Information and Communication Technologies (ICT) Programme

Project N°: H2020-ICT-2016-1-732105

D4.2: Self-adaptation Manager

Lead Beneficiary:	INSA
Workpackage:	WP4
Date:	30/06/2019
Distribution - Confidentiality:	Public

Abstract: This document presents an update of the tools integration activities, conducted for building the CERBERO self-adaptation management. It focuses on the integration plan of the CPS and CPSoS adaptation managing solutions.

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
Julio Oliveira	TNO
Claudio Rubattu	UniSS/INSA
Carlo Sau	UniCA
Tiziana Fanni	UniCA
Leonardo Suriano	UPM
Daniel Madroñal	UPM
Katiuscia Zedda	AI
Leszek Kaliciak	AS
Hans Myrhaug	AS
Ayse Goker	AS
Stuart Watt	AS
Michael Masin	IBM
Pablo Sanchez de Rojas Mendez	TASE
Francesco Regazzoni	USI

Document Revision History

Date	Ver.	Contributor (Beneficiary)	Summary of main changes
2019.05.06	0.4	Maxime Pelcat (INSA)	Table of content draft, Ack by partners.
2019.05.23	0.5	Maxime Pelcat (INSA)	Individual invitations to edit.
2019.05.31	0.6	Maxime Pelcat (INSA)	Integration of sections by UNICA, UNISS, UPM, and TNO
2019.06.11	0.9	Maxime Pelcat (INSA)	Integration of sections by other partners.

2019.06.12	1.0	Maxime Pelcat (INSA)	Finalization of a version for review.
2019.06.26	1.1	Maxime Pelcat (INSA)	Integration of reviews by F. Palumbo and A. Goker.

Table of contents

1. Executive Summary

This document is an update of D4.4, detailing the status of CERBERO work on self-adaptation management at M30.

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

1.1. Structure of the Document

While Section 2 relates advances since M15 on the different tool-to-tool integrations, Section 3 focuses on adaptation framework integration as a whole. Section 4 gives updates on the applicability of the CERBERO self-adaptation capabilities to the CERBERO use cases. Finally, Section 5 concludes on the built self-adaptation framework.

1.2. Related Documents

Deliverable D4.4 (M15) has detailed the rationale for the selected tools and the integration activities conducted within the CERBERO project in order to support self-adaptation in the contexts of CERBERO use cases. This update aims at explaining the actions conducted since M15 and at comparing them to the initial plan. The presented tool integration activities follow the integration methodology presented in D5.5.

Deliverable D2.1 has defined the Key Performance Indicators (KPIs) to be observed in the CERBERO use cases. The technologies presented in this document, while aiming at a generic adaptation support of applicative and architectural modifications, target primarily the KPIs listed in D2.1.

Deliverable D4.1 (M30) has described the different multi-layer adaptation strategies of the project. The CERBERO approach for self-adaptation has also been compared in Deliverable D4.4 to the state-of-the-art of system adaptation management. This document complements D4.1 by explaining how tool-to-tool and multi-tool integrations are conducted in order to support these adaptation strategies.

Deliverable D5.6 and its update D5.2 present the features of individual tools in the CERBERO framework, including design-time and run-time tools. This document concentrates on run-time tools and focuses on their integration. For more details on design-time tools and their integration, please refer to Deliverable D5.2.

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.

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

CTR id	CTR Description	Link with the D4.2 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.
0009	CERBERO SHALL develop integration methodology and framework.	The adaptation infrastructure and tools are part of the CERBERO framework.
0016	CERBERO tools SHOULD be tested vs. state-of-the-art.	The CERBERO integrated tools are tested vs. state-of-the-art solutions. The built self-adaptation manager brings unique design automation features, as explained in the following sections and in Deliverable D4.1.
0019	CERBERO technology providers SHALL coordinate technical support for their tools with use case engineers.	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. M30 Progress w.r.t. Self-Adaptation Manager Integration Plan

Figure 1 and Figure 2 recall the tool integration plan stated in D4.4. Five parallel activities have been conducted within the consortium to build the CERBERO self-adaptation management, numbered (1) to (5) hereafter. Self-adaptation is a combination of awareness and reconfiguration. To implement self-adaptation, the CERBERO self-adaptation manager relies on of four types of elements: *fabrics, monitors*, *managers* and *engines*. While monitors provide sensing capabilities to the decision process conducted by managers, engines provide processing reconfiguration capabilities at different levels. Fabrics are either software or hardware facilities that implement processing, reconfiguration, or sensing operations. The details of the CERBERO multi-layer runtime adaptation strategies are explained in Deliverable D4.1.

The CERBERO integration activities aim at the management of adaptivity at two different scales, as depicted in Figure 1 and explained in Deliverable D4.1:

- At the Cyber-Physical System of Systems level (CPSoS),
- At the Cyber-Physical System level (CPS).

The two scales of adaptivity can be combined for the study of a single system. The adaptation technologies built in CERBERO are all generic to the different triggers of adaptation (e.g. user or environment related), and cover software, hardware, and sensor fabrics.

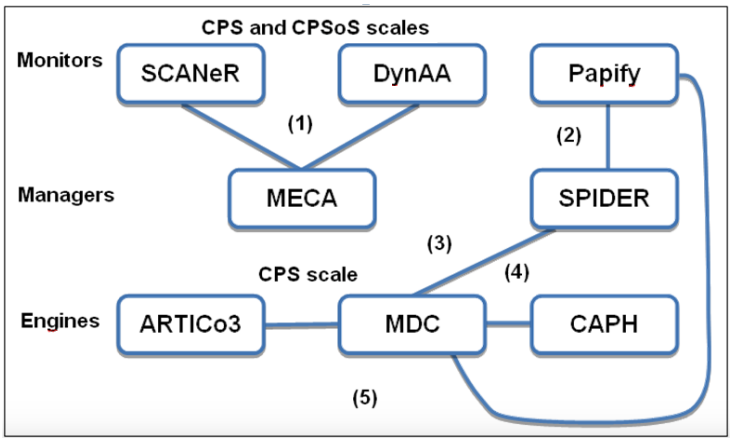

Figure 1: Overview of the main runtime tools integration activities.

The next sections will refer to the integration plan displayed in Figure 2 and stated at M15 in D4.4. The main objective of this document is to present an update of this plan at M30.

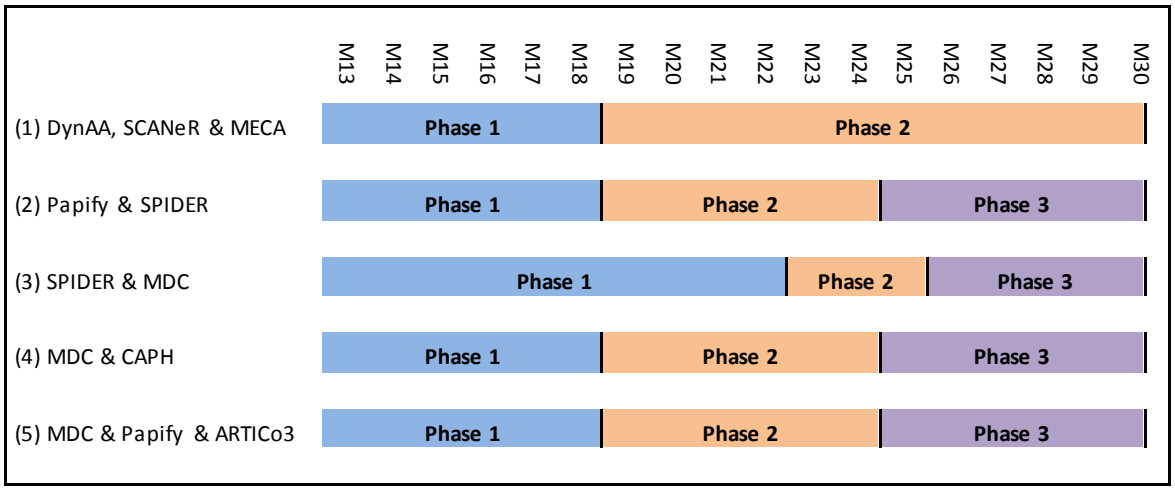

Figure 2: Self-Adaptation Manager Plan for Integration

The following list details the different planned phases, as displayed in D4.4:

- (1) 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 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
- (2) 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.
- (3) 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) 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
- (5) MDC & Papify & ARTICo3 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

Next sections detail the tool integration activities conducted until M30 for building CERBERO self-adaptation management, and progress w.r.t. the plan drawn in D4.4 (M15).

2.1. Integration Activity (1): DynAA, SCANeR & MECA

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 3, 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 receives 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 determines if adaptation is required.

The adaptation itself can be, for example:

- starting an investigation of alternative routes, in case planned charging poles are found to be out of service. This is an adaptation triggered by the changes in the environment;
- proposing advised routes based on impact analysis performed on DynAA, or triggered by route request from driver. This is an adaptation triggered by the user; or
- advising the user to reduce energy consumption by reducing or switching off the air- conditioning. This happens in case the battery charge is found to be critically low. Such adaptation is triggered by the system state.

MECA allows for some self-adaptation: the adaption is either triggered by the user (through the UI), or by modifications of the system's environment and detection of potential problems (e.g. limited availability of charging poles). The test of self-adaptation based on environment is done via manipulating charging-pole status (force out of service), which triggers MECA to re-plan. Environmental changes such as traffic jams and weather conditions are currently not simulated.

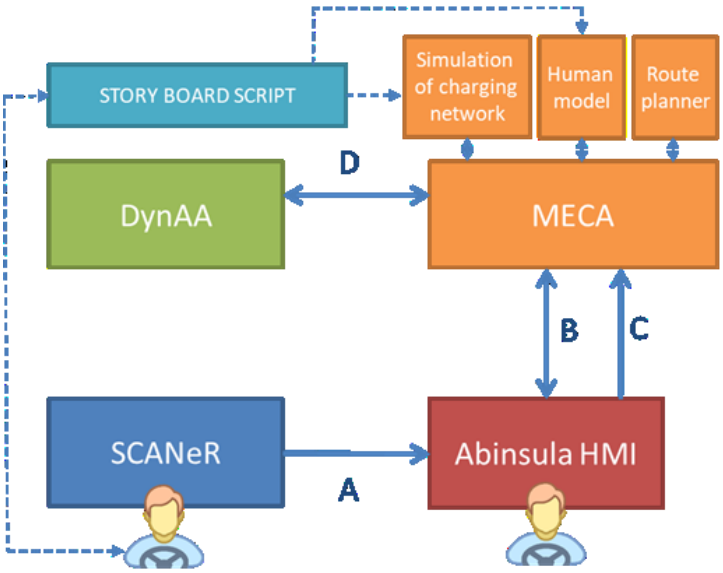

Figure 3: Schematic overview of MECA, DynAA, SCANeR, and Abinsula HMI interworking.

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

Current state of integration:

There are no significant modifications to the integration plan, nor to the adaptation mechanisms in the actual demonstration setup. All continue to work as depicted in the picture. At the current integration state, all major integration activities between the different components have been completed. We are currently refining and extending the interfaces with more capabilities. For example, MECA will report to DynAA the battery

state-of-charge read from SCANeR. This information is used to improve the DynAA routing analysis.

Specifically, TNO included advanced simulation library versions in DynAA for the Battery and Motor models. They will be used in future demonstrators. Implementation using DynAA system-in-the-loop version for this part will be an optional extension for the final demonstration in December 2019.

Furthermore, a Human-Machine user Interface (HMI) was added between SCANeR and MECA to allow for real time car monitoring on the map at the same time reduce the previous number of interfaces between the modules. The Human Machine Interface is implemented and provided by Abinsula.

2.2. Integration Activity (2): Papify & SPIDER

Current state of integration:

The Papify toolbox performs automatic PAPI-based instrumentation of dynamic dataflow applications. This toolbox is the result of combining Papify with the dataflow Y-chart based design framework, PREESM, and its counterpart run-time reconfiguration manager, SPIDER. The built toolbox takes the form of an automatic code generation for static and dynamic applications, a dedicated library to manage run-time monitoring and two User Interfaces (UIs) to ease both the configuration and the analysis of the monitoring.

Phase 1 of integration is finished, i.e. Papify eventLib function calls are automatically inserted within SPIDER jobs and Local Runtimes (LRT). Additionally, the support for monitoring heterogeneous architectures has been included.

For Phase 2, four goals have been defined to derive generalized models to translate LRT (Papify Parameters) measurements into a relevant KPI. A bottom-up approach has been adopted, starting from implementation and measurements to build a relevant model. System energy consumption has been chosen as the main target KPI, as it is important to both Planetary Exploration and Ocean Monitoring use cases (cf. Deliverable D2.1). A Massively Parallel Processor Array (MPPA-256-N) platform has been selected as a challenging experimental platform. This platform gathers 16 clusters with 16 cores within each of them, i.e., there are 256 processors working in parallel.

The four successive conducted activities have been:

1. Analyze the different ways to report instantaneous power consumption of the platform for system adaptation

Two different mechanisms have been studied: 1) to use an application called k1-power, which has a high sample rate but low precision; 2) to use a multimeter plugged on the power supply of the platform, which has a low sample rate but high voltage resolution. Finally, it has been decided to develop a custom monitor based on accessing a sensor (not available from k1-power) and to correct its low precision by applying a correction

function. This correction function has been obtained by characterizing the current measured by the sensor and comparing it with the one measured with the multimeter. Finally, a linear regression technique has been applied to obtain the correction function. With this new methodology, both a high sample rate and a high precision are reached.

2. Include Performance Monitoring Counters (PMCs), and test KPI retrieval on a relevant embedded platform

As a preliminary step, the available monitoring infrastructure already existing within the platform has been analyzed. During this analysis it has been stated that the infrastructure was complete enough to be used to characterize the different resources of the platform, as each of the 256 cores of the platform has its own counters and all can be monitored simultaneously. As a second step, PAPI, the standard monitoring library has been ported to the platform. This library provides access to the PMCs and the different events that occur in the processor during the execution of applications. Additionally, this library is divided into two layers: the lower one is architecture-specific and it is aimed to deal with the peculiarities of each platform, while the upper layer aims at transparently managing the low level and to provide the user with an abstraction layer to uniformly access to the PMCs of any of the employed platforms. The work done supports the low level of both the second and third generations of the MPPA architecture, where the second generation is the one previously referred as MPPA-256-N (Bostan version), while the third one will be called Coolidge (available soon).

3. Create a benchmark for testing Papify/Spider integration

During this step, a set of stress applications has been built, where the resources of the platform (cache memory, communication resources, floating point units, etc) are stressed differently. In this sense, the benchmark has been divided in two: communication stress and processing stress.

The former is composed of applications where the different clusters (16) existing within the platform exchange data with other clusters or with the Input/Output (I/O) subsystem. In these tests, 4 communication situations may happen:

a. Cluster sending (or receiving) data to (from) another cluster. In this test the shared memory (SMEM) existing within the clusters will be both read to and written from.
b. Cluster sending data to the I/O subsystem. In this test the SMEM will be read from while the DDR memory existing in the I/O will be written to.
c. Cluster receiving data from the I/O subsystem. In this test the SMEM will be written to while the DDR memory will be read from.
d. I/O subsystem sending data to itself. In this test the DDR memory will be both read from and written to. This test is based on stressing the computation resources inside each cluster.

Each of the tests composing this part of the benchmark aims at stressing the computation resources differently. For example, there are tests focused on stressing the instructions executions, the cache memory usage, or performing matrix multiplications to stress the whole system.

4. Demonstrate the use of the built Model of Architecture in SPIDER/Papify

The objective of this part is to develop a Linear System-Level Architecture (LSLA) Model of Architecture (MoA) of the MPPA-256-N platform. This model is characterized for isolating the communication and the processing from each other. Consequently, the benchmark developed has been used to obtain linear models representing each part of the architecture. Additionally, the technique used to extract the equations that characterize the resources, again, has been the linear regression.

For the communication part, it has been decided to use the size of the data token transmitted to compute the energy employed during its transmission. To properly characterize this part of the platform, the source and the destination of the token are taken into account, as using the DDR memory has been proven to be 10 times more energy consuming than using the SMEM of the clusters. Figure 4 gathers the results obtained for this part of the model. As it can be seen, the differences in the energy consumption when using (or not) the DDR memory are clear and properly reflected in the estimation.

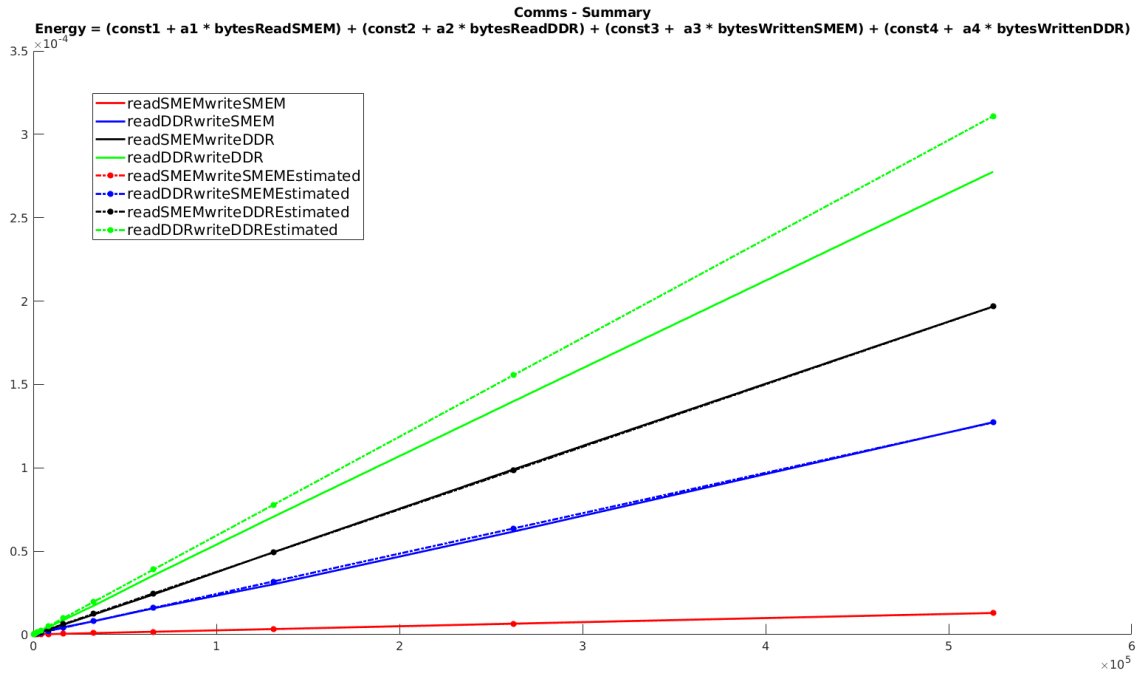

Figure 4 - Communication model of the MPPA.

For the computational part, the different workloads of the tests composing the stress benchmark have been characterized using the events extracted through the PMCs and the PAPI library. By doing so, a linear and cumulative model has been extracted and, as a first approach, it has been validated with some samples of the benchmark obtained an average accuracy of 85%. Figure 5 depicts the estimation for the computation when stressing the MPPA-256 board with different workloads and different type of computation.

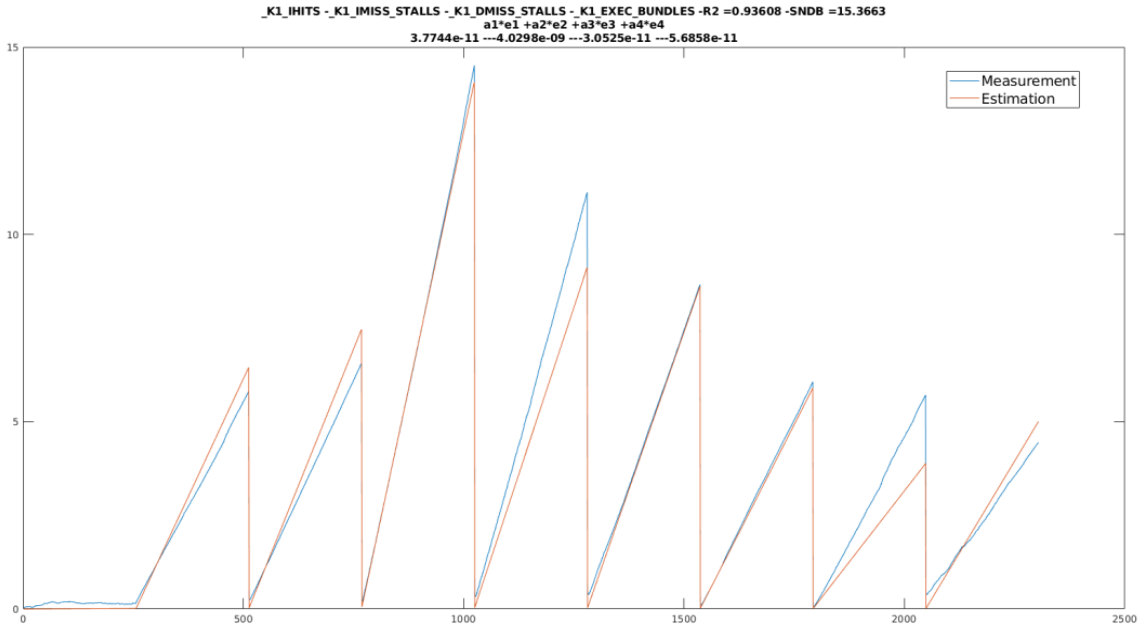

Figure 5 - Computational model of the MPPA.

For phase 3, to enable system self-adaptation based on internal system state triggers, there is one component of SPIDER that suffers the deepest change: the mapping algorithm. Using multi-objective optimization, both timing and energy can be taken into account, considering: 1) a performance objective (e.g. a minimum of X fps), 2) the static energy consumed by the system (the more PEs are used, the more static energy will have the system), 3) the communication energy cost (for example, on the MPPA, sending to the I/O (represented as one type of PE) costs more than sending data between clusters (represented as another type of PE) and 4) the dynamic energy consumption (one actor could consume different amount of energy depending on the type of PE executing the actor). A multi-objective optimization algorithm has been assessed by simulation and offers promising results on trading-off between time and energy, validating the proposed multi-KPI API an efficient way to feed self-adaptation.

Progress w.r.t. the initial plan:

- Phase 1: *automatically insert Papify eventLib function calls within SPIDER jobs and Local Runtimes (LRT)*: this work has been conducted w.r.t. the plan, and the obtained support is generic to many types of platforms,
- Phase 2: *derive generalized models to translate LRT (Papify Parameters) measurements into relevant CERBERO KPIs*: this work has been conducted w.r.t. the plan. Model building has been conducted on many-core energy estimation models and experimented on a complex MPPA-256-N platform.
- Phase 3 : *enable system self-adaptation, including KPI estimated values as inputs to the Global Runtime Self-Adaptation manager*: this work has been conducted w.r.t. the plan. The Papify time PMC interface combined with the precise energy

KPI model offer the appropriate input for multi-objective optimization algorithms in the Global Runtime Self-Adaptation manager.

Maturity of the built self-adaptation management:
The integration has led to Papify being tightly integrated to SPIDER and to both PMC and model-based KPIs being extractable from an execution. Manual steps are still required for using model-based adaptation while PMC-based adaptation is fully automated.

2.3. Integration Activity (3): SPIDER & MDC

Current state of integration:
The integration activity between the SPIDER dataflow-based runtime manager and the MDC dataflow-to-hardware suite has been conducted in order to combine Coarse-Grain Reconfigurable (CGR) hardware accelerators with multi-core software architectures. Both tools are based on a dataflow Model of Computation (MoC, cf. deliverable D3.3) that can be used to separate temporal/parallelism problems from functional problems in hardware design. Moreover, the modularity of the dataflow representations favors a natural splitting of the computation into different blocks without side effects, making it possible to automatically map these blocks onto heterogeneous software and hardware Processing Elements (PEs). Thus, the idea behind this SPIDER-MDC integration is to leverage on a software-hardware datapath design flow and to develop and manage dataflow-based autonomous reconfigurable systems, as requested by the CERBERO model-based approach to system design.

A proof of concept of the proposed approach has been derived and was fully functional at M24. As a first step, the PiSDF-based description of an edge detection application, that implies one actor with reconfigurable workloads requiring hardware acceleration with low reconfiguration overheads, has been implemented. In a second step, starting from the description of two single kernels (Sobel and Roberts filtering) and using the CAPH language to generate hardware, a Coarse-Grained Reconfigurable accelerator using MDC has been generated. One instance of such accelerator has been deployed onto an FPGA device implemented in a system on chip including 2 ARM processors (Xilinx Zynq-7000 ARM/FPGA SoC). In this setup, the SPIDER runtime manager schedules and maps at runtime the whole application graph. It sends the execution orders to different slave (processors and accelerator) transparently. The user can trigger the hardware reconfiguration using switches present on the PYNQ-Z1 board.

SPIDER-MDC integration requires the decision-making code of SPIDER to have information on the performance of the execution to decide mapping, scheduling and memory management (among others). This information is either gathered reactively

(from execution of previous iterations of the algorithm by e.g. Papify) or predictively (from a model of future performances). The capacity to predict performances is particularly important in the context of SPIDER-MDC where parameters may change constantly and a very limited and imprecise knowledge may be extracted from the study of previous code iterations. As a consequence, a study on latency-based adaptation methods has been conducted within the SPIDER-MDC integration activities. This study complements the energy-oriented study presented in the previous section and necessity has arisen from the need of real-time systems to act predictively. The idea of this study is to build and train simple Models of Architecture (MoA), executable at run-time, to predict the full hardware/software performance of SPIDER-MDC based on limited information on the platform and algorithm. To handle the complete heterogeneous system, the runtime system can then use either the MoA predicted latency or the Papify measured latency. To build the MoA, it is necessary to retrieve a certain "application activity" related to the latency of the I/O paths that are present in the application graph. Indeed, only one path (i.e. one chain of processing actors and communications) causes the latency between arrival of data and production of results, even if data and task parallelisms create many concurrent paths. Once a model of application activity is built, a training phase has to be performed with multiple applications in order to get a model of the architecture that is capable of providing latency estimation valid for several applications. This model should be usable from limited application and architecture information. The objective of this MoA study is to offer early information for SPIDER to decide the adaptation strategy starting from a latency model for the chosen hardware solution.

As the connection to PREESM and SPIDER have a strong impact on the internal behavior of MDC, the tool-to-tool connection is also discussed extensively in Deliverable D5.2.

Progress w.r.t. the initial plan:

The implementation is quite aligned with respect to the initial plan:

- Phase 1 (Integrate MDC & SPIDER by combining software and hardware adaptation based on varying application parameters) has been implemented and tested on a proof of concept application;
- Phase 2 (verify this approach with respect to relevant CERBERO KPIs) has been conducted, focusing on the processing latency KPI that is essential to the different use cases. Indeed, deliverable D2.1 "CERBERO Scenarios Description" shows that latency (also called "response time") is central to CERBERO use cases. The built model of architecture is defined to be very lightweight and support adaptivity at runtime with respect to the different configurations of the system;
- Phase 3 (derive a proof of concept of the proposed approach in the context of CERBERO use case scenarios). Proof-of-concept algorithms (SIFT keypoint detector, and stereo matching) have been implemented and used

to train and test the built the model. These algorithms are coherent with the contexts of the PE and OM use cases.

Maturity of the built self-adaptation management:

The combination of the MDC and SPIDER tools has been achieved and allows a user to switch between hardware and software tasks at runtime through the rescheduling and the remapping the application onto the proper processing elements depending on the parameters. The new Model of Architecture offers a valuable input to the self-adaptation loop, making it possible to schedule based on predicted future performance. The use of the built MoA in SPIDER still requires manual steps while user-triggered adaption is fully automated.

2.4. Integration Activity (4): MDC & CAPH

CAPH is external to the CERBERO consortium but its integration has been a strong asset for the support of coarse grain reconfiguration within the CERBERO engines. Indeed, the CAPH compiler provide advanced constructs to build and manipulate complex data-paths to which MDC can insert coarse grain reconfiguration support.

Current state of integration:

The integration between MDC and CAPH was already in a good shape at M15: the tools were connected together by providing CAPH with a RVC-CAL backend, in order to generate also dataflow models compliant with the MDC supported ones; and by providing MDC with a hardware communication protocol generalization, so that it has been made able to manipulate any kind of hardware actor, generated either manually or by means of high level synthesis tools like CAPH.

Going from M15 to M30, only refinements and tests have been applied to the integration between MDC and CAPH. In particular, besides debugging the integration, on the MDC side the protocol generalization has been aligned with the MDC co-processor generator additional feature (see D5.2). Moreover, the integration has been validated on a real test case involving several versions of interpolation filters adopted in latest video coding algorithms (HEVC) for motion estimation/compensation purposes. The MDC/CAPH integrated flow has been compared with commercial solutions aiming at the development of reconfigurable HEVC interpolation filters, resulting in more efficient and predictable (in terms of latency, one of the CERBERO KPIs) solutions [RUBATTU 2018].

Progress w.r.t. the initial plan:

With respect to the initial plan:

- Phase 1 (complete, debug and assess the MDC & CAPH integration for coarse grain adaptive HW) has been completed;
- Phase 2 (verify this approach with respect to relevant CERBERO KPIs) has been done, since the CERBERO KPI that is currently relevant for the test case (that is latency) has been verified with the integrated MDC and CAPH approach;

- Phase 3 (derive a proof of concept of the proposed approach in the context of the CERBERO use case scenarios) has been partially done, since HEVC coding is an interesting test case transversal for all the CERBERO use cases, especially the ones where environmental monitoring is necessary. Even if they can be possible topics of extensions for the CERBERO use cases, video coding aspects will not be investigated during the CERBERO project.

Maturity of the built self-adaptation management:

This integration is not properly part of the self-adaptation management, but serves as a design time support to complete the automation of the adaptable hardware accelerators design flow.

2.5. Integration Activity (5): MDC & Papify & ARTICo³

The planned Papify-MDC-ARTICo^3 integration aimed at providing monitoring support to HW accelerators. This integration activity has built a tightly connected set of technologies on hardware reconfiguration. A first step in this work was done by the preliminary Papify-ARTICo^3 integration [SURIANO 2018]. This integration provided a custom approach that used Papify for reading the hardware monitors available in the ARTICo^3 slots.

The following steps have been necessary to complete the integration:

- Integrating MDC and ARTICo^3 to provide support for a multi-grain reconfigurable approach, capable of managing both Coarse-Grain Reconfiguration (CGR) and Dynamic Partial Reconfiguration (DPR). *This integration required the ARTICo^3 toolchain to be extended to take into consideration both the memory banks and the configuration registers in the slots, and MDC code generation to be modified to provide processor-coprocessor systems 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 slave template).*
- Providing a generic monitoring approach for HW accelerators, suitable to provide proper feedbacks of both MDC accelerators and ARTICo^3 architecture, as well as feedbacks of their combination given by the multi-grain reconfiguration. *This integration required developing a generic PAPI-based approach for monitoring the HW accelerators, that could be valid for both MDC-generated accelerators (in which the number and kind of events may change according to the profiled application), and for ARTICo^3 accelerators (in which the number of slots may vary according to the user-requested level of parallelism and to the available board).*

Current state of integration:

The MDC-ARTICo^3 integration has been successfully achieved according to the plans. Figure 6 depicts the integrated multi-grain toolflow. Applications to be accelerated are defined as dataflow specifications and merged in an ARTICo3-compliant CGR kernel.

Then ARTICo^3 toolchain processes this kernel to embed it in the HW architecture [FANNI 2018, RODRIGUEZ 2018].

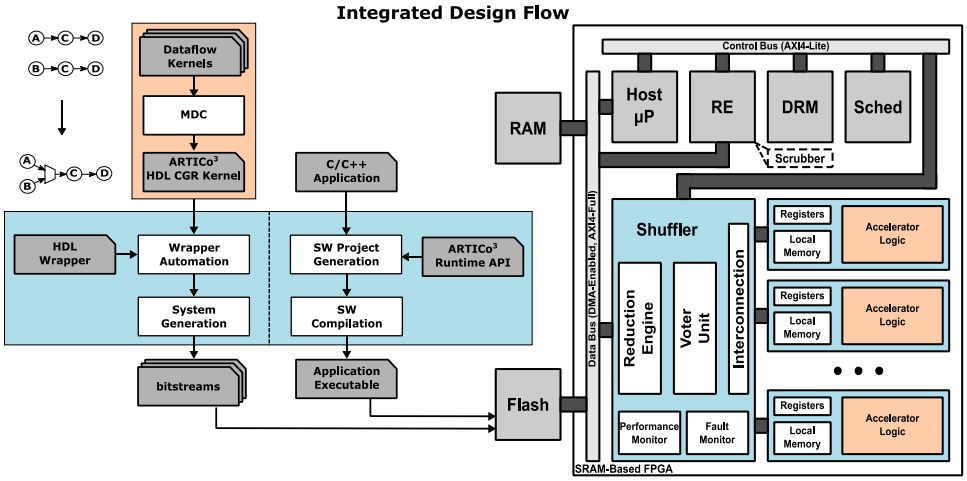

Figure 6 - Multi-grain design flow and adaptation.

The first step, offering support for monitoring DPR HW accelerators, has been conducted in the preliminary Papify-ARTICo^3 integration [SURIANO 2018]. In this integration a first PAPI-ARTICo^3 component, to be read using Papify monitors available in the ARTICo^3 slots, was developed. Figure 7 illustrates a schematic view of ARTICo^3 infrastructure and highlights the integrated PMCs for Performance and Fault monitoring [SURIANO 2018].

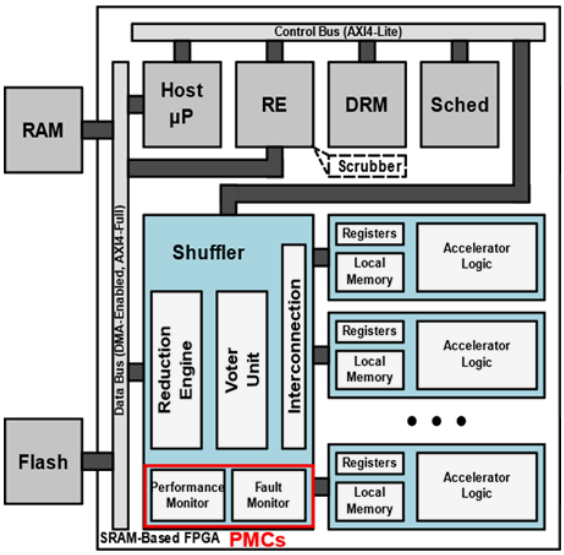

Figure 7 - ARTICo³ hardware architecture with Performance and Fault Monitors.

In order to make this monitoring support more generic, and provide a monitoring for CGR accelerators, the integration of Papify with MDC has then been conducted [MADRONAL 2019]. In this work, we have defined two levels of HW monitoring (see Figure 8):

1. the accelerator-level: homogeneous for every ARTICo^3-supported accelerator;
2. the low-level: specific to a given accelerator whose code is generated with MDC, e.g. by profiling the bottleneck PEs internally.

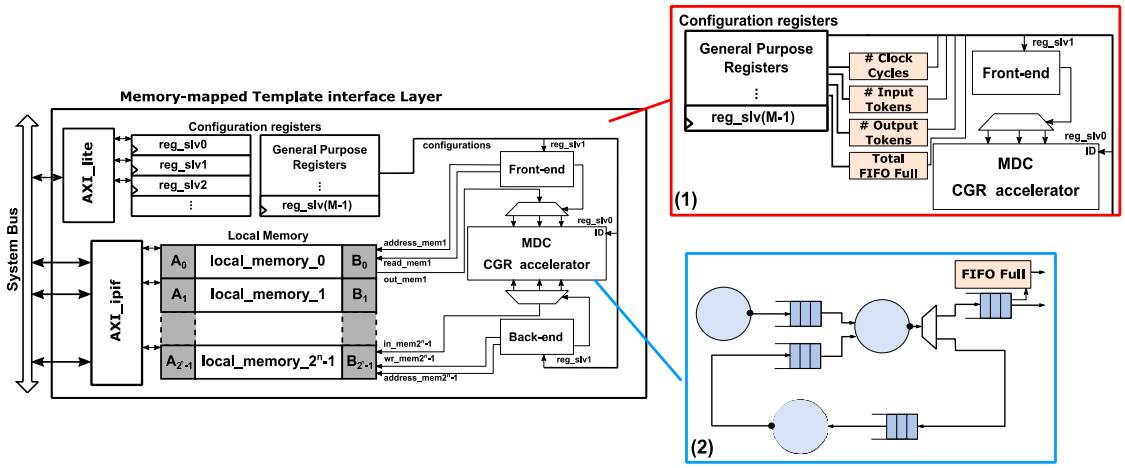

Figure 8 - HW Monitoring at two level of abstraction: (1) accelerator level; (2) low-level.

As illustrated in Figure 8 the HW monitors are accessed through the configuration registers of the accelerator. Since the base address of the accelerator may change in different accelerators, as well as the number and type of events to be monitored, we developed a configurable PAPI-MDC component that is automatically configured when the application is launched. The configuration xml file is similar to the following one:

```
<?xml version="1.0" encoding="UTF-8"?>
<mdcInfo>
<baseAddress>0xADDRESS</baseAddress>
<nbEvents>N</nbEvents>
<event>
<index>M</index>
<name>MDC_EVENT_NAME</name>
<desc>Event Description</desc>
</event>
</mdcInfo>
```

When configured through this API, the PAPI-MDC component is compliant with the existing standard SW components and events can be naturally accessed using Papify from a software interface to monitor the current behavior of the system.

Following the same approach to provide a consistent methodology for accessing instrumented HW, an XML-based PAPI-ARTICo³ configuration procedure was also designed in order to be compliant with PAPI and Papify from one side and with ARTICo^3 accelerators from the other side. The following listing shows the structure of this xml description:

```
<?xml version="1.0" encoding="UTF-8"?>
<artico3Info>
        <nbEventsArtico3>N</ nbEventsArtico3 >
```

```
        <eventArtico3>
                <index>M</index>
                <name>ARTICo3_EVENT_NAME</name>
                <desc>Event Description</desc>
        </eventArtico3>
        <nbKernels>K</nbKernels>
        <kernel>
                <kernelName>ARTICo3_KERNEL_NAME</kernelName>
                <nbEvents>N</nbEvents>
                <event>
                        <index>M</index>
                        <name>ARTICo3_KERNEL_EVENT_NAME</name>
                        <desc>Event Description</desc>
                </event>
        </kernel>
<nbEvents>N</nbEvents>
</artico3Info>
```

It is worth noting that there are two different types of events:

- the generic events associated to the ARTICo³ infrastructure (always present) such as ERRORS and CLOCK-CYCLES (as explained in [SURIANO 2018]). Other generic events are planned to be developed for extending the monitoring capabilities of the ARTICo³ hardware structure.
- Specific kernel events associated to hardware accelerators. These are not part of the ARTICo³ infrastructure but, instead, are part of the internal structure of the accelerators and can be associated, also, to MDC-specific events as well as to any other custom event.

Regarding the second type of events, the built hardware component is compliant with standard PAPI calls and events can be naturally accessed using Papify. Moreover, knowing that an ARTICo³ slot can host an MDC hardware accelerator, the internal structure of the XML file in charge of describing the events is fully compatible with the PAPI-MDC component:

```
<?xml version="1.0" encoding="UTF-8"?>
<artico3Info>
        <nbEventsArtico3>N</ nbEventsArtico3 >
        < eventArtico3>
                <index>M</index>
                <name>ARTICo3_EVENT_NAME</name>
                <desc>Event Description</desc>
        </ eventArtico3>
        <nbKernels>K</nbKernels>
```

```
        <kernel>
                <kernelName>ARTICo3_KERNEL_NAME</kernelName>
<baseAddress>0xADDRESS</baseAddress>
<nbEvents>N</nbEvents>
<event>
<index>M</index>
<name>MDC_EVENT_NAME</name>
<desc>Event Description</desc>
</event>
        </kernel>
<nbEvents>N</nbEvents>
</artico3Info>
```

The portion of the xml file containing the PAPI-MDC-powered accelerator description is highlighted in red. This file is known and generated at design time because it must reflect the hardware monitor infrastructure of the designed hardware. At runtime, the Papify/PAPI layer is in charge of selecting and initializing only the meaningful events. In the ARTICo^3 architecture each DPR slot may have its own registers (in any number, including none), as specified in the hardware project and reflected in the configuration file.

Progress w.r.t. the initial plan:

Phase 1 of the working plan has been completed: MDC and ARTICo^3 have been integrated and Papify is able to provide a HW/SW monitoring interface.

The automated instrumentation methodology for heterogenous HW/SW setups, envisioned in *phase 2*, has been developed for MDC and for ARTICo^3 by

- extending the use of PAPI and Papify to every generic hardware accelerators [SURIANO 2018],
- proposing a runtime monitoring approach for hardware accelerators based on PAPI and Papify [MADRONAL 2019].

Within the monitoring of MDC accelerators, we have defined the accelerator-level monitoring insertion, to keep trace of important dataflow metrics during execution, such as the execution time, the number of input tokens and the number of output tokens. However, low-level monitoring still requires manual steps to be used within the code generation flow.

The multi-grain adaptivity has been demonstrated to be effective [FANNI 2018]. We have also performed *phase 3*, providing PoCs of the MDC-ARTICo^3 integration [FANNI 2018, RODRIGUEZ 2018], a PoC on the MDC-Papify integration [MADRONAL 2019] and a PoC on the ARTICo^3-Papify integration [SURIANO 2018].

Maturity of the built self-adaptation management:

The different steps of integration of MDC, Papify and ARTICo^3 are rather mature, and have been tested on different standalone PoC applications. The feedback on the current status of the HW execution to the SW adaptation manager (SPIDER) is less mature and requires manual steps to feed the scheduler with hardware-related information. The presented work opens for generalizing the models, combining measurements and models to drive CERBERO adaptation.

3. CERBERO Framework-Level Integration to Support the CERBERO Self-Adaptation Strategy

This section goes beyond tooling integration and explains how the complete adaptation framework benefits from CERBERO advances on security, mathematical programming and from the CERBERO integration approach. The following table summarizes the state of the integrated tools implementing the CERBERO self-adaptation strategy described in Deliverable D4.1. It is an update of Deliverable D4.4 Table 1 and shows that the set of integrated tools now covers the span of originally targeted adaptation capabilities.

Table 2: CERBERO Integrated tools for self-adaptation management - UPDATED.

	CERBERO Internal	Modeling	Optimization	HW/SW Design	Runtime Support	In Loop Simulation	Open Source
DynAA	Yes	S	S			S	
SCANeR	No					S	
MECA	Yes	S			S		
SPIDER	Yes		S	S	S		S
Papify	Yes				S		S
ARTICo³	Yes		S	S	S		S
MDC	Yes		S	S	S		S
CAPH	No	S		S			

Supported: S.

The next sections successively detail framework-level integration activities, mathematical programming and security related advances and their impact on the self-adaptation manager.

3.1. Integration of CPS and CPSoS Multi-Layer Strategies

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 an advanced self-adaptivity of the target highly-heterogeneous CPS is supported.

CERBERO integration approach, as discussed in WP5 deliverables, has progressed in two directions: tool-to-tool direct integration (when same models and semantics were adopted) and the definition of a middleware to provide the interconnection of the tools together, in a layered fashion, to facilitate exchange of information and control data between these components or subsystems and insuring that the integrated system meets

performance and behavioural expectations. This latter is called the CERBERO Intermediate Format (CIF), and aims at achieving easy exchange of relevant information between all connected tools. Unlike tool-to-tool integration, CIF provides a unified platform for model and data transformation that allows for implementing automatic transformation capabilities. Within the CIF framework, meta-models (or schemas) of all input and output data are defined in a declarative way, supporting the definition of transformation processes as a mapping between corresponding schemas. Such unification results in achieving easy integration of multiple tools having multiple views and/or providing multiple functionalities.

Currently, the CIF integration approach has been demonstrated thought a Proof of Concept (PoC) connection of 3 CERBERO design-time tools: PREESM – AOW – DynAA. The purpose of the PoC has been to calculate optimized scheduling of a software application, provided as an SDF graph, on a hardware, provided as a hardware architecture description. The optimization can be performed with respect to several goals, such as minimal latency, maximum throughput, and minimum energy, and subject to different constraints, such as computation and memory capacity. At the moment there is no support of CIF in CERBERO run-time tools but the methodology built in CIF for rich tool inter-operability remains fully valid in the CERBERO runtime management context.

3.2. Novelties on Assessing CERBERO Adaptivity at Design Time through Mathematical Programming

Adaptivity of CPS should be considered at design time in order to find CPS architectures capable to adapt themselves to potential changes in environment and internal state. As CERBERO methodology for design space exploration considers the modeling of system and environment uncertainty at design time (please see D3.3. 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.

In CERBERO we can apply stochastic optimization during HW/SW co-design using two-stage and multi-stage models. 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 may provide both robust designs that require rare adaptations and Affinely Adjustable Robust Counterpart (AARC) that provides adaptation policies. AARC defines two kinds of decision variables: adjustable and non-adjustable. Adjustable variables 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. In CERBERO we extend theory of Robust Optimization to hybrid environments in continuous time.

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 and internal state.

3.3. Novelties on Enhancing CERBERO Adaptive Runtime Security and Reliability

Cyber-physical systems (CPS) should work in a reliable and secure way. The functionality needed to guarantee the secure operation of a CPS are strongly dependent on the specific application and could vary from the simple presence of an encryption algorithm to the request of implementing the system in way that is resistant against physical attacks. CPS are often deployed in hostile environments and have a lifespan usually much longer than usual consumer electronics. Furthermore, these systems should be capable of reacting to changes in the environment. As a result, it is fundamental that also security and reliability included in the system is adaptable. Despite being critical however, the capability of adapting security and reliability to a desired level is still generally not provided by current system design methods and, when available, is extremely limited. Previous approaches addressing this need are developed for specific applications and, often, they are not sufficiently general to be applied to a large variety of systems.

In CERBERO we addressed this problem by proposing architectures that allow adapting security functionalities to the changes of environment. The first adaptation explored in CERBERO has been the dynamic selection of different implementation of the same algorithm for trading the performance with the energy consumed by the accelerator.

In the last months we added the capability of adapting the level of security and the amount of redundancy added to provide reliability. As discussed in D4.1, we concentrated on AES used in GCM mode, with a MAC size of 128 bit, since this was is the standard required by our target application. The architecture we propose is capable of dynamically changing the size of the encryption key used (from 128 to 256) and the type of error detection and correction used (from simple parity to complex hamming codes). The accelerator can be enhanced to provide first order resistance against physical attacks. These choices are triggered by monitors. Currently, the self-adapting capability of the accelerator have been demonstrated using dedicated monitors embedded in the accelerator and using triggers external to the accelerator (for instance, the ones coming from the global monitors). In the next months, the technology will be demonstrated within the CERBERO use cases.

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

The following sections update the discussion, started in Deliverable D4.4, on the applicability of CERBERO self-adaptation strategies within the context of the project's use cases.

4.1. Planetary Exploration (PE)

Since the achievements discussed in the deliverable D4.4, many advances have taken place. The adaptation motion planning algorithm based on Nelder-Mead optimization has already been implemented and validated through both the physical arm and a Python simulator. The development and integration of the different CERBERO tools and technologies have been completed on M30 and set up different capabilities at design time and run time, i.e. automatic instrumentation for ARTICo³ accelerators, a novel MDC + ARTICo³ integrated design toolchain and multi-grain reconfiguration, etc.

A number of layers were considered for the PE use case runtime adaptation support in this deliverable. According to the advances previously mentioned, specific updates in each one of these layers are outlined below:

- **Triggers:** information for adaptation comes from both external sensors and internal monitors. On one hand, time-of-flight laser sensors provide measurements of distance to physical objects, enabling the exteroceptive adaptation to the environment. On the other hand, the performance monitors provide the different KPIs that will be used for evaluating the internal state of the computing platform and feed proprioceptive adaptation (i.e. adaptation triggered from self-awareness information). Current sensors will not be used in the final demonstrator; a model of the power consumption of the actuators will be inserted instead in order to estimate this parameter
- **Adaptation fabrics**: The crux for adaptation in this use-case will be the integration between the different CERBERO tools and technologies at computation level. SPIDER will be used to dynamically change the processing scheduling at runtime. ARTICo³ will implement dynamic partial reconfiguration to enable transparent scalability and automatic degree of ruggedization of the system. MDC virtual reconfiguration will provide fast switching between different implementations of the algorithm. Just-In-Time hardware composition introduces a higher degree of granularity for reconfiguration, although the usage of this technology in the final demonstrator is still being assessed and will depend on its degree of maturity.
- **Adaptation monitors**: the automatic instrumentation of the code provided by Papify facilitates the implementation of performance monitors inside the processing system. The integration between Papify, ARTICo³ and MDC makes it possible to measure the number of errors and determine the value of internal parameters of the accelerators such as execution times, power consumption, etc.

- **Embedded models**: two main models are being used. First, a physical model that implements the forward kinematics of the robotic arm; and second, a power consumption model that predicts the energy requirements of the calculated trajectory.
- **Adaptation manager**: the manager will put together all the information from the adaptation monitors and respond to changes in the environment or inside the computing system itself, according to the different triggers already described. Considering the movement and energy embedded models, it will command the architectural and functional reconfiguration fabrics to achieve adaption to the new situation.

4.2. Ocean Monitoring (OM)

The Ocean Monitoring (OM) use-case incorporates CERBERO technologies and methodologies to allow different notions of adaptation. Specifically, CERBERO adaptation loop is used to separate and define the interaction between the adaptation components. The different types of information used for adaptation purposes are combined through information fusion models.

The OM use-case recent progress relates to the following notions of environment triggered sensor adaptation:

- Object detection and tracking based on fusion of colour and background subtraction approaches.
- Image rectification method based on keypoints matching.
- Real-time optimization for the motion detection and tracking.

In all cases the adaptation fabric provides the processing systems used for visual information sensed from the environment. In the case of object detection and tracking for stationary camera, adaptation monitors incorporate background subtraction which is applied to the current frame and the history of reference frames in order to look for motion. When motion is detected, the adaptation manager identifies the nature of motion and decides whether to adapt. Next, the adaptation engine identifies the location of the moving object and extracts the colour pattern from the motion area. It continuously adapts processing fabrics to the colour range of the colour-based object detection approach so that the object can be tracked based on the fusion of two different frameworks. The aforementioned hybrid approach was developed and implemented in the OM use-case demonstrator.

The goal of adaptive image rectification method based on keypoints matching approach is to adaptively find more and better keypoints matches to continuously improve the rectification results over time. This would allow for the self-correction process in the case when the viewpoints of the camera sensors change. Although this has been developed for the OM use case, it could also be applicable to the PE and ST use cases too. This could be due to the physical stress or damage the sensors may sustain as a result of a rough landing on Mars surface, or an electric car or underwater vehicle collision with another object (for OM). Please refer to D4.1 for the details of the proposed framework.

The OM use-case has a need to detect and track objects not only from stationary but also moving cameras. The camera movement direction can be detected from a clustered

motion vector field, for example. There are a number of alternative techniques for detecting motion, with advantages and disadvantages in different contexts – evaluation of the best techniques for the video data typical of ocean monitoring scenarios is currently in progress. However, all techniques, whether they are based on block matching, blob tracking, or optical flow, depend on constrained optimization techniques. The OM use-case therefore explores the possibility of using relevant CERBERO tools capable of performing the aforementioned optimization in real-time.

4.3. Smart Travelling (ST)

In the Smart Travelling (ST) 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 ST use case considers CPSoS interactions and the driver actions as triggers to launch the reconfiguration of the current execution status, based on SW fabrics. 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 his/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.

5. Conclusions

This deliverable has reported the final progress of Task 4.4 "System Self-Adaptation". In terms of the CERBERO operational objectives detailed in the Table 1 "CERBERO Operational Objectives" of the CERBERO project sealed proposal, Task 4.4 has contributed primarily to the following objectives:

- CH1.2 - Provide a comprehensive framework, customizable upon the UC needs, extending and making interoperable a large set of tools.
    - On this challenge, T4.4 has merged highly customizable tools into an implementation of CERBERO adaptation strategies, compatible with a large set of platforms and with a large set of stream processing applications representative of the ones present in use cases.
- CH1.1 - Provide reusable Libraries of Key Performance Indicators (KPIs), Cross-Layer Models and Adaptivity support.
    - On this challenge, T4.4 has built the adaptivity support and KPI retrieval facilities necessary to the adaptation loop, as well as models to extract KPIs from indirect hardware measurements.
- CH2.2 - Reduce by 50% the design efforts required to build a CPS of a given performance.
    - On this challenge, T4.4 has provided adaptation technologies that automate the generation of complex hardware behaviors, strongly saving design efforts when building an adaptable system.
- CH2.3 - Reduce by 50% cost of maintenance.
    - On this challenge, T4.4 has implemented a technology capable of strongly modifying the processing of a system to adapt to system modifications, to environment modifications, or to user commands autonomously.
- CH1.3 - Reduce by 30% the energy consumed by a fully CERBERO compliant CPS or CPSoS, while maintaining its performance.
    - On this challenge, T4.4 has aggregated a set of reconfiguration technologies offering hardware and software customization to the application, and capable of large energy savings through specialization.

As an updated version of Deliverable D4.4, this document has explained how the CERBERO M15-M30 activities have built the CERBERO self-adaptation advanced technology that combines monitoring and reconfiguration features, as well as hardware-software coordinated adaptations and application-level adaptation. This document shows that the plan provided for in Deliverable D4.4 has been followed and that the tool integration efforts have conducted to the implementation of the CERBERO self-adaptation management strategies presented in Deliverable D4.1.

6. References

[FANNI 2018] Fanni, T., Rodríguez, A., Sau, C., Suriano, L., Palumbo, F., Raffo, L., & de la Torre, E. (2018, December). Multi-Grain Reconfiguration for Advanced Adaptivity in Cyber-Physical Systems. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)* (pp. 1-8). IEEE.

[MADRONAL 2019] Madroñal, D., & Fanni, T. (2019, April). Run-time performance monitoring of hardware accelerators: POSTER. In *Proceedings of the 16th ACM International Conference on Computing Frontiers* (pp. 289-291). ACM.

[RODRIGUEZ 2018] Rodriguez, A., & Fanni, T. (2018, December). Multi-Grain Adaptivity in Cyber-Physical Systems. In *2018 30th International Conference on Microelectronics (ICM)* (pp. 44-47). IEEE.

[RUBATTU 2018] Rubattu, C., Palumbo, F., Sau, C., Salvador, R., Sérot, J., Desnos, K., ... & Pelcat, M. (2018). Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators. *IEEE Embedded Systems Letters*.

[SURIANO 2018] Suriano, L., Madroñal, D., Rodríguez, A., Juárez, E., Sanz, C., & de la Torre, E. (2018, July). A Unified Hardware/Software Monitoring Method for Reconfigurable Computing Architectures Using PAPI. In *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)* (pp. 1-8). IEEE.