Information and Communication Technologies (ICT) Programme

Project N^{o}: H2020-ICT-2016-1-732105

D5.4: CERBERO Holistic Methodology and Integration Interfaces

Lead Beneficiary:	AI
Work package:	WP5
Date:	
Distribution - Confidentiality:	[Public]

Abstract: This is a report on integration activities with emphasis on cross-layer cross abstraction levels integration methodology as well as operational interfaces among tools. Here presented is a definition of CERBERO holistic design framework, integration roadmap in addition to elaboration on available and planned interfaces. This release of the deliverable is the first of three releases, of which the second and last, D5.5 and D5.1 respectively, will be the outcome of

iterative development (Deliverable 5.4), throughout the lifetime of the CERBERO project.

Disclaimer

The CERBERO Consortium is the following:

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

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

Document Authors

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

Name(s)	**Organization Acronym**
Gasser Ayad	AI
Michael Masin	IBM
Francesca Palumbo	UNISS
Karol Desnos	INSA

Document Revision History

Date	**Ver.**	**Contributor (Beneficiary)**	**Summary of main changes**
	v0.1	AI	First draft
	v0.2	AI	Second draft
19/09/2017	v0.3	AI, IBM, UNISS	Third draft
08/10/2017	v0.4	AI, UNISS	Fourth draft

Table of Contents

1. Executive Summary

This is a report on integration activities with emphasis on interfaces among tools. This is the first release that will define CERBERO methodology and will be updated at the beginning of Phase II of the project (Iterative development scheme of the project), according to possible technical requirements updates. Hence the second release, D5.5, is released at M21. Final reporting on integration activities and the complete framework characteristics and interfaces are included in D5.1 delivered at M32. Please note that this non-chronological order of the deliverable is due to a recent amendment.

1.1. Structure of Document

The document starts by an introduction to modeling of CPSs and vision of CERBERO integration. Then it delves into state of the art of CPS component specification and design space exploration (DSE). Then the report discusses CERBERO design framework integration approach with required interfaces between the myriad of tools across all layers of the toolchain (model, application, runtime, and hardware layers). The report mentions a number of successful integration endeavors by some of the consortium members, and finally sheds some light on how CERBERO integrated framework is intended to be demonstrated.

1.2. Related Documents

D2.6: CERBERO Technical Requirements

2. Introduction

Cyber-Physical Systems (CPS) are engineered systems comprising interacting physical and computational components. In CPS, computation and communication are deeply embedded in and interacting with physical processes to add new capabilities and characteristics to physical systems.

In this report, we investigate how the various system components could be integrated into a holistic operational framework respecting the requirements imposed by the overall system and seeking a new foundation for CPS design, integration and operation. The goal of the project is to deliver a model-based, heterogeneous and robust solution that is going to be customizable (upon scenario needs) and generalizable (to different scenarios).

2.1. Modeling for Systems Engineering

Systems Engineering (SE) dictates the design process associated with the development of large-scale products through defining systems and subsystems requirements, their architecture, and critical parameters. With recent increase in product complexity and business competition, mere intuition of system design engineers has proven insufficient for finding feasible, safe, reliable and affordable designs[1]. Hence modeling started to emerge to overcome these shortcomings and provide well-thought-out plan for a solid design and smooth implementation and refinement.

In the context of systems engineering, models are created to deal with complexity. In doing so they allow us to understand an area of interest or concern and provide unambiguous communication amongst interested parties, models are leveraged in nearly all stages of the development process. During analysis, models provide an abstract representation of the desired solution, e.g. in terms of dynamic behavior diagrams such as sequence diagrams and state machines. In the design phase, the software architecture could be abstracted through UML component structure diagrams and class diagrams. In addition, ports and interfaces facilitate modeling of data flows between components. From these models code can be partially generated in the implementation phase. Moreover, many models can be used at several levels of specification of the system. For example Finite State Machines are useful at system, and component levels, both for HW and SW. For testing purposes, models are used to generate test cases. To sum it up, Model-Based Systems Engineering (MBSE) or, more generically, Model-Driven Engineering (MDE) has models as the primary data source. Model Driven Development uses the activities associated with modelling to drive the whole development process[16].

From that perspective, specialized modeling tools come into play to increase productivity. There is a plethora of specification languages such as SysML (a UML extension), AADL, and modeling tools such as Excel, IBM Rhapsody, PTC Artisan Studio, Sparx Enterprise Architect, domain specific tools (e.g. medical, avionics, automotive, marine, space, etc.), and simulation environments such as Simulink and Modelica[1]. The Functional Mock-up Interface (FMI) standard facilitates the exchange of simulation models between suppliers and OEMs. FMI adopts a tool-independent approach for both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code[15]. Its success could be attributed, at least partially, to a minimalistic API allowing high flexibility for tool providers.

There are also black box tools integration tools that help with reducing complexity, improving efficiency and cutting development time. modeFRONTIER - by ESTECO - provides an innovative optimization environment with modular, profiled-based access. ESTECO's integration platform for multi-objective and multi-disciplinary optimization offers a seamless coupling with third party engineering tools, enables the automation of the design simulation process and facilitates analytic decision making. ModelCenter Integrate - by Phonix Integration - allows users to automate any modeling and simulation tool from any vendor, integrate these tools together to create repeatable simulation workflows, set simulation parameters, and automatically execute the workflow. Hence ModelCenter Integrate increases productivity by enabling users to execute significantly more simulations with less time and resources.

The benefits of using models in systems engineering are manifold: Building models is usually easier than building the actual system as a whole from ground up. Modeling helps to capture, structure, and understand the system and to reveal, early enough, possible problems and potential bugs lurking at every stage of the system design and implementation. It has proven more appropriate for high-stakes and increasingly complex applications such as reconfigurable and adaptive cyber physical systems. Last but not least, integration per se is not a scientific activity, it is rather an engineering problem full of accidental issues (i.e. tools adopting different languages, running on different platforms, vendor specific components, etc.) CERBERO is no exception; therefore, we decided to adopt a model-based, semantically oriented, approach.

2.2. Systems Integration from CERBERO Perspective

Systems integration is a process whereby a cohesive system is created from components that were not specifically designed to work together. Components of an integrated system are often systems in their own right. And integration aims at interconnecting these components together in a layered fashion in order to provide a useful exchange of information, data and/or control between these sub-systems and assuring that the integrated system meets requirements and performs according to user expectations.

To facilitate systems interconnection, an interface is defined and created as a point of interaction between communicating systems. Interfacing may also mean using a common message format, or intermediate representation, to provide kind of a unified communication paradigm across the system entirely, or partially. Translation would be required from the interface of one component to the intermediate representation, or vice versa.

Generally, integration is done considering subsystems as black-boxes, hence creating a middleware to “glue” together these disparate subsystems without them needing to know anything about each other. We support the “open world” assumption, where each tool should assume that all objects may have more properties than it knows about. Inspired by FMI standard, we are looking for as simple formalism as possible to exchange objects with properties that tools recognize with middleware support for simple property mappings when it is semantically clear.

A recommended integration process adopts an iterative – that is, continuous or constantly evolving - integration model rather than a static or fixed model. Hence, it is essential to create a holistic and customizable framework for subsystem integration as these subsystems and tools undergo continuous development. For tracking of integration progress, the integrated system must be verified and validated periodically against system and user requirements toward a mature integration framework for the overall platform or toolchain.

3. State of the Art

Systems Engineering governs the design process associated with the development of large-scale products through defining systems and subsystems requirements, their architecture, and critical parameters. Recent increase in product complexity and business competition dictates the necessity for finding feasible, safe, reliable and affordable designs. Consequently, three techniques have become popular in helping handle the complexity: layering design process into several levels of abstraction, separation of concerns, and using computerized tools for automation of modeling, optimization and analysis.

Embedded systems are commonly subject to data intensive processing applications where huge amounts of data are handled in a regular way by means of repetitive computations. These applications deal with intensive or massive parallelism either on the data level or on the task level. High-level analysis of data-intensive applications becomes a complex task necessitating a refinement step toward low levels of abstraction specifying both computation and communication costs in the system.

The following works address the challenge of abstracting system design in a layered fashion that maintains computation and communication specifications of the system and facilitates analysis and optimization hence provides a competitive business edge for the end cyber physical system.

3.1. Usage of Formal Semantics

Model-Based Engineering of Cyber-Physical Systems (CPS) needs correct-by-construction design methodologies, hence CPS specification languages require mathematically rigorous, unambiguous, and sound specifications of their syntax and corresponding model semantics. Cyber-physical systems are software-integrated physical systems often used in safety-critical and mission critical applications, for example in automotive, avionics, chemical plants, or medical applications. In these applications sound, unambiguous and formally specified modeling languages can help developing reliable and correct solutions.

Traditional systems engineering is based on causal modeling (e.g., Simulink), in which components are functional and a well-defined causal dependency exists between the inputs and outputs. It is known that such a causal modeling paradigm is imperfect for physical systems and CPS modeling since physical laws are inherently acausal.

Recently, acausal modeling has gained traction and several languages have been introduced for acausal modeling (e.g., Modelica, bond graphs)[17]. Every time a new specification language is introduced, there is a natural demand to extend it to support as many features as possible. Unfortunately, this often leads to enormously large and generic languages, which have many interpretations and variants without a clear, unambiguous semantics. Because of the size of these languages, there is not much hope for complete formalization of their semantics.

A fundamental problem is that generic languages provides support for many more features than a specific problem needs, still they often lack support for some essential functions that would be otherwise needed. Thus, in most cases it is more feasible to use Domain Specific Modeling Languages (DSML)[18][19], which are designed to support exactly the necessary functions. Additionally, because DSMLs are usually significantly smaller than generic languages, their formal specification is feasible.

The work in [2] discusses the challenges to develop the formal semantics of a CPS-specific modeling language called Cyber-Physical Modeling Language (CyPhyML). The paper formalizes the structural semantics of CyPhyML by means of constraint rules, and the behavioral semantics by defining a semantic mapping to a language for differential algebraic equations. The specification language is based on an executable subset of first-order logic, which facilitates model conformance checking, model checking and model synthesis.

3.2. Viewpoint modeling

Viewpoint modeling is an effective approach for analyzing and designing complex systems. Splitting various elements and corresponding constraints into different perspectives of interests, enables separation of concerns such as domains of expertise, levels of abstraction, and stages in lifecycle. Specifically, in Systems Engineering different viewpoints could include functional requirements, physical architecture, safety, geometry, timing, scenarios, etc. The first development and refinement step is referred to as Engineering Modeling, and the second optimization and analysis step is referred to as Design Space Exploration (DSE). Consequentially, there are no automatic tools for holistic DSE based on libraries of previously developed and tested Analysis Viewpoints.

Despite partial inter-dependences, models are usually developed independently by different parties, using different tools and languages. However, the essence of Systems Engineering requires repetitive integration of many viewpoints in order to find feasible designs and to make good architectural decisions, e.g., in each mapping between consecutive levels of abstraction and in each design space exploration. This integration into one consistent model becomes a significant challenge from both modeling and information management perspectives.

The work in [1] suggests: (1) a unique modular algebraic viewpoint representation robust to design evolution and suitable for generation of the integrated optimization/analysis models, and (2) an underlying ontology-based approach for consistent integration of local viewpoint concepts into the unified design space model. The paper shows also an example of an optimization model with different combinations of partially interdependent Analysis Viewpoints. Using the proposed modeling and information management approach the underlying viewpoints' equations can be applied without modification, making the approach pluggable.

3.3. MARTE and PiSDF

At the present time, embedded systems are commonly dedicated to data intensive processing applications where huge amounts of data are handled in a regular way by means of repetitive computations. These applications deal with intensive or massive parallelism. Indeed, parallel applications can implement two levels of parallelism: data parallelism and task parallelism. High-level analysis of data-intensive applications becomes a complex task necessitating a refinement step toward low levels of abstraction specifying both computation and communication costs in the system. Accurate performance numbers can be reached at the cost of very detailed modeling. On the other hand, a moderate effort for modeling leads to a high-level evaluation task, but the accuracy is lost.

The work in [3] proposes a new approach that takes advantage of Model-Driven Engineering (MDE) foundations and Modeling and Analysis of Real-Time and Embedded Systems (MARTE) profile. The paper defines a transformation to a new level of abstraction that alleviates the exploration and analysis tasks of real-time data-intensive processing applications. This level is based on a novel extension of the famous Synchronous Data Flow (SDF) Model-of-Computation (MoC), the Parameterized and Interfaced Synchronous Dataflow (PiSDF) model. PiSDF facilitates the specification, and especially the analysis of data-intensive applications as it gathers a lot of features including hierarchy, configurability and dynamism. This MoC introduces analysis techniques facilitating the design space exploration task. Then, a high-level analysis of the data-parallel application is performed using the PREESM rapid prototyping tool.

4. CERBERO Integration Methodology

4.1. Tool Suite

As a first step toward building CERBERO toolchain, IBM and AI in collaboration with the entire consortium have surveyed and agreed upon an array of tools that are either state-of-the-art or already exist within the consortium as fully mature technologies. This survey helped to collect and consolidate information about functionality of each tool, setup requirements, documentation availability, maturity/readiness level, licensing, and usability domains, and would serve as a reference point for integration efforts. These tools cover the entire stack of a CPS: system model layer, application layer, OS or runtime layer, and hardware abstraction layer. This survey analysis resulted in Appendix A, which elaborates on all tools in full detail. Collaboration between consortium members on tools is meant to steer WP5 efforts toward defining a feasible CERBERO integration plan and to specify the interfaces for newly designed tools.

4.2. System Design and Operational Framework

For the model-based design of complex systems, such as CPS and CPSoS, a structured and well-defined design framework is essential to guarantee high quality products that fulfill all requirements of the stakeholders and to enable handling the complexity of such systems by introducing different viewpoints or abstraction layers to model the system under development. Therefore, an important step to obtain a system implementation is the Design Space Exploration (DSE), where design decisions are taken based on goals and requirements defined in previous phases. The set of valid solutions may be restricted by constraints, which could be derived from previously defined requirements considering aspects such as functional requirements, physical architecture, safety, geometry, timing, etc. Valid solutions are rated with respect to defined goals to facilitate a decision toward choice of a final system implementation. Based on this abstract framework, concrete DSE methods can be implemented addressing different kinds of DSE problems that are relevant in industrial practice. In the CERBERO framework we intend to carry out DSE at different levels, at system level to define the proper distribution among computing nodes, and within the single node to identify the optimal HW/SW partitioning and components set-up. The selectedoptimal design resulted from the system level DSE should be available (i.e., machine readable) for HW/SW co-design to define, partially, its functional requirements.

CERBERO design framework aims at shifting designers' work at a higher abstraction level and earlier in the design process, relieving them from manual and complex HW/SW tuning phases. It creates and promotes model-based cross-layer optimization, design, verification and simulation methods that aim at deeply modifying CPS system design approach, shifting from a V model paradigm to a ladder model paradigm, see *Figure 1*, thus slashing the time to market. In the model-based approach all the properties and system characteristics are tackled concurrently right at the model level at design-time to allow the cross-optimization of physical components, as well as the cross-configuration of the HW and SW layers. Dealing with CPS and CPSoS poses challenges for the overall system assessment (dimensioning and characterizing communication channels, profiling power consumption, etc.) To cope with these challenges, CERBERO features adequate analysis tools and provide sufficient breakdown of all related layers of the system model to guarantee continuous validation.

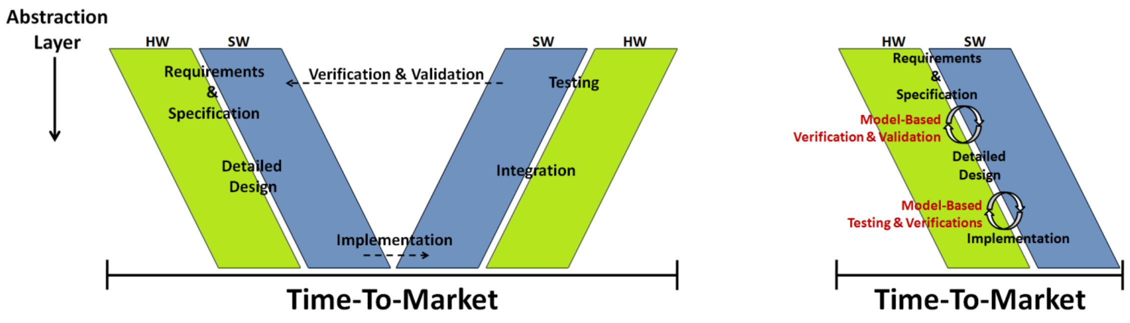

Abstraction Layer
HW
SW
Requirements & Specification
Verification & Validation
Testing
Detailed Design
Integration
Implementation
Time-To-Market
Model-Based Verification & Validation
Model-Based Testing & Verifications

Figure 1: V-Model vs. Ladder System Design Approach

Tool integration is meant to be cross-layer, see *Figure 2*, in such a way that consistency is maintained between individual layers (high-level system model, application architecture, runtime manager, and low-level implementation) through collaborating with respective partners for development of interfaces or intermediate semantics that map the functionality of each block or tool in the toolchain and wrap it for integration with other blocks or tools accordingly. Using intermediate semantics for integration will allow requirements analysis and verification at the model level. Integration of some HW design tools, however, is more implementation oriented since interfaces between these tools have already been coded or currently being developed and tested. Integration will be based on methods and interfaces in agile cycles performing cross-layer integration and combining modelling, simulation, verification and code generation activities.

It is essential before attempting the modeling, and hence integration efforts, to plan the following system characteristics:

- Functional and non-functional requirements.
- Available system libraries, including models that require new developments.
- System structure and composition.
- A unified naming convention for all properties and attributes.
- Open world assumption (more tools, more viewpoints)

Consequentially, constraints and KPIs (such as jitter, delay, latency, KPI, QoS, energy consumption) are verified before starting integration activities.

o

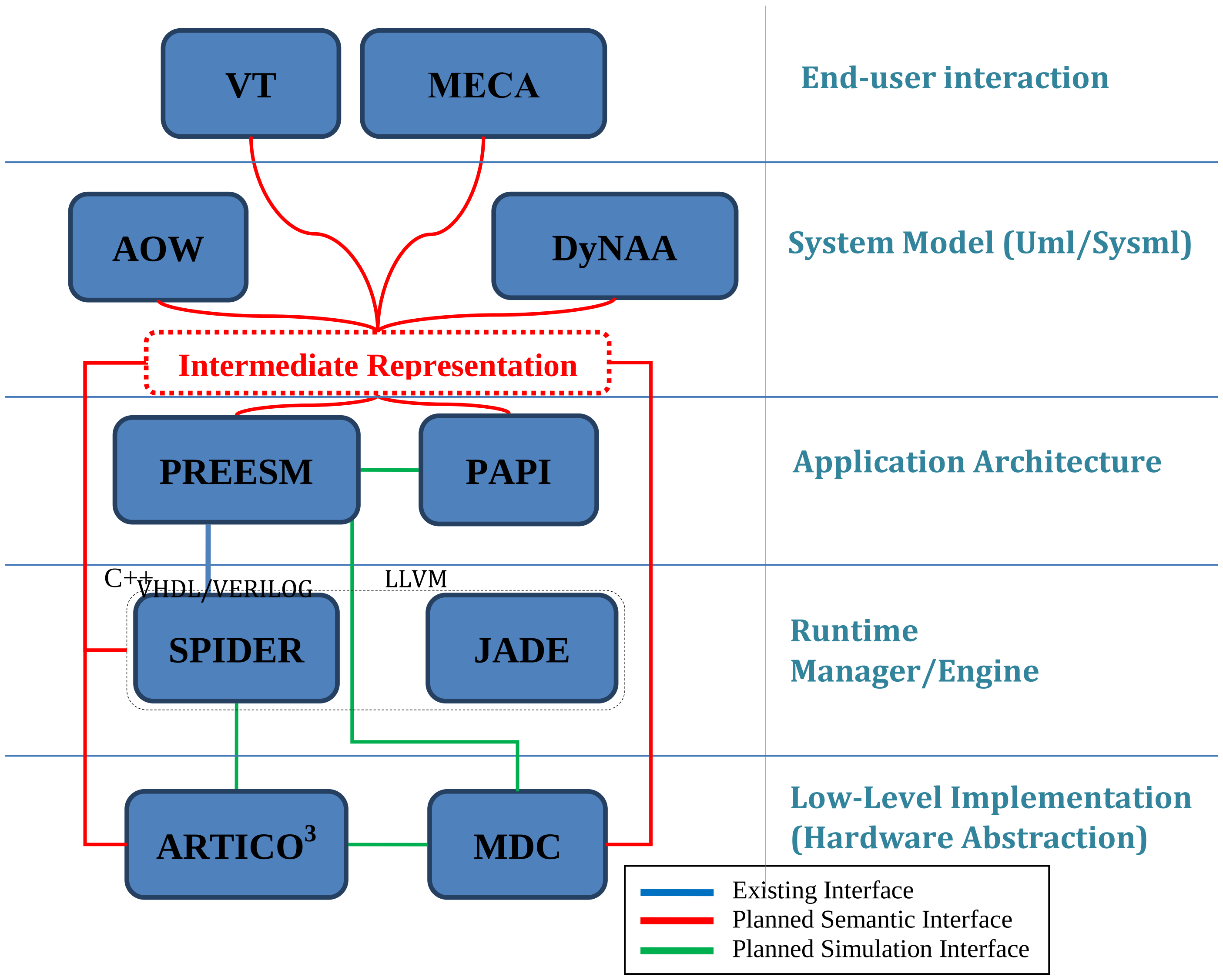


Figure 2: CERBERO Cross-Layer Integration Topology

Integration also involves gap analysis in order to discover gaps and overlaps and reveal points of interoperability. Understanding the opportunities for integration or gap-filling informs holistic tradeoff decisions about integrating systems and capabilities [8].

Execution of CERBERO integration methodology adopts a Continuous Engineering approach that guarantees reuse of design time models for generating novel and more accurate design and operational models up to the runtime environment. In this deliverable we describe an initial integration methodology. Semantic model based integration supports cross-layer and cross-levels of abstraction modeling, usually dealing with system integration and system modeling. In this case we leverage on ontologies “passing” relevant properties among layers and levels of abstraction. On the other hand, each layer (that can handle more than one model) has its own tools that usually much more compatible among each other and that should pass information by means of the same ontologies focusing on fast execution (important, e.g., for simulation).

We implement a simplified version of ontology based on things with properties: each tool will look for relevant properties of intermediate models, constraints or KPIs from other tools (potentially in other layers / levels of abstraction); while the framework supports mapping model properties to tools/analysis namespace properties and provides shared directories for model exchange. The methodology will be further co-refined together with project advance in cross-layer integration framework combining modeling, DSE, simulation, verification and code generation capabilities of CERBERO tools. Two iterations are planned during the total duration of the project, see Figure 3, delivering a “beta” and a “final” version of CERBERO integration strategies and internal interfaces.

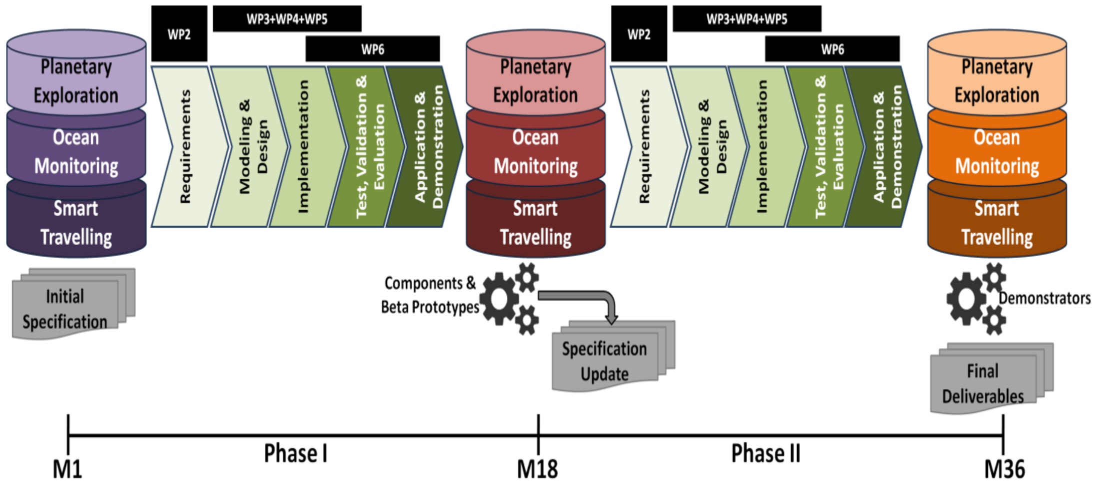


Figure 3: Iterative Development Scheme of CERBERO Integration Framework

4.3. Simplified Ontology-based Integration with Intermediate Representation

As mentioned earlier, integration is traditionally a complex engineering problem. It is characterized by several different accidental issues, as usage of different modeling paradigm or languages, that make the creation of the infrastructure particularly effort hungry. This is particularly true in the CPS environment where you need to combine together components suitable for the different aspects of the CPS. These motivations led designer to opt for semantic integration of tools, and ontology-based integration is particularly suitable, in our opinion, to the case.

The term “ontology” derives from ancient Greek “ontos”, which means “being” and logos, which means, “discourse”. Ontology -- or roughly the "science of stuff" and how it is represented -- used to be a rather obscure branch of philosophy. It still is in some cases, but it is also an important and growing area of computer science and the web of things (WoT). Then, ontology has assumed other relevant meanings, such as:

“A formal, shared and explicit representation of a domain concept.”

or:

“A method for formally representing knowledge as a set of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts.”

or:

“A formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains.”

Ontology-based data integration involves the use of ontology(s) to effectively combine data or information from multiple heterogeneous sources. So we are starting CERBERO integration efforts by creating some kind of ontology, by which we mean a collection of terms that identify the real things and relationships that are relevant to CERBERO-supported domains and industry-driven requirements. Maintaining an ontology design facilitates keeping track of the terms and ensures integration efforts quickly get up to speed.

CERBERO features heterogeneous technologies and tools. As discussed above, we retained the idea that model-2-model transformation would not necessarily be the main mean of communication between tools (also, the feasibility of having fully automated model to model transformations from system of system level down to hardware is unlikely). Instead, each tool will manage its own model(s), and the intermediate representation will be used to exchange “cross-layers” and “cross-models” information between tools.

The intermediate format is therefore necessary to achieve the mediation between the application's class model conceptualization and the common domain ontology conceptualization, since objects in the original format cannot be handled directly in the framework[8]. In CERBERO, the exchange format will follow the Resource Description Framework (RDF)-like meta-model underlying common ontology.

CERBERO proposes an application layer solution for interoperability. The key idea is to utilize semantics provided by existing specifications and dynamically wrap them in a middleware fashion into semantic services in such a way that automates interoperability without any modifications to existing standards, devices, or technologies, while providing to the framework user an intuitive semantic interface with services that can be obtained by executing all CERBERO technologies. In particular, a semantic layer is proposed for simple mapping of KPIs to model properties (i.e., connecting name spaces) as a semantic service; we may conventionally call that middleware the “Service Layer.”

CERBERO's integrated framework aims at a one model for the entire platform. That is, a single model with several different fields and properties that are detail-rich. Each tool and /or analysis viewpoint will recognize some properties, enrich others and ignore the rest, allowing (partial) transformation of otherwise inconsistent or incompatible models or semantics.

A property is defined by a set of properties, such as an ID or a name, a type, a value, and (if applicable) a measurement unit. A property can be simple (flat), or object (composite of sub-properties). And since information available at system level are relevant at lower levels, properties in a model structure are inter-linked in such a way that allows navigation from one property in the system to another across different layers. Some links between properties may carry decisions. Ontology helps with revealing meaning and relations of each property from the whole graph by referring to a property by its name. All properties relevant to the model are present in Ontology. Ontologies can be either simplified (i.e. system model features only a subset of all properties of the real system), or full ontology where all properties in the system model are presented in the ontology.

The system-level of abstraction is capable of having a complete view of the system infrastructure. However, system level is not fully or directly aware of the “internals” of the underlying more detailed levels. Underlying levels can communicate/report to the system level through backward annotations. At system level, a clue is needed about the underlying layers. Such a clue can be obtained through design space exploration of features of the underlying HW architectures, such as nominal values of speed and power consumption per core, number of cores, etc.

Properties have to be exported in some form of intermediate representation, such as XML or JSON. A parser is to be built to read properties and sub-properties. This intermediate representation or semantics is used in to define the cross-layer interfaces for the CERBERO integration framework. Interfaces are to be developed iteratively, that is, in agile cycles so as to reach full maturity by the end of the CERBERO project.

To sum it up, intermediate representation or schema is the file or message format that one level of abstraction, layer or tool is going to produce and the other is going to consume. It may not necessarily be human understandable/comprehensible, but it’s necessarily interpretable from one level/layer to another. We need to extend class-based representation (i.e. the system-level model on IBM Rhapsody or DynAA) to the property-based intermediate representation (e.g. in XML format as with DynAA, or JSON format as with the SEMI tool of IBM). In addition, feedback can be communicated to the originating level/layer to facilitates self-adaptivity and reconfigurability.

4.4. Empirical Approach to Cross-Layer Integration

In this case study each tool from a subsequent layer will look for relevant properties of intermediate models from tools of the preceding abstraction layer. Model properties would be mapped to namespace properties. Properties are exchanged through a mechanism for sharing or information, following generations conventions of KPIs.

From the modeling perspective, paper [9] presents a toolchain integration methodology for abstracting existing legacy IPs modules into SysML components. During the abstraction flow, it is possible to set the level of detail to be maintained in SysML, such as hierarchical structure and data types of the legacy modules, in order to allow designers to choose the level of abstraction to be preserved in the SysML model. The methodology aims at producing SysML models with both structural and behavioral information.

4.5. Mapping of Model Properties to Analysis Metrics

Any metrics library consists of three main parts: attributes, constraints, and filters[13]. This library is mapped to the system and/or application high-level models in UML/SysML using a mapping UI in the modeling environment, such as IBM Rational Rhapsody[14].

Attributes

Here we give an example to illustrate the attribute set. In figure 6 below, the attribute set itsUnPlanCostElements is a set of parts having at least following attributes: MaitenanceUnPlanCost, MTTF (only elements with MTTF > 0 included), and subSysId. Attributes of metric can be mapped to any attributes in the model via UI, as appears in figure 7.

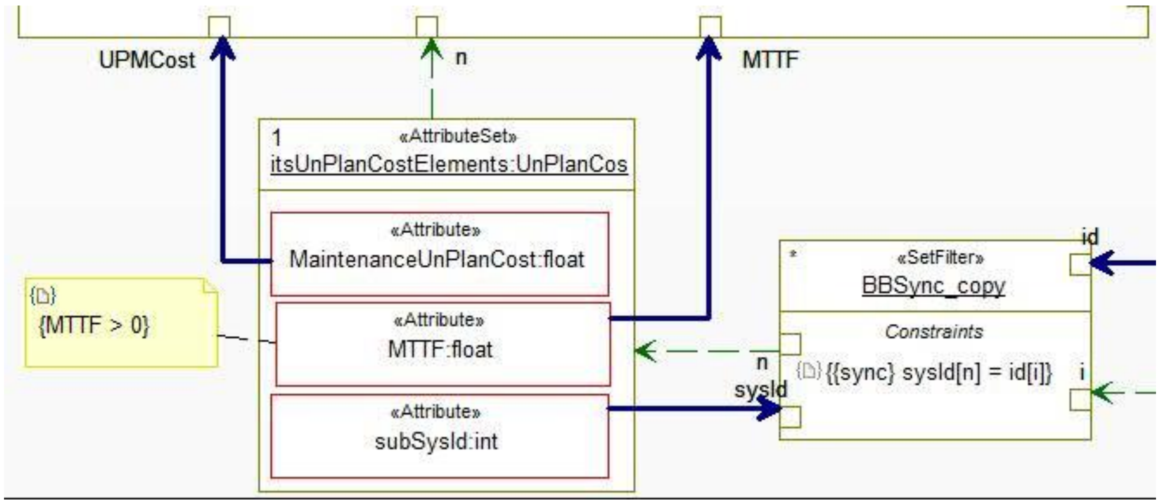
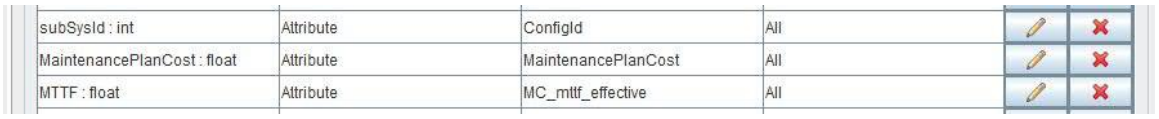


Figure 6: Metric Attribute Set.



subSysId : int	Attribute	ConfigId	All	✎	✖
MaintenancePlanCost : float	Attribute	MaintenancePlanCost	All	✎	✖
MTTF : float	Attribute	MC_mttf_effective	All	✎	✖

Figure 7: Mapping of Metric Attributes to Model Attributes

Hence, attribute set in the metric represent a set of the parts in the model. All parts having specific attributes and/or stereotypes in the model (after mapping) are members of attribute set

Currently all attributes in the attribute set must be mapped to attribute or stereotype in the model. Potential elements of the set can be additionally filtered depending on attribute values. To do this – attach SysML constraint to attribute set, as we will see shortly.

Constraints

A set constraint is constraint calculated (or valid) for elements of attributes set. It must have incoming set flows from attribute sets connected via set constraint parameters, and must have incoming attribute flows from attributes of attribute set connected via constraint parameters. Calculated attributes must rather have outgoing attribute flows. Figure 8 shows a set constraint.

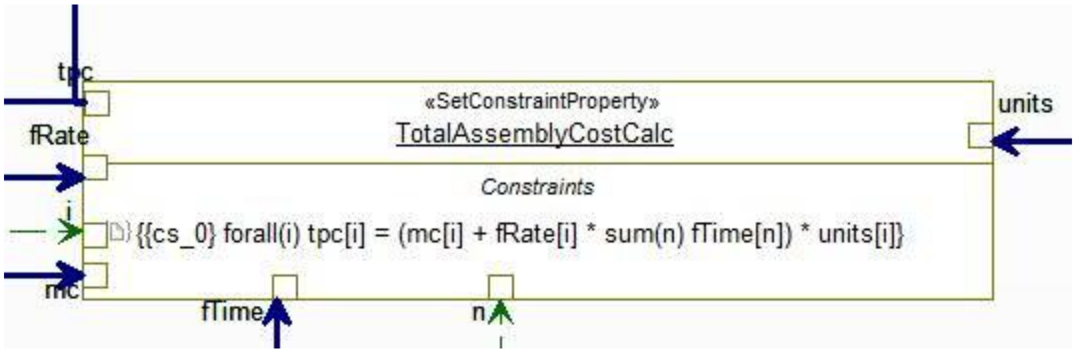


Figure 8: Set Constraint.

Filters

A set filter is constraint which define filter of one set based on values of elements of other sets . Outgoing set flow points to filtered attribute set. Here a constraint defines a filtering criteria based on attributes of filtered set. As appears below:

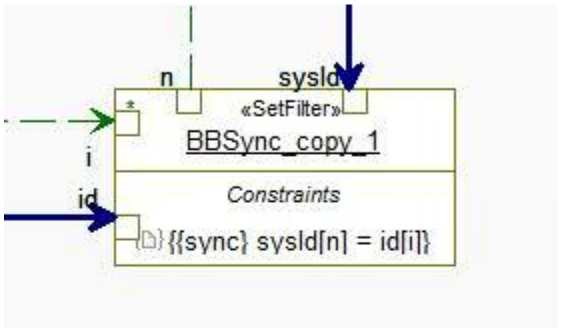
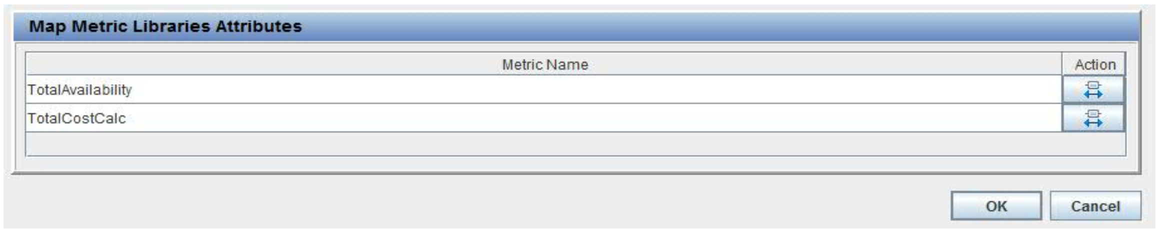


Figure 9: Set Filter.

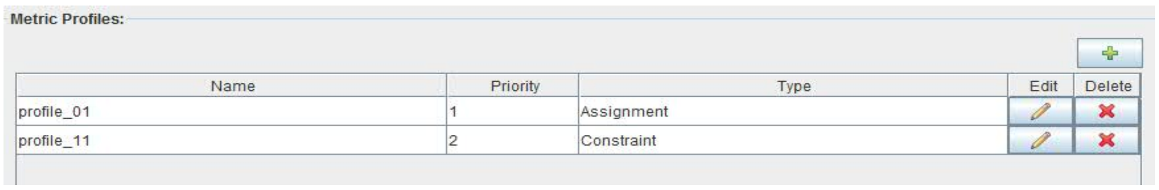
Mapping UI

To apply metric to the model, one should create matching profile. In Rhapsody there exists two types of metric profiles, namely assignment and constraint. For assignment profiles one also should set priority to define order of calculations. Each metric can be applied many times to the same model. A metric attribute can be mapped to model attribute, a stereotype, or to different model attributes for different blocks, as appears in Figure 10 below.



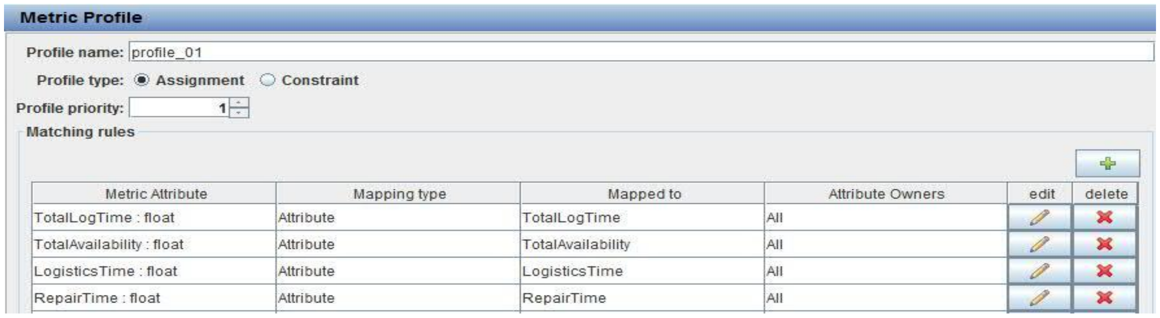
Map Metric Libraries Attributes

Metric Name	Action
TotalAvailability	
TotalCostCalc	



Metric Profiles:

Name	Priority	Type	Edit	Delete
profile_01	1	Assignment		
profile_11	2	Constraint		



Profile name: profile_01
Profile type: [✓ Assignment] Constraint
Profile priority: 1

Metric Attribute	Mapping type	Mapped to	Attribute Owners	edit	delete
TotalLogTime : float	Attribute	TotalLogTime	All		
TotalAvailability : float	Attribute	TotalAvailability	All		
LogisticsTime : float	Attribute	LogisticsTime	All		
RepairTime : float	Attribute	RepairTime	All		

Figure 10: Attribute Mapping, Metric Profiles, and Metric Profile Properties

4.6. OS Qualified as a Toolchain Host

CERBERO array of tools and technologies widely support both Windows and Linux. However, Windows has been qualified by the consortium as the host operating system of choice for the project since it has a larger market share and is more familiar with respect to Linux, more feature-complete, enjoys commercial support, and is not subject to the disintegrity imposed by a wide range of different Linux distributions that are only maintained by their respective communities.

5. Internal Interfaces

Cyber Physical Systems (CPSs) are an evolution of embedded systems and are based on a tight combination of collaborating computational elements (i.e. micro computing units or embedded systems interconnected by a communication system) that control physical entities. Therefore, in CPS all types of smart equipment (i.e. sensors, actuators, devices, machines, robots) are interconnected creating a smart community with data capture and action capability from/to the physical world.

A CPS-based architecture for manufacturing is made of smart but independent manufacturing components without any knowledge of the role they have to play together in the real application. Ontologies can supply such kind of knowledge, playing a very crucial role in CPS design.

Ontology in computer terms is concerned with meaning of and relationship between entities. It derives its importance from its ability to organize raw or unstructured data by semantics (i.e. meaning, such as classes, properties or attributes, and relationships), rather than merely by strings or keywords, thus facilitating more efficient data operations (storing, querying, sending and receiving). An ontology is often referred to as a "schema".

Ontologies are typically far more flexible than class representations and hierarchies as they are meant to represent information coming from all sorts of heterogeneous data sources. Class hierarchies on the other hand are meant to be fairly static and rely on far less diverse and more structured sources of data.

Therefore, CERBERO consortium postulates simplified ontologies as the right tool to implement cross-layer information sharing and data flow in order to implement internal interfaces and hence realize the CERBERO holistic integration methodology. As discussed above, by simplified ontology we mean using things with properties where all analysis viewpoints are defined by a set of properties and all objects that hold them.

5.1. RDF/JSON Intermediate Representation for Data Serialization

Successful tool integration necessitates that system model data to be serialized or rendered into a particular, preferably standard, format or syntax that can be parsed later on and transformed into another format as per need of subsequent layers.

JSON representation of a set of *RDF* triples as a series of nested data structures has become increasingly popular as a data serialization format thanks to its more lightweight structure compared to XML, making it a useful format for data exchange in a way that requires less bandwidth than a bulky XML document. JSON-LD is a JSON-based format that stands for *JavaScript Object Notation for Linked Data*. Structured data expressed in JSON-LD uses the familiar JSON structure but has been especially developed for expressing information using structured data (or, linked data) vocabularies - such as *schema.org*. There are several valid formats for representing RDF data. JSON-LD is one of these formats. Other serialization formats exist for RDF, such as RDF/XML, OWL/XML, Turtle, TriG, and N3. In other words, an *RDF document* is a document that encodes an RDF graph or RDF data set in a *concrete RDF syntax*, such as JSON-LD, to enable the exchange of RDF data between systems. JSON-LD format is lightweight and is easy for humans to grasp.

JSON-LD is available in a number of popular programming environments. Each implementation of JSON-LD is fully conforming to the official JSON-LD specification, and is available at www.json-ld.org. The specification is already deployed in production by major companies and has recently received the official W3C Standard status.

As RDF ontologies can be efficiently expressed in JSON, CERBERO consortium adopts JSON and JSON-based extensions for data intermediate representation and cross-layer interfacing. JSON is compatible with IBM semantic middleware (SEMI). Thorough investigation is currently taking place to assure that JSON as format for serializing and transmitting structured data would fulfill the integration needs of CERBERO. D5.5 - the next version of this deliverable - would feature well-defined CERBERO interfaces.

5.2. Ontology Data Serialization with JSON

Many software packages are now available for creating ontologies, among which are:

- Stanford University's Protégé, a free, open-source ontology editor.
- TopBraid Composer from TopQuadrant.
- Generally any text editor.

Apart from RDF/XML, RDF ontologies can also be expressed in human readable formats, most notably JSON.

Before delving into a full example of a JSON-represented ontology, here below is a quick recall of the most relevant terminology:

JSON: is a lightweight format for data exchange (as XML, but less verbose yet more human-readable). [By "Data" we mean is the set of classes and attributes that will be shared by tools and technologies that interoperate within the CERBERO framework.]

JSON-Schema: a JSON document according to which the ontology is defined. [For example, if there are required or mandatory attributes, if they are of number datatype, if they can be null, etc.] In XML equivalence, that would correspond to an XML schema or a Document Type Definition (DTD).

Ontology: formally defines a common set of terms used to describe and represent a domain (according to the JSON-Schema).

Instance of Ontology: a specific element of an ontology.

Template: a preloaded JSON schema that can be used and expanded by the user to create their ontologies.

Context: a list of templates that simplifies the ontology presentation preserving its contents.

Here we provide a simple Ontology such as representing a temperature sensor that stores an identifier, timestamp, measurement, unit and GPS coordinates. An instance of this ontology may look like:

```
"SensorTemperature": {
  "identifier":"ST-TA3231-1",
  "timestamp": {
    "$date": "2014-01-27T11:14:00Z"
  },
   "measurement":25.1,
   "unit":"C",
   "geometry": {
     "type": "Point",
     "coordinates": [90, -10.1]
   }
}
```

Following is a JSON Schema that describes the Ontology:

```
"$schema":"http://json-schema.org/draft-04/schema#",
"title":"SensorTemperature Schema",
"type":"object",
"required": ["SensorTemperature"],
"properties": {
   "_id": {
     "type":"object",
     "$ref":"#/identifier"
   },
  "SensorTemperature": {
    "type":"string",
    "$ref":"#/data"
   },
   "additionalProperties": false,
   "identifier": {
     "title":"id",
     "description":"ID for sensor temperature",
     "type":"object",
     "properties": {
       "$oid": {
         "type":"string"
```

```
          }
        },
        "additionalProperties": false
},
"data": {
    "title":"data",
    "description":"Info about sensor temperature",
    "type":"object",
    "required": ["identifier","timestamp",
                 "measurement","unit","geometry"],
    "properties": {
      "identificador": {
       "type":"string"
      },
      "timestamp": {
        "type":"object",
        "required": ["$date"],
        "properties": {
          "$date": {
            "type":"string",
            "format":"date-time"
          }
      },
      "additionalProperties": false
    },
    "measurement": {
      "type":"number"
    },
    "unit": {
      "type":"string"
    },
    "geometry": {
      "$ref":"#/gps"
    },
    "additionalProperties": false
 },
 "gps": {
   "title":"gps",
   "description":"Gps SensorTemperatura",
   "type":"object",
   "required": ["coordinates","type"],
   "properties": {
    "coordinates": {
      "type":"array",
      "items": [
        {
          "type":"number",
          “maximum”:180,
          “mininum”:-180
        },
        {
          "type":"number",
          “maximum”:180,
          “mininum”:-180
        }
      ],
      "minItems":2,
      "maxItems":2
    },
    "type": {
      "type":"string",
      "enum": ["Point"]
    }
    "additionalProperties": false
}
```

6. Software Packaging Plan

Software packaging for CERBERO integration demos will move in either one of two possible directions: virtualization or containerization. Both directions are currently being assessed. Both technologies generally serve the same purposes of portability, isolation, reusability, effective use of the hardware, better DevOps and continuous deployment and testing. However, they are quite different in terms of how they are implemented. Figure 11 illustrates how a virtual machine compares to an application container.

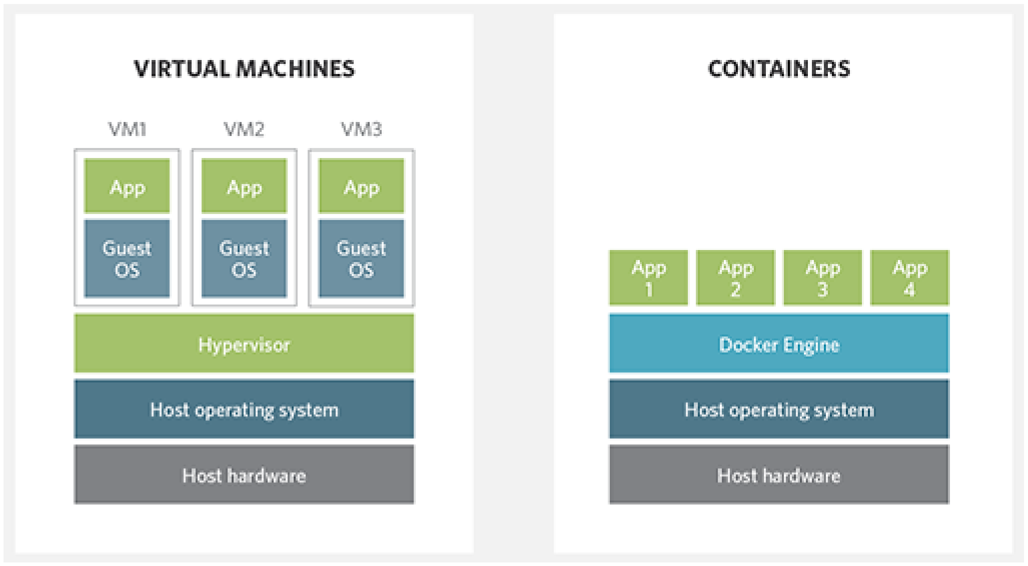


Figure 11: Schematic Comparison of Virtual Machines (left) and Containers (right)

Each approach has its advantages and drawbacks. Below we discuss each approach from CERBERO perspective:

6.1. Virtualization

A virtual machine has a full OS with its own memory management installed with the associated overhead of virtual device drivers. In a virtual machine, valuable resources are emulated for the guest OS and hypervisor, which makes it possible to run many instances of one or more operating systems in parallel on a single machine (or host). Every guest OS runs as an individual entity from the host system.
Currently two virtual machines are being maintained and updated in the context of CERBERO project integration and demonstration activities. A Windows virtual machine hosts the Windows-based tools, namely IBM AOW and Dynaa, and a Linux virtual machine hosts Linux-based tools including Preesm and Xilinx Vivado.

6.2. Containerization

Docker is the world’s leading software container platform. Developers use Docker to eliminate “works on my machine” problems when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density. Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

Each tool provider would provide a docker container - that is, a virtual environment with their respective tools installed and technologies or frameworks set up. That is considered a better alternative to sharing just one machine on which all tools and technologies are to be present. All tool providers can work in parallel preparing their respective containers, hence we would slash the overall time preparing the work environment, and also avoid any possible inter-operability problems/conflicts due to Docker powerful application isolation capability.

7. Relevant Works of Tool Integration

7.1. Integrating PREESM with ARTICO3

PREESM (a rapid prototyping and multi-core code generation and scheduling tool by INSA), and SDSoC (a hardware accelerators generation tool by Xilinx) have been integrated in collaboration between INSA and UPM for the ReCoSoC conference. Work is titled "Analysis of a Heterogeneous Multi-Core, Multi-HW-Accelerator-Based System Designed Using PREESM and SDSoC"[10].

A design flow that combines, on one side, PREESM as a dataflow-based prototyping framework and on the other side Xilinx SDSoC as a HLS-based framework to automatically generate and manage hardware accelerators has been developed. This design flow leverages the automatic, static task scheduling obtained from PREESM with asynchronous invocations that trigger the parallel execution of multiple hardware accelerators from some of their associated sequential software threads. An image processing application is used as a proof of concept, showing the interoperability possibilities of both tools, the level of design automation achieved and, for the resulting computing architecture, the good performance scalability according to the number of accelerators and SW threads.

Hence, this design flow provides developers with transparent deployment capabilities to efficiently execute different applications on complex devices such as multi-core devices featuring CPUs, GPUs and large FPGAs, such as Xilinx Zynq-7000 or Zynq UltraScale+ MPSoC architectures that are now prevalent in the market. This work serves as a proof-of-concept for implementation-based integration and paves the way for target HW infrastructure integration (between PREESM actors and ARTICo3 HW accelerators) to fulfill part of integration objectives of the CERBERO project.

7.2. Integrating MDC with HLS Tools

Coarse-Grained Reconfigurable systems may sound particularly appealing in the definition of modern runtime adaptive computing systems. Nevertheless, their design is complex and effort-hungry, which exacerbate the already present design productivity gap [11]. Therefore, high-level synthesis and automated design environment sound particularly useful, though very often limited by given platform- or vendor-specific tools.

In CERBERO, we are attempting to decouple as much as possible our technologies from specific vendors. Hence, we are in the process of integrating a target-agnostic dataflow-based design environment for coarse-grained reconfigurable platforms. Targeting this need, embedded system developers have invested a lot of effort in the development of advanced methodologies and tools for system design and test. The trend is the following. First, to leverage on model-based design methodologies, increasing the level of abstraction and re-using as much as possible IPs. Second, to adopt automated design flows to relieve designers from the burden of going deep down with the embedded system definition. High-Level Synthesis (HLS) tools are hailed as one of the most promising solutions to cope with the design productivity gap. Different tools and methodologies have been proposed by academy and industry, mainly differing for the adopted high-level model/language and/or targeted hardware. C code is the most commonly adopted high-level representation, together with its object-oriented (C++) and system-oriented (SystemC) versions. These representations are supported by the main commercial tools, such as Vivado HLS from Xilinx[12]. FPGA company tools (Vivado HLS and HLS Compiler) are typically limited to the vendor specific devices. Due to their modularity and abstraction capabilities, dataflow models of computation (MoCs) have also been adopted as high-level representations for HLS.

We are currently working on a platform-agnostic design flow for heterogeneous coarse-grained configurable (CGR) systems. Designers are requested to provide only the dataflow models of the applications/kernels to be implemented on the reconfigurable substrate. Those models are automatically analyzed and combined in a unique optimal reconfigurable multi-dataflow specification, which is then instantiated in hardware. The environment is built upon the combination of a dataflow-based design suite for CGR systems, MDC, and a platform-agnostic dataflow-to-hardware synthesizer (aka dataflow language), CAPH. The actors of that specification are synthesized using CAPH and the CGR platform in deployed by MDC. Designers do not have to deal with resource minimization, HDL definition and reconfiguration management. As a status quo, there are no other integrated environment capable of providing the same support in a completely platform-agnostic way.

8. Appendix A. Brainstorming on CERBERO Tools and Technologies.

In the context of defining CERBERO integration framework and operational interfaces, CERBERO consortium have conducted an internal survey of all relevant tools developed and owned by the partners. Proposed tools for application architecture and runtime development widely support Eclipse IDE and are already available as Eclipse plugins. High-level systems modelling and simulation are to be carried out under IBM AOW and TNO Dynaa. All proposed tools - apart from AOW and MECA - support both Microsoft Windows OS as well as various Linux distributions running on mainstream hardware architectures, and require C/C++, Python, or Java as programming languages.

Proposed tools vary in their level of maturity, though they currently assume TRL "3" as an overall minimum, which would allow CERBERO to progress smoothly focusing on tool interfacing and integration to fulfill the target use cases covered in WP2. Some of the tools are fully developed and already released while others are the outcome of completed EU research projects. CERBERO tools are available for free distribution under varying licensing schemes. Following is a breakdown of the survey outcome into details of each tool.

8.1. AOW

Architecture Optimization Workbench (AOW) is a tool for System of System (SoS) and System level multi-objective multi-view cross-level design using Mathematical Programming techniques. Currently, AOW allows models of metrics, views and levels using algebraic mixed-integer linear equations and inequalities. In CERBERO AOW will be extended to models using linear differential equations and inequalities as well. Using AOW, system (or SoS) designers can simultaneously find the optimal system (or SoS) architecture and control, including routing and reconfiguration decisions. In control domain, AOW will focus on Robust Optimization, Markov Decision Process (MDP) and Partially Observable MDP (POMDP).

Inputs:

* SysML or UPDM models of functional requirements, topology of physical architecture, geometry, potential mappings from functional to physical and allocations of physical to geometry

* Excel catalogs and inventories of components, geometry, mappings and allocations

* Library of metrics and goals

* Optimization configuration, aliases

Outputs: Set of Pareto-optimal designs (textual, graphical outputs and detailed SysML models) and control policies.

Solver: Cplex

Possible contributions to UCs

1. Self-healing system for planetary exploration

Reconfiguration controller using MDP or POMDP, optimal arm routing.

2. Smart traveling for electric vehicles

Routing decisions for most scenarios taking into account transportation network congestion, road and battery physical conditions (e.g., elevation of roads, discharging of batteries, etc.).

3. Ocean monitoring

Battery re-configuration, routing/navigation of vessels taking into account ocean and battery physical conditions.

8.2. DYNAA

DynAA (Dynamic Adaptive Multi Sensor Networks Architecture) is a system analysis and design tool based on a fully fledged DEVS simulator. The DynAA simulation engine combines features from system simulators (such as Matlab Simulink and Modellica) and network simulators (such as NS3 and OMNet++). With DynAA, the designer can model complex systems and simulate the interaction between its components in different scenarios and/or with different system parameter settings. During simulation, DynAA records system states which can later be visualized or quantified. In this way, DynAA offers insight in the key performance indicators of an application (model) and on its evolution through time – and as such facilitates early design and architecture decision making.
The DynAA tool provides:

1) Semi-integrated cross-layer modelling of computing and network components. DynAA has its own modeling language, similar to sysML, but emphasizing the functional view, the physical view, the communication view, and the mapping view. Models can be saved in XML format for integration with other tools or code generation.
2) The DynAA modelling language includes special modelling constructors for large and distributed system and modelling of reconfigurable aspects. For example, the DynAA language has:
   a. cardinality constructs that allows to describe and parameterize the number of component instances in the system.
   b. Network topological constructs that allows to describe how components are connected and how the network evolves when new components are added or removed
   c. Reconfiguration handle constructors that allows to explicitly indicates which components or model aspects are modified during adaptation/reconfiguration of the system.

   The features above make DynAA specially interesting for modelling and simulationg self-adaptive signal processing systems.
3) Models written in the DynAA modelling language can be transformed by code generators (also available) into Matlab code for simulation purposes in the DynAA simulator.

DynAA simulations can be controlled within a closed loop with an optimizer (e.g. using local/global hill climbing search, genetic algorithms, monte carlo, or grid-search). At each loop iteration, the DynAA model is automatically parameterized with a specific set of settings for which simulation produces KPI indicators. This loop allows an effective design space exploration. Such feature also makes DynAA an effective design tool, as it can go beyond analysis and search optimal solution within a parameter space.

TNO offers the DynAA modelling language and part of the tool chain, which was developed within the DEMANES project as a base and starting point for the CERBERO modelling and development approach.

Inputs for DynAA:

1) Models for the components of the system in the DynAA modeling language, similar to sysML, but emphasizing the functional view, the physical view, and the mapping view. Input models are able to fully model all three CERBERO layers: functional, physical, and communication, as well as the function-to-physical relationships.
2) For the code generation, models in XML file. Simulation code can be directly written in Java or Matlab as well, in this case no code generation takes place.
3) Modelling with graphical user interface is possible, by means of using a commercial software (MetaEdit+), but appropriate licensing is needed
4) The automated optimization loop (design space exploration) needs a parameter space file describing the design space, and key performance indicator cost functions written in Matlab or Java.

Outputs for DynAA:

1) Simulations in DynAA produce logfiles with historical track of system states (configurable for each system model). Such logfiles can be used to visualize and quantify key performance indicators for the system development. Also, correctness of operation can be used as a first validation trial.
2) When combined with the optimizer loop, DynAA automatically searches the design parameter space and indicates the best parameter setting according to a custom performance function.

DynAA will be used to simulate the models in the Smart Travelling for Electric Vehicles use case and will be further enriched with more formal models of computation and extended to support new system properties. Moreover, it will cope with system-in-the-loop simulation.

Possible contribution to use cases

1. Self-healing system for planetary exploration

Potential contributed that should be evaluated later:

1) DynAA could be used to simulate several different self-healing strategies in the early design phase.
2) DynaAA could also be used on evaluating the performance of the path planner under different scenarios, such as problems in sensor data.

2. Smart traveling for electric vehicles

DynAA can be used to run simulation modules in parallel with the SCANR simulation environment already used by CRF. Here the SCANR system is viewed as a system-in-the-loop and DynAA need to simulate its modules in real time. DynAA can also be used to model simulation modules and possibly generate part of the code needed to implement specific simulation modules. Finally, DynAA could be used to provide mechanism to run simulation (faster than real time) for different alternatives to calculate predictions and generate advice to the driver (needed in more complex use case scenarios). The generated advice could potentially also be used to automatically modify the behavior of the system (vehicle), as is done in autonomous driving. In the foreseen use case the advice will initially only be used to inform the driver.

3. Ocean monitoring

Suggestions:

1) DynAA can be used to simulate different resource allocations, for example, variants of the system where data processing is done local in the unmanned vehicles or done remotely in a base station far from the unmanned vehicle.
2) If the unmanned vehicles have to communicate among themselves under water, DynAA can be used to analyse their communication performance:
    a. when different communication technologies are used (WiFi, Bluetooth, USB, etc. (?));
    b. when different communication protocols technologies are used (e.g. gossiping, broadcast, etc…);
    c. under different scenarios that affect the communication medium. For that, models of the communication channels under water (noise, attenuation, reflection) can be modelled in DynAA
3) DynAA can be used to estimate the energy performance and battery lifetime under different system workloads and communication.

8.3. Verification Technologies

Verification Technologies (VT) will be involved in the implementation of several features of the CERBERO framework, in particular (i) cross-layer and multi-objective features; (ii) model learning and adaptation from data; and (iii), management of functional and non-functional requirements and their formulation in formal properties.
With respect to CERBERO cross-layer and multi-objective features, quantitative verification techniques will be integrated in order to provide correct-by-construction design. Quantitative verification techniques that will be implemented in VT can fill the gap and iterate with AOW and DynAA in order to verify and correct the models when needed. An important requirement for VT is support for initial creation of verification models with following update and maintenance based on changes in user requirements. AOW models and levels of abstraction. State-of-the-art model checkers will be integrated (and extended, if it will be the case) to formally verify properties related to functional and non-functional requirements in probabilistic and real-time models.

Concerning model learning and adaptation from data, VT will be used for formal verifying learning algorithms and adaptive learning agents. To support reconfiguration and adaptiveness, it will be developed new techniques aimed to cope with adaptation and automated repair, i.e., once a behaviour unsatisfies a requirement, the agent should be fixed without manual inspections.

Regarding the management of functional and non-functional requirements, VT can be used to verify several different properties that span from generic requirements (e.g., security, energy, dependability) to highly application-specific (e.g., the availability of charging points on electric vehicle network, etc.). The idea is to leverage Natural Language Processing techniques and domain ontologies to automatically extract requirements expressed in natural language and map them in Probabilistic Computational Tree Logic.

Possible contributions to UCs

1. Self-healing system for planetary exploration

Verification of arm trajectory with respect to properties regarding the generation of collision free motion paths.

Verification of properties related to forward and inverse kinematic models.

2. Smart traveling for electric vehicles

Management and verification of functional and non-functional requirements described in Section 4.2 of D2.1a

3. Ocean monitoring

In scenario “Marine robot propulsion and transport”, management and verification of requirements related to “reconfiguration of battery module in runtime”.

8.4. PREESM

The main features of PREESM are:

1) A fully automated mapping of computational tasks (actors) to multiple processing cores with the objective of optimizing statically the execution latency and the load balancing of cores.

2) A state-of-the-art and fully automated memory optimization of the generated application memory footprint.

The supported Model of Computation of PREESM is the PiSDF dataflow MoC. It offers a balance between predictability and application adaptivity to modified parameters. The automated mapping is based on a set of heuristics and coarse-grain simulation methods.

Possible contributions to UCs:

1. Self-healing system for planetary exploration

Hardware/software co-prototyping for self-healing system with hardware reconfiguration, automating tasks currently manually performed.

2. Smart traveling for electric vehicles

Support of computationally intensive real-time tasks of the driving simulator with a mapping to specialized processors or hardware/software platforms.

3. Ocean monitoring

Modeling and prototyping an environment-aware video compression for the rovers.

Inputs: heterogeneous MPSoC-level modelling.

* PiSDF Dataflow model of functionalities.

* Model of target components + means of communication (S-LAM Model of Architecture).

* Model of deployment constraints and some “physical” info.

Output:

* System Simulation.

* Metrics: Predicted values for optimization.

* Code generation for MPSoC.

Structure: Java, based on Eclipse

Planned developments in PREESM / SPIDER: 1) add modeling and DSE for non-functional requirements, 2) connect to HW tools on one hand and to system level tools on another hand with required extensions in DSE, 3) potentially, extend Spider to low frequency event based scenarios. There is a direct connection to the Space use case, for Smart Traveling it depends if these tools will be used for the implementation, in Ocean Monitoring there is a potential of connecting all CERBERO tools, more details are required.

8.5. PAPIFY / PAPIFY-VIEWER

Papify is an instrumentation tool based on the Performance Application Programming Interface (PAPI) library to monitor data-flow specifications in run-time. Papify can monitor different type of events which can be related directly or indirectly to performance metrics and energy consumption measurements.

Papify-viewer is a visualization tool for events, once a data-flow specification has been *papified*. Events are shown either in a per-actor or a per-partition basis.

Possible contributions to UCs

1. Self-healing system for planetary exploration

Papify and papify-viewer can provide the instrumentation and visualization required to monitor the self-healing system for performance and energy consumption in a heterogeneous platform. The feedback provided by papify might be employed to make system reconfiguration decisions at run-time based on the selected key performance metrics.

2. Smart traveling for electric vehicles

Papify and papify-viewer can provide the instrumentation and visualization required to monitor the involved cyber-physical systems for the actual implementation of the distributed smart travelling system over a network. The feedback provided by papify might be employed to make system reconfiguration decisions at run-time.

3. Ocean monitoring

Papify and papify-viewer can provide the instrumentation and visualization required to monitor a heterogeneous platform which implements vision-based algorithms. The feedback provided by papify might be employed to make system reconfiguration decisions at run-time to trade-off quality of experience (QoE) and energy consumption.

8.6. SPIDER

Spider performs the mapping of a real-time application at run-time and adaptively depending on application parameters and available cores. Spider optimizes the application latency and supports the PiSDF application representations developped in PREESM. Spider complements PREESM with dynamic application mapping.

Possible contributions to UCs:

1. Self-healing system for planetary exploration

Management of hardware/software reconfiguration based on the health state of the robotic arm.

2. Smart traveling for electric vehicles

Adaptive execution of the real-time tasks of the simulator with a mapping to specialized processors or hardware/software platforms

3. Ocean monitoring

Adapting to energy availability the video compression module for the rovers.

Inputs:

* heterogeneous MPSoC-level modelling

* PiSDF Dataflow model of functionalities,

* Model of deployment constraints and some “physical” information.

Output: System

* Execution and profiling,

* Runtime management of MPSoC resources.

Structure: C++, Based on Linux.

8.7. ARTICo^3

ARTICo^3 is a hardware-based computing architecture for high-performance embedded scenarios. It is able to provide dynamic and adaptable behavior in terms of computing performance, energy consumption, and fault tolerance, by using Dynamic and Partial Reconfiguration (DPR) of FPGAs to load one or several copies of one or several hardware accelerators.

Possible contributions to UCs

1. Self-healing system for planetary exploration

ARTICo^3 can provide the required levels of fault tolerance to the aerospace scenario. In addition, and since ARTICo^3 features a self-monitoring infrastructure, the system can autonomously recover from unexpected faults in the hardware accelerators, by first detecting them and then performing either DPR to mitigate transient errors or task migration (to another reconfigurable region using DPR or even to a software processor) to mitigate permanent faults.

2. Ocean monitoring

ARTICo^3 can provide different levels of computing performance and energy consumption for hardware-based implementations of the vision-based algorithms described in this use case. In-node processing can be performed on video streams of changing resolution (e.g. from VGA up to 4k) to provide the required Quality of Service (QoS) while at the same time extending battery life in the autonomous system.

8.8. Multi-Dataflow Composer (MDC)

Tool purpose:

Dataflow to Hardware tool. MDC is capable of automatic deployment of Coarse-Grained Reconfigurable accelerators. It is composed of two subparts.

- The Multi-Dataflow Generator (MDG) - Given an input set of dataflow specifications (functionalities to be executed in hardware), MDG applies datapath merging techniques in order to derive unique high-level specifications of the multi-purpose acceleration. At this stage, dataflow-level optimization can be performed to define constraints-optimal implementations.
- The Platform Composer (PC) – Given a multi-dataflow description, the HW blocks capable of implementing its actors and the communication protocol to be implemented among them, PC deploys a multi-functional hardware accelerator. You can derive pure RTL description of the accelerator datapath or a whole Xilinx Vivado compliant peripheral.

Features:

* Beneficial in highly constrained scenarios, MDC allows:

- static mapping of different specifications over the same reconfigurable coarse-grained substrate
- both area and power saving if input specifications have a common subset of actors.

* MDC is RVC-CAL compliant and is integrated with other MPEG-RVC tools capable of analysing (e.g. Orcc), optimizing (e.g. Turnus) and synthesizing (e.g. Xronos) dataflow networks.

* The automated flow at the moment supports Dataflow Process Network models but, in theory, any kind of dataflow MoC is compliant with the proposed approach.

Inputs:

* RVC-CAL compliant dataflow models (Dataflow Process Network) of the functionalities to be accelerated.

* RTL descriptions of the actors (either manually or automatically derived).

* Xml file with the communication protocol to be implemented among the actors

Output:

* Dataflow specification of the multi-functional network.

- * RTL description of the multi-functional hardware accelerator including a configuration Look-Up Table that, according to the required functionality ID, is capable of programming the switching elements in charge of the substrate programmability.

<u>Possible contributions to UCs</u>

1. Self-healing system for planetary exploration

MDC can allow effective power saving. In particular, ARTICo^3 is capable of implementing different functionalities over different architecture slots, MDC would allow a sort of slot merging to enable different functionalities over the same slot. Therefore, we foresee the possibility of combining MDC with ARTICo^3 to offer both effective power reduction on top of fault management in the aerospace scenario. Moreover, the coarse grained reconfigurable approach at the base of MDC may allow to automatically tune conflicting requirements (as specified below for the Ocean Monitoring case) quicker than adopting finer grained approaches.

2. Ocean monitoring

The Ocean Monitoring use case has planned to deliver an embedded system for acquisition, encoding and transmission of environmental images. This system is meant to continuously monitor F (e.g. decoding quality) and NF (e.g. remaining battery level) requirements. The coarse grained reconfigurable approach at the base of MDC may allow to automatically tune the encoding quality according to the remnant battery level thanks to approximation techniques.

8.9. MECA

MECA (Mission Execution Crew Assistant) is a tool that provides decision support for user of a CPS. It was originally developed for the European Space Agency for use in (hypothetical) space exploration missions. It facilitates the collaboration between humans and CPSs by aligning the communication at the right level of abstraction.

In simple terms, MECA gathers all information available, decided what information is needed to make a decision (discarding superfluous data), and present the options to the user. This presentation will be ranked based on the user model, which based on previous decisions by the user and represents the “preferences” of the user.

MECA will only be active in case interaction with the user is needed. In other words, as long as the system can optimize itself given the (changes in) the mission, the user is not involved in the decision making process. However, once the conditions of the mission change so much that the system can no longer find an optimization that satisfies all mission parameters, the user is prompted to make a decision (effectively changing the mission parameters such that a solution can be found again).

An example from the EV use case of this could be that two different routes are found that have identical distance and energy consumption, but one is a rural road and the other a high way. In this case, there is no clear optimal solution; both possibilities have the same solution in terms of mission parameters. In this case MECA may, for example, suggest the rural road option, based on the fact that the uses tends to prefer these kinds of roads.

In CERBERO, MECA will be further developed from a tool designed specifically for space exploration missions, into a generic tool that can be used for user decision support in general, and will be applied to one or more use cases.

Inputs:

* Possible solutions for optimization of the current scenario: For instance, for the EV use case this will be possible routes the driver can take.

* Additional relevant data: For example, for the EV use case this could be expected arrival times, points of interest along the route, traffic conditions, etc.

* User model (internal input): MECA will build a user model over time. It will record decisions made by the user and will in this way build up a model of the users "preferences".

Possible contributions to UCs

2. Smart traveling for electric vehicles

MECA will be responsible for the interaction between the driver and the car; in other words the "human in the loop". MECA will only be triggered if the conditions in the scenario change in such a way that the system can no longer find a unique optimal solution and input of the user is needed.

Based on the changes of the state of the car, or the scenario, DynAA will provide several different alternatives for routing for the driver. MECA will assess these alternatives using the user model, and suggest the user with the solution that is most likely preferred by the user.

3. Ocean monitoring

Like, in the scenario above, MECA could be used to provide the operator with a ranked list of options to alter operations if conditions in a mission change.

This option is still to be discussed.

9. References

1) Michael Masin, Lior Limonad, Aviad Sela, David Boaz, Lev Greenberg, Nir Mashkif, Ran Rinat, “Pluggable Analysis Viewpoints for Design Space Exploration”, Procedia Computer Science, Volume 16, 2013, Pages 226-235.

2) Gabor Simko, David Lindecker, Tihamer Levendovszky, Sandeep Neema, Janos Sztipanovits, “Specification of Cyber-Physical Components with Formal Semantics – Integration and Composition”, Springer, Part of the Lecture Notes in Computer Science book series (LNCS, volume 8107).

3) Manel Ammar, Mouna Baklouti, Maxime Pelcat, Karol Desnos, Mohammed Abid. “MARTE to PiSDF transformation for data-intensive applications analysis”, Design & Architectures for Signal & Image Processing (DASIP), Oct 2014, Madrid, Spain. 2014.

4) CPS Public Working Group Cyber-Physical Systems (CPS) Framework Release 1.0 (https://pages.nist.gov/cpspwg/).

5) Framework for Cyber-Physical Systems, Release 1.0, May 2016, Cyber Physical Systems Public Working Group.

6) CERBERO website, http://www.cerbero-h2020.eu

7) Smart Cyber-Physical Systems - The EU Framework Programme
https://ec.europa.eu/programmes/horizon2020/en/h2020-section/smart-cyber-physical-systems

8) Heiko Paulheim, Chapter 9, Ontology-based Application Integration, Springer, Berlin, ISBN 1461414296.

9) Ayad, Gasser; Nittala, Ramakrishna; Lemaire, Romain (2015)
“Automatic runtime customization for variability awareness on multicore platforms.” In: IEEE 9th International Symposium on Embedded Multicore/Manycore SoCs (MCSoC 2015), Turin, Italy, 23-25 September 2015. pp. 143-150.

10) L. Suriano, A. Rodriguez, K. Desnos, M. Pelcat and E. de la Torre, "Analysis of a heterogeneous multi-core, multi-hw-accelerator-based system designed using PREESM and SDSoC," 2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Madrid, 2017, pp. 1-7.

11) M. Pelcat et al., “Design productivity of a high level synthesis compiler versus HDL”, International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016.

12) Xilinx® Vivado® for high-level synthesis,
www.xilinx.com/products/design-tools/vivado/integration/esl-design

13) Evgeny Shindin, “Metrics Library Concepts and Usage”, Haifa Research Lab, IBM Corp., 2014.

14) IBM® Rational® Rhapsody® for model-based systems engineering,
http://www-03.ibm.com/software/products/en/ratirhapfami

15) Functional Mock-up Interface (FMI) Standard,
fmi-standard.org

16) MBSE case study, Internaitonal Council on Systems Engineering, January 2012. incoseonline.org.uk

17) Presentation of Modelica, Sébastien FURIC, INSA Lyon and LMS Engineering Innovation, France.

18) Janne Luoma, Steven Kelly and Juha-Pekka Tolvanen, "Defining Domain-Specific Modeling Languages: Collected Experiences", MetaCase, Ylistönmäentie 31, FI-40500 Jyväskylä, Finland

19) Domain-Specific Modeling Languages: Moving from Writing Code to Generating It, Steven Kelly, Microsoft, December 2007. https://msdn.microsoft.com/en-us/library/cc168592.aspx