconfiguration criteria of SPIDER. Specifically, this combination of tools will be able to optimize system execution based on one or several KPIs at runtime.

To integrate both tools, a library to (1) monitor each actor and to (2) extract hardware usage information is being developed. The library (*eventLib*) is built on top of the PAPI interface. Specifically, energy consumption estimation models will be developed based on performance events. These estimation models are employed when the instrumentation circuitry is not present within the platform setup.

Three main steps can be distinguished in the SPIDER-Papify plan: (1) the integration of Papify with PREESM to automatically insert *eventLib* function calls within the generated code; (2) the study of KPI (initially, energy) estimation models based on performance events for the targeted CPS architectures; (3) the inclusion of KPI estimated values as inputs to the GRT Self-Adaptation Manager.

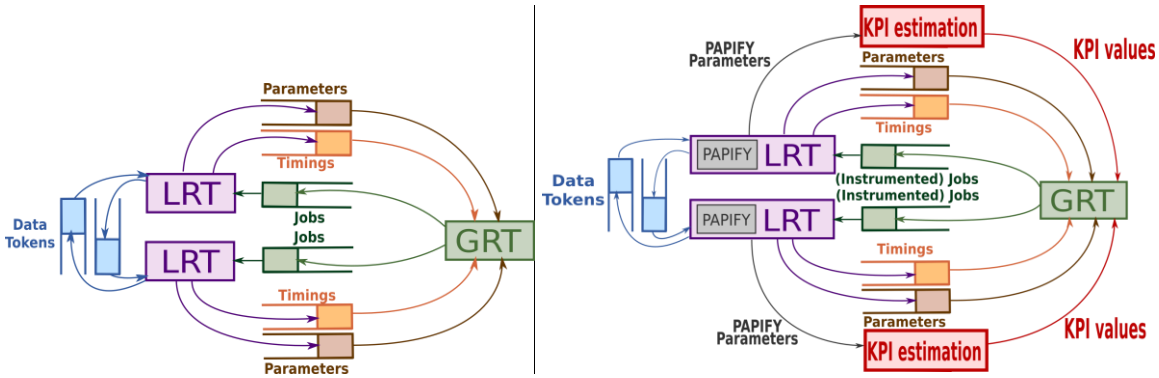

Figure 3: Original SPIDER workflow (left) vs Papify-SPIDER workflow (right)

4.4. SPIDER & MDC Integration

This integration activity combines SPIDER SW reconfigurability management with MDC HW reconfigurability management.

SPIDER and MDC show complementary characteristics that motivate for their integration. SPIDER provides software scheduling and memory management at runtime for general-purpose multi-core architectures. In this context, several processors and memory units are reprogrammed in order to exploit instantaneous parallelism. However, SPIDER supported processing elements do not include reconfigurable hardware blocks and adaptivity is based on an a priori knowledge of several metrics (latency, throughput and memory utilization) evaluated on changes in software parameters. Moreover, no strategy is considered that takes into account the computational energy consumption KPI. On the contrary, MDC provides a model-to-model compiler capable of merging several

dataflow applications, as well as a dataflow-to-hardware synthesizer that implements coarse-grained reconfigurable (CGR) systems. MDC profiles CGR system configurations, providing different metrics (area, power, frequency) and includes a power manager that enables clock- and power-gating techniques. The SPIDER and MDC tool combination will support energy and time adaptivity in heterogeneous multicore + CGR hardware. The proposed approach is to use CGR blocks as slave processing elements in the target system, and reschedule these processing elements from a host processor at runtime based on models of the instantaneous hardware behavior. At the moment, the SPIDER and MDC integration is focused on predicting the CERBERO computational KPIs latency, throughput and energy. The adaptation architecture is illustrated in Figure 4. An application graph, conforming to a dataflow Model of Computation, is dynamically scheduled by SPIDER. Depending on the scheduling, a hardware system composed of ARM cores and CGR architectures performs the computation. Software and hardware monitoring provides feedback to SPIDER with respect to the correct execution of the tasks. With regards to hardware monitoring, this feature will be provided through the integration of MDC and Papify (Section 4.7). In addition, rescheduling will also be triggered by sensors in order to adapt the computing layer to the environment changes or system needs.

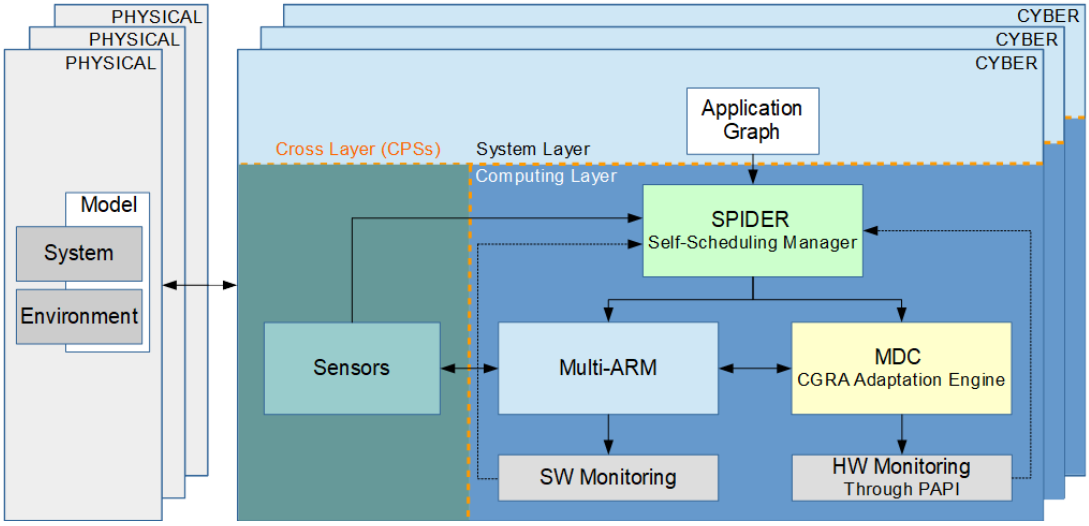

Figure 4: Integration of SPIDER – MDC Self-Adaptation Architecture

4.5. MDC & CAPH Integration

This integration activity combines the high-level hardware synthesis capabilities of CAPH with the hardware adaptation capabilities of MDC.

In order to provide a fully automated flow going from dataflow representations (high level models describing the applications to be accelerated) to coarse-grained (CG) reconfigurable architectures (hardware accelerators able to execute the different applications on a common substrate), the Multi-Dataflow Composer (MDC) tool (by UNISS and UNICA), that is basically a composition and optimization tool, requires the Register Transfer Level (RTL) hardware descriptions of the dataflow actors. One possibility is to derive such RTL descriptions directly from the dataflow models by

means of High-Level Synthesis (HLS) engines. In literature, HLS is a hot topic and several HLS engines have been proposed either from academy (e.g. Xronos [BEZATI 2013], CAPH [CAPH 2017], Bambu [BAMBU 2017]) and industry (e.g. Vivado HLS [XILINX 2018], Altera HLS Compiler [ALTERA 2018], Cadence Stratus [CADENCE 2018]). In the past, MDC was interfaced with the Xronos HLS engine and shared the same dataflow models of Computation to derive the RTL representation of the actors [SAU 2016]. In spite of benefits in terms of design time, Xronos adoption lead to a strong limitation: the target platforms were limited to platforms from one vendor, Xilinx FPGAs. The CERBERO toolchain having for objective to support a wide range of systems, this limitation led to the CAPH-MDC integration activities.

Generally speaking, there is no perfect HLS engine; the efficiency of the obtained systems is linked to the context of applications, and highly depends on the target device/technology, as well as on the initial specification format. A novel choice in this sense is CAPH, an open source HLS engine supporting dataflow models as specification format (close to the MDC ones) and target independent (CAPH generates generic RTL descriptions for any kind of FPGA vendor or even for ASIC flows)[3]. During the first year of CERBERO, MDC has been integrated with CAPH to provide a generic fully automated CG reconfigurable flow. The aim of this process is not to add another HLS engine, besides Xronos, among the MDC supported ones, but to allow the designer to choose any kind of HLS engine. This goal required two main actions:

- a CAPH-to-XDF parser has been defined in cooperation with Prof. Jocelyn Serot from the Blaise Pascal University of Clermont-Ferrand (creator of CAPH) to implement model-to-model transformations from CAPH dataflow [SEROT 2014] to MDC compliant dataflow (MPEG-RVC [BHATTACHARYYA 2011]).
- a generalization of the supported actor-to-actor communication protocol (so far fixed and compliant with MPEG-RVC actors only) to support in hardware any user-defined actor-to-actor communication handshake. Besides customizing the communication handshake, users have now additional features: they can link model parameters to the RTL, put additional modules between actors (such as FIFOs and fanouts) and specify system level signals (e.g. clock and reset).

With the MDC & CAPH integration it will be possible to automatically generate generic CG reconfigurable accelerators for the CERBERO adaptivity support. Such kind of reconfigurability can be also reached by means of imperative (non dataflow oriented) HLS engines, such as Vivado HLS or Altera HLS Compiler. A comparison between these design choices is being performed and a research article (with the joint effort of UNISS, UNICA, INSA and UPM) is to be submitted soon. To anticipate some of the achieved results, with respect to imperative HLS, MDC & CAPH integration provides unprecedented predictability for KPIs such as latency and throughput, before the synthesis stage. By contrast, Vivado HLS and Altera HLS Compiler tools provide latency only after synthesis and only for simple designs: if reconfiguration is implemented on the top of the imperative language, latency estimation may not be accurate or even available.

[3] Further details on the CAPH tool are provided in deliverable D5.6 - CERBERO Framework Components (Ver 1).

4.6. MDC & ARTICo3 Integration

This integration activity brings together the coarse grain HW adaptation of MDC and the fine grain HW adaptation of ARTICo3[3].

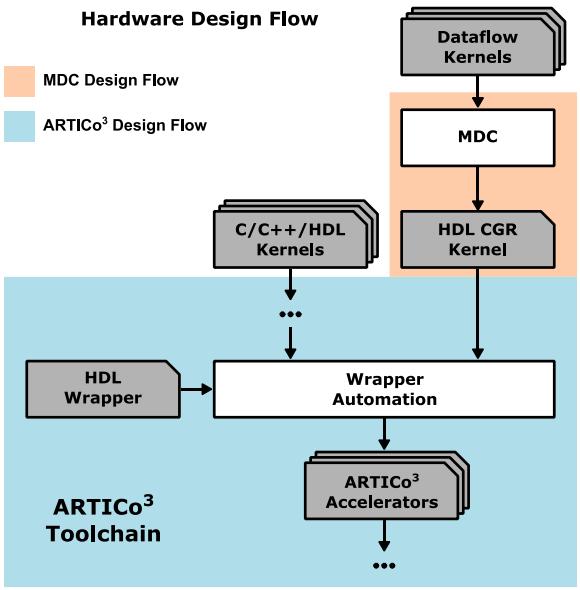

Figure 5: Integration of ARTICo3 – MDC Design Flow

The ARTICo[3] design flow, as described in D5.6, makes it possible to automatically integrate a Custom Hardware Accelerator (described in HDL or in C/C++) in a standard wrapper that provides a common interface with the rest of the processing architecture.

On the other hand, MDC provides N:1 composition/synthesis, as also defined in D5.6. Starting from a set of input dataflow specifications, MDC automatically generates a Coarse Grain Reconfigurable (CGR) HDL accelerator.

Figure 5 depicts the integrated ARTICo[3] – MDC Design Flow, where the generated CGR HDL accelerator acts as an additional entry point for the ARTICo[3] toolchain.

ARTICo[3]-based hardware accelerators are connected to the communication infrastructure in the system using a custom gateway, called Data Shuffler, which is able to dynamically alter its internal datapath to meet specific requirements of computing performance, energy consumption and fault tolerance. The gateway hides custom point-to-point interfaces (reconfigurable partitions) behind a standard AXI4 interface (static partition). Plug-and-play capabilities are enabled in user-defined custom accelerators by instantiating them in a wrapper module that provides:

- Local memory banks: configurable number of parallel access ports
- Configuration register bank
- Address translation logic: uniform memory map for the microprocessor; independent memory maps for user-defined logic

One of the MDC extensions generates processor-coprocessor systems, where the CGR accelerator is automatically wrapped with the logic necessary to communicate with the processor. This logic includes:

- Local memory
- Configuration registers bank
- Front-end and back-end that manage communication between the CGR accelerator and previous listed logic.
- AXI4 standard interface

As depicted in Figure 6, integrating an MDC CGR accelerator in ARTICo^3 requires that both front-end and back-end logic in the MDC accelerator have access to the memory banks and configuration registers of the ARTICo^3 wrapper. This is achieved by extending the MDC code that takes care of the processor-coprocessor system generation to be compliant with the ARTICo^3 register/memory structure (i.e. embedding the custom logic in the ARTICo^3 wrapper instead of doing it in a standard AXI4 template). This modification will be developed and demonstrated within the CERBERO project.

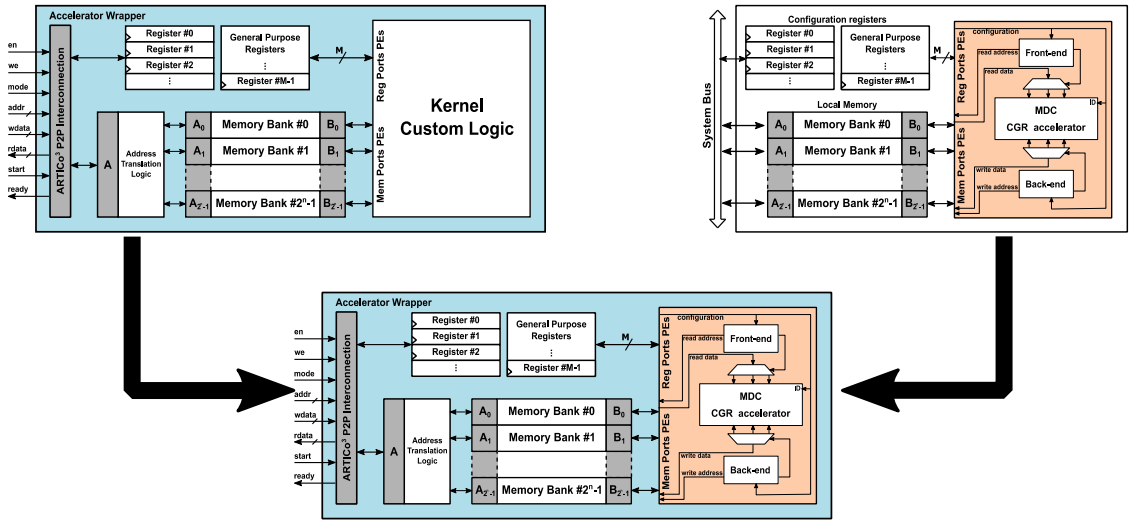

Figure 6: Integration of a MDC coarse grain reconfigurable accelerator in ARTICo^3.

4.7. Papify Monitors & Hardware Integration

This integration activity combines Papify monitoring and ARTICo3 HW reconfiguration.

As evoked in Section 4.4, Papify is a tool based on the Performance Application Programming Interface (PAPI). PAPI aims at providing event information directly extracted from a set of Performance Monitor Counters (PMC) existing in current modern processors. On heterogeneous platforms, a global overview of the platform performance is needed for driving self-adaptation, relying on a simultaneous access to key event occurrences of Hw and Sw components.

Hw-Papify is based on the access to a set of Hw-mapped registers that monitor events such as

- the *Number of Errors*, to increase reliability while monitoring multiple accelerators taking charge of the same functionality;
- the *Number of Clock-cycles*, to obtain the accelerator latency; and
- the *FIFO occupancy*, to detect communication bottlenecks within the accelerator to decide whether or not a specific kernel is parallelized.

To achieve this goal with Papify, user-defined PAPI components[4] are developed to access a Hw register file. Figure 7 (left) depicts a block diagram explaining the procedure to access these components. Specifically, the steps to connect Hw components in Papify are:

1. direct mapping of user-space virtual addresses to Hw accelerators[5] physical addresses using mmap(…);
2. command/data writing into Hw accelerators using virtual addresses;
3. inclusion of Hw register file write/read actions in the Hw PAPI component.

Additionally, in order to unify the access to performance events of any processing element (PE), a new abstraction layer is included to automatically access the specific PAPI component through a new library called eventLib. The behavior of this library is shown in Figure 7 (right), where PE monitoring is managed with three functions:

1. a configuration call to set the PE and PMC;
2. a start call to begin monitoring;
3. a stop function to end monitoring and retrieve performance information.

KPI estimation models based on events provided by Papify will be developed to replace actual measurements in those setups where the required instrumentation circuitry is not present.

A first integration of Papify with ARTICo³ has been achieved. Specifically, an ARTICo³ PAPI component and the *eventLib* library to unify the access methodology for both Sw and Hw PAPI components have been developed. In the planned integration activities, the *eventLib* library will be enhanced to support the access to any kind of user-defined PAPI component, hence generalizing the *Papify*-instrumentation of Hw accelerators, e.g. the coarse-grain reconfigurable accelerators generated by MDC.

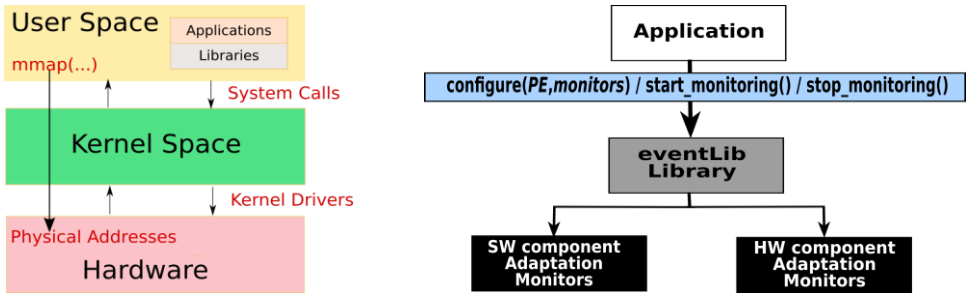

Figure 7: HW PAPI component diagram (left); *eventLib* abstraction layer diagram (right)

4.8. System-level Perspectives for Homogenizing Tool Integrations

From the previously presented integration activities, one of the key goals of CERBERO is to produce an integration framework capable of combining and interlinking consortium tools semantically across different tooling layers, in such a way that allows for self-adaptivity of the highly-heterogeneous CPS. The anticipated outcome of such point-to-point integration efforts is a fully-integrated toolchain that, beyond point-to-point

[4]PAPI Component Manual: http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Component_Developers_Manual
[5] The Hw accelerator has to be generated with a standard AXI4 interface.

integrations, features a high degree of self-adaptivity at system level. CERBERO semantic interfaces are being envisaged in such a way that fulfills this critical requirement of system-level adaptivity taking into account the overall system to autonomously handle the dynamic changes in its operational environment, thus making the system as transparent as possible to the application and enhancing overall system flexibility and self-maintainability.

In that sense, a group involving AI, INSA, UNISS, IBM, and TNO is preparing the system-level perspective by shaping system-level adaptation strategies considering current progress in CERBERO integration framework development. The following ongoing integration activities contribute to the realization of an adaptive framework at system level:

- Developing different technologies for implementing the CERBERO semantic interfaces for cross-layer data flow and tool-to-tool communication. These interfaces represent - or rather abstract - each layer of the CERBERO toolchain.
- Designing the architecture for cross-layer analysis, optimization, verification, rapid prototyping and continuous deployment.
- Building the framework demonstrator and verify the system-level integration results (first iteration) by means of both simulation and mathematical analysis (Section 2.3) of TNO and IBM, respectively, as well as by system prototyping with the CERBERO self-adaptation toolchain (built from Section 4 activities).

These activities complement the point-to-point tool integration activities in building a rich and consolidated CERBERO self-adaptation runtime manager.

5. Applicability of the CERBERO Self-Adaptation Capabilities to Use Cases

The objective of this section is to explain the intended effects of CERBERO self-adaptation in the context of the use cases. As shown in Deliverable D2.6 "Technical Specification", adaptation within the CERBERO project applies to all three use cases.

5.1. Planetary Exploration (PE)

The Planetary Exploration demonstrator has unique requirements such as a strong need of self-* properties, including self-adaptation, self-healing and self-awareness. For this reason, the use case will foster the capabilities of CERBERO runtime adaptation at different layers:

- **Triggers**: the developed technology will adapt the system starting from information acquired from sensors in the physical part (current sensors of the motors of the robotic arm), as well as information coming from monitors of the computing platform.
- **Adaptation fabrics**: CERBERO HW and SW adaptation fabrics, including managers and engines, will be used to solve the trajectory planning problem. In principle, SPIDER, ARTICo^3, ARTICo^3+MDC and ARTICo^3 + JIT HW composition will be tested.
- **Adaptation monitors**: execution will be monitored via using PAPI compatible functions through Papify.
- **Embedded models**: KPIs will be drawn from lightweight models, embedded in the system so they can operate autonomously.
- **Adaptation manager**: It will respond to specific situations, such as performance and energy utilization improvement or fault mitigation techniques. Design diversity, fault detection and HW accelerator migration within the FPGA fabric will be used to extend reliability, which will be demonstrated by a fault injection mechanism to measure fault detection and fault recovery times.

5.2. Ocean Monitoring (OM)

The Ocean Monitoring use case is based on a platform with strong real-time and energy constraints, and which delivers a computer vision processing and image evaluation pipeline within that platform. Adaptation against these constraints is a cross-cutting concern, and the case requires adaptation monitors, managers, and engine aspects at different levels of granularity to implement that adaptation.

The primary models of adaptation are planned, addressing the three primary sources of adaptation:

- **System**: The primary system adaptation involves tracking power levels and remaining data storage capacity, and ensuring that both usage levels are optimized, and that navigation is modified when safe levels require returning to an access point.

- **Environmental**: These range from location adaptation (using GPS sensors and navigation planning, with a world model to represent travel in the 3D medium), to light adaptation (using broad histogram data from camera outputs to detect colour and illumination and adjust light accordingly).
- **Human**: Adapting to human requirements focuses more on the informational systems and the image processing pipeline. Because the image processing will be implemented using existing Java platforms (likely either FastCV or VisionWorks), the initial Ocean Monitoring case will only adapt externally for image processing, to user image quality and relevance requirements as well as overall environmental and system monitors.

Because these components of adaptation vary in authority and source, the Ocean Monitoring case will adopt an established architecture for autonomous underwater vehicles, NIST's reference model architecture [ALBUS 1994]. In this model, there are adaptation monitors, managers, and engines within each component node within the hierarchical architecture, each of which can also relay to monitors above and to managers below, as shown in Figure 8 below.

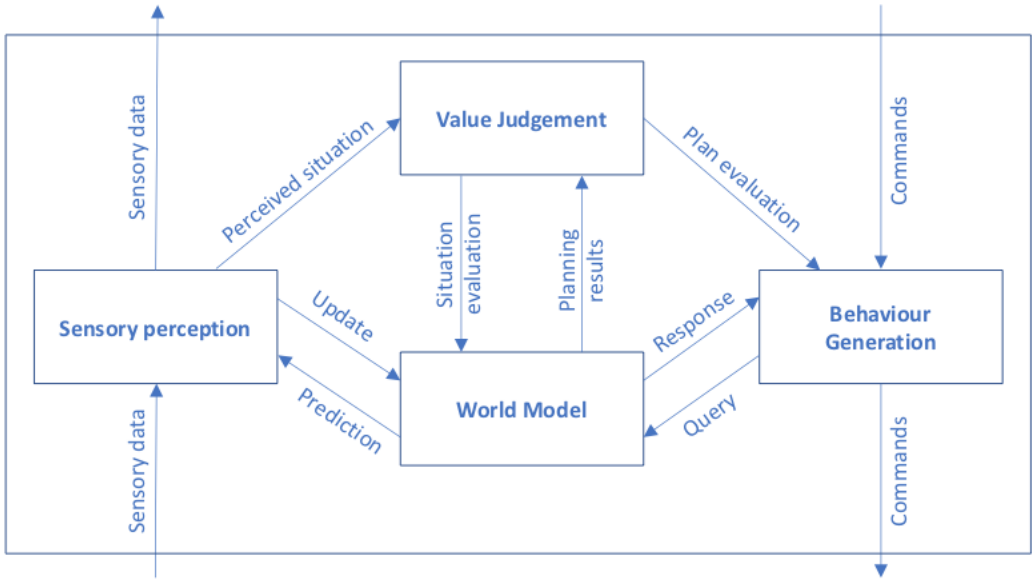

Figure 8. Overview of the RCS-4 model, from [ALBUS 1994]

In the Ocean Monitoring instantiation of this architecture, DynAA will form the primary adaptation model components. Value judgment will principally be implemented through custom logic, as well as behavior generation; however, in most cases the complexity of the aspects will be low.

The Ocean Monitoring vision processing pipeline has strong real-time and energy constraints, which allow CERBERO technologies to be tested on a cross-cutting test bed even beyond CERBERO direct usage within OM. The self-adaptive CERBERO toolchain constituted by SPIDER, MDC, CAPH, ARTICo3 and Papify offers a versatile solution to support such a stream processing pipeline over heterogeneous architectures. Subsets of the toolchain can be employed to target fully software pipelines (SPIDER + Papify), software /CGRA hardware pipelines (SPIDER + MDC + Papify) or software /DPR

hardware pipelines (SPIDER + ARTICo3 + Papify) on any type of image processing pipeline. Proofs of Concept will be developed to demonstrate the performance of the self-adaptive CERBERO toolchain on image processing pipelines.

5.3. Smart Travelling (ST)

In the Smart Travelling use case, the targeted self-adaptivity is at the application level. New driver support functionalities are developed, relying on the self-adaptive DynAA, SCANeR & MECA toolchain (Section 4.2), which will provide advice to the driver, based on predictions for possible routes and knowledge on the status of the car. As the driver support functionality needs to adapt its advice based on changes in the vehicle and the environment, the vehicle and environment will need to be monitored continuously and actions will need to be triggered in case situation has changed.

MECA will need to pose knowledge on the preferences of the driver and the actual situation (e.g. speed of the car) to determine the most appropriate advice on each given moment. In case of heavy or fast traffic, the driver support functionality should for example reduce the number of alternatives given to the driver to select from, and thus reduce distraction in the given moment.

The adaptivity in the smart travelling use case must consider CPSoS interactions and the driver actions to provide reconfiguration of the current execution status. Particularly, the system cannot perform self-adaptation without the authorization of the driver (except for critical failures or dangerous situations such as a battery short circuit), so the use case demonstrator will enable a decision-making process in which the driver is another layer of the system. Then, the adaptivity will take place in the three following levels:

- **System**: Based on the information provided by the different CPSs (e.g., the car status, charging poles, battery consumption prediction, etc.) the system can adjust different parameters for holding the required safety constraints.
- **Environmental**: The interaction with the environment has a fundamental role as the driving activities are influenced by the environment (e.g., traffic jams, weather conditions, etc.). The system shall monitor the environment to adapt the current route based on the current and predicted conditions.
- **Human**: The driver has the final decision about the proposed routes given by the demonstrator. Such routes could take into consideration driver's history and/or agenda to provide personalized routes adapted to him/her profile. Moreover, the demonstrator could monitor the driver health to adapt the route if the driver is tired or want to stop at an unplanned location for instance.

To achieve this adaptivity, the smart travelling use case will exploit the MECA tool to perform monitoring of the various levels and trigger the adaptive behaviors. To obtain information from the driver and from the system (car), MECA will be integrated with SCANeR for the demonstration. In the case of the environmental status, interfacing with map providers and weather forecast services will be done. For performing the route planning, a specific decision module will be implemented for this study case, which will obtain information from map providers, obtaining a set of potential routes. These routes, including information of in-route charging poles, will be supplied to the DynAA tool. Using that information, DynAA performs a simulation based on the battery model,

discarding those routes that cannot be completed and providing a set of itineraries which include charging stops (if required). MECA will filter and rank (based on user preferences) these itineraries, enabling the user to choose one. Then, during driving, MECA sets up route monitoring regarding the user and car status, meanwhile DynAA performs battery monitoring and simulation in the loop to ensure that the battery performs according to the predictions.

6. Conclusion: Self-Adaptation Manager Integration Agenda and Advances w.r.t State-of-the-Art

The point-to-point tool integration activities, described in Section 4, will follow the agenda of Figure 9. Vertical bars depict demonstration setups.

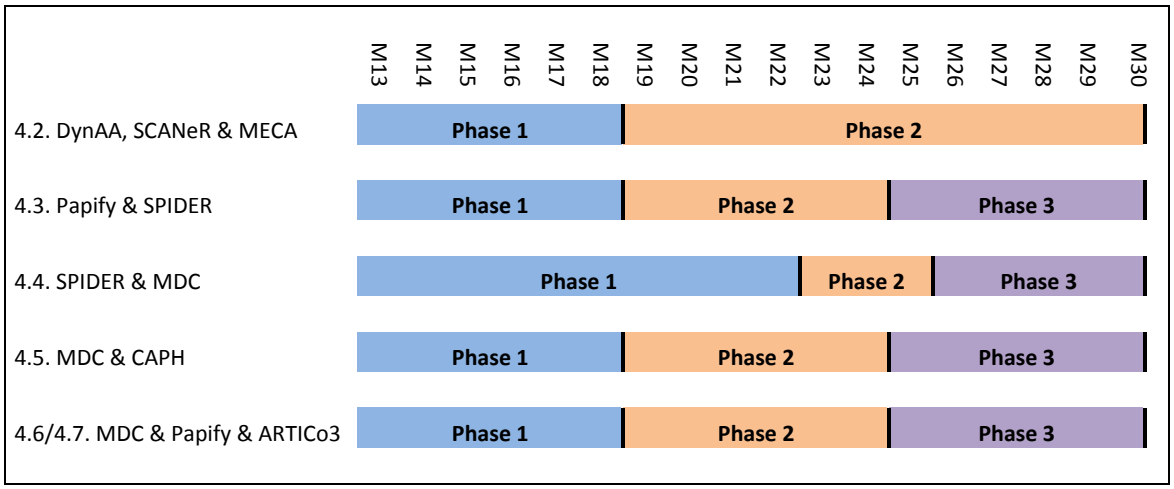

Figure 9: Self-Adaptation Manager Plan for integration

Some details follow on the self-adaptation manager integration activities:

- 4.2 – "DynAA, SCANeR & MECA Integration"
  - Phase 1
    - Detail one reference scenario, set up test environment with SCANeR, MECA and DyNAA, data fusion and synchronization,
    - Define and implement tool interfaces, integration verification.
  - Phase 2
    - Detail all scenarios, develop and integrate CERBERO intermediate format, add AOW for optimization of route planning
    - use additional CERBERO tools (like Preesm/Spider and Verification tool) to optimize / validate solution
- 4.3 - Papify & SPIDER Integration
  - Phase 1
    - automatically insert Papify eventLib function calls within SPIDER jobs and Local Runtimes (LRT),
  - Phase 2
    - derive generalized models to translate LRT (Papify Parameters) measurements into relevant CERBERO KPIs,
  - Phase 3
    - enable system self-adaptation, including KPI estimated values as inputs to the Global Runtime Self-Adaptation manager.

- 4.4 - SPIDER & MDC Integration
  - Phase 1
    - Integrate MDC & SPIDER by combining software and hardware adaptation based on varying application parameters,
  - Phase 2
    - verify this approach with respect to relevant CERBERO KPIs,
  - Phase 3
    - derive a proof of concept of the proposed approach in the context of CERBERO use case scenarios.
- 4.5 - MDC & CAPH Integration
  - Phase 1
    - complete, debug and assess the MDC & CAPH integration for coarse grain adaptive HW,
  - Phase 2
    - verify this approach with respect to relevant CERBERO KPIs
  - Phase 3
    - derive a proof of concept of the proposed approach in the context of the CERBERO use case scenarios
- 4.6/4.7 - MDC & ARTICo3 Integration, Papify Monitors & Hardware Integration
  - Phase 1
    - provide a unified hardware/software monitoring interface using Papify,
    - extend MDC generation code, to generate ARTICo3 compliant CGR accelerators,
  - Phase 2
    - provide an automated instrumentation methodology for heterogeneous hardware/software setups
    - experiment with multi-grain adaptivity, proposing different reconfiguration strategies according to relevant CERBERO KPIs
  - Phase 3
    - derive generalized models to translate heterogeneous hardware/software measurements into relevant CERBERO KPIs and enable system self-adaptation, providing KPIs to the CERBERO Self-Adaptation Manager.
    - derive a proof of concept of the proposed approaches in the context of the CERBERO use case scenarios

The previous sections have covered the on-going activities and plan for building the CERBERO self-adaptive management. The [MACIAS 2013] survey on self-adaptive systems gives insights on the main challenges of such management. The survey states that *"commonly used software modeling notations (e.g., UML) provide no means of describing and analyzing control and procedure to deal with uncertainty"*. The model-based, unified self-adaptation approach of CERBERO tackles this challenge by **developing methods and tools to represent and exploit adaptation opportunities**. Moreover, the models fostered by the CERBERO consortium are **tailored to the parallel**

and heterogeneous systems that constitute the state-of-the-art of cyber physical hardware platforms.

The same survey also claims that *"one of the shortcomings of software engineering for self-adaptive applications is the lack of actual case studies."* The CERBERO project overcomes this problem by **applying the proposed self-adaptation methods to *real-life* use cases of significant size**.

7. References

[ADHIANTO 2010]	Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J. and Tallent, N. R. (2010), HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurrency Computat: Pract. Exper, 22: (pp, 685–701). doi:10.1002/cpe.1553.
[ALBUS 1994]	Albus, J. S., & Rippey, W. G. (1994, September). RCS: A reference model architecture for intelligent control. In From Perception to Action Conference, 1994, Proceedings (pp. 218-229). IEEE.
[ALIREZAEYAN 2015]	Alirezaeyan, J., Yousefi, S., & Doniavi, A. (2015). Adaptive reliability satisfaction in wireless sensor networks through controlling the number of active routing paths. Microelectronics Reliability, 55(11), 2412-2422.
[ALOUANI 2017]	Alouani, I., Wild, T., Herkersdorf, A., & Niar, S. (2017, August). Adaptive Reliability for Fault Tolerant Multicore Systems. In Digital System Design (DSD), 2017 Euromicro Conference on (pp. 538-542). IEEE.
[ALTERA 2018]	www.altera.com/products/design-software/high-level-design/intel-hls-compiler/overview
[ASSAYAD 2017]	Ismail Assayad, Alain Girault. Adaptive Mapping for Multiple Applications on Parallel Architectures. Third International Symposium on Ubiquitous Networking, UNET'17, May 2017, Casablanca, Morocco. <hal-01672463>
[BAMBU 2017]	https://panda.dei.polimi.it/?page_id=31
[BENNACEUR 2014]	Bennaceur, A., France, R., Tamburrelli, G., Vogel, T., Mosterman, P. J., Cazzola, W., ... & Emmanuelson, P. (2014). Mechanisms for leveraging models at runtime in self-adaptive software. In Models@ run. time (pp. 19-46). Springer, Cham.
[BETTY 2009]	Betty, H. C., Rogério, D. E., Holger, G., Inverardi, P., & Magee, J. (2009). Software engineering for self-adaptive systems. Lecture Notes in Computer Science, 5525.
[BEZATI 2013]	E. Bezati, S. Casale-Brunet, M. Mattavelli & J. Janneck, Synthesis and optimization of high-level stream programs. In Proceedigns of the *Electronic System Level Synthesis Conference, 2013.*
[BHATTACHARYYA 2011]	S.S. Bhattacharyya, J. Eker, J.W. Janneck, C. Lucarz, M. Mattavelli, and Mickaël Raulet, Overview of the MPEG Reconfigurable Video Coding Framework. In Journal of Signal Processing Systems, Volume 63, Issue 2, pp.251-263, 2011.
[BRUN 2009]	Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., ... & Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In Software engineering for self-adaptive systems (pp. 48-70). Springer, Berlin, Heidelberg.
[CADENCE 2018]	https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/synthesis/stratus-high-level-synthesis.html

[CAMARA 2017]	Cámara, J., Bellman, K. L., Kephart, J. O., Autili, M., Bencomo, N., Diaconescu, A., ... & Tivoli, M. (2017). Self-aware Computing Systems: Related Concepts and Research Areas. In Self-Aware Computing Systems (pp. 17-49). Springer International Publishing.
[CAPH 2017]	http://caph.univ-bpclermont.fr/CAPH/CAPH.html
[CERBERO 2017]	http://www.cerbero-h2020.eu
[GAUTIER 2013]	Gautier, T., Lima, J. V., Maillard, N., & Raffin, B. (2013, May). Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on (pp. 1299-1308). IEEE.
[GEROSTA 2016]	Gerostathopoulos, I., Bures, T., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., & Plouzeau, N. (2016). Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations. Journal of Systems and Software, 122, 378-397.
[HAN 2017]	Han, M., Park, J., & Baek, W. (2017, March). CHRT: a criticality- and heterogeneity-aware runtime system for task-parallel applications. In Proceedings of the Conference on Design, Automation & Test in Europe (pp. 942-945). European Design and Automation Association.
[HOFFMANN 2011]	Hoffmann, H., Maggio, M., Santambrogio, M. D., Leva, A., & Agarwal, A. (2011). SEEC: a general and extensible framework for self-aware computing. MIT-CSAIL-TR-2011-046 Technical Report, MIT.
[HORMATI 2014]	Hormati, A. H., Choi, Y., Kudlur, M., Rabbah, R., Mudge, T., & Mahlke, S. (2009, September). Flextream: Adaptive compilation of streaming applications for heterogeneous architectures. In Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on (pp. 214-223). IEEE.
[KNÜPFER 2008]	Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., ... & Nagel, W. E. (2008). The vampir performance analysis tool-set. In Tools for High Performance Computing (pp. 139-155). Springer, Berlin, Heidelberg.
[MACIAS 2013]	Macías-Escrivá, F. D., Haber, R., Del Toro, R., & Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. Expert Systems with Applications, 40(18), 7267-7279.
[PALUMBO 2017]	Palumbo, F., Sau, C., Fanni, T., Raffo, L. (2017) Challenging CPS Trade-Off Adaptivity with Coarse-Grained Reconfiguration. [Proceedings to be published]Applications in Electronics Pervading Industry, Environment and Society Conference (ApplePies).
[QUAN 2016]	Quan, W., & Pimentel, A. D. (2016). A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems. Design Automation for Embedded Systems, 20(4), 311-339.
[ROBSON 2017]	Robson, M. P., Buch, R., & Kale, L. V. (2016, November). Runtime coordinated heterogeneous tasks in charm++. In Proceedings of the Second Internationsl Workshop on Extreme Scale Programming

Models and Middleware (pp. 40-43). IEEE Press.

[SALEHIE 2012] Salehie, M., & Tahvildari, L. (2012). Towards a goal-driven approach to action selection in self-adaptive software. Software: Practice and Experience, 42(2), 211-233.

[SAU 2016] C. Sau, P. Meloni, L. Raffo, F. Palumbo, E. Bezati, S. Casale-Brunet & M. Mattavelli, Automated Design Flow for Multi-Functional Dataflow-Based Platforms. In Journal of Signal Processing Systems, Vol. 85, Issue 1, pp. 143-165, 2016.

[SCHLÜTTER 2014] Schlütter, M., Mohr, B., Morin, L., Philippen, P., & Geimer, M. (2014). Profiling Hybrid HMPP Applications with Score-P on Heterogeneous Hardware. In International Conference on Parallel Computing (No. FZJ-2014-01861). Jülich Supercomputing Center.

[SCHNEIDER 2009] Schneider, S., Andrade, H., Gedik, B., Biem, A., & Wu, K. L. (2009, May). Elastic scaling of data parallel operators in stream processing. In Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on (pp. 1-12). IEEE.

[SEROT 2014] J. Serot & F. Berry, High-Level Dataflow Programming for Reconfigurable Computing. In Proceedings of the IEEE 26th International Symposium on Computer Architecture and High Performance Computing Workshops, 2014.

[STREIT 2013] Streit, K., Hammacher, C., Zeller, A., & Hack, S. (2013, January). Sambamba: runtime adaptive parallel execution. In Proceedings of the 3rd International Workshop on Adaptive Self-Tuning Computing Systems (p. 7). ACM.

[THOMAS 2004] Thomas A., Becker J. (2004) Dynamic Adaptive Runtime Routing Techniques in Multigrain Reconfigurable Hardware Architectures. In: Becker J., Platzner M., Vernalde S. (eds) Field Programmable Logic and Application. FPL 2004. Lecture Notes in Computer Science, vol 3203. Springer, Berlin, Heidelberg

[VANDEWOUDE 2003] Vandewoude, Y., Rigole, P., Urting, D., & Berbers, Y. (2003). Draco: An adaptive runtime environment for components. Appendix of the EMPRESS deliverable for Run-time Evolution and Dynamic (Re) configuration of Components.

[WOLF 2014] Wolf, M. (2014). High-performance embedded computing: applications in cyber-physical systems and mobile computing. Newnes.

[XILINX 2018] www.xilinx.com/products/design-tools/vivado/integration/esl-design

[VENKATASUBRAMANIAN 2014] K.K. Venkatasubramanian, C. Shue. "Adaptive Information Security in Body Sensor-Actuator Networks." In Proc. of 2014 Usenix Summit on Health Information Technologies Aug 2014.

[WEN 2010] Wen, M., Yin, Z., Long, Y., & Wang, Y. (2010). An adaptive key management framework for the wireless mesh and sensor networks. Wireless Sensor Network, 2(09), 689.

[YUN 2015] Yun, J., Park, J., & Baek, W. (2015, June). HARS: A heterogeneity-aware runtime system for self-adaptive multithreaded applications. In Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE (pp. 1-6). IEEE.

[ZHANG 2015] Zhang, F., Cao, J., Khan, S. U., Li, K., & Hwang, K. (2015). A task-level adaptive MapReduce framework for real-time streaming data in healthcare applications. Future Generation Computer Systems, 43, 149-160.