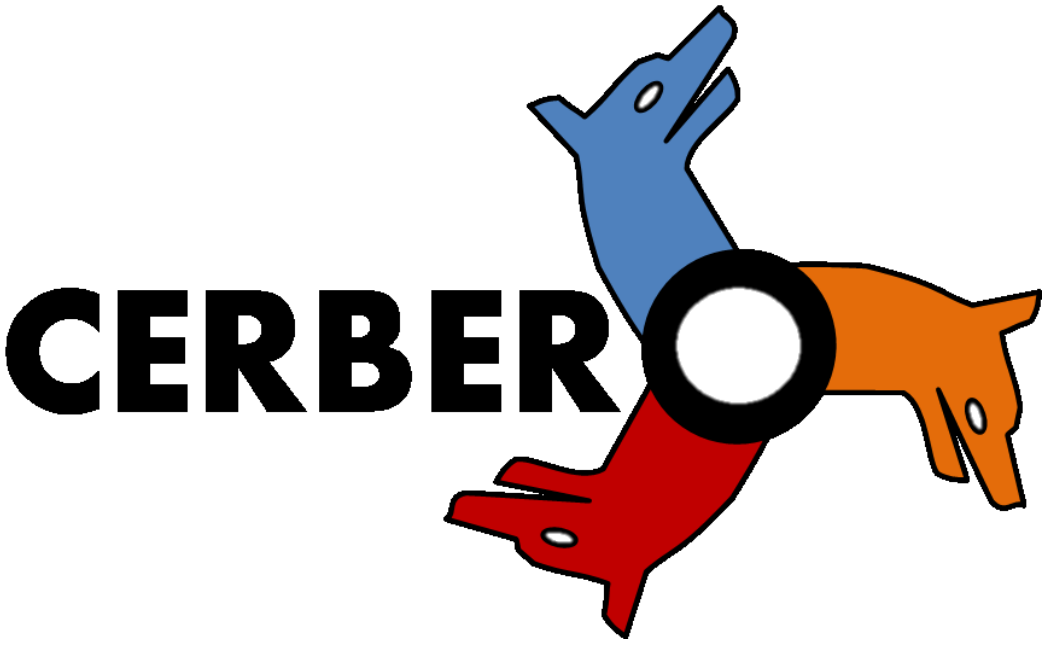
Information and Communication Technologies (ICT) Programme

Project Nº: H2020-ICT-2016-1-732105



D3.1: CERBERO Modelling of KPI

Lead Beneficiary: USI
Workpackage: WP3
Date:
Distribution - Confidentiality: [Public]

Abstract:

This document presents CERBERO KPI-based design methodology for adaptive CPS and describes how the KPIs are supported by the tools included in the CERBERO toolchain. To make the document self-contained, it includes a summary of the results reported in D3.4.

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**
Francesco Regazzoni	USI
Karol Desnos	INSA
Francesca Palumbo, Luca Pulina	UNISS
Carlo Sau, Tiziana Fanni	UNICA
Leszek Kaliciak, Stuart Watt, Hans Myrhaug, Ayse Goker	AS
Alfonso Rodrigues, Daniel Madronal, Andres Otero	UPM
Michael Masin, Evgeny Shindin	IBM
Julio Oliveira Filho, Coen van Leeuwen, Adriaanse Joost	TNO
Edo Loenen	S[&]T
Pablo Sánchez de Rojas Méndez	TASE

Document Revision History

Date	**Ver.**	**Contributor (Beneficiary)**	**Summary of main changes**
23/01/2019	0.1	Francesco Regazzoni (USI)	ToC and draft of Use Case
04/04/2019	0.1	Karol Desnos (INSA)	Contribution related to PREESM and SPIDER
05/04/2019	0.1	Francesca Palumbo (UNISS), Carlo Sau (UNICA), Tiziana Fanni (UNICA)	Contribution related to MDC tool
10/04/2019	0.2	Francesco Regazzoni (USI)	Complete sections till section 4
12/04/2019	0.2	Leszek Kaliciak, Stuart Watt, Hans Myrhaug, Ayse Goker (AS)	Added definition of disparity and geometric distortion (included at the end of section 5.3)
12/04/2019	0.2	Alfonso Rodrigues (UPM)	Contribution on Artico^3 and JIT-HW
15/04/2019	0.3	Francesco Regazzoni (USI)	Received contribution from Adriaanse Joost (TNO, modification of KPIs), and Pablo Sánchez de Rojas Méndez (TASE, confirmed not change in KPIs)

15/04/2019	0.3	Francesco Regazzoni (USI)	Assembled the documents with contributions from TNO, INSA, UNICA, UNISS, IBM, UPM, TASE, AS
16/04/2019	0.3	Michael Masin (IBM)	Contribution on AOW
17/04/2019	1	Francesco Regazzoni (USI)	First complete version of the document
24/04/2019	1.3	Julio Oliveira Filho, Coen van Leeuwen (TNO)	Contribution on DynAA
24/04/2019	1.3	Daniel Madronal (UPM)	Contribution on PAPIFY
24/04/2019	1.3	Andres Otero (UPM)	Update on JIT-HW
26/04/2019	1.4	Edo Loenen (S[&]T)	Contribution on MECA
28/04/2019	1.5	Francesco Regazzoni	Wrote section 3, final assembly
29/04/2019	1.6	Luca Pulina	Correction on SAGE description
30/04/2019	1.7	Michael Masin, Evgeny Shindin, Francesca Palumbo	Changes on section 3 and section 4, general document feedback
30/04/2019	1.8	Francesco Regazzoni	Assembled feedback described in point 1.7, added table in section 4
30/04/2019	1.9	Francesco Regazzoni	Minor changes, closing the document.
30/04/2019	2.0	Francesco Regazzoni, Michael Masin	Preparation for submission

Table of contents

1. Executive Summary

This document completes and updates the definition of Key Performance Indicators (KPIs) and their definition initiated in D3.4, presents a complete design methodology for CPSs based on KPIs, and described how KPIs are used in the tools composing the CERBERO toolchain.

The objective of this deliverable is to complete and finalize the definition of the KPI based design methodology. KPIs are following the CPS during its whole lifespan. This implies that KPIs should be considered at design time, but also that the infrastructure supporting adaptation should be conceived starting from KPIs. To do so, design steps should be revised and completed. We add specific steps to the design flow for selection of KPI, synthesis of requirements, and synthesis of monitors, and we completed standard design steps to carry out also task related to the use of KPIs.

This deliverable is completed by the detailed description of the way in which KPIs are supported by the tools included in the CERBERO tool chain, and by an update of the KPI needed in the three use cases of this project.

1.1. Structure of Document

The document is organized as follows. Section 2 summarizes the KPI concepts and features extensively discussed in D3.4 (this section is in the document for making is self-contained). Section 3 presents the whole KPI based design methodology discussing in detail each step. Section 4 discuss how the KPIs are mapped to the tools composing the toolchain of CERBERO. Section 5 provides an update on the KPIs of the CERBERO use cases, summarizing the ones previously discussed in D3.4 and defining new ones when needed.

1.2. Related Documents

- D2.1 - CERBERO Scenario Description: KPIs are defined on the specific use case described in the updated scenario description (final version)
- D2.2 - CERBERO Technical Requirements: D3.1 contributes to satisfy D2.2 requirements (KPIs will be used at the design time and during the whole lifetime of the CPSs)
- D3.5 – Models of Computation: KPIs will be used to represent the system properties which will be verified with different Models of Computation
- D3.6 - Cross-layer Modelling Methodology for CPS: KPIs have to be robust and consistent across different layers and modeling methodologies. This is a fundamental contribution of the CERBERO project.
- D5.6 - CERBERO Framework Components: ultimately, KPIs will be integrated and used by the tools composing the CERBERO framework.
- D3.4 – CERBERO Modeling of KPI: this document provides an update of D3.4 and completes it presenting the whole KPI based design methodology.

1.3. Related CERBERO Requirements

- Deliverable D2.2 of the CERBERO project [CERBERO_D2.2] defines the CERBERO Technical Requirements (CTRs) of the project (identified with an

ID ranging from 0001 to 0020). Research topics related to KPIs activities are reported in Table 1.

Table 1: Links to CERBERO technical requirements

CTR id	CTR Description	Link with the D3.1 document on Modelling KPIs
0001	CERBERO framework SHOULD increase the level of abstraction at least by one for HW/SW co-design and for System Level Design.	The proposed methodology is applicable at any level of abstraction and requires to define Key Performance Indicators that are robust against change of layers of abstraction.
0002	CERBERO framework SHOULD provide interoperability between cross-layer tools and semantics at the same level of abstraction.	The methodology proposed here uses Key Performance Indicators that have to be defined using a formalism which guarantee interoperability between tools and layers.
0004	CERBERO framework SHOULD provide software and system in-the-loop simulation capabilities for HW/SW co-design and System Level Design.	The design methodology proposed here uses model-based design space exploration driven by the Key Performance Indicators. Adaptivity will be also driven by KPIs.
0005	CERBERO framework SHOULD provide multi-viewpoint multi-objective correct-by-construction high-level architecture	Key Performance Indicators used in the proposed methodology allow to define the metrics of multiple viewpoints and the objectives of the multi-objective architecture.
0007	CERBERO framework SHALL define methodology and SHOULD provide library of reusable functional and non-functional KPIs.	As defined in D3.4 and completed here, Key Performance Indicators are organized in family of reusable components. This deliverable presents a complete design methodology based on the concept of KPIs.
0009	CERBERO SHALL develop integration methodology and framework.	The library of reusable Key Performance Indicators is a fundamental component of the CERBERO framework.
0020	CERBERO framework SHALL provide methodology and tools for development of adaptive applications.	Key Performance Indicators are the way for evaluating the state of the system and to trigger the adaptation. This deliverable presents how adaptivity is explicitly integrated into a complete KPI based design flow.

2. Summary of Key Performance Indicators Definition and Properties

In this section, for completeness, we summarize the main concepts introduced and extensively presented in deliverable 3.4. In particular, we report the definition of Key Performance Indicators (KPIs) as they have been presented in Section 3 of deliverable D3.4 and, from the same section, we recall the concept of family of KPIs, and we remind the properties that KPIs should have in the CERBERO project and in the design of CPS in general.

In the context of CERBERO, we have defined a KPI as a quantifiable parameter associated with a metric. A single KPI evaluates one critical parameters of a CPS and evaluates the discrepancies from its long term goal. The whole CPS is evaluated using a number of KPIs, grouped in what we called set of KPIs. In our context, KPIs should have the following properties:

- KPIs have to be quantifiable. For this reason, KPIs will always be defined together with a metric which allows to measure them or rank them.
- The set of KPIs that are used to evaluate its performance are specific of the target CPS. To measure effectively the performance of the CPS, in fact, KPIs have to be very tailored to it.
- KPI definitions could be reused in designing different CPS or provide basis for system specific KPIs. If the set of KPIs have to be specific, the single KPI can the same over several CPS or can be generic (and specialized on the target CPS afterwards).
- KPIs would drive the evaluation of the system during the whole live cycle of the CPS. In CERBERO, KPIs will be used to drive the conception of the system, but also to steer the continuous adaptation.

A key concept of CERBERO is the one of "family of KPIs". This concept was introduced to allow reusability of KPIs (which, by nature, are instead intrinsically tailored on the specific CPS). A family of KPIs has been defined as a set including all the KPIs accommodated by the same properties and to which the same algebraic operations are valid. KPIs and the way in which they are evaluated are described with an algebraic formulation. The evaluation of a KPI falls always into some definable *algebra,* and often exhibiting a well know structure. CERBERO proposes ways to formally define, model, and classify KPIs according to the mathematical structure they exhibit (or need) in their calculations. We call each class of KPIs obtained in this way family of KPIs. Since the properties and the algebraic operations are defined at the family level, once that they are defined or demonstrated for one KPI, they are immediately valid for all the other ones belonging to the same family. The properties and the algebraic operations associated to the KPIs are thus the main reusable components. A critical point for leveraging on KPIs is to measure them, since for some

KPIs define a metric is not straightforward. The KPIs, for which a metric is too complex to be defined, will be modeled as a list, containing all the possible options, and a partial order between the elements in the list, given by the designer to each of the options.

One possible example of KPI is the one that we called additive, namely that the KPIs are calculated by an addition operation (examples of this is for instance area). Area, can be defined in a template fashion, as the sum of all the elements composing the system (without specifying, at the template level, if elements are gates, or look-up-tables). For all the KPIs belonging to area, we can assert certain properties (still using the example of area, we can state that the area of the whole system is certainly greater or equal to the area of the processor included in the system). In addition to additive, other families of KPIs we identified are multiplicative, maximum, minimum, average, hierarchical, weight sum, weighted average, complex

3. The Complete KPI Based Design Methodology

In deliverable D3.4 we have presented the generic approach for classifying, modeling and measuring KPIs. In this section we use these concept to describe a complete KPI based design methodology. Although the concept of KPI is widely used since several years in many fields and the notion of KPI based methodology is often used also in the context of design of CPSs, a clear identification of the steps that composes this methodology is still missing.

The starting point of our design methodology is the model based design methodology [7] proposed to design cyber-physical systems. We maintain the main concepts and steps of that methodology, extending them to include the notion of KPIs and adding the needed steps to carry out a design flow based on them.

1. **State the problem:**

   The first step of the whole design process it to have a clear problem statement. This step is identical to the one propose by the Jensen et al. [7]. The problem to be solved has to be firstly described in simple language, without use of mathematical formalism or specific terminology. In fact, as pointed out, the design of cyber-physical system problem is a multidisciplinary one and "this step is necessary to effectively communicate design requirements" [7].

2. **Select the KPIs**

   Together with the statement of the problem, it is necessary to select the KPIs which the system should try to optimize. In fact, KPIs are effective only if they are specific to the system they are applied to. At this stage of the design flow, KPIs should still be defined in a simple language. As what is happening in Step 1, the outcome of Step 2 should be a simple and accessible reference for all the teams involved in the design of the CPS. We separated the step of KPI selection from the one of the statement of the problem because we believe that the selection of KPIs, being related with the evaluation of the system performance, should be emphasized.

3. **Model the physical processes and the KPIs**

   As in the work of Jensen et al [7], we need to model the physical processes in an iterative fashion. However, together with the physical systems, we need to model also the KPIs. A model is a simplified representations of a system or of a phenomena. Models of KPIs are the same: a simplified representation of the real KPIs and of a metric to measure it. We would like to stress on this aspect of KPIs: we believe that a KPIs should necessary be defined together with a metric to evaluate it. So has to be the model. Models of KPIs should also be defined in an iterative step, till the model is sufficiently accurate to provide an representation of the reality that is sufficient to measure the needed quantity. In fact, KPIs may lead model's fidelity – given the model of a KPI, all its inputs should be contained in the CPS model.

4. **Characterize the problem and the KPIs**

For the problem characterization, this step is similar to the one proposed by Jensen et al [7]. We add to this step the characterization of KPIs. We introduced in deliverable D3.4 the concept of family of KPI that is a set including all the KPIs accommodated by the same properties and to which the same algebraic operations are valid. Here we need to characterize the KPIs to assign them to the correct family of KPIs. The KPIs characterization has to be completed to take full advantage of the properties of the family of KPIs during the design phase. The characterization of the KPIs will also dictate the model of computation. Similarly to the model of physical processes, when it is clearly defined to what is needed to be measure to evaluate the CPS, it is immediately possible to discard the models of computation that do not allow to evaluate the KPIs. For this reason, we removed the step five from the work of Jensen et al. [7], since the model of computation is dictated here in Step 4 by the characterization of KPIs.

5. **Derive the control algorithm**

   This is the same step of Jensen et al. [7], which determines the conditions under which a system can be controlled and in the selection of an appropriated algorithm for controlling it.

6. **Define the toolchain**

   One of the pillars of KPI based design is that KPIs are following the CPS for its whole lifespan. This however has to be supported. In fact, it is useless to define and model KPIs that can not be used during the design phase since tools do not support them. Also, it is useless to define KPIs that can be monitored at runtime to drive adaptivity if there are no tools able to generate the needed structure to expose the parameters needed to evaluate KPIs at runtime. For this reason, we the selection of the tool chain is as fundamental as the selection of the mode of computation. Select the tool chain that supports the KPIs that have been selected in Step 2. We state that a tool supports a specific KPI is it has the capability of driving its decision based on that KPI or if it has the capability to generate the infrastructure needed to expose the KPI at runtime. Problems can occur when well established tools do not have the possibility to compute the KPIs as specified in Step 2. Designers should pay attention to this designing, when possible, dedicated wrappers to adapt to the situation.

7. **Synthesize the low level requirement**

   Low level requirements should be derived directly from higher level KPIs. Where automatic synthesizers of implementation requirements from KPIs are not available, hand written low level requirements should be prepared. However it is necessary that the requirements carefully adhere to the specified KPIs. Synthesis of the requirements include the specification of constraints for the design tools to drive their decisions. In this step, what it is needed is to express the KPIs by means of a design constrain tool. For example, if a KPIs is critical path that has to be minimized and the tool is a synthesis tool, then a way to express this KPI in terms of constrains of the target tool is to set the clock for the synthesis to a minimal level. This step could require some iterations, since the imposed requirements might be too tight.

8. System Architecture

At this stage, the partitioning between hardware and software portions of the system have to be carried out. Partitioning of monitors that control both parts have also to be carried out here. Hardware/software co-optimization is also carried out in this step. Hardware/software partitioning is pretty dynamic and iterative process. Because of this dynamicity, the order of Step 8, Step 9, Step 10 and Step 11 is not strict and these steps are carried out in an iterative and not necessary sequential fashion.

9. **Define the hardware**

This is similar to the step of Jensen et al. [7], which consists in the selection of the hardware capable of fulfill the requirements of the CPS. We extend this step including explicitly, when needed, in the requirement that the selected hardware should support adaptivity (for instance, the hardware should support voltage scaling if power consumption is one of the KPIs).

10. Synthesize the monitors

As requirements, monitors for evaluating KPIs at runtime and support adaptivity should be derived directly from KPIs. Again as for low level requirements, where automatic synthesizers of monitors from KPIs are not available, hand written monitors strictly adhering to the KPIs should be prepared. Depending on the selected hardware, it could be possible that the needed monitors are not directly available, or that the KPI to be monitored can be inferred by a combination of existing monitors. When not possible automatically, should be designer's care to develop the needed wrapper or routine to build the desired monitor starting from the existing ones.

11. Synthesize the software

This is similar to the step of Jensen et al. [7], which consists in realizing the software to be executed by the system. We extend this step including explicitly, the need for a adaptivity algorithm which has to be included in the CPS. Software is needed for hardware software co-optimization, so it is anticipated here compared to the work Jensen et al. [7].

12. Assemble and Simulate

This is similar to the "Simulate" step of Jensen et al. [7], which consists of solving the problem to be solved, identified in Step 1 using simulation tools running at the appropriated model of computation. This step needs to be extended to simulate also the adaptivity of the CPS. Here, using simulation, we will carry out early validation, early verification and early testing of each component and of the whole systems.

13. Construct

This is similar to the "Construct" step of Jensen et al. [7], which builds the system according to the specifications.

14. Verify, validate and test

This is similar to the step "Verify, validate and test" of Jensen et al. [7], which consists in verifying validating and testing the system of the global system completely assembled. We extend this step including explicitly the need for testing the adaptivity and the correctness of the evaluation of the KPI carried out at run time.

4. Mapping to the CERBERO tool chain

This section provides an update on which KPI are mapped to the CERBERO tool chain and describe how KPIs are supported by each component of the tool chain. One of the pillars of the CERBERO project is the KPI based design. KPIs are following the CPS during the whole lifespan of the CPS, starting from the requirements, and are used to drive adaptivity. Based on this concept, we can see that tool composing a toolchain that supports KPIs based design can do so in two ways:

- being able to directly use, at design time or at run time, one or more KPI during the generation of the output of the tool. This implies the capability of calculate the given KPI. In the rest of this document, we will call this support of type one.
- producing the infrastructure needed for exposing one or more KPI at runtime or at design time (if needed in simulation) or producing the infrastructure for modifying, at run time, the behavior of a system in a way that affect the value of one or more KPIs. We will call this support of type two.

Taking as example power consumption as KPI defined as addition of the power consumption of each component of a CPS and that as to be minimized. For a tool, the first way of supporting this KPI (directly use of KPI) would be accepting, as input, a parameter that allows to specify constraints on power consumption or relevant objective functions, such as "generate a solution that minimize the power consumption". This can be done by means of constraints (for instance, an ideal parameter definition "set power consumption = minimal"). As mentioned, this way of supporting KPIs implies that the tool has the capability of computing the given KPI. This assumption can be pretty strong, thus users of the tool should be particularly careful with compatibility aspects. This mean that the way in which the tool computes the KPI must be consistent with the one used in the KPI definition. The second way to support a KPI would be to be able to produce the infrastructure to monitor the power consumption of a CPS at run time and expose it to the system, so that the system can dynamically adapt its behaviour to reduce the power consumption. In the rest of this section we explore the tools composing the CERBERO toolchain. For each tool, we quickly recall the tool functionality and we detail on how the considered KPIs are supported.

MECA:

In recent years we have reached significant automation level in environments such as industrial production or transport. Applications for these complex environments require enhanced technologies for allowing human supervision and decision support.

The MECA (Monitoring Execution Control Advice) technology is a framework designed to properly monitor and trigger adaptation behavior, keeping the user informed and providing advice and explanation about how to better interact with a system to improve the efficiency, safety and user wellness. MECA provides supports for KPIs of type one. In general, MECA support the optimization of all KPIs. However, it does this in a way that is somewhat different of most of other tools. MECA allows the system to be extended beyond hardware and software, by involving human operators in decision making. As such, it allows humans to provide input to the system, which can serve as constraints to KPIs.

There is one specific KPI that applies to MECA:

- **User satisfaction**: In the smart traveling use case, the role of MECA is to ask input from the user regarding the trip they wish to make. This information is

collected essentially as constraints, then it is sent to the rest of the system (in particular DynAA). However, rather than automatically optimizing the system to the constraints given to the user (in terms of e.g. travel time and energy), the system supplies MECA with possible optimization solutions. MECA then ranks these solutions based on various parameters (see also `Figure 1`), such as:

- Distance
- Travel time
- Number of stops
- Cost

These ranked features are presented to the user, which makes the ultimate decision. By giving the user on the one hand the freedom to choose the optimization, but on the other also decision support due to ranking, user satisfaction is optimized.

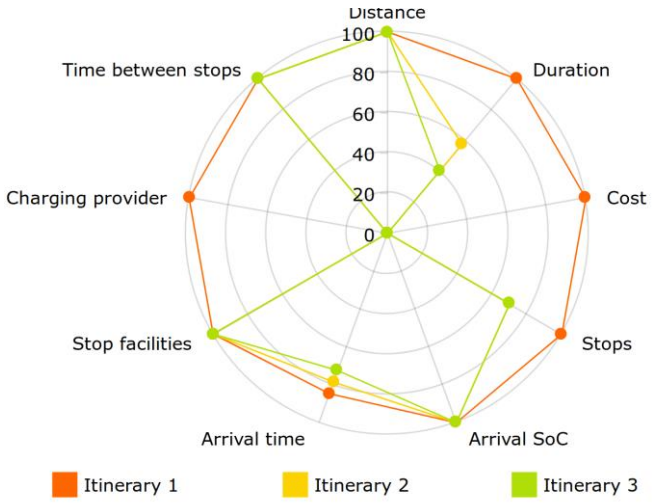


Figure 1: visualization of ranking performed by MECA.

DynAA:

DynAA is a generic event-based simulation environment for evaluating dynamic models. Its key strengths are its capabilities to deal with modifications that occur during the simulation without having to describe the exact behavior of the system when that happens. For instance when a component fails or disappears it will simply stop responding to events, and as soon as it reappears it will start to work as it did previously. Especially in a large-scale simulation with adaptive components this is beneficial when adaptation is frequent, or the system must be able to deal with failing components.

As a general remark, DynAA supports KPIs in a very broad and flexible sense. Users can define their own KPIs by indicating Java functions that are invoked during simulations. These Java functions may be triggered upon user defined events, and have access to measurement points in the model, as well as to variable status at the given simulation time. In this way, DynAA users are able to fine tune their KPIs according to their need, and for every system simulation. Here, we discuss the definition in DynAA of the KPIs specific to the Smart Travelling use case, which are the following:

- **Energy**: The electric vehicles have a key limiting resource, which is the battery. Different itineraries put the battery under differing constraints, and may or may not be feasible providing the vehicle battery. In the DynAA simulation this is

computed by cumulatively adding all segment energy consumptions provided by the EV model. The input for the model is the predicted vehicle speed, the segment length and the road type.

- **Travel time**: The travel time will be a considerable KPI for the end user as it will have a direct impact on his agenda. In the DynAA simulation environment the itineraries are evaluated by taking the predicted segment speed and length, and combined with the duration of the required stops (for charging the EV) the total travel time is computed.
- **Number of stops:** During the smart travelling simulation, a huge amount of trajectories are followed by cars (agents). Cars are allowed to stop at certain stop points during the trajectories (for example, to recharge the car or give the driver resting time). Every time a car stops at one of these points a specific event is generated. The total number of stop events is computed per agent, and per trajectory, and constitutes this KPI. The number of stops is a KPI that can be used later on as constraint (selecting trajectories with a maximum number of stops) or as optimization criteria (minimize the number of necessary stops in the way).
- **Cost:** During the smart travelling simulation, the cars (agents) stop for recharging at different charging poles. Each charging pole has its own price rate, incurring in different total monetary cost at the end of a trajectory and depending on where the car (agent) decided to stop. The **cost** KPI in this case refers to the added (cumulative) costs accumulated during the whole trajectory. Measured per agent, per trajectory.
- **Response time to trigger:** The response time to trigger is the time spent between the arrival of a reconfiguration trigger and the end of the reconfiguration phase. As DynAA is a discrete event based simulation/analysis tool, the natural way to calculate the response time to trigger is to associate DynAA events to both simulation moments: an event is associated to the occurrence of the trigger, and an event is associated to the moment when the system finishes all reconfiguration phase actions. The **response time to trigger** is measured by subtracting the time of the first event from the time of the second event. For a given simulation, the user has to indicate the event types he associates with start and end of a reconfiguration phase.
- **Battery lifetime**: the battery lifetime is the average time measured between full battery discharging cycles, under typical and/or constant load. Each discharging cycle begins when the charging of the battery stops (charging circuit is disconnected). That is indicated in DynAA by a specific event. Each discharging cycle ends when the battery level reaches a minimum level (minimum level is indicated as part of the user model); this moment is also associated with a specific event. For each fully completed discharging cycle, DynAA registers the time difference between the indicated events. The **battery lifetime** KPI is an average of the registered discharging cycle times for a given battery. Such KPI is included in the battery model.

DynAA provides support of KPIs of type one and of type two. In fact DynAA is able to generate the infrastructure to compute KPIs that can be used externally and it also contains an optimization engine that can optimize specific KPIs.

AOW:

Architecture Optimization Workbench (AOW) is a tool for Design Space Exploration (DSE) in different levels of abstraction. In CERBERO, AOW is mostly used for HW/SW co-design, with the goal of finding, over a specific architecture, the best mapping and scheduling of application activities. The application is given to the tool as a dataflow model. The available software and hardware components are the model of the given architecture. The tool also receives a list of possible mapping scenarios. AOW provides KPI support of type one. The following KPIs are currently directly used or can be used to drive the HW/SW co-design optimization

- **Monetary cost of the used components (output)** – sum of cost of all components needed for application's execution. Cost of all components is defined in the model of architecture.
- **Hardware and network communication utilization (output)** – weighted sum of all activities or communication packages using hardware or network channel, respectively. Activities and communication packages are defined by the design space exploration model based on model of application, model of architecture and mapping scenarios.
- **Energy consumption (output)** – weighted sum of energies consumed by hardware, communication channels and memory / storage. Defined by design space exploration and model of architecture.
- **Throughput (input)** – in streaming applications it is defined by number of application executions per cycle time. Defined by model of application.
- **Latency (output)** – in streaming application, it is defined as time from receiving the data until output based on the data. In AOW latency is calculated based on a system of linear inequalities that eventually provide a lower bound on the latency of each application activity, while the application latency bound is defined by their maximum. Defined by design space exploration, model of application, model of architecture and mapping scenarios.

PREESM/SPIDER:

PREESM and SPIDER have similar behavior with regards to the management and optimization of KPIs. The two tools take as inputs a model of the architecture, a model of the application which has to be mapped on the architecture, expressed as data-flow graph, and a set of constraints, as for instance which task can be mapped to which core. The output of the tool is an optimized mapping of the application on the target architecture, an optimized scheduling and an optimized memory allocation for the selected resources. The tool also generates the codes and wrappers needed for the allocation and the scheduling selected. The main difference between the two tools is that PREESM operates at compile time, based on a static description of the application, while SPIDER operates at runtime, and make all resource allocation decision, and KPI optimizations, during the execution of the application, thus supporting adaptive reconfiguration of the application. KPIs here are used to drive the

decisions of the tool, thus both tools provide support of type one. Among them, the one relevant for us are:

- **Throughput**: Throughput as a KPI is passed to the tool using the model of the application. This choice was made because the application may dictate the

throughput requirements, for instance, if a sensor should have a specific sampling rate, this is coming from the application.

- **Latency**: Latency of computations is a KPI which is dependent both on the application and the targeted architecture. Because, of this dual dependency, latency of the actor, is defined in the set of constraints given to the tool, for each pair of application-architecture model, and reporting the worst case execution time, or the mean execution time of actors.

- **Resource utilization**: Resource utilization, is both an input and an ouptut KPI to PREESM. As an input, constraints on the resource utilization are given to the tool, constraining for example the cores on which the actors can be executed, or limiting the capacity of available memories. As an output, PREESM maps and allocates computing and communication of the application to the resources of the architecture. Some useful resource utilization KPIs are thus obtained, such as load balancing and processing load of computing resources, or the percentage of available memory being used.

The following KPI could be or will drive the decision of the tool

- **Energy consumption**: The energy consumption is not yet supported by PREESM. However, the work to integrate energy consumption is pretty advanced, and it is already possible to discuss how energy consumption will be included in the tool. Energy consumption will be passed to the tool mainly by annotating it in the model of the architecture and in the constraints but energy consumption will be also included in the description of the application. The tool, internally, will take the contribution from each of the inputs to obtain a complete picture of the energy consumption to be used during the optimization.

- **Reconfiguration time**: This KPI is not yet included in the tool, but a preliminary analysis demonstrates that also this KPI could be easily supported by PREESM. From this initial exploration, it seems that the most suitable place to include reconfiguration time is in the model of the architecture.

ARTICo^3:

ARTICo^3 is a hardware-based reconfigurable processing architecture for high-performance embedded computing systems. Dynamic and Partial Reconfiguration (DPR) is used to change the number and type of hardware accelerators in the FPGA, creating a run-time adaptive solution space with selectable tradeoffs between computing performance, energy consumption and fault tolerance. ARTICo^3 provides KPI support of type two. Hence, the main KPIs in the ARTICo^3 framework are:

- **Latency/Throughput:** ARTICo^3 architecture exploits task- and data-level parallelism by using one or more hardware accelerators per functionality. Its optimized communication infrastructure enables almost linear scalability in terms of execution time when working with heavily computing-bounded tasks.

Hence, latency and throughput are reduced and increased, respectively, transparently and at run-time by dynamically changing the number of accelerator instances in the FPGA. ARTICo^3 toolchain does not require inputs to specify latency and throughput, however the tool generate the support to promptly react at run time to measures of latency and throughput coming from the dedicated monitors.

- **Energy consumption:** while it is true that increasing the number of hardware accelerators inside an FPGA leads to increased power consumption, it can also lead to a significant decrease in terms of energy consumption if execution time is also drastically reduced. In ARTICo^3, the multi-accelerator setups that can be configured in the FPGA enable energy-efficient execution of heavily computing-bounded tasks. As in the case of latency and throughput, ARTICo^3 does not use energy consumption directly, but generates the infrastructure for reacting to run time measures of energy consumption coming from dedicated monitors.
- **Reliability:** one of the built-in features of the ARTICo^3 architecture is its capability of performing redundant execution on demand. Hence, the reliability of the hardware-accelerated processing is increased by using two or three copies of the same accelerator but working in Double or Triple Modular Redundancy (DMR, TMR). The specific configuration of a task is set at run-time from the host code defined by the user/developer. As in the case of latency and throughput, ARTICo^3 does not use reliability directly but generates the infrastructure for reacting to run time measures of reliability coming from dedicated monitors.
- **Resource utilization (in terms of area of basic components):** reconfigurable computing systems based on DPR technology enable time-multiplexing of the FPGA fabric. However, in ARTICo^3 the FPGA is partitioned in several reconfigurable regions called slots, where only a limited amount of resources is available for designers to describe their custom logic. To estimate the resource utilization ARTICo^3 uses the external tool chain specific of the target FPGA, and uses the results of that toolchain to verify the feasibility of the proposed design.
- **Reconfiguration time:** DPR is a time-consuming procedure that cannot be neglected when computing the overall execution time for a given task. However, this time is usually bounded, and can be used at run-time to perform more accurate predictions/estimations on the behavior of any task being executed in the architecture. As latency and throughput, reconfiguration time is measured at runtime using dedicated monitor. For reconfiguration time the monitor can be as simple as a counter.

Just-In-Time Hardware Composition (JIT-HW):

Just-In-Time HW composition is a tool for composing accelerators at run-time using Dynamic Partial Reconfiguration (DPR). It consists of a multi-grain reconfigurable overlay which can be used to map different accelerators. Two different approaches are envisaged. An evolutionary/iterative approach that has been implemented using Block-Based Neural Networks (BBNNs) which finds a solution of a problem finding the weights, biases and the topology of the network using an evolutionary algorithm, and a deterministic approach for composing accelerators from dataflow graphs obtained from SW descriptions (still under development), by relying on intermediate representations.

JIT-HW provides KPI support of type one and type two. The KPIs relevant to this tool are the following ones:

- **Implementation/training time:** one of the main objectives of this tool is to compose accelerators in the minimum possible time, from high-level goals or from the software descriptions. In the evolutionary/iterative approach, there is a training phase where different parameters are modified using an evolutionary algorithm until finding a valid solution. In the deterministic approach, there is a mapping phase where the nodes of an intermediate representation in the form of a dataflow graph are mapped onto the overlay and a routing phase where the different processing elements of the overlay are routed as indicated in the dataflow graph. There is a trade-off between the time dedicated to these phases and other KPIs as resource utilization, latency and quality of service. This KPI is used to decide, based on the implementation or training time, the number of accelerators (composable overlays) being trained in parallel. Specific monitors getting information from the fitness function have been implemented with this aim. This information is available also for future uses, for instance to improve the training algorithms (for instance, by adjusting the parameters of the evolutionary/iterative mapping algorithms).
- **Reconfiguration time:** the implementation/training phases use DPR to adapt the reconfigurable overlay to compose the required accelerator. Traditional dynamic partial reconfiguration may be not fast enough in some cases and can lead to very long implementation/training phases. In order to improve reconfiguration times, we have developed IMPRESS, a tool made specifically for JIT-HW composition that allows to use different reconfiguration granularities. Coarser granularities can be used to add or change processing elements of the overlay while finer granularities can be used to reconfigure LUT-based components that are part of a processing elements (e.g., constants, multiplexers and functional units) in a fast way by only changing LUT truth tables. In that way, a coarser granularity is only used for changing the overlay (i.e., adapt its size in the BBNN or modify the PE in the deterministic overlay) and a finer granularity is used to fine-tune a specific overlay. As in the case of ARTICo^3, reconfiguration time is measured at runtime using dedicated monitor. Based on the information provided by the monitor, the adaptation manager could take decisions on which granularity (fine or coarse) and the flavour of reconfiguration to be used to provide adaptation.
- **Resource utilization (in terms of area of basic components):** an accelerator can be mapped in overlays with different size. In resource constrained systems the smaller the overlay, the better. However, there are trade-offs between this KPI and other KPIs. On the one hand, in the deterministic approach, a smaller overlay can lead to longer place and route implementation times. On the other hand, in the iterative approach, a smaller overlay will have shorter training phase at the expense of a possible worst solution. As in the case of ARTICo^3, resource utilization report is generated by an external tool chain and the result is used to verify that the proposed design fits the constraints.
- **Quality of service and latency**: these KPIs indicate the quality of the accelerator. The total latency of the accelerator is one of the metrics that can be used in both approaches to see how good the accelerator is. Moreover, in the iterative approach it is possible to build accelerators that can be

approximated to the desired functionality with different levels of quality of service. Information about quality of service are collected at run time using dedicated monitor specifically designed to compute the quality of service (i.e. fitness resulting from evolution) function.

- **Energy consumption:** when JIT-HW composition is used in embedded systems, is important to reduce energy consumption. Energy consumption can be divided in the energy used to reconfigure the overlay in the implementation/training phase and the energy consumed by the accelerator once it has been composed. As in the case of ARTICo^{3,} energy consumption is measured at runtime by dedicated monitor. Energy measurements could also be used at run-time as a metric to guide the evolution (composition) of the architecture (accelerators).

MDC:

MDC is a tool for dataflow to hardware composition. Starting from different input specifications, it is capable of assembling a target-agnostic coarse-grained reconfigurable datapath that can be used within a co-processing unit. MDC provides support of KPIs of type one and of type two. Below the set of KPI that are relevant for MDC or the co-processing units created using MDC.

- **Resource utilization (in terms of area of basic components):** This KPI is relevant for the tool to deploy constraint-aware system configurations. MDC exploits datapath merging techniques to merge the different input dataflow specifications within a single reconfigurable datapath. Datapath merging process, in genral, is meant to minimize the number of actors and connections. MDC has an extension specifically related to structural profiling, which is for ASIC technology only. The profiler explores all the possible topologies in the design space and extracts some pareto curves, which are used from by users to select the configuration respecting his/her area and frequency constraints. In the FPGA case, Pareto analysis in not yet available and actors and connection minimization are minimized [3].
- **Throughput, Latency, and Energy Consumption:** These KPIs are relevant for the deployed accelerator and can be used at runtime to drive it. Given a fixed operation, for example a filter or a jpeg decoder [6], different configurations of the actors in the input kernel could lead to architectural implementations offering different throughput, latency and energy consumption profiles. The different profiles can be mixed together to support different runtime trade-offs exploitable at runtime, in a self-automated manner or in a user-commanded manner, to meet non-functional constraints [2]. Latency and throughput are predictable at design time at the dataflow specification level [1]. The energy consumption can be evaluated at runtime by measuring the power consumption of each single configuration and then multiplying it by the latency of the selected execution [2]. The possibility of using these KPIs within MDC compliant accelerators depends by the users and the context of application. It is user's responsibility to provide different profiles as input to MDC.
- **QoS** This KPI is relevant for the deployed accelerator and can be used at runtime to drive it. Similarly to the throughput, latency, and energy consumption case it is user's responsibility to provide different profiles as input to MDC. In this case the profiles shall match different quality profiles. For instance, in the case of the interpolation filters of an HEVC decoder, an MDC compliant coarse-grained accelerator can support different approximated

computations offering different precisions of computation, by changing the number of taps in the input kernels [5]. The different profiles, in [5], corresponds to different power (and energy) consumptions values; therefore, at runtime, profiles can be tuned to meet the current requirements.

- **Reconfiguration time:** This KPI is relevant for the deployed accelerator when different types of reconfiguration are supported. Generally speaking, to configure and MDC compliant accelerator, the host processor writes the configuration register of the co-processing unit, then 1 clock cycle is needed to reconfigure the datapath infrastructure of the central computing core of the co-processing unit itself. This series of actions is fixed for MDC-compliant accelerators. Therefore, in stand-alone MDC usage reconfiguration time may not be considered as a KPI. Nevertheless, it becomes a KPI of the deployed accelerator in a multigrain reconfigurable infrastructure as that one presented in [4].

PAPIFY:

PAPIFY is a tool for dataflow application monitoring and instrumentation. Starting from a dataflow specification and relying on components included in the Performance Application Programming Interface (PAPI) library, PAPIFY provides both the instrumented code and the monitoring infrastructure required to access, in run-time, low-level performance information. With this information, the different KPIs required (in run-time) by the tools/applications can be inferred. For this reason, PAPIFY has not relevant KPIs itself, but generating the support to expose low-level performance information, enables the run time KPI estimation associated to the CERBERO tools. PAPIFY provides KPI support of type two.

SAGE:

The SAGE verification suite is actually composed of two tools, namely ReqV, that is devoted to the automated (formal) consistency checking of requirements expressed in natural language, and HyDRA, which allows the automated synthesis of high-level task-oriented optimal “correct-by-construction” policies. As in the case of PAPIFY, the tools in the suite do not manage KPIs in specific ways. In particular, in the case of ReqV, KPIs could be involved in the requirements formulation by the designer as input variables or state variables in the high-level system model under verification. SAGE provides KPI support of type one.

Table 4.1 summarizes the type of KPI support provided by each tool.

TOOL	**Support type one**	**Support type two**
MECA	X	
DynAA	X	X
AOW	X	
PREESM/SPIDER	X	
ARTICo^3		X
JIT-HW	X	X
MDC	X	X
PAPIFY		X
SAGE	X	

2. Table: the X indicates that the tool provide the support for KPIs of that type.

5. KPIs within use cases

This section provides an update on the KPIs and the metrics that are used to measure them in CERBERO use cases. For the previously defined KPIs, for completeness we provide a summary of the definition present in D3.4. For the newly introduced KPIs, or for the one that have been modified after D3.4, we provide the new definition.

5.1. Smart Traveling

The Smart Traveling use case address the problem of driving assistance for electric vehicle, considering several parameters and constrains, including the insurance of the sufficient level of battery, use driving styles, and different types of cars.

The following KPI, have been initially defined in deliverable D3.4 for this case of study. Below we summarize and update these definitions. Also, for this use case, we introduce here a new separation of KPIs into two categories, the first category contains measures on the primary process of the system, being the proposed routes, whereas the second category contains KPIs that are about the reconfiguration process (secondary process). Note that for the primary KPIs only the KPIs of the eventual best recommendation is taken into account, and not all rejected itineraries. As such on a global level the *primary process* KPI is of type maximum since only the best performing recommendation is used.

Primary KPIs

- **Energy**, defined as the total amount of electric energy required to bring the car from the start to the end of the itinerary. This is of type minimum since the minimal amount of used energy in the simulation is the one that is taken into account in the final user recommendation
- **Cost,** defined as the financial cost of the charged energy during the travel. This KPI is of type additive since the total cost can be computed by the cost of the individual charging actions.
- **User compliance.** The systems' proposed itinerary should be in compliance with defined user preferences. This includes the number of proposed stops, as well as usage of preferred charging services, or other facilities that are available at intermediate stopping points. This type of KPI is a ranked feature.
- **Total travel time.** The time to traverse the itinerary is the use case specific physical variant of the latency KPI. However, this KPI is of type minimum, as only the shortest proposed total travel time is taken into account.
- **Driver status.** The driver support functionality should be able to verify the current status of the driver in order to provide or adapt advice to the driver. For this special sensors will be added to the simulator to monitor eye movement and eye lids during the trip to detect status of the driver (like tiredness). This KPI belongs to the family of minimum, because we conservatively select the minimum threshold for parameters that allows to identify the status of the driver in order to react to the situation.

Secondary KPIs

- **Latency:** defined as the delay time of signals within the system. Both the latency of the reconfiguration process (for communication between system components) is used as a performance indicator. This KPI is of type additive.
- **Throughput:** defined as the amount of data transmitted from sources to destinations. This KPI is of type minimum.
- **Quality of service:** defined as the overall performance of the simulations executed for the of the Smart Travelling scenario and related to correctness of events, response to events, precision of the simulation, accuracy of the timing events, accuracy of the responses. This is a ranked feature.
- **Response time to triggers:** defined as the number of seconds or milliseconds needed for the simulation to respond to triggers. This total response time contains the latency, but also the time needed to run the simulations. This KPI is of type additive.

5.2. Self-Healing for Planetary Exploration

Self-healing for planetary exploration explores the use of self-monitoring and self-healing capabilities to overcome the failures caused by cosmic radiations. For this use case, there is no change in the type of KPIs and in the definition of them with respect to deliverable D3.4. For completeness, we report a summary of the previously defined KPIs in the following bullet lists:

- **Latency** is the delay before the robotic arm starts moving after solving the kinematic equations. This KPIs is of type additive.
- **Throughput** is maximum rate that inverse kinematics equations can be solved. This KPI is of type minimum.
- **Energy** due to computation and movements of the robotic arm. This KPI is of type additive.
- **Power** due to computation and movements of the robotic arm. This KPI is of type additive.
- **Resources utilization** is the amount of resources used in rad-tolerant Zynq FPGA. This KPI is additive.
- **Security** is the confidentiality in communication with the robotic arm. This KPI is a ranked feature.
- **Reliability** is the reliability of the system in harsh environments with radiation effects. This KPI is a ranked feature.
- **Response time to triggers** Reconfiguration time due to radiation effects. Different topologies will be used to analyze response time to triggers and reconfiguration. This KPI belongs to the family maximum.
- **Availability** Availability of communication with the robotic arm. This KPI is of type ranked feature.
- **Cost:** the development cost. This KPI is additive.

5.3. Ocean Monitoring

The Ocean Monitoring use case exploits video-sensing, mounted on underwater ocean robots, to serve as marine eyeballs that can capture live videos and images. For this use case, we updated the list of KPIs with respect to deliverable D3.4. Below we report a summary of the previously defined KPIs together with the new KPIs.

- **Throughput** is mainly related with the throughput of the communication and it is defined as the amount of data transferred per time unit within a given communication channel. It is a KPI belonging to the family minimum.
- **Energy** is the amount of energy available in a battery used for the motor and electronic equipment in a marine robot. This KPI belongs to the family maximum.
- **Power** is the power needed for propulsion, steering, and other digital equipment onboard the marine robot. This KPI is additive.
- **Response time [to triggers]** is related with the time needed to retrieve the data once they are requested. This KPI is of time maximum.
- **Cost** are of two types 1) the financial cost to prepare robot for trip and 2) the time cost to prepare robot for trip and duration of trip. Both KPIs are additive.
- **Image Quality** can be defined in terms of measurable characteristics, perceived degrees of quality, or a combination of these. In our case, image quality is a ranked feature.

The following KPIs have been added with respect to deliverable D3.4

- **Disparity.** The difference in image location of the corresponding points from two images of scene taken from different viewpoints when projected under perspective transformation.

- **Geometric distortion.** The dissimilarity between rectified and original image.

  The new KPIs are needed to measure the quality of image rectification which in turn is needed for certain image fusion techniques such as depth maps and super-resolution.

6. References

[1] C. Rubattu *et al*., "*Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators*". In *IEEE Embedded Systems Letters*. doi: 10.1109/LES.2018.2882989.

[2] F. Palumbo *et al.*, (2019) "*Challenging CPS Trade-off Adaptivity with Coarse-Grained Reconfiguration*". In: De Gloria A. (eds) Applications in Electronics Pervading Industry, Environment and Society. ApplePies 2017. Lecture Notes in Electrical Engineering, vol 512. Springer, Cham.

[3] F. Palumbo *et al.*, "*Power-Awarness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy*". In the Journal of Signal Processing Systems, vol. 87, pp. 81–106, Apr 2017.

[4] T. Fanni *et al.*, "*Multi-Grain Reconfiguration for Advanced Adaptivity in Cyber-Physical Systems*". In the 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 2018, pp. 1-8. doi: 10.1109/RECONFIG.2018.8641705

[5] C. Sau *et al*., *"Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multifrequency Approximate Computing"*. In the IEEE Embedded Systems Letters, vol. 9, no. 3, pp. 65-68, Sept. 2017. doi: 10.1109/LES.2017.2703585

[6] C. Sau *et al*., *"Automated design flow for multi-functional dataflow-based platforms"*. In Journal of Signal Processing Systems, v 85, pp. 143-165. Doi: https://doi.org/10.1007/s11265-015-1026-0

[7] Jensen, Jeff C., Danica H. Chang, and Edward A. Lee. "A model-based design methodology for cyber-physical systems." *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE, 2011.