Ref. Ares(2020)7968314 - 28/12/2020

D1.3 – PRELIMINARY VERSION OF CPSOSAWARE SYSTEM ARCHITECTURE

Authors	Pavlos Kosmides (CTL), Eleni Adamopoulou (CTL)
Work Package	WP1 – Requirements, Use Cases, Specifications and Architecture

Abstract

This document focuses on the capturing and presentation of technical specifications of the system components, including functional and non-functional requirements. It introduces the **CPSoSaware Technical Specification Elicitation Framework**: based on established requirements engineering processes, the framework is aimed at eliciting a comprehensible list of system and component requirements that will facilitate reaching a precise architecture design for the CPSoSaware system. The results from applying the framework are manifested as an aggregate collection of knowledge items related to technical component specifications and requirements that take the form of a **living reference document** accessible by all involved stakeholders, in the sense that it will be frequently revisited and reiterated throughout the project lifetime. The presentation of this living document, along with an account of the defined architectural blocks and the preliminary version of the system architecture, constitute the main outputs from this deliverable.

Funded by the Horizon 2020 Framework Programme
of the European Union

Deliverable Information

Work Package	WP1 – Requirements, Use Cases, Specifications and Architecture
Task	T1.3 – CPSoSaware System Specifications and Architecture
Deliverable title	Preliminary Version of CPSoSaware System Architecture
Dissemination Level	Public
Status	F
Version Number	1.0
Due date	31/12/2020

Project Information

Project start and duration	01/01/2020 – 31/12/2022, 36 months
Project Coordinator	Industrial Systems Institute, ATHENA Research and Innovation Center 26504, Rio-Patras, Greece
Partners	1. ATHINA-EREVNITIKO KENTRO KAINOTOMIAS STIS TECHNOLOGIES TIS PLIROFORIAS, TON EPIKOINONION KAI TIS GNOSIS (ISI) * Coordinator 2. FUNDACIO PRIVADA I2CAT, INTERNET I INNOVACIO DIGITAL A CATALUNYA (I2CAT), 3. IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD (IBM ISRAEL 4. ATOS SPAIN SA (ATOS), 5. PANASONIC AUTOMOTIVE SYSTEMS EUROPE GMBH (PASEU) 6. EIGHT BELLS LTD (8BELLS) 7. UNIVERSITA DELLA SVIZZERA ITALIANA (USI), 8. TAMPEREEN KORKEAKOULUSAATIO SR (TAU) 9. UNIVERSITY OF PELOPONNESE (UoP) 10. CATALINK LIMITED (CATALINK) 11. ROBOTEC.AI SPOLKA Z OGRANICZONA ODPOWIEDZIALNOSCIA (RTC) 12. CENTRO RICERCHE FIAT SCPA (CRF) 13. PANEPISTIMIO PATRON (UPAT)
Website	www.cpsosaware.eu

Control Sheet

Version	Date	Summary of changes	Author
0.1	09/11/2020	*Document outline created*	CTL
0.2	10/11/2020	*v1 of Chapters 2 and 3*	CTL
0.3	12/11/2020	*v1 of Chapters 4 and 5*	CTL
0.4	13/11/2020	*Added Subsection 4.2*	CTL
0.5	17/11/2020	*Added executive summary and Chapter 6*	CTL
0.6	02/12/2020	*Added abstract and Appendix A*	CTL
0.7	09/12/2020	*v2 of Chapter 5*	CTL
0.8	15/12/2020	*Internal review*	CTL
1.0	18/12/2020	Final version submitted to EC	CTL

	Name
Prepared by	CTL
Reviewed by	Pekka Jääskeläinen (TAU), Georgios Keramidas (UoP)
Authorised by	CTL

Date	Recipient
9/12/2020	Project Consortium
18/12/2020	European Commission

Table of Contents

List of Figures

List of Tables

Executive Summary

This document constitutes D1.3 "Preliminary Version of CPSoSaware System Architecture" and reports on the outcomes of the first phase of Task 1.3 during the first year of the project. The focus is on the technical specifications of the system components, including functional and non-functional requirements. Towards this direction, D1.3 introduces the **CPSoSaware Technical Specification Elicitation Framework**: based on established requirements engineering processes, the framework is aimed at eliciting a comprehensible list of system and component requirements that will facilitate reaching a precise architecture design for the CPSoSaware system. The results from applying the framework are manifested as an aggregate collection of knowledge items related to technical component specifications, requirements, and interfaces that take the form of a **living reference document** accessible by all involved stakeholders, in the sense that it will be frequently revisited and reiterated throughout the project lifetime. The presentation of this living document, along with an account of the defined architectural blocks and the preliminary version of the system architecture, constitute the main outputs from this deliverable.

1 Introduction

When referring to software and hardware systems, the term "**architecture**" is used metaphorically, with a meaning equivalent to the architecture of a building, referring to an outline for the system to be developed and the tasks that need to be completed in order to reach the final result [Perry & Wolf, 1992]. Therefore, **system architecture** is aimed at specifying the fundamental components of a (sophisticated) system, the involved software and hardware elements, their interrelations, and the properties of both elements and relations [Clements et al., 2003].

Specifying the architecture of a system is heavily interlinked with the process of **requirements engineering**, which is aimed at assessing whether the developed system meets the purpose for which it was initially intended [Nuseibeh & Easterbrook, 2000]. In fact, the two processes are often seen as complementary: architecture is aimed at the "how", while requirements engineering is aimed at the "what"; both of them, nevertheless, revolve around stakeholder concerns, needs and wishes [Shekaran et al., 1994].

In this context, the overarching goal of this document is to present the technical specifications of the CPSoSaware system components, which have been elicited based on established requirements engineering processes and will lead to deriving a preliminary version of the system architecture.

1.1 Document Structure

The rest of the document is structured as follows:

- **Chapter 2** is an introduction to requirements engineering, requirements elicitation methodologies and the CPSoSaware objectives within this context;
- **Chapter 3** describes the methodologies applied for capturing of CPSoSaware technical component specifications and requirements;
- **Chapter 4** presents the collected knowledge per technical component of the CPSoSaware system;
- **Chapter 5** presents a preliminary version of the CPSoSaware system architecture;
- **Chapter 6** concludes the document with some final remarks and directions for the next steps.

1.2 Definitions and Acronyms

Below is a list of the most relevant acronyms used in the document together with their recurring definitions:

Acronym / Term	Definition
CICL	CPSoSaware Intra-CPS Communication Layer
CPS	Cyber-Physical System
CPSoS	Cyber-Physical System of Systems
CSAIE	Cognitive System AI Engine
DCCI	Distributed, Cognitive and Cooperative Intelligence

DoA	Grant Agreement No. 871738 – CPSoSAware. Annex 1 Description of the Action.
DSL	Domain-Specific Language
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
IP	Intellectual Property
ME	Monitoring Engine
ML	Machine Learning
MRE	Modelling and Redesign Engine
ODE	OpenCL Description Execution
P2P	Peer to Peer / Point to Point
PoCL	Portable Computing Language
RE	Requirements Engineering
SICL	System Inter-Communication Layer
SoC	System-on-Chip
SRMM	Security Runtime Monitoring and Management
SW	Software
TRL	Technology Readiness Level
XRT	Human in the loop situational awareness using XR tools

2 Requirements Elicitation and Analysis - Background

This chapter briefly introduces Requirements Engineering and focuses on two of its key activities, requirements elicitation and analysis. The most popular methodologies for eliciting requirements are then presented, followed by an overview of the objectives of the requirements capturing process within CPSoSaware.

2.1 A Broad Definition of Requirements Engineering

As discussed in the introduction, within the context of software and hardware systems development, **Requirements Engineering** (**RE**) is the process of assessing whether the developed system meets the purpose for which it was initially intended [Nuseibeh & Easterbrook, 2000]. This process entails the identification of stakeholders and their respective needs from the system, as well as the documentation of these needs in a way that will facilitate their analysis and will drive the subsequent implementation of components that will address them. Due to the numerous challenges involved (e.g., communication gaps with stakeholders, unclear or even conflicting goals and needs, etc.), RE is considered highly critical for delivering an accurate software architecture design and plays a key role particularly during the first steps in the development process, as illustrated in **Figure 1** [Liao, 2002].

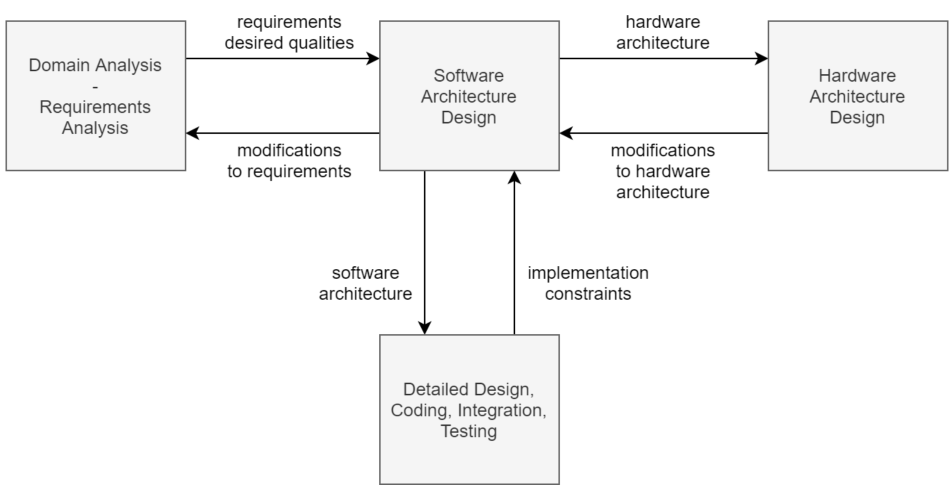

Figure 1. The process from requirements analysis to system architecture [Liao, 2002].

RE involves two main activities: (a) **requirements elicitation**, which is aimed at specifying the system to be developed in a form that the end-user understands, and, (b) **requirements analysis**, which promises to deliver an analysis model that can be unambiguously interpreted by the developers of the system [Bruegge & Dutoit, 2009].

Regarding requirements, a typical distinction is often made between **user requirements**, i.e., the requirements derived from the potential end-users, and **technical requirements**, i.e., the requirements referring to the technical aspects of the components to be developed. Although the elicitation of the former usually precedes the latter, a successful system design should involve the collection and analysis of both types of requirements.

More specifically, user requirements typically describe aspects of the system from the end-user perspective that are not directly related to the functional behaviour of the system, like, e.g., response time, accuracy, etc., and are also known as **non-functional requirements**. On the other hand, **functional requirements** specify the explicit functions of a system and its interaction with the environment, and give an outlook of its technical aspects, processes, and dependencies. Consequently, functional requirements are the key driver in defining the architecture of the system.

In the case of a complex software system consisting of numerous software components, the system requirements include the functional and non-functional requirements of each component, as well as the specifications that refer to the system-wide behaviour and functionality. In order to avoid confusion between the two levels (i.e., system vs component) and to better organize the pertinent knowledge, it is common practice to come up with lists of system and component-level use cases and then map requirements to them accordingly.

2.2 Methodologies for Capturing Requirements

During requirements elicitation, it is common practice to apply a variety of related methodologies and techniques, in order to obtain a more holistic outlook of the domain and to acquire knowledge from various stakeholders, such as end-users, domain and technical experts [Eid, 2015]. This subsection briefly presents the most established methodologies in gathering requirements.

2.2.1 Document Analysis

The analysis of existing documentation is a valuable first step in requirements elicitation, since it leads to a better understanding of the domain and the system to be developed and can help substantially during the next steps, e.g., for formulating more accurate questions for the interviews with stakeholders (see next subsection). And, inversely, if certain responses from the interviews are unclear, analysing existing documentation may help in clarifying things. The downsides to this approach are that (a) it is a time-consuming process, and, (b) the documentation may be out of date.

2.2.2 Interviews with Stakeholders

There are two types of interviews with stakeholders in order to elicit requirements: one-on-one sessions and group interviews. One-on-one sessions are arguably the most common technique for gathering requirements and should be well-prepared beforehand, in order to get the most out of them. The appropriate stakeholders to be interviewed should be identified and a list of both open-ended and close-ended questions must be prepared. The former questions facilitate retrieving more holistic and high-level knowledge and allow the interviewer to focus on more specific aspects with more elaborate subsequent questions, while the latter are more useful in covering more topics in a deeper manner and in less time. Once the list of questions is complete, it is typically a good practice to send them to the interviewee prior to the interview so that they better prepare (see also next subsection). An additional good practice is to record the session (with the interviewee's consent), so that the responses can be revisited at a later time.

Group interviews, on the other hand, are similar to one-on-one interviews, with the exception that more persons are being interviewed at the same time. This type of interview is ideal when the interviewees all have similar positions in the organization and/or experience and background. The key advantage of group interviews is that responses by one interviewee may trigger further discussion by the others, which leads to eliciting richer information during the interview. The major drawback is that group interviews are hard to schedule, since establishing a date/time slot that works well for all parties can prove quite challenging.

2.2.3 Requirement Specification Templates

Requirement specification templates are structured questionnaires circulated to stakeholders prior to interviews and are very well suited for involving multiple parties at once, guiding them to provide focused responses and descriptions on specific aspects of the system to be developed. In case of unclear responses, clarifications may be discussed during the interviews.

In the literature one can find several established requirement specification templates, like, e.g., Volere[1], which can be extended, in order to include additional fields specific to the system at hand.

2.2.4 Use Case Analysis

Use cases in software engineering are the sets of actions or events that define the interactions between a system and the involved agents; the latter being human or machines external to the system [Bruegge & Dutoit, 2009]. Use cases do not directly point to requirements, but analysing them leads to identifying desired system behaviours and qualities, which may be implicitly converted to requirements. In the cases of complex systems consisting of multiple components, use cases are typically defined per component or per group of components, and their analysis will lead to extracting functional and non-functional requirements at the component level.

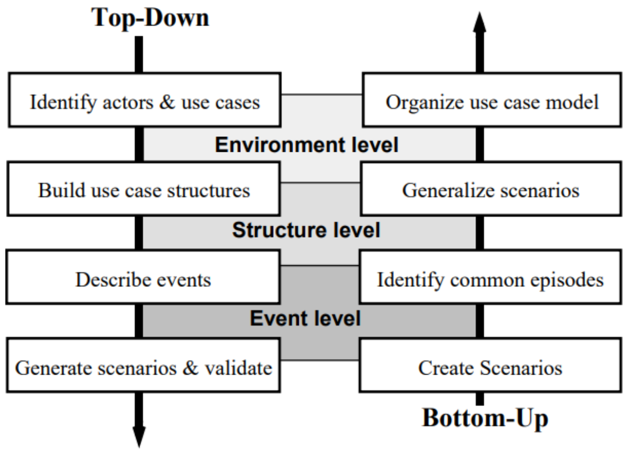

Figure 2. Top-down vs bottom-up use case analysis approaches [Regnell, 1996].

According to Regnell (1996), the process of extracting requirements from use cases can be either bottom-up, with the aggregation of component-level use cases comprising the system-level use cases, or top-down, where high-level use cases and related actors are initially identified, and then sub-scenarios and requirements are specified towards the component-level (see **Figure 2**).

2.3 Requirements Elicitation Objectives in CPSoSaware

Our overarching objective for the requirements elicitation process during this first period (M1-M12) of the project was focused on generating a project-wide reference document collaboratively with the rest of the involved partners that will focus on the following:

(a) **Functional and non-functional requirements** both on the component and the system level, identifying the desired functionalities of each component and of the system as a whole.
(b) **Mapping of use cases and roles** to the technical requirements.
(c) **Dependencies** per system component.

[1] https://www.volere.org/templates/volere-requirements-specification-template/

(d) **Potential implications and conflicts** between requirements.

Towards this objective, we adopted the requirements elicitation methodologies presented in the previous subsection and came up with a reference document, which we intend to have in the form of a "**living document**" that will be frequently revisited and reiterated. This way, the requirements will be refined and re-adjusted throughout the project's lifecycle. The first version of the document and its elements is described in the rest of this deliverable, while the next iterations of the document will be reported in the upcoming deliverables D1.4 "Second Version of CPSoSaware System Architecture" (due M24) and D1.5 "Final CPSoSaware System Architecture" (due M36).

3 CPSoSaware Technical Specification Elicitation Framework

Within CPSoSaware and under the scope of WP1, a task force has undertaken the application of established requirement engineering processes towards fulfilling the requirement elicitation objectives and enabling a precise architecture design for the CPSoSaware system. This chapter is aimed at presenting the activities that were performed to capture technical component specifications, driven by the team's expertise and methodologies found in literature.

3.1 Component Specification Templates

Inspired by the Volere methodology and its variations (see Subsection 2.2.3), a detailed requirement specification template was assembled and distributed to partners involved in the design and implementation of technical components. The template intended to provide a common vocabulary for the homogeneous expression of technical and non-technical details. Divided into four logical sections, the template incorporated free text fields for questions regarding the component high-level description, expected inputs and outputs, functional and non-functional requirements, and deployment/integration conditions.

Since this effort started early in the project and proceeded in parallel with the definition of pilot use cases, the participants were encouraged to only insert available information - omitting fields that remained yet undecided - and periodically revisit the document with updates deriving from the evolvement of system-level requirements.

A detailed presentation of the template segments is presented in the following sections. The complete specification template can be found in *Appendix A: CPSoSaware Component Specification Template*.

3.1.1 Descriptive Component Specification

The first part of the component specification template intended to capture high-level component metadata. More specifically, it included the following fields:

- **Task name:** The task(s) from the Description of Action (DoA) where the component implementation is detailed.
- **Task leader:** The partner(s) assigned to lead the corresponding task(s) in the DoA.
- **Component name:** The name of the technical component.
- **Type:** An indication whether the component is a software, a hardware, or a combination of both.
- **Short description:** An abstract of the component's functionality and purpose within CPSoSaware.
- **Methodologies that will be used:** A short description of the technical and/or scientific methodologies orchestrated for the component implementation.
- **User-defined scenarios (non-technical):** A set of component-level use case descriptions in a non-technical manner.
- **Map to project objectives:** An association of the technical component with the CPSoSaware project objectives as described in the DoA.
- **Relevant Use Cases:** An association of the technical component with the CPSoSaware pilot use cases.

- **Estimated date of first release that can be deployed/integrated:** An early estimation for the first delivery expressed as the month of the project's lifecycle.

3.1.2 Component Inputs and Outputs

A template section is dedicated to collecting information regarding a component's interfaces and expected inputs/outputs. Along with the headway in the pilot use case definitions, the acquisition of such knowledge will prove critical for the design of meaningful pipelines for the CPSoSaware system. A mapping between data owners and data consumers is expected to facilitate development and integration. The corresponding questionnaire fields for data inputs are:

- **Main inputs:** A description of the expected input sources for this component.
- **Input data from partner:** The partner(s) responsible for providing the input(s), either via their technical components or in the form of datasets, etc.
- **Nature of expected input:** The expected format for the requested input (e.g. JSON format, video streams, image files, etc.).
- **Related Scenarios:** Use case scenarios and pilots that associate with the production and/or processing of these data.
- **Interfaces:** The interface(s) provided or required by the technical component for the retrieval of input data.
- **Triggered by:** The event(s) or condition(s) that will trigger the execution or functionality of the component.

Similarly, the fields for expected outputs are:

- **Main outputs:** A description of the output(s) that will be produced by this component.
- **Output data to partner:** The partner(s) responsible for consuming the generated output(s), either via their technical components or in the form of datasets, etc.
- **Nature of expected output:** The expected format for the produced output (e.g., JSON format, video streams, image files, etc.).
- **Related Scenarios:** Use case scenarios and pilots that associate with the consumption and/or processing of these data.
- **Interfaces:** The interface(s) provided or required by the technical component for the delivery of output data.

3.1.3 Functional and Non-functional Requirements

The definition of requirements at the technical component level is meant to portray functional and non-functional necessities for the design, operation, and integration of the component itself. This bottom-up approach suggests that the collection of requirements per component will constitute a subset of the system-level requirement set. Consequently, this effort will be associated with the top-down definition of requirements at the pilot use case level, reported in the upcoming D1.2 "Requirements and Use Cases". Therefore, the corresponding fields in the template are:

- **Main functional requirements:** A set of technical requirements for the design, development, and functioning of the component.

- **Main non-functional requirements:** A set of qualitative and quantitative conditions that the component should cover. These can be related with topics such as scalability, security, accessibility, availability and more.

3.1.4 Deployment and Integration Requirements

The final section of the component specification template aims at eliciting technical details related to the development, deployment, and integration with the rest of the CPSoSaware system. More specifically, the included fields are:

- **Development environment:** The development environment incorporates the operating system(s), IDE(s) and programming language(s) used for the implementation of the component.
- **Execution time:** An estimation for processing time of the component, in case of including heavy processing tasks, etc.
- **Execution frequency:** This indicates how often the execution is expected to take place, in case of periodic executions.
- **Software requirements:** The component's dependency from external software.
- **Hardware requirements:** The component's dependency from external hardware.
- **Communications:** Connectivity requirements, such as access to the Internet, Bluetooth interface, etc.
- **Integration requirements:** Specific requirements regarding the integration of the component with the rest of the system.
- **Deployment requirements:** Specific requirements regarding the deployment of the component.
- **Security requirements:** Specific requirements to avoid any potential security issues.
- **Privacy requirements:** Specific requirements to avoid any potential privacy issues.
- **Critical factors:** Any critical factors that might affect the development or functionality of the component.
- **Containerization:** The ability to be containerized (e.g. with Docker) if the component of discourse is a software module.

3.2 Consulting with Stakeholders

Extensive discussions with involved technology experts allowed the extraction of requirements, system specifications, and potential architecture designs. A series of project meetings, periodic WP1 meetings, ad-hoc sessions with stakeholders and offline communications enabled the dissemination of the component specification templates, the collection of the component list, the elicitation of useful information and, finally, the establishment of a common understanding on the overall CPSoSaware system design. Recurring communications also ensured better comprehension over the applied requirements elicitation framework and allowed the iteration of the process towards a more precise requirement elaboration.

3.3 Analysis of Description of Action

The requirements elicitation process at the early stages of the project demanded the detailed study and analysis of the main reference document, the DoA. The detailed descriptions of the overall project objectives, the proposed system architecture, and the use cases offer a playground for the extraction of functional and non-functional requirements. Work package and task descriptions extensively portray the expected technical components and features. As a result, details from the document were initially studied to identify the main architectural blocks and sub-blocks, while the use case analysis resulted in a preliminary requirement set. Since the DoA was authored prior to the project kick-off, the knowledge derived from the document acted as the basis for the requirements elicitation process, and needed to be verified, refined, and extended along with the maturing of the project's objectives.

3.4 Formalization of Captured Requirements

A critical part of the requirements engineering process is requirement management. This includes the mechanisms for documenting, prioritizing, tracking, agreeing, and communicating specifications to relevant stakeholders. Therefore, we have pursued the aggregation of collected knowledge related to technical component specifications and requirements into a single living document, accessible by all involved stakeholders, where information is encoded in a uniform format. This document intends to limit ambiguity and facilitate the reference and update of requirements throughout the project lifetime.

The document is in MS Excel format, containing several tables that are briefly described below. Information is encoded using appropriate prefixes/suffixes and colour codes to facilitate browsing and search.

Technical component list

This table includes the collection of CPSoSaware technical components, along with short descriptions, architectural blocks, state-of-the-art, and requirements. More specifically, the table contains the following columns:

- **Code:** A unique identifier assigned to the technical component, by assembling the prefix *TC*, the task number, and an incrementing integer (e.g., *TC3.1.2*). This field also acts as a hyperlink to the component specification template previously filled by the corresponding technology expert, in order to enable the fast review of provided information.
- **Component name:** The name of the technical component.
- **Type:** Indicates whether this is a software, hardware, or both.
- **Task:** Indicates the task(s) involved in the design and implementation of the component.
- **Partner:** The partner(s) leading the design and implementation of the component.
- **Short description:** A textual description of the scope, functionality, and objectives of the component.
- **Architectural block:** A set of architectural blocks and sub-blocks has been identified by the study of DoA (see Section 0). This field allows the selection from a pre-defined list of values [*CPSoS system layer*, *CPS/CPHs layer*, *Simulation and training*]. This classification of components enables the conceptualization of the architecture.
- **Architectural sub-block:** Similarly, the component is assigned to the appropriate sub-block. The list of sub-blocks also derives from the DoA analysis and conferencing with stakeholders.

- **Functional requirements:** A set of technical/functional requirements that characterize the component as a standalone module. The requirements are encoded using the component's code, along with the suffix *.R* and an incrementing integer (e.g. *TC3.1.2.R2*) to enable unambiguous references to requirements.
- **Non-functional requirements:** Just like above, non-functional requirements are listed and encoded using the suffix *.NFR* and an incrementing integer (e.g. *TC3.1.2.NFR1*).
- **State-of-the-Art / Innovation:** A short textual description for the state-of-the-art in the technological domain of the technical component, along with potential innovation and advance beyond the state-of-the-art.
- **Current TRL:** The technology readiness level of the component.

Use cases list

This table enlists the identified use cases at the component level (see Section 2.2.4). Use cases are expressed with the simple pattern <u>*Actor -> interacts (Use case or functionality) ->* Component</u>. The table contains the following columns:

- **Component code:** A reference to the technical component using its unique identifier.
- **Component name:** The name of the technical component.
- **Use cases:** The set of use cases for this technical component. Each use case is assigned a code combining the component code with the suffix *_UC* and an incrementing integer (e.g. *TC3.1.2_UC1*).
- **Actor:** The entity that interacts with the component within the context of the specific use case. An actor, which assumes a *role*, can be a human that interfaces with the component or another software/hardware. System-compatible roles are extensively described on a dedicated table.
- **If Actor is HW/SW, identify:** If the actor interacting with the use case has been defined as some hardware or software component, it should be denoted here which external or system component that is. The corresponding component code is the required value (e.g. *TC3.1.1*). This is meant to facilitate the precise architecture definition. In several cases, an actor's appropriate use case code is also defined in this field, indicating connections like: *The use case of component TC3.1.1 (TC3.1.1_UC1) will interact with the use case of component TC3.1.2 (TC3.1.2_UC2)*. The expected result of this drill is to generate sequences of interconnected use cases which will be later used for the definition of sequence diagrams and system-level use cases.
- **Role's interaction:** A textual description of the interaction between the actor and the component within the context of this use case.

Roles list

This is a table for listing actor types (roles), and - in case of human roles - their responsibilities, rights, and duties towards the system.

4 Presentation of Captured CPSoSaware Specifications

This section presents the knowledge captured via the application of the specification elicitation framework described in Sections 2 and 3. This knowledge is contained in the *living document*, arranged in structures, as presented in Section 3.4.

4.1 Roles

As already mentioned, *roles* indicate the types of actors that interact with the CPSoSaware system and its technical **components**. These actors may be other system components, external systems or humans participating in use cases. The identification of the CPSoSaware-compatible roles, along with their potential interactions with CPSoSaware, is crucial for the definition of precise use cases and meaningful architecture designs. Defining roles is a continuous process, highly affected by the ongoing design of pilot use cases, therefore it is performed on an iterative basis. Table 1 presents the currently defined actor types.

Table 1. CPSoSaware compatible roles

Actor	Interaction
Analyst	Uses data-driven processes to gain insights about required actions and potential improvements
CPS/CPSoS Designer	Designs the models for CPSs/CPSoS simulations, communication, and deployment.
End-user	Provides specifications, requirements and preferences for the modelling and system design of CPSs and CPSoS
HW/SW component	-

4.2 Technical Component Specifications

This section outlines the specifications for the technical components provided by the responsible partners who filled in the respective information in the shared reference document. The information was then homogenized into a uniform format by lead partner CTL and is presented subsequently per component.

4.2.1 Data Collection Module

The Data Collection Module is the component based on ElasticSearch that will be developed in order to ingest, store and manage data that is obtained from the activities in T2.1 covering the analysis of user skills/factors, virtual cognitive user/environment models and metrics modelling.

Code: TC2.1.1	**Task:** T2.1	**Partner**: 8Bells	**Type**: Software	**TRL:** 2
Architectural block:		**Architectural sub-block:**		

<table>
<tr><td colspan="2">CPSoS system layer</td><td colspan="2">Cognitive System AI Engine (CSAIE)</td></tr>
<tr><td colspan="4">State-of-the-Art / Innovation:
Still unclear if a well-known tool will be used. For the first step of the questionnaires, we can use google forms in order to collect the data from the pilots and then we will study the way to store them, analyse and graphically represent the outputs (e.g., ElasticSearch Kibana is considered)</td></tr>
<tr><td colspan="2">Functional requirements</td><td colspan="2">Non-functional requirements</td></tr>
<tr><td colspan="2">TC2.1.1.R1 Use an events list to register events.
TC2.1.1.R2 Be able to demonstrate in graphical way the HF models (that can be initially designed in UML format).
TC2.1.1.R3 Use statistics to track important changes in variables.
TC2.1.1.R4 Should provide library of human metrics.
TC2.1.1.R5 Should be able to display in a graphical way these metric.
TC2.1.1.R6 The operator should be able to query the Data Collection data metrics.
TC2.1.1.R7 The operator will be able to input data from a CSV/Spreadsheet into the DCM.
TC2.1.1.R8 The system shall ensure the confidentiality and integrity of the data being transmitted in the system.
TC2.1.1.R9 The system shall ensure the availability of its services to the relevant stakeholders.</td><td colspan="2">TC2.1.1.NFR1 The DCM should scale automatically to meet the demand of new DCM metric data.
TC2.1.1.NFR2 The DCM should provide a secure housing of The metrics data.</td></tr>
<tr><td colspan="4">Component-level Use Cases</td></tr>
<tr><td>Name</td><td>Actor type</td><td>Actor (SW/HW)</td><td>Interaction</td></tr>
<tr><td>TC2.1.1_UC1
Store end-user specifications</td><td>Analyst
End-user</td><td></td><td>Collects and stores end-user requirements and preferences (skills, gender, expertise with ICTs, health condition, daily routines, etc) via surveys, interviews, group sessions, etc.</td></tr>
<tr><td>TC2.1.1_UC2
Analyse end-user specifications</td><td>Analyst
End-user</td><td></td><td>Models users by analysing activities, behavioural parameters, etc, by retrieving stored knowledge. Specifies the number and type of users to be involved in each pilot use case.</td></tr>
</table>

4.2.2 Intra-Communication Sim Tool

Tool designed and implemented to match network requirements imposed by the application and deployed CPSoS to proposed network technologies and configurations (e.g., modulation, signal strength, duty cycle etc.) and network topologies. The tool will be based on the NS3 simulator and it will be built based on experimentation on models of dominant wireless protocols for intra-communication, e.g., BLE, ZigBee/802.15.4, Wi-Fi.

Code: TC2.2.1	**Task:** T2.2	**Partner**: UoP	**Type**: Software	**TRL:** 4

Architectural block: Simulation and training	**Architectural sub-block:** -
State-of-the-Art / Innovation: We are not aware of any platform that is able to deliver the NS3 simulator through a well-defined API.	
Functional requirements	**Non-functional requirements**
TC2.2.1.R1 The user should be able to feed the simulator with specific network scenario configuration. **TC2.2.1.R2** The tool will be able to record the simulation results in log files. **TC2.2.1.R3** The tool will be able to process the log files and extract the evaluation results of the simulation. **TC2.2.1.R4** The simulation outcomes will be able to be indexed in database and visualized (e.g. Prometheus/Grafana, Elasticsearch/Kibana) **TC2.2.1.R5** The evaluation results will be formulated and fed back to the input of the tool to perform optimizations through iterations.	**TC2.2.1.NFR1** The tool should be able to scale according to the load. **TC2.2.1.NFR2** Adoption of microservices paradigm (e.g. containerization). **TC2.2.1.NFR2** Authentication/authorization schemes will be supported. **TC2.2.1.NFR3** The tool will expose well-defined APIs to allow third-party services to integrate **TC2.2.1.NFR4** The tool will be available online.

Component-level Use Cases

Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.2.1_UC1 Define network requirements	CPS/CPSoS Designer		Defines and models performance and energy parameters.
TC2.2.1_UC2 Simulate network topologies	CPS/CPSoS Designer		Simulates various network technologies, topologies and configurations based on defined network requirements.
TC2.2.1_UC3 Generate proposed network technologies and optimizations	CPS/CPSoS Designer		Retrieves the proposed network configurations.

4.2.3 PoCL-Remote

Scalable distributed OpenCL runtime layer with P2P event synchronization capabilities.

Code: TC2.2.2	**Task:** T2.2	**Partner**: TAU	**Type**: Software	**TRL:** 3

Architectural block:	**Architectural sub-blocks:**
CPSoS system layer	CPSoSaware Modelling and Redesign Engine (MRE) CPS Commissioning and CPS to System Inter-Communication Layer components

State-of-the-Art / Innovation:
Previous similar projects did not focus on low latency aspects of CPS or edge offloading.

Functional requirements	**Non-functional requirements**
TC2.2.2.R1 Provide access to all OpenCL-supported devices in a network distributed platform from a single host application. **TC2.2.2.R2** Support peer-to-per synchronization of devices without host-application round trips.	**TC2.2.2.NFR1** At most 15% overhead in latency on top of the unavoidable network latencies. **TC2.2.2.NFR2** Can utilize 80% of the theoretical bandwidth for buffer transfers.

Component-level Use Cases

Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.2.2_UC1 Configure the OpenCL runtime layer	CPS/CPSoS Designer		Configures the parameters for the required OpenCL runtime environment.
TC2.2.2_UC2 Commision/deploy new HW/SW components			Requests the commissioning of a new component to the OpenCL runtime.
TC2.2.2_UC3 Decommision deployed HW/SW components			Requests the decommissioning of a component from the OpenCL runtime.

4.2.4 Slice Manager

The i2CAT 5G platform is an open architecture software platform that facilitates the sharing and slicing of 5G infrastructure elements. This is achieved by leveraging new network virtualization solutions and dynamic configuration enabled by 5G technologies (based on ETSI NFV and Network Slicing). The Slice Manager is the main “entry point” and the “heart” of this platform, and it “hides” OpenStack, OSM (OpenSourceMano), Racoon (i2CAT’ s RAN controller), and potentially other controllers behind it. It provides a REST interface which can be used for either performing or triggering (via delegation to other components) the following main functionalities:

- Manage (create/read/update/delete) computes, physical NWs, and Wi-Fi APs (mainly for Infrastructure owners via the Dashboard);
- Manage (create/read/update/delete) chunks of the above resources (mainly for Slice Users via the Dashboard);
- Manage (create/read/update/delete) slices as collections of the aforementioned chunks;
- Manage (create/read/update/delete) users, which are authenticated by AAA;
- Trigger the deployment of Network Services (NS) on specific slices, while performing related configuration actions (that can be useful for diverse NSs during deployment, so that they function properly).

Code: TC2.2.3	**Tasks:** T2.2, T4.2	**Partner**: i2CAT	**Type**: Software	**TRL:** 4

Architectural block: CPSoS system layer	**Architectural sub-blocks:** System Inter-Communication Layer
State-of-the-Art / Innovation: Past projects did not focus on providing an end-to-end intelligent slice managing and orchestrating facilities for the mobile users.	
Functional requirements	**Non-functional requirements**
TC2.2.3.R1 Secure identification of CPSs. **TC2.2.3.R2** Monitors and assures the behaviour and performance of the various slices through collecting network function and infrastructure data. **TC2.2.3.R3** Slice automation and orchestration. **TC2.2.3.R4** Need to support slice modelling by changing various network functions, connection, and links to create specific network services.	**TC2.2.3.NFR1** Delay of service instantiate **TC2.2.3.NFR2** Service recovery failure and service continuity

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.2.3_UC1 Create slices	CPS/CPSoS Designer End-user		Creates slices as collections of chunks containing computes, physical NWs, and Wi-Fi APs.
TC2.2.3_UC2 Update slices	CPS/CPSoS Designer End-user		Updates slices, slice chunks and contained computes, physical NWs, and Wi-Fi APs.
TC2.2.3_UC3 Delete slices	CPS/CPSoS Designer End-user		Deletes slices, slice chunks and contained computes, physical NWs, and Wi-Fi APs.

TC2.2.3_UC4 Trigger the deployment of Network Services	CPS/CPSoS Designer End-user		Triggers the deployment of NSs on specific slices, while performing related configuration actions (that can be useful for diverse NSs during deployment, so that they function properly).

4.2.5 LightEdge

LightEdge is responsible for bringing the MEC solutions for the mobile network operators and enable the edge facilities among the end-users. Importantly, LightEdge helps end-users to move from the current 4G-based system to 5G-enabled system.

Code: TC2.2.4	**Tasks:** T2.2, T4.2	**Partner**: i2CAT	**Type**: Software	**TRL:** 4
Architectural block: CPSoS system layer		**Architectural sub-blocks:** System Inter-Communication Layer		
State-of-the-Art / Innovation: Currently providing the MEC facilities to the end-users for smoothly transiting into 5G-based system from the 4G-enabled system. However, enabling the functionalities intelligent horizontal or vertical scaling of VNFs is still far from achieved.				
Functional requirements		**Non-functional requirements**		
TC2.2.4.R1 Potentially able to leverage the features and functionality of OSM MANO. **TC2.2.4.R2** Complying with the Cloud native solutions and allowing the containerized edge application. **TC2.2.4.R3** Capable of supporting the Local breakout for enterprise application.		**TC2.2.4.NFR1** Handling the user mobility **TC2.2.4.NFR2** Scalability between the LightEdge enabled MEC servers **TC2.2.4.NRF3** Service failure recovery		

Component-level Use Cases

Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.2.4_UC1 Enable the MEC service facilities to the End-users	CPS/CPSoS designer End-user		Bring the MEC facilities to the end-users

4.2.6 Hardware Accelerator IP Cores

This refers to FPGA-based IP core components. The FPGA IP cores will be automatically generated from higher-level models by using an appropriate ML framework, whenever feasible. The IP cores will be seamlessly integrated in the PoCL-based OpenCL run-time system by means of a hardware abstraction layer (AlmaIF).

Code: TC2.3.1	**Tasks:** T2.3	**Partner**: UoP/ISI/TAU	**Type**: Hardware & Software	**TRL:** 6
Architectural block: CPSoS system layer		**Architectural sub-blocks:** HW/SW Component Library		
State-of-the-Art / Innovation: Previous work is fixed function, our work attempts to improve load balancing scenarios via software-based task switching via (optional) soft core overlays. End to end generation of HW blocks with various computational requirements. HW blocks will be automatically generated from high-level models, and the focus is on making this seamless with the OpenCL-based platform.				
Functional requirements		**Non-functional requirements**		
TC2.3.1.R1 Accelerate DNN inference in comparison to software running in ARM.		**TC2.3.1.NFR1** At least 20% faster 8b convolutions achieved.		
Component-level Use Cases				
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**	
TC2.3.1_UC1 Configure FPGA-based IP accelerators	CPS/CPSoS Designer		Configures metrics (e.g. memory and computational throughputs and quantization techniques of various levels).	
TC2.3.1_UC2 Commission ML Hardware to OpenCL stack	CPS/CPSoS Designer HW/SW component(s)	**TC2.2.2** PoCL-Remote (**TC2.2.2_UC2**)	Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware/software interface visible to PoCL run-time system.	

4.2.7 Security Accelerators for CPS Security Agents/Sensors

FPGA based IP core components (interfaces) focused on security/cryptography.

Code: TC2.3.2	**Tasks:** T2.3, T2.4	**Partner**: USI	**Type**: Hardware & Software	**TRL:** 4
Architectural block: CPSoS system layer		**Architectural sub-blocks:** HW/SW Component Library		
State-of-the-Art / Innovation: Algorithmic modifications are provided in order to increase the performance of the cryptography primitives. Such modifications are aiming to increase the parallel processing capabilities of the algorithm and rely on performing analysis of the Data and control flow so that operations within the cryptographic primitive can be disassociated so as to be parallelized. Apart from that the current state of the art is focused on providing resistance against side channel attacks i.e. secret information leaking from an implementation as the algorithm is executed. Latest research is focused on postquantum cryptography algorithms that are in the process of getting standardized by NIST.				

The SoC platform used as starting point features a Linux operating system but do not fully exploit the capabilities of FPGA for accelerating/enforcing security. Reconfigurable hardware needs to be used to provide more advanced security functionalities and to improve the performance.

Functional requirements	**Non-functional requirements**
TC2.3.2.R1 Confidentiality: The components should provide cryptography services for popular public and private encryption algorithms). Support for public key infrastructure should be possible). **TC2.3.2.R2** Data Integrity: The components should provide cryptography services for popular message integrity mechanisms include MAC functions, digital signature, and authenticated encryption. **TC2.3.2.R3** Authentication: The components should be able to provide authentication services including message authentication, machine to machine (M2M) authentication. The above security/cryptography components should be able to operate in both pilots, in supporting the security of the CAN bus protocol, and V2V communication. Also, in supporting the security of on device attacks in the industrial domain and the industrial network protocols.	**TC2.3.2.NFR1** Resistance against security attacks (side channel attacks). **TC2.3.2.NFR2** Strong cryptographic strength. **TC2.3.2.NFR3** Reliability fault-tolerance. **TC2.3.2.NFR4** Efficiency (response time). **TC2.3.2.NFR5** Efficiency (constrained memory and chip covered area resources). **TC2.3.2.NFR6** Flexibility. **TC2.3.2.NFR7** Interoperability.

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.3.2_UC1 Configure FPGA-based IP accelerators	CPS/CPSoS Designer		Configures metrics (e.g. memory and computational throughputs).
TC2.3.2_UC2 Commission security agents/sensors to OpenCL stack	CPS/CPSoS Designer HW/SW component(s)	**TC2.2.2** pocl-remote (**TC2.2.2_UC2**)	Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware interface.

4.2.8 Model Transformation to OpenCL

Code: TC2.3.3	**Task:** T2.3, T3.6, T4.6	**Partner**: UoP/ISI	**Type**: Software	**TRL:** 5
Architectural block: CPSoS system layer		**Architectural sub-block:** Modelling and Redesign Engine (MRE)		
State-of-the-Art / Innovation:				

ML/DNN models will be automatically transformed to OpenCL kernels of different complexity levels. The transformation will be based on well know.			
Functional requirements		**Non-functional requirements**	
TC2.3.3.R1 Profiling **TC2.3.3.R2** HLS based SW to HW Transformation		**TC2.3.3.NFR1** Development of HW-SW Library with reliable Components.	
Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC2.3.1_UC1 Configure FPGA-based IP accelerators	CPS/CPSoS Designer		Configures metrics (e.g. memory and computational throughputs and quantization techniques of various levels).
TC2.3.1_UC2 Commission ML Hardware to OpenCL stack	CPS/CPSoS Designer HW/SW component(s)	**TC2.2.2** pocl-remote (**TC2.2.2_UC2**)	Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware/software interface visible to PoCL run-time system.

4.2.9 Xilinx XRT KPI Monitoring

Code: TC2.4.1	**Task:** T2.4, T3.6, T4.1	**Partner**: UoP	**Type**: Software	**TRL:** 6
Architectural block: CPSoS system layer		**Architectural sub-block:** Modelling and Redesign Engine (MRE)		
State-of-the-Art / Innovation: XRT FPGA monitoring services visible to remote nodes. Coordinated FPGA reconfiguration decisions at multiple CPS levels.				
Functional requirements		**Non-functional requirements**		
TC2.4.1.R1 Accelerate DNN inference in comparison to software running in ARM. (TC2.3.1.R1) **TC2.4.1.R2** HLS based SW to HW Transformation (TC5.1.1.R1) **TC2.4.1.R3** Commissioning: The component should be able to collect hardware bitstreams IP Cores and download them on the FPGA fabric of a Multiprocessor System on Chip FPGA board. (TC4.6.1.R1) **TC2.4.1.R4** Reconfigurability: The components should be able to reconfigure the commissioned hardware IP Cores on the FPGA fabric of a TC4.6.1.R3		**TC2.4.1.NFR1** Development of HW-SW Library with reliable Components (TC3.6.1.NFR1) **TC2.4.1.NFR2** The component should be able to handle efficiently the configuration updates and resolve any possible dependencies (TC4.6.1.NFR2) **TC2.4.1.NFR3** The component should be able to provide integrity validation method in both ends (e.g. hashes of the transferred payloads). (TC4.6.1.NFR3) **TC2.4.1.NFR4** The component should be aware of the commissioning process' status and handle failures (e.g. rollback to previous versions). (TC4.6.1.NFR4)		

Multiprocessor System on Chip FPGA board and replace existing hardware IP Cores. (TC4.6.1.R2) **TC2.4.1.R5** Removal: The component should be able to remove existing hardware IP Cores in the FPGA fabric of a Multiprocessor System on Chip (MPSoS) FPGA board. (TC4.6.1.R4) **TC2.4.1.R6** Accessibility: The component should be able to communicate with the model based design mechanism of the CPSoSaware layer in order to deploy hardware IP Cores in the MPSoC board. (TC4.6.1.R5)	

Component-level Use Cases

Name	Actor type	Actor (if SW/HW)	Interaction
TC2.4.1_UC1 Configure FPGA-based IP accelerators	CPS/CPSoS Designer		Configures metrics (e.g. memory and computational throughputs and quantization techniques of various levels). Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware interface.
TC2.4.1_UC2 Commission ML Hardware to OpenCL stack	CPS/CPSoS Designer HW/SW component(s)		Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware/software interface visible to PoCL run-time system. Deploys the FPGA IP accelerator to the OpenCL system software stack via a simple common hardware interface.

4.2.10 Modelling Orchestration Tool

The modelling orchestration tool captures the CPS overall, manages individual CPS inputs and outputs between other CPSs, and orchestrates the CPSoS components in order to achieve a model of models.

Code: TC2.5.1	**Tasks:** T2.5	**Partner**: 8BELLS	**Type**: Software	**TRL:** 4
Architectural block: CPSoS system layer		**Architectural sub-blocks:** Modelling and Redesign Engine (MRE)		
State-of-the-Art / Innovation: The evaluation phase so far of the orchestrator component has selected a generic component which can be further specified to the requirements of creating a decentralized and autonomous CPSoS model operation. Occupus[2] cloud orchestrator is the primary candidate to serve as the basis for the system. Furthermore,				

[2] Cloud agnostic orchestrator, flexible, allows for complex, constantly changing topologies.

Prometheus[3] is being considered as a data collection and distribution system between CPS running models at present.

Functional requirements	Non-functional requirements
TC2.5.1.R1 User-driven orchestration control events to initiate orchestration. **TC2.5.1.R2** Autonomic model-driven orchestration control events by models. **TC2.5.1.R3** Integrate existing CPS modelling-simulation tools.	**TC2.5.1.NFR1** Minimize centralized control of the orchestration. **TC2.5.1.NFR2** Reliable and secure autonomic operations.

Component-level Use Cases

Name	Actor type	Actor (if SW/HW)	Interaction
TC2.5.1_UC1 Integrate new CPS/CPHS	CPS/CPSoS Designer		Provide CPS model for a new CPS to be integrated in the CPSoS.
TC2.5.1_UC2 Associate CPS inputs/outputs			CPS-specific inputs/outputs are associated to other CPS inputs/outputs according to the provided models.

4.2.11 Visual Localization

This component will handle the robustification to GPS spoofing attacks: Create a database in which every record will include the descriptor of each image and the GPS coordinates. Given a set of images, a visual search is applied against a set of relevant geo-tagged images from the database. As a result, a ranked list of images is obtained sorted by descriptors distances. A weighted average of the GPS coordinates of the extracted descriptors yield a corrected GPS value for the query image, providing a coarse localization fix.

Code: TC3.1.1	**Tasks:** T3.1	**Partner**: ISI	**Type**: Software	**TRL:** 3

Architectural block:	**Architectural sub-blocks**:
CPSoS system layer	Security Runtime Monitoring and Management (SRMM)

State-of-the-Art / Innovation:

TC3.1.1 will act as complement to TC3.3.1, facilitating the vehicle to localize itself, without relying on other vehicles. Moreover, it will reduce the impact of GPS spoofing.

Functional requirements	Non-functional requirements
TC3.1.1.R1 Trajectory of vehicle generated by CARLA.	**TC3.1.1.NFR1** Minimize the computational time of visual search in the database.

[3] Input sensory data from CPS to drive autonomic functionality and de-centralized approach.

TC3.1.1.R2 Database of geo-tagged images available.			
Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.1.1_UC1 Configure visual localization	CPS/CPSoS Designer	**TC3.1.2** Deep Multimodal Scene Understanding (**TC3.1.2_UC2**)	Provides a set of geo-tagged images for a given environment/workspace and configures the visual localization.
TC3.1.1_UC2 Sense user location	HW/SW component(s)	**TC3.1.2** Deep Multimodal Scene Understanding (**TC3.1.2_UC2**)	Inputs a query image for TC3.1.1 to identify user's GPS coordinates.

4.2.12 Deep Multimodal Scene Understanding

The main objective of this module is to derive the semantic information within a given scene, namely, understanding a scene. This is the basis for autonomous driving, traffic safety, vision-guided manufacturing, or activity recognition. This module will deploy deep architectures to derive semantic information from a fusion of sensor data and the fusion of their semantic interpretation, since understanding a scene from an image or sequence of images requires more effort than simple feature extraction. RGB/Lidar, RGB/depth data will be deployed, and this module will include algorithms and deep architectures operating in distributed or centralized manner to define the operation of CPSs.

Code: TC3.1.2	**Tasks**: T3.1	**Partner**: ISI	**Type**: Software	**TRL**: 3
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Monitoring Engine		
State-of-the-Art / Innovation: Literature encompasses a multitude of approaches to achieve multimodal data fusion and scene understanding using RGBD, LIDAR, or a combination of the two. We aim to improve the current state-of-the-art proposing accelerated yet robust deep architectures that fuse data either in the input layer or in the deeper layers.				
Functional requirements		**Non-functional requirements**		
TC3.1.2.R1 Availability of RGBD and point cloud data. **TC3.1.2.R2** Camera mapping strategy and LIDAR processing approach for effective data fusion. **TC3.1.2.R3** Post-processing semantic analysis functionality.		**TC3.1.2.NFR1** Real-time execution. **TC3.1.2.NFR2** Efficient semantic representation to reduce required training data.		
Component-level Use Cases				

Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**

TC3.1.2_UC1 Produce high-level semantics	HW/SW component(s)	TC3.1.2 itself	Provide multimodal sensor data (RGB/Lidar, RGB/depth) to be analysed by Computer Vision/Deep Learning mechanisms and produce high-level observations/detections.
TC3.1.2_UC2 Semantic fusion of detections	HW/SW component(s)	TC3.1.2 itself	Detections deriving from TC.3.1.2_UC1 are fused in a domain-specific semantic schema.

4.2.13 User Behaviour Monitoring

The user behavioural monitoring will be based on CPSoSaware' s collaborative sensory multi-modal fusion mechanism and will be based on algorithms for physiological and behavioural monitoring that will facilitate the evaluation of cognitive load/situational awareness development of a smart sensing module to allow inertial and optical sensor fusion, providing 6DoF pose estimation, thus dealing with occlusions and drifts. The specificities of the algorithms will be defined by the system requirements and use cases.

Code: TC3.1.3	**Tasks**: T3.1	**Partner**: UPAT/ISI	**Type**: Software	**TRL**: 4
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Monitoring Engine (ME)		
State-of-the-Art / Innovation: In literature, there are a lot of implementations trying to evaluate the driver's drowsiness, however, most of them focus only on a specific measurement (direct or non-direct). We attend to improve the current state-of-the-art situation using a fusion scheme, integrating different architectures, and providing more accurate results.				
Functional requirements		**Non-functional requirements**		
TC3.1.3.R1 Pre-trained model of faces for the real-time face recognition and face tracking via markers. **TC3.1.3.R2** Continuously recording of the driver's face. **TC3.1.3.R3** Optimization of algorithms for running in real-time. **TC3.1.3.R4** Decision making based on the drowsiness level, indicating the appropriate warning signs. **TC3.1.3.R5** Continuous monitoring and recording of the driver's pulse rate from a wearable device.		**TC3.1.3.NFR1** Computational efficiency. **TC3.1.3.NFR2** Security of the driver. **TC3.1.3.NFR3** Maximization of the situational awareness. **TC3.1.3.NFR4** Robustness under different light conditions.		
Component-level Use Cases				

Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.1.3_UC1 Sense	HW/SW component(s)	**TC3.1.2** Deep Multimodal Scene	Retrieve the pose/posture characteristics of the user.

physiological condition of user		Understanding (**TC3.1.2_UC2**)	
TC3.1.3_UC2 Sense cognitive condition of user	HW/SW component(s)	**TC3.1.2** Deep Multimodal Scene Understanding (**TC3.1.2_UC2**)	Retrieve the estimation of the user's cognitive load/condition.

4.2.14 AI Acceleration

DCNNs achieve ground-breaking performance in a great variety of applications, including classification tasks such as object recognition. However, DCNNs are computationally expensive, meaning that they usually demand high-performance platforms for their implementation.

The goal is the study of DCNN acceleration / compression techniques for their effective implementation in embedded platforms, lower the computational cost (number of operations, storage requirements). With the least possible loss in accuracy. Specifically, our efforts are focused on pruning and sharing techniques:

- Can achieve considerable acceleration without significant performance loss
- Can be applied to pre-trained DCNNs.

These are orthogonal and could potentially be combined.

Code: TC3.1.4	**Tasks**: T3.1	**Partner**: ISI	**Type**: Software	**TRL**: 3
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Monitoring Engine		
State-of-the-Art / Innovation: Currently, simple vector quantization is employed. Here, we aim to employ novel vector quantization techniques that considerably increase the acceleration / storage gain by exploiting the theory of dictionary learning and sparse representations.				
Functional requirements		**Non-functional requirements**		
TC3.1.4.R1 Pre-trained DCNN availiable: Original DCNN model, pre-trained for the target application, available in ONNX, or Matlab, or TF format. **TC3.1.4.R2** Data availability: Training/validation dataset, for the target application, available for retraining/finetuning purposes. **TC3.1.4.R3** Accelerated DCNN runtime functionality: Availability of parameter-sharing enabled convolutional layer implementation.		**TC3.1.4.NFR1** Accelerated model accuracy within user specifications. **TC3.1.4.NFR2** Minimize accelerated DCNN model storage space. **TC3.1.4.NFR3** Minimize accelerated DCNN model inference time execution.		
Component-level Use Cases				

Name	Actor type	Actor (if SW/HW)	Interaction
TC3.1.4_UC1 Optimize AI performance	CPS/CPSoS Designer		Requests the acceleration of given DCNNs to perform on embedded platforms.

4.2.15 PoCL-accel

This is a generic OpenCL driver (for PoCL) to interface with custom devices (hardware accelerators) from the OpenCL API.

Code: TC3.2.1	**Tasks**: T3.2	**Partner**: TAU	**Type**:	**TRL**: 3
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: OpenCL Description Execution (ODE)		
State-of-the-Art / Innovation: Previously different IPs needed largely new drivers, with pocl-accel + AlmaIF integrating to a common OpenCL platform is made easier.				
Functional requirements		**Non-functional requirements**		
TC3.2.1.R1 Must be able to support at least OpenCL 1.2 based command queues on the AlmaIF.		**TC3.2.1.NFR1** Driver overhead less than 1%.		
Component-level Use Cases				

Name	Actor type	Actor (if SW/HW)	Interaction
TC3.2.1_UC1 Deploy driver	CPS/CPSoS Designer		Deploys the POCL driver and integrates with the OpenCL API.

4.2.16 Multimodal Localization API

This component will implement a software library (written mostly in Python Programming Language) of novel techniques for multi-modal localization. Combination of LiDAR data and angle of arrival/departure will be investigated for improved cooperative localization. The studied techniques will be implemented via distributed approaches.

Code: TC3.3.1	**Tasks**: T3.1, T3.3	**Partner**: ISI	**Type**: Software	**TRL**: 3
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Distributed, Cognitive and Cooperative Intelligence (DCCI)		

State-of-the-Art / Innovation:
Our aim is to study and develop both non-Bayesian and Bayesian cooperative localization and tracking methods. Moreover, we will focus on distributed implementations that increase the robustness and safety, but also converge to the centralized solution.

Functional requirements	**Non-functional requirements**
TC3.3.1.R1 Trajectories of vehicles : CARLA simulator will generate the trajectories of vehicles moving in a city. **TC3.3.1.R2** Measurements availability: It is assumed that absolute position and range measurements from GPS and LIDAR sensor will always be available. **TC3.3.1.R3** Cooperation: Multi-modal fusion will be performed in a collaborating manner, by representing the VANET as a graph.	**TC3.3.1.NFR1** Measurements degraded by Gaussian noise. **TC3.3.1.NFR2** Exchange of measurements and estimation of locations before the new GPS measurement.

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.3.1_UC1 Perform cooperative localization	HW/SW component(s)	**TC3.1.2** Deep Multimodal Scene Understanding (**TC3.1.2_UC2**)	Requests the production of localization data from the combination of measurements (e.g. GPS, LiDAR) for metrics like arrival/departure and trajectories.
TC3.3.1_UC2 Submit localization data for semantic fusion	HW/SW component(s)	**TC3.1.2** Deep Multimodal Scene Understanding (**TC3.1.2_UC2**)	The produced localization data are submitted for semantic fusion in the scene understanding module.

4.2.17 PathPlanning API

This component will implement a software library (written mostly in Python Programming Language) of novel techniques for collaborative path planning.

Code: TC3.3.2	**Tasks**: T3.1, T3.3	**Partner**: ISI	**Type**: Software	**TRL**: 3
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Distributed, Cognitive and Cooperative Intelligence (DCCI)		
State-of-the-Art / Innovation:				

Currently collaborative path planning is focused on centralized solutions. We aim for the study and the implementation of distributed techniques. This will increase the robustness, the safety, and the awareness levels of the CPS platform.

Functional requirements	**Non-functional requirements**
TC3.3.2.R1 Location Logging Mechanism: The component should be able to collect logs of the node's position. **TC3.3.2.R2** Control Error Logging Mechanism: The component should be able to collect logs related to the path planning control error. **TC3.3.2.R3** Execution Time: The component should be able to collect the measured time between update of sensor inputs till response to updated inputs for each node. **TC3.3.2.R4** Connectivity graph: The component should be able to store the nodes that are actively collaborating to optimize the path planning control. **TC3.3.2.R5** Awareness level: The component should be able to store the awareness level (SAL) metric.	**TC3.3.2.NFR1** Minimize centralized control. **TC3.3.2.NFR2** Minimize collision risk. **TC3.3.2.NFR3** Maximize fault tolerance. **TC3.3.2.NFR4** Maximize situational awareness.

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.3.2_UC1 Measure and store the position measurements fusing several sensors	HW/SW component(s)	**TC3.3.1** Multimodal Localization API	Provides position logs for a given node or set of nodes.
TC3.3.2_UC2 Obtain the required force (torque) for each vehicle to perserve the path	HW/SW component(s)	**TC3.3.2_UC2** itself	Requests the generation of paths between given nodes
TC3.3.2_UC3 Obtain the flow topology information	HW/SW component(s)	**TC3.3.2_UC3** Obtain the flow topology information	TC3.3.2 preserves a connectivity graph of the active nodes to optimize the path planning.

4.2.18 XR Tools for Increasing Situational Awareness

AR-based enhancement tools to improve the human in the loop awareness. The tools should facilitate the transfer of information (streams, reminders, or visual aids) to the user to improve focus on the current task, remember other parallel or scheduled tasks, improve response time, avoid imminent dangers or accident-related factors. More specifically, the AR methods will be used for increasing the situational awareness of the drivers in the following applications:

- Pothole detection approaches.
- Rendering of occluded objects that will be identified by cooperative localization methods.

<table>
<tr><td colspan="2">Code: TC3.4.1</td><td colspan="2">Tasks: T3.4</td><td colspan="2">Partner: UPAT</td><td colspan="2">Type: Software</td><td colspan="2">TRL: 3</td></tr>
<tr><td colspan="5">Architectural block:
CPS/CPHS layer</td><td colspan="5">Architectural sub-blocks:
Extended reality tools and interfaces</td></tr>
<tr><td colspan="10">State-of-the-Art / Innovation:
State-of-the-art geometrical analysis approaches will be used, applied to point clouds, in order to extract information from the captured scene (e.g., object detection). Consequently, this information will be visualized using AR-based technology, providing also warning signs to the drivers. Additionally, coalition information of other neighbouring vehicles will be used to highlight occluded or partially observed moving objects like other vehicles or pedestrians.</td></tr>
<tr><td colspan="5">Functional requirements</td><td colspan="5">Non-functional requirements</td></tr>
<tr><td colspan="5">The AR situational awareness application should be able to provide:
TC3.4.1.R1 Information streams regarding the task underway improving focus,
TC3.4.1.R2 Personalized reminders regarding other parallel or scheduled tasks significantly improving response time,
TC3.4.1.R3 Notifications and visual aids regarding imminent dangers or accident-related factors (e.g., pothole and obstacle detection),
TC3.4.1.R4 KPIs visualizing the effectiveness of the CPSoS functionality,
TC3.4.1.R5 Cooperative situational awareness. Visualization and use of coalition information provided by other vehicles or interactive robots (e.g., highlighting of occluded vehicles and pedestrians).</td><td colspan="5">TC3.4.1.NFR1 Computational efficiency (real-time).
TC3.4.1.NFR2 User-friendly interface.
TC3.4.1.NFR3 Reliability and robustness of the provided aware sign.
TC3.4.1.NFR4 Improve situational awareness without disturbing the user's attention.
TC3.4.1.NFR5 Provide only useful information based on personalized user's preferences.</td></tr>
<tr><td colspan="10">Component-level Use Cases</td></tr>
<tr><td colspan="2">Name</td><td colspan="2">Actor type</td><td colspan="2">Actor (if SW/HW)</td><td colspan="4">Interaction</td></tr>
</table>

TC3.4.1_UC1 View detected potholes	End-User		The end-user (driver) reviews detected potholes using AR-enabled technologies.
TC3.4.1_UC2 View occluded objects	End-User		The end-user (driver) reviews rendered occluded objects using AR-enabled technologies.

4.2.19 CPS Layer Security Sensors/Agents

CPS layer Security sensors/agents that collect security related data and pre-process them before transmitting them to the CPSoSaware SRMM at the system layer.

Code: TC3.5.1	**Tasks**: T3.5	**Partner**: USI/ISI/ATOS	**Type**: Hardware & Software	**TRL**: 4
Architectural block: CPS/CPHS layer		**Architectural sub-blocks**: Distributed, Cognitive and Cooperative Intelligence (DCCI)		
State-of-the-Art / Innovation: Recent research is focused mainly on software agents deployed in cybersecurity devices. There are few works that introduce a hardware-software co-design approach to increase the performance of the security agent by using hardware acceleration of cryptography/security primitives.				
Functional requirements		**Non-functional requirements**		
TC3.5.1.R1 Logging Mechanism: The component should be able to collect logs of event that take place in a CPS platform. **TC3.5.1.R2** Data Integrity: The component should be able to ensure integrity of collected data that are forwarded to the CPSoSaware Runtime Monitoring System. **TC3.5.1.R3** Data Authenticity: The component should be able to ensure authenticity of collected data that are forwarded to the CPSoSaware Runtime Monitoring System. **TC3.5.1.R4** Detectability: The component should be able to detect simple anomalous events in the CPS system (e.g. related to false data injection, security attacks on the device and CPS network issues). **TC3.5.1.R5** Secure channel communication: The component should be able to transmit in a secure and trusted way the collected logs to the CPSoSaware Runtime monitoring system. This can be managed through end to end secure communication.		**TC3.5.1.NFR1** Efficiency (response time). **TC3.5.1.NFR2** Efficiency (constrained memory and chip covered area resources). **TC3.5.1.NFR3** Flexibility so that sensor components can be updated dynamically. **TC3.5.1.NFR4** Interoperability so that sensors can be used in various CPSs and both pilots. **TC3.5.1.NFR5** Trusted computation following security by design approach and use of trusted execution environments.		

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.5.1_UC1 Collect security data	HW/SW component(s)	Security sensors and agents	Security agents/sensors provide security-related data.
TC3.5.1_UC2 Pre-process and normalize security data			TC3.5.1 pre-processes collected security data and normalizes format.
TC3.5.1_UC3 Transmit security data	HW/SW component(s)	**TC4.3.1** Security Runtime Monitoring	TC3.5.1 provides the pre-processed security data to the Security Runtime Monitoring (SRMM).

4.2.20 TCE (openasip.org) Soft Cores

Customized processors designed using TTA-Based Co-design Environment (TCE), an open source application-specific instruction set toolset based on the transport-triggered architecture (TTA). Various hardening features can be added via replication of functionality and special instructions.

Code: TC3.6.1	**Tasks**: T3.6	**Partner**: TAU	**Type**: Hardware & Software	**TRL**: 3-5
Architectural block: CPSoS system layer		**Architectural sub-blocks**: HW/SW Component Library		
State-of-the-Art / Innovation: Other ASIP tools are commercial / closed source and utilize traditional operation-triggered architectures, while TCE has an open source repository available, and the used TTA template provably improves also soft core usage. The tools have various parts which are under development and in different TRLs.				
Functional requirements		**Non-functional requirements**		
TC3.6.1.R1 Programmable co-processor for cases where hardware customization is useful, but runtime programmability is needed. **TC3.6.1.R2** Ability to execute at least two different tasks defined by switching the software binary only.		**TC3.6.1.NFR1** Performance requirements are task/application specific. Overall, acceleration or improved energy-efficiency over similar software on a general-purpose processor is required to justify an ASIP.		

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC3.6.1_UC1 Customize	CPS/CPSoS Designer		Customizes processor using TCE

processor architecture			
TC3.6.1_UC2 Evaluate customized processor	CPS/CPSoS Designer		Evaluates customized processor statistics

4.2.21 OpenCL Wrapper for Hardware IP Cores

OpenCL kernel description interface to associate Hardware IP cores with the OpenCL models.

Code: TC4.1.1	**Tasks**: T4.1	**Partner**: TAU	**Type**: Hardware & Software	**TRL**: 3
Architectural block: CPSoS system layer		**Architectural sub-blocks**: HW/SW Component Library		
State-of-the-Art / Innovation: Previously different IPs needed largely new drivers, with pocl-accel and AlmaIF integrating to a common OpenCL platform is made easier.				
Functional requirements		**Non-functional requirements**		
TC4.1.1.R1 Ability to easily add IPs and co-processors to OpenCL platforms that are orchestrated from a single OpenCL runtime.		**TC4.1.1.NFR1** The implementation overhead of the wrapper should be less than 1% of the wrapped design.		
Component-level Use Cases				
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**	
TC4.1.1_UC1 Associate HW IP cores to OpenCL models	CPS/CPSoS Designer HW/SW component(s)	**TC2.3.1** ML Hardware Accelerator IP Cores	CPS/CPSoS designer associates OpenCL models to HW IP cores from TC2.3.1	

4.2.22 HW/SW profiling and analysis based on Vitis Tools

Profiling for a highly heterogeneous platform consisting of multicore ARM processor, ASIP processors as well as FPGA fixed logic IP. FPGA logic is a “morphable” computation resource without predefined computational capabilities. All SW nodes will be handled by PoCL, enabling dynamic remapping and re-scheduling opportunities.

Code: TC4.1.2	**Tasks**: T4.1	**Partner**: UoP	**Type**: Hardware and Software	**TRL**: 5
Architectural block: CPSoS system layer		**Architectural sub-blocks**: Modelling and Redesign Engine (MRE)		

State-of-the-Art / Innovation:
Xilinx profiling framework, based on Vitis, taking as input a set of profiling parameters extracted by LLVM framework by implementing a new set of LLVM API calls.

Functional requirements	Non-functional requirements
TC4.1.2.R1 HLS based SW to HW Transformation (TC5.1.1.R1) **TC4.1.2.R2** Commissioning: The component should be able to collect hardware bitstreams IP Cores and download them on the FPGA fabric of a Multiprocessor System on Chip FPGA board. (TC4.6.1.R1) **TC4.1.2.R3** Reconfigurability: The components should be able to reconfigure the commissioned hardware IP Cores on the FPGA fabric of a (TC4.6.1.R2) **TC4.1.2.R4** Multiprocessor System on Chip FPGA board and replace existing hardware IP Cores. (TC4.6.1.R3) **TC4.1.2.R5** Removal: The component should be able to remove existing hardware IP Cores in the FPGA fabric of a Multiprocessor System on Chip (MPSoS) FPGA board. (TC4.6.1.R4) **TC4.1.2.R6** Accessibility: The component should be able to communicate with the model based design mechanism of the CPSoSaware layer in order to deploy hardware IP Cores in the MPSoC board. (TC4.6.1.R5) **TC4.1.2.R7** IP Core Software Support: The component should be able to deploy appropriate software driver components on the runtime system (embedded OS or bare metal API) been executed on a MPSoC FPGA board so that hardware IP Cores are accessible. Support for POCL tool could be offered. (TC4.6.1.R6) **TC4.1.2.R8** Accelerate DNN inference in comparison to software running in ARM. (TC2.3.1.R1)	**TC4.1.2.NFR1** Reliability and robustness of the suggested assembly steps. (TC5.3.1.NFR3) **TC4.1.2.NFR2** Programmable co-processor for cases where hardware. (TC3.6.1.R1) customization is useful, but runtime programmability is needed. **TC4.1.2.NFR3** Performance requirements are task/application specific. Overall, acceleration or improved energy-efficiency over similar software on a general-purpose processor is required to justify an ASIP. (TC3.6.1.NFR1)

Component-level Use Cases

Name	Actor type	Actor (if SW/HW)	Interaction
TC4.1.2_UC1 Commission HW component at the System layer (TC4.6.1_UC1)	CPS/CPSoS Designer		Input: LLVM generated statistics Output: Library of SW/HW components
TC4.1.2_UC2 Commission HW component at the	CPS/CPSoS Designer		Input: LLVM generated statistics Output: Library of SW/HW components

CPS layer (TC4.6.1_UC2)			

4.2.23 Architecture Optimization

This component aims to provide all necessary optimizations in order to reconfigure and redesign the System's CPSs/CPHSs so as to holistically match the systemic design and operational goals/parameters achieving reliability, robustness, responsiveness, CPS/CPHS criticality, energy efficiency, and security/trust.

Code: TC4.1.3	**Tasks**: T4.1	**Partner**: IBM	**Type**: Software	**TRL**: 3

Architectural block: CPSoS system layer	**Architectural sub-blocks**: Modelling and Redesign Engine (MRE)
State-of-the-Art / Innovation: IBM Architecture Optimization Workbench (AOW)	
Functional requirements	**Non-functional requirements**
TC4.1.3.R1 Input. The component must handle input in a mathematical optimization format, providing the necessary abstractions to model (with decision variables) CPSs/CPHSs including both hardware and software components and their connections. **TC4.1.3.R2** Objective. The component should be capable of optimizing a variety of objective functions. This does not include simultaneous multiple objectives (Pareto front). **TC4.1.3.R3** Constraints. The component must be able to handle connection, application, and resource constraints. **TC4.1.4.R4** Output. The component should produce as output a hardware-software partitioning that is optimal according to the specified mathematical optimization problem.	**TC4.1.3.NFR1** Efficiency (response time) **TC4.1.3.NFR2** Efficiency (optimality) **TC4.1.3.NFR3** Feasibility of solution

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC4.1.3_UC1 Schedule Software tasks on Hardware	CPS/CPSoS Designer		CPS/CPSoS designer creates policies for optimal partitioning of software tasks on hardware components of CPSs/CPHSs

4.2.24 Intra-Communication Manager

On one hand mechanisms to supervise a running network configuration in a real deployment. The metrics that reflect the application requirements will be monitored to provide feedback on whether the application requirements are met. Feedback will be extracted as a structured file by the end of each experiment on real deployments. The extracted feedback file will be used for further optimization during the simulation time. On the other hand, mechanisms allowing the reception of new network interface firmware or/and configuration file and application of these on the embedded platform.

<table>
<tr><td>Code: TC4.2.1</td><td>Tasks: T4.2</td><td>Partner: UoP</td><td>Type: Software</td><td>TRL: 4</td></tr>
<tr><td colspan="2">Architectural block:
CPS/CPHS layer</td><td colspan="3">Architectural sub-blocks:
CPSoSaware Intra-CPS Communication Layer</td></tr>
<tr><td colspan="5">State-of-the-Art / Innovation:
Performance evaluation during runtime. Reconfiguration of network parameters during runtime.</td></tr>
<tr><td colspan="2">Functional requirements</td><td colspan="3">Non-functional requirements</td></tr>
<tr><td colspan="2">TC4.2.1.R1 SW agents running on the HW platform monitor the network performance under the current network configuration for specific application scenario.
TC4.2.1.R2 The performance outcome is processed in order to evaluate whether the application requirements are met.
TC4.2.1.R3 SW agent running on the HW is responsible to receive new network configuration and/or network interface firmware to apply on the device.</td><td colspan="3">TC4.2.1.NFR1 The device should be able to recover from failing network firmware/configuration update.
TC4.2.1.NFR2 SW agent should be able to verify the integrity of the received payloads.
TC4.2.1.NFR3 Versioning of the applied configurations should be supported.
TC4.2.1.NFR4 Authentication/Authorization for receiving configuration updates.</td></tr>
<tr><td colspan="5">Component-level Use Cases</td></tr>
<tr><td>Name</td><td>Actor type</td><td>Actor (if SW/HW)</td><td colspan="2">Interaction</td></tr>
<tr><td>TC4.2.1_UC1
Define network metrics</td><td>Analyst</td><td></td><td colspan="2">Defines the network-related metrics that correspond to the application requirements.</td></tr>
<tr><td>TC4.2.1_UC2
Export feedback file</td><td>Analyst</td><td></td><td colspan="2">Generates a structured file of metrics/values after an experiment.</td></tr>
<tr><td>UC4.2.1_UC3
Apply new network interface</td><td>CPS/CPSoS Designer</td><td></td><td colspan="2">Feeds TC4.2.1 with new network interface firmware and/or config file.</td></tr>
</table>

4.2.25 Security Runtime Monitoring

The Security Runtime Monitoring will be based on the ATOS XL-SIEM which will be receiving events from security sensors deployed in the infrastructure, normalizing them and using them to generate alerts based on a set of correlation rules.

<table>
<tr><td>Code: TC4.3.1</td><td>Tasks: T4.3</td><td>Partner: ATOS</td><td>Type: Software</td><td>TRL: 7</td></tr>
<tr><td colspan="2">Architectural block:
CPSoS system layer</td><td colspan="3">Architectural sub-blocks:
Security Runtime Monitoring and Management (SRMM)</td></tr>
<tr><td colspan="5">State-of-the-Art / Innovation:
Innovation 1 - collect and process security events from new data sources (CPS level, Communication layer)
Innovation 2 - detect new threats and attacks from the specific landscape considered in CPSoS: automotive and manufacturing.
Innovation 3 - adapt to the HW/SW and deployment requirements of the CPSoS architecture.</td></tr>
<tr><td colspan="2">Functional requirements</td><td colspan="3">Non-functional requirements</td></tr>
<tr><td colspan="2">TC4.3.1.R1 Input: The component must receive normalized security events through TCP/41000 from agents/sensors deployed remotely, in the infrastructure that is under surveillance. Events comply with a predefined JSON format.
TC4.3.1.R2 Configuration: The component should be configured using the component's graphical dashboard, to define the security monitoring infrastructure in use (topology of sensors/agents deployed and active), the security detection rules and the correlation directives.
TC4.3.1.R3 Events Processing: The component must process security events received as input, correlate them using the security detection rules configured, and generate security alarms as ouput, as defined in the correlation directives configured.
TC4.3.1.R4 Output: The component should produce as output security alarms. Alarms comply with a predefined JSON format. Alarms can be configured to be persisted in a DB, logged into a file, transmitted to a third-party component (using a middleware such as Message Queue/Broker) and displayed in the SRMM graphical dashboard.
TC4.3.1.R5 Cross-correlation: Security alarms produced as output by the SRMM can be configured to be input into the SRMM correlation engine, for cross-correlation processes.</td><td colspan="3">TC4.3.1.NFR1 Scalability: of the SRMM correlation engine and data collection module
TC4.3.1.NFR2 High-performance: of the SRMM correlation engine and the data persistence layer
TC4.3.1.NFR3 Integrity: of the security events transmitted from sensors/agents to the SRMM component, and of the security alarms generated as output by the SRMM
TC4.3.1.NFR4 Confidentiality: of the security events transmitted from sensors/agents to the SRMM component, and of the security alarms generated as output by the SRMM
TC4.3.1.NFR5 Accountability: of the security events transmitted from sensors/agents to the SRMM component, of the correlation process and of the security alarms generated as output by the SRMM</td></tr>
</table>

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC4.3.1_UC1 Configure security monitoring and detection rules	Analyst		Defines the topology of the infrastructure to be monitored and configures the security detection rules.
TC4.3.1_UC2 Receive security events	HW/SW component(s)	Security sensors and agents	Security agents/sensors provide security-related events.
TC4.3.1_UC3 Analyse and Correlate security events	HW/SW component(s)		TC4.3.1 correlates security events according to the configured security rules.
TC4.3.1_UC4 Generate alerts	HW/SW component(s) / Analyst	RabbitMQ queue	Transmits security alerts in JSON format and displays alerts in the Graphical Dashboard.

4.2.26 V2X Simulator

First implementation of V2X simulator is a simulator based on OMNeT++, Vanetza, and SUMO modules. It implements IEEE 802.11p and LTE C-V2X Mode 4. It can represent realistic scenarios based on OpenStreetMaps.

Code: TC4.4.1	**Tasks**: T4.4	**Partner**: i2CAT/ROBOTEC	**Type**: Software	**TRL**: 4
Architectural block: Simulation and training		**Architectural sub-blocks**: -		
Functional requirements		**Non-functional requirements**		
TC4.4.1.R1 ROS/ROS2 interface. **TC4.4.1.R2** V2X representation of state in AV simulator.		**TC4.4.1.NFR1** Possibility of running in real time. **TC4.4.1.NFR2** Modular architecture integrated in simulation framework.		

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC4.4.1_UC1 Receive data from central simulator	HW/SW component(s)	**TC4.4.3** AV Simulator	Provide scene configuration, vehicles' data (location, velocity, etc.)

TC4.4.1_UC2 Perform simulation			TC4.4.1 performs simulation
TC4.4.1_UC3 Generate traces	HW/SW component(s)		Perception, location & velocity data

Another module for V2X simulation is developed based on OMNeT++ network simulator, but without using SUMO. The communication with AV Simulator will be done using ROS/ROS2 interface directly. Thanks to such approach, V2X module can work with any AV Simulator with ROS/ROS2 communication implemented.

Code: TC4.4.1	**Tasks**: T4.4	**Partner**: ROBOTEC	**Type**: Software	**TRL**: 4
Architectural block: Simulation and training		**Architectural sub-blocks**: -		
State-of-the-Art / Innovation: Developed simulation module will be compatible with any AV simulator using ROS/ROS2 communication. Current solutions (OMNeT++) cannot be directly integrated with AV Simulators, without intermediate modules.				
Functional requirements		**Non-functional requirements**		
TC4.4.1.R1 ROS/ROS2 interface. **TC4.4.1.R2** V2X representation of state in AV simulator.		**TC4.4.1.NFR1** Possibility of running in real time. **TC4.4.1.NFR2** Modular architecture integrated in simulation framework.		
Component-level Use Cases				
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**	
TC4.4.1_UC1 Working as a submodule of central simulator	HW/SW component(s)	**TC4.4.3** AV Simulator	Getting scene configuration, vehicles' data (location, velocity, etc.) from AV simulator, sending back V2X messages received by each trafic agent	

4.2.27 Manufacturing Environment Simulation

This component is a simulator based on one of the available solutions (Gazebo, Nvidia Isaac) with additional CPSOSaware related modules enabling advanced simulation of all scenarios and integration with other simulations (factory sensors, human behaviour modelling, cybersecurity).

Code: TC4.4.2	**Tasks**: T4.4	**Partner**: ROBOTEC	**Type**: Software	**TRL**: 4
Architectural block: Simulation and training		**Architectural sub-blocks**: -		

Functional requirements	Non-functional requirements
TC4.4.2.R1 Machine learning support. **TC4.4.2.R2** Possibility of modelling additional elements of use case scenarios: humans, light curtain, safety eye, etc. **TC4.4.2.R3** Available models of robotic arms used in CRF factory. **TC4.4.2.R4** Integration with middleware: Simulation solution should offer integration with state-of-the-art robotics middleware (e.g. ROS and ROS2). **TC4.4.2.R5** User control: Simulation should allow users to control all critical aspects in the simulation through dedicated API (e.g. agents behaviour or sensors).	**TC4.4.2.NFR1** Fast execution - Software should offer a fast execution simulation for which graphics are not required. **TC4.4.2.NFR2** Diagnostic and Error Handling - Simulation should offer diagnostic and error handling **TC4.4.2.NFR3** Determinism - Simulation should ensure determinism. **TC4.4.2.NFR4** Modular System Architecture - Simulation should have modular system architecture.

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC4.4.2_UC1 Receive input	End-User		Provides vehicle models, control algorithms, predefined control scenarios
TC4.4.2_UC2 Run simulation			TC4.4.2 performs simulation
TC4.4.2_UC3 Generate training data	End-User		TC4.4.2 generates datasets for perception & reports of scenario validation

4.2.28 AV Simulation

This is a simulator based on one of the available open-source solutions with additional CPSoSaware-related modules, enabling advanced simulation of all scenarios and integration with other simulations (V2X, HIL, cybersecurity, DMS, etc.). Simulation of sensors, cyberattacks, communication with vehicles and infrastructure.

Code: TC4.4.3	**Tasks**: T4.4	**Partner**: ROBOTEC	**Type**: Software	**TRL**: 4
Architectural block: Simulation and training		**Architectural sub-blocks**: -		
State-of-the-Art / Innovation: AV Simulator will be used for validation all automotive related use cases. The AV simulator will be used as central module for related scenarios simulations. Important requirements are possible simulation of sensors, control of multiple agents and possible integration with other simulators.				

Functional requirements	Non-functional requirements
TC4.4.3.R1 Machine learning support for perception algorithms. **TC4.4.3.R2** User control: Simulation should allow users to control all critical aspects in the simulation through dedicated API (e.g. agents behaviour or sensors). **TC4.4.3.R3** Integration with middleware: Simulation solution should offer integration with state-of-the-art robotics middleware (e.g. ROS and ROS2).	**TC4.4.3.NFR1** Simple way of defining test scenarios. **TC4.4.3.NFR2** Scalability to multiple agent control - Simulation should provide multiple clients that can control different actors. **TC4.4.3.NFR3** Scalability to cloud services - Simulation should be able to run on scalable cloud services to run. multiple simulation scenarios (e.g. Google Cloud, Microsoft Azure or other). **TC4.4.3.NFR4** Fast execution: Software should offer a fast execution simulation for which graphics are not required. **TC4.4.3.NFR5** Diagnostic and Error Handling - Simulation should offer diagnostic and error handling. **TC4.4.3.NFR6** Determinism - Simulation should ensure determinism. **TC4.4.3.NFR7** Modular System Architecture - Simulation should have modular system architecture.

Component-level Use Cases			
Name	**Actor type**	**Actor (if SW/HW)**	**Interaction**
TC4.4.3_UC1 Receive input	End-User		Provide human behaviour models, predefined control scenarios
TC4.4.3_UC2 Run simulation			TC4.4.3 performs simulation
TC4.4.3_UC3 Generate training data	End-User		TC4.4.3 generates datasets for perception & reports of scenario validation

4.2.29 Commissioning of Hardware Components in CPSs

The Developed Hardware components after HW/SW partitioning will need to be deployed in the CPS. We focus on the dedicated HW accelerator components designed in other tasks and we aim at structuring the deployment/commissioning mechanism in the CPS SoC FPGA Fabric. In T4.6 we will focus on the commissioning mechanism from the system layer perspective while in task T5.2 we will focus on the commissioning mechanism infrastructure (support) at the CPS layer (in each CPS).

Code: TC4.6.1	**Tasks**: T4.6, T5.2	**Partner**: UoP/IBM	**Type**: Software	**TRL**: 4
Architectural block: CPSoS system layer		**Architectural sub-blocks**:		

	CPS Commissioning and CPS to System Inter-Communication Layer components
State-of-the-Art / Innovation: To the best of our knowledge there is poor work in automated commissioning processes for cyber physical systems that consider software and hardware components as well.	
Functional requirements	**Non-functional requirements**
TC4.6.1.R1 Commissioning: The component should be able to collect hardware bitstreams IP Cores and download them on the FPGA fabric of a Multiprocessor System on Chip FPGA board. **TC4.6.1.R2** Reconfigurability: The components should be able to reconfigure the commissioned hardware IP Cores on the FPGA fabric of a TC4.6.1.R3 Multiprocessor System on Chip FPGA board and replace existing hardware IP Cores. **TC4.6.1.R4** Removal: The component should be able to remove existing hardware IP Cores in the FPGA fabric of a Multiprocessor System on Chip (MPSoS) FPGA board. **TC4.6.1.R5** Accessibility: The component should be able to communicate with the model-based design mechanism of the CPSoSaware layer in order to deploy hardware IP Cores in the MPSoC board. **TC4.6.1.R6** IP Core Software Support: The component should be able to deploy appropriate software driver components on the runtime system (embedded OS or bare metal API) been executed on a MPSoC FPGA board so that hardware IP Cores are accessible. Support for PoCL tool could be offered.	**TC4.6.1.NFR1** The component should be able to validate that connectivity exists and recover from possible network failures. **TC4.6.1.NFR2** The component should be able to handle efficiently the configuration updates and resolve any possible dependencies. **TC4.6.1.NFR3** The component should be able to provide integrity validation method in both ends (e.g. hashes of the transferred payloads). **TC4.6.1.NFR4** The component should be aware of the commissioning process' status and handle failures (e.g. rollback to previous versions).

Component-level Use Cases

Name	Actor type	Actor (if SW/HW)	Interaction
TC4.6.1_UC1 Commission HW component at the System layer	CPS/CPSoS Designer		Input: HW accelerator bitstreams to be deployed Output: CPS deployed HW executable
TC4.6.1_UC2 Commission HW component at the CPS layer	CPS/CPSoS Designer		"Input: HW accelerator bitstreams to be deployed Output: CPS deployed HW executable"

4.2.30 HLS based SW to HW Transformation

HLS based synthesized HW components with PoCL compatible interfaces.

<table>
<tr><td>Code: TC5.1.1</td><td>Tasks: T5.1</td><td>Partner: UoP/ISI</td><td>Type: Hardware & Software</td><td>TRL: 4</td></tr>
<tr><td colspan="2">Architectural block:
CPSoS system layer</td><td colspan="3">Architectural sub-blocks:
Modelling and Redesign Engine (MRE)</td></tr>
<tr><td colspan="5">State-of-the-Art / Innovation:
HLS based synthesized HW components with PoCL compatible interfaces.</td></tr>
<tr><td colspan="2">Functional requirements</td><td colspan="3">Non-functional requirements</td></tr>
<tr><td colspan="2">TC5.1.1.R1 Profiling (TC4.1.2.R1)
TC5.1.1.R2 Commissioning of Hardware Components in CPSs (TC4.6.1.R1)
TC5.1.1.R3 Reconfigurability (TC4.6.1.R2)
TC5.1.1.R4 IP Core Software Support (TC4.6.1.R6)
TC5.1.1.R5 ML Hardware Accelerator IP Cores (TC2.3.1)
TC5.1.1.R6 Accelerate DNN inference in comparison to software running in ARM. (TC2.3.1.R1)
TC5.1.1.R7 Provide access to all OpenCL-supported devices in a network distributed platform from a single host application. (TC2.2.2.R1)</td><td colspan="3">TC5.1.1.NFR1 Development of HW-SW Library with reliable Components. (TC3.6.1.NFR1)
TC5.1.1.NFR2 Performance requirements are task/application specific. Overall, acceleration or improved energy-efficiency over similar software on a general-purpose processor is required to justify an ASIC. (TC3.6.1.NFR1)</td></tr>
<tr><td colspan="5">Component-level Use Cases</td></tr>
<tr><td>Name</td><td>Actor type</td><td>Actor (if SW/HW)</td><td colspan="2">Interaction</td></tr>
<tr><td>TC5.1.1_UC1
Commission HW component at the System layer (TC4.6.1_UC1)</td><td>CPS/CPSoS Designer</td><td></td><td colspan="2">Input: HW accelerator bitstreams to be deployed
Output: CPS deployed HW executable</td></tr>
<tr><td>TC5.1.1_UC2
Commission HW component at the CPS layer (TC4.6.1_UC2)</td><td>CPS/CPSoS Designer</td><td></td><td colspan="2">"Input: HW accelerator bitstreams to be deployed
Output: CPS deployed HW executable"</td></tr>
</table>

4.2.31 Extended Reality lifelong learning tools/Interfaces for integrated CPSoS

An AR-based CPHS user training toolkit will be developed so as to help the user adapt to changes in the environment and the dynamic CPSoS, whether these may concern a new machine that is added in the system or some new task process. Users often encounter strong outer constraints such as time or occupation, thus more immersive technologies aim to better exploit the uniqueness of AR and designing more effective virtual environments to improve the learning process. Virtual training scenarios will cover a broad range of user-desired activities.

<table>
<tr><td>Code: TC5.3.1</td><td>Tasks: T5.3</td><td>Partner: UPAT</td><td>Type: Software</td><td>TRL: 1</td></tr>
<tr><td colspan="2">Architectural block:
CPS/CPHS layer</td><td colspan="3">Architectural sub-blocks:
Extended reality tools and interfaces</td></tr>
<tr><td colspan="5">State-of-the-Art / Innovation:
AR technologies will be used to support on-site learning by giving contextually relevant and personally adapted guidance to the user. On-site learning has a close connection to knowledge sharing as learning can be supported both by formal guidance and knowledge shared by peers. The AR tools and virtual workplace simulations make guidance lively and engaging. AR-based life-long learning tools.</td></tr>
<tr><td colspan="2">Functional requirements</td><td colspan="3">Non-functional requirements</td></tr>
<tr><td colspan="2">The AR technology, as a learning tool, should be able to:
TC5.3.1.R1 Involve gamification of learning which makes the process fun and interactive.
TC5.3.1.R2 Provide visual cues in a distraction-free environment which helps the users to better understand the concepts.
TC5.3.1.R3 The technologies come with intelligent learning content and provide real-time responses.
TC5.3.1.R4 The trainee can easily accomplish the mapping between the training and the real task and is also able to access additional training material or information about the virtual objects.
TC5.3.1.R5 AR can support assembly tasks in hybrid human-machine manufacturing lines, improving efficiency and ergonomics, and reduce costs in industrial environments that require manual assembly operations.</td><td colspan="3">TC5.3.1.NFR1 Computational efficiency (real-time).
TC5.3.1.NFR2 User-friendly interface.
TC5.3.1.NFR3 Reliability and robustness of the suggested assembly steps.
TC5.3.1.NFR4 Improve the learning procedure without disturbing the user's attention.
TC5.3.1.NFR5 Provide only such type of help and instructions based on personalized user's preferences.</td></tr>
<tr><td colspan="5">Component-level Use Cases</td></tr>
<tr><td>Name</td><td>Actor type</td><td>Actor (if SW/HW)</td><td colspan="2">Interaction</td></tr>
<tr><td>TC5.3.1_UC1
Execute on-site learning scenario</td><td>End-user</td><td></td><td colspan="2">Eye gaze, gestures, physiological signals etc.</td></tr>
<tr><td>TC5.3.1_UC2
Execute off-site learning scenario</td><td>End-user</td><td></td><td colspan="2">Eye gaze, gestures, physiological signals etc.</td></tr>
</table>

TC5.3.1_UC3 Build Personal training plan	End-user		Personalized information related to user.
TC5.3.1_UC4 Execute collaborative learning scenario	End-user		Collaborative information from instructor and/or other colleagues.
TC5.3.1_UC5 Generate recommendations	Analyst / End-user		Retrieve recommendations based on trainee performance.

5 CPSoSaware System Architecture – Preliminary Version

The analysis of the collected information, as presented in Section 4, along with architecture descriptions deriving from the DoA (**Figure 3**), has led to the identification and refinement of architectural blocks and sub-blocks for the efficient conceptualization of the overall architecture and the classification of technical components into interconnected, sensible and coherent groups. At this design stage, technical components are treated as *black boxes*, meaning that their internal functionality and architecture are not yet taken into consideration. However, the progress in the definition of pilot use cases, technical requirements, and system-level use cases will feed the architecture designing process at the next stages of the project.

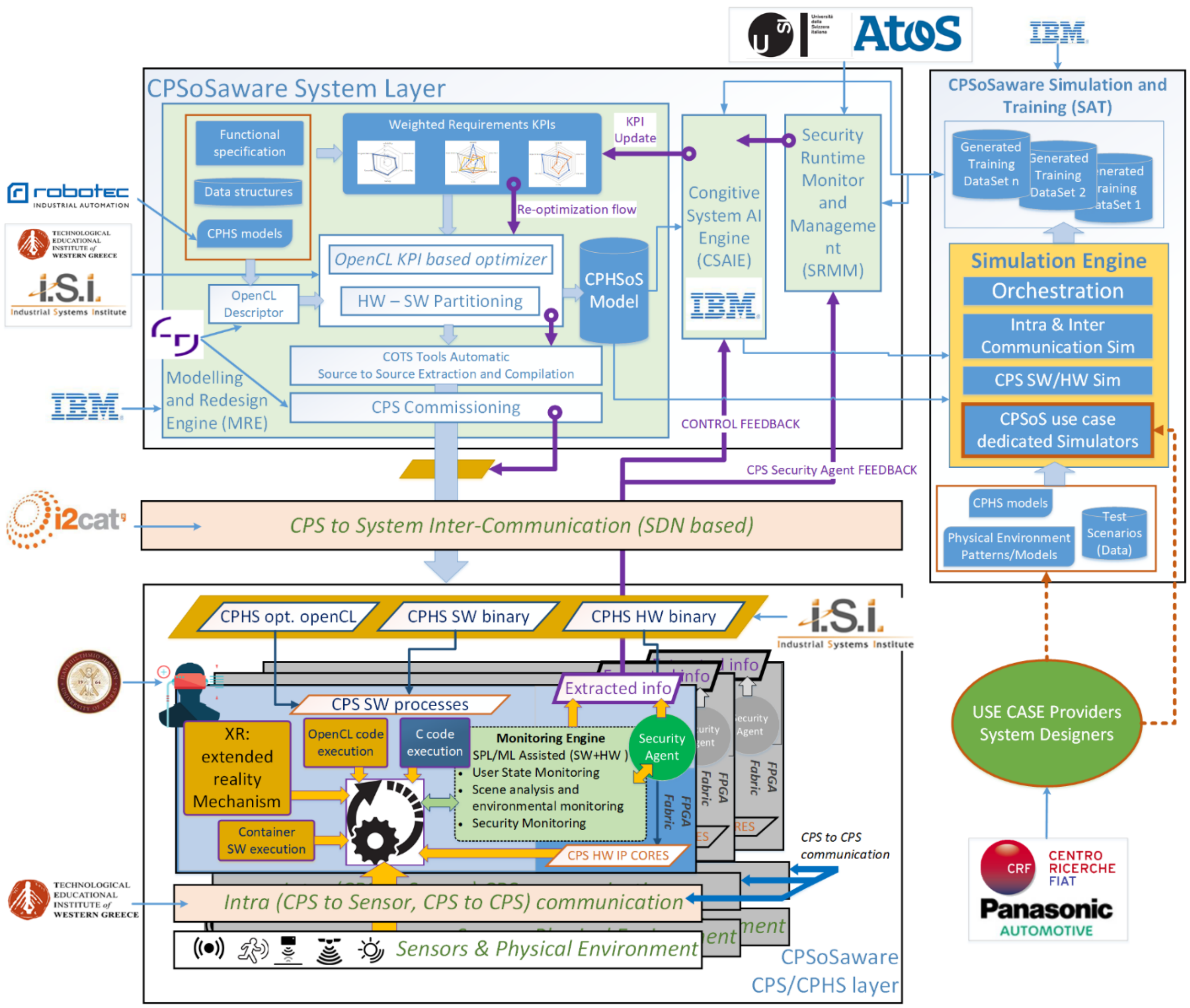

Figure 3. CPSoSaware proposed architecture

This section shortly presents the defined architectural blocks and a preliminary version of the architecture that includes the - so far - identified interconnections.

5.1 Architectural Blocks

5.1.1 CPSoS System Layer

This layer serves the modelling, configuration, redesign, evaluation, optimization, commission, communication, and orchestration of CPS/CPHs in the CPSoS of discourse. Included sub-blocks are:

- **Security Runtime Monitoring and Management (SRMM)**: This block contains components aimed at providing security awareness to CPSoSaware. By deploying anomaly detection and threat assessment mechanisms, it will provide information for mitigation strategies and feed the CSAIE.
- **Cognitive System AI Engine (CSAIE)**: This block intends to add a layer of cognitive control over the system KPIs. It periodically collects and analyzes data from SRMM and ME, and it provides input to the MRE regarding the collaborative decentralization strategy to be followed in the various CPSs.
- **Modelling and Redesign Engine (MRE)**: MRE incorporates system components that enable the modelling, optimization, and redesign of the CPSoS. It allows the definition of models and meta-models considering system requirement KPIs. With the use of OpenCL, the MRE components manage the implementation of the CPSoSaware *Model, Optimize, Design, Deploy* (MODD) approach.
- **System Inter-Communication Layer (SICL)**: This layer undertakes the responsibility to deploy communication technologies between the CPSoS and the CPSs.
- **CPS Commissioning and CPS to System Inter-Communication Layer**: This block is responsible for commissioning resources and strategies to the variety of participating CPSs.

5.1.2 CPS/CPHS Layer

This block provides the mechanisms for the preparation and deployment of CPS/CPHS to programmable SoC computing devices, such GPUs and FPGAs. It contains the following architectural sub-blocks:

- **OpenCL Description Execution (ODE)**: With the use of Portable Computing Language (PoCL[4]) and its distributed extension PoCL-Remote, ODE manages the deployment of OpenCL configurations to FPGAs at the CPS and CPSoS level.
- **Distributed, Cognitive and Cooperative Intelligence (DCCI)**: This block provides a multitasking mechanism to be shared between CPSs without the involvement of the CPSoS System Layer after deployment. Thus, the participating CPSs will collectively present reliability and fault tolerance towards the fulfilment of system-wide objectives.
- **CPSoSaware Intra-CPS Communication Layer (CICL)**: CICL is aimed at establishing efficient and reliable communications between a) CPSs and their respective sensors, and b) CPSs with other CPSs in the system, in accordance with the DCCI objectives.
- **Extended reality tools and interfaces (XRT)**: This block will provide CPS users with appropriate interfaces and tools for high engagement, optimal experience, situational awareness, and reduced reaction times.

[4] http://portablecl.org/

- **Monitoring Engine (ME)**: The CPS Monitoring Engine collects information regarding the status of CPSs in order to extract appropriate knowledge (e.g. features, decisions). The collected information includes input from involved humans and the cyber-physical environments of CPSs.

5.1.3 Simulation and Training Layer

In a nutshell, this layer serves and orchestrates the simulation of the various CPS/CPHS and their communications, generating training data required for effective optimizations.

5.2 Preliminary Architecture Block Diagram

For the visualization of architectural blocks, we used *PlantUML*[5], which provides a thoroughly documented domain-specific language (DSL) for the description and automatic generation of UML diagrams. More specifically, we utilized a free web-based PlantUML editor named *PlantText*[6] to define the PlantUML code. This section presents the generated UML block diagrams that illustrate the preliminary version of the CPSoSaware architecture.

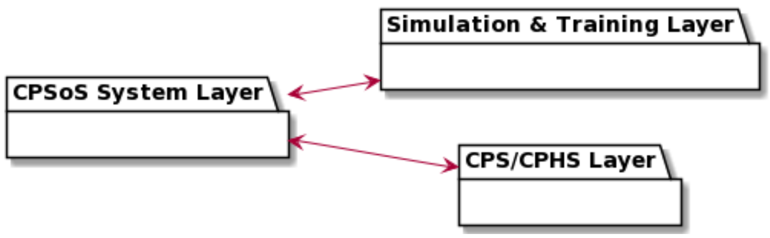

Figure 4. UML diagram - Architectural blocks

Figure 4 displays the top-level architectural blocks of CPSoSaware. The UML source within the PlantText editor can be reviewed via this link. Consequently, the sub-blocks were added to the appropriate layers, resulting in the diagram of **Figure 5**. The UML source of the latter - and an image of higher quality - can be found in this link.

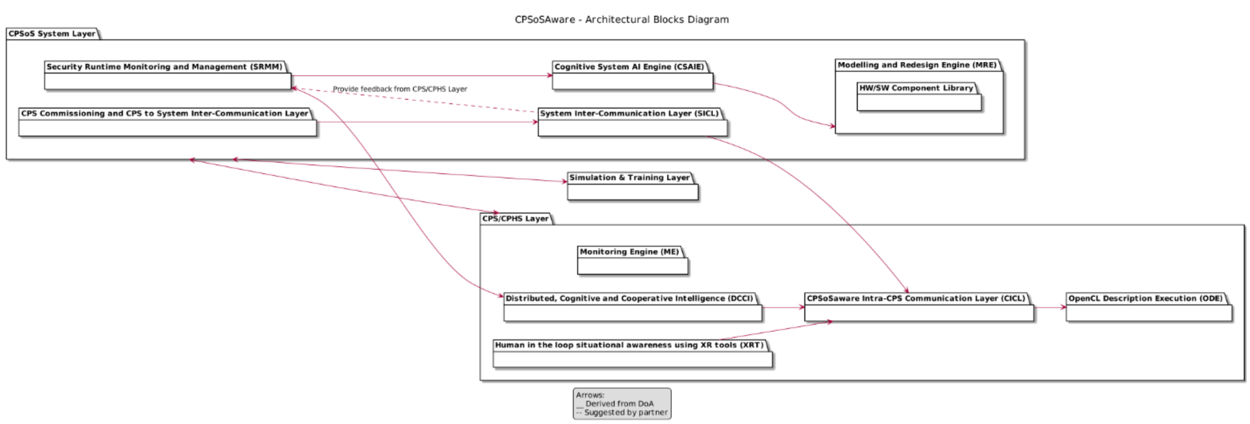

Figure 5. UML diagram - Architectural blocks and sub-blocks

---

[5] https://plantuml.com/

[6] https://www.planttext.com/

Finally, we corresponded the technical components to the appropriate blocks and sub-blocks by utilizing information collected via the component specification templates. By consulting the technology experts and the DoA, a preliminary set of connections (arrows) among blocks and components has also been identified. The resulted UML diagram, shown in **Figure 6**, can also be found in high resolution via this link.

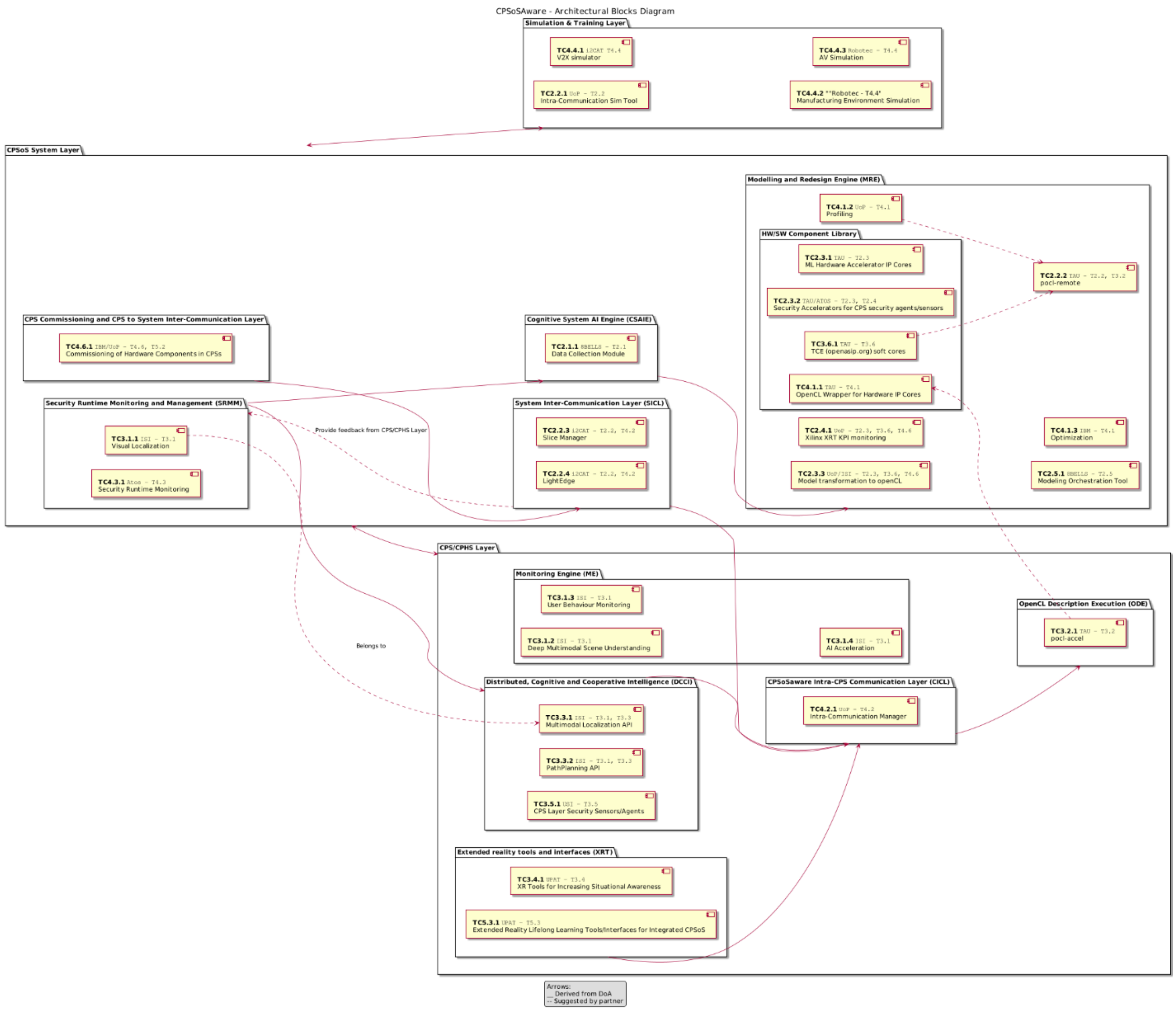

Figure 6. UML diagram - Architectural blocks and sub-blocks with technical components

The next steps in the CPSoSaware architecture include the elicitation of all component interconnections and the definition of interfaces for their communications, in order to facilitate the precise system development and deployment.

6 Conclusions and Next Steps

This deliverable initially introduced the background notions of requirements engineering along with the established requirement elicitation methodologies, and then focused on presenting the outputs from the first phase of Task 1.3. The key outcome from our work on this task during the first year of the project was the **CPSoSaware Technical Specification Elicitation Framework** presented in Chapter 3. Based on a combination of requiring technical partners to fill a circulated requirement specification template, analyzing the existing documentation (DoA), and consulting with the involved stakeholders, we produced a uniform information-rich reference document (Chapter 4). This document contains technical component specifications, including the respective functional and non-functional requirements, and is a **living document** in the sense that it will constantly be accessible by all involved stakeholders and will be frequently reiterated throughout the project lifetime. The overarching aim is to reach a precise architecture design for the CPSoSaware system. An additional key outcome from this work is an overview of the defined **architectural blocks** and a **preliminary version of the system architecture**, which were also presented in this document (Chapter 5).

Relying on the requirements presented in Chapter 4 as the foundation, the next steps in this thread of work involve working towards clearly specifying the interfaces between the functional modules and components of the CPSoSaware framework. The key outcome will be a detailed system architecture, provided in a format that can be parsed by the CPSoSaware framework.

References

Bruegge, B., & Dutoit, A. (2009). *Object-Oriented Software Engineering: Using UML, Patterns and Java* (Third Edition). Prentice Hall. ISBN 978-0136061250.

Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003, May). Documenting Software Architectures: Views and Beyond. In *Proceedings of 25th Int. Conf. on Software Engineering*, 2003. (pp. 740-741). IEEE.

Eid, M. (2015). *Requirement Gathering Methods*. Available at: https://bit.ly/38uuXYe, last accessed: Nov'20.

Liao, L. (2002). From Requirements to Architecture: The State of the Art in Software Architecture Design. *Department of Computer Science and Engineering*, University of Washington, (pp. 1-13).

Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements Engineering: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46).

Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes, 17*(4):40. doi:10.1145/141874.141884.

Regnell, B., Andersson, M., & Bergstrand, J. (1996, March). A Hierarchical Use Case Model with Graphical Representation. *In Proceedings IEEE Symposium and Workshop on Engineering of Computer-Based Systems* (pp. 270-277). IEEE.

Shekaran, C., Garlan, D., Jackson, M., Mead, N. R., Potts, C., & Reubenstein, H. B. (1994, April). The Role of Software Architecture in Requirements Engineering. In *Proceedings of IEEE Int. Conf. on Requirements Engineering* (pp. 239-245). IEEE.

Appendix A: CPSoSaware Component Specification Template

Task name	*Name of the Task X.X*
Task Leader	*Name of the Task X.X leader*

Component Name	*Name of component/module*
Type (Software/Hardware)	*Pls. Indicate if it is a software or hardware*
Short Description	*Pls. provide a general description of your software/hardware infrastructure components/modules that will be used.*
Methodologies that will be used	*Pls. indicate the methodologies that will be used*
User-defined scenarios (no technical)	*Pls. provide the use case-based scenarios which are requested for the system requirements*
Map to project objectives	*e.g. O.1 …*
Relevant Use Cases	*e.g. Pilot/Use Case 1 …*
Estimated date of first release that can be deployed/integrated	*e.g. M12*

Main Inputs	*Specify main inputs;*
Input Data from Partner	*Pls. Indicate the partner you will be getting data from*
Nature of Expected Input	*Pls. Indicate the input format that your component would expect (e.g. JSON, image files, etc.)*
Related Scenarios	*Pls. Indicate the use case scenarios requiring this data*
Interfaces	*Pls. Indicate the connection interfaces - APIs*
Triggered by	*Pls. Indicate the events or conditions that trigger the component's functionality*

Main Outputs	*Specify main outputs;*
Output Data to Partner	*Pls. Indicate the partner you will be providing data to*

Nature of Expected Output	*Pls. Indicate the output format that your component would be expected to produce (e.g. JSON, image files, etc.)*
Related Scenarios	*Pls. Indicate the use case scenarios requiring this data*
Interfaces	*Pls. Indicate the connection interfaces - APIs*

Main functional Requirements	*Pls. Specify main functional requirements - functional-1*
Main non-functional Requirements	*Pls. Specify main non-functional requirements – non-functional-1*
Development Environment	*Pls. Specify the development environment and the programming language to be used*
Execution Time	*Pls. Specify a rough estimation of the execution time of the component*
Execution Frequency	*Pls. Specify a rough estimation of the execution frequency of the component*
Software Requirements	*Pls. Specify any SW requirements or dependencies*
Hardware Requirements	*Pls. Specify the minimum HW required for the best functionality of the component.*
Communications	*Pls. Indicate specific communication requirements between inputs, outputs or between submodules.*
Integration Requirements	*Pls. Indicate any specific integration requirements.*
Deployment Requirements	*Pls. Indicate any specific deployment requirements.*
Security Requirements	*Pls. Indicate any specific security requirements.*
Privacy Requirements	*Pls. Indicate any specific privacy requirements.*

Critical Factors	*Pls. Describe any critical factors that might affect the development or functionality of the component*
Containerization	*Pls. Indicate if the component can be containerized (e.g. Dockerized)*