Model-compilation challenges [for Cyber-Physical systems (CPS)]

Belgacem Ben Hedia, Etienne Hamelin, Chokri Mraidha, Sara Tucci Piergiovanni

Model-compilation challenges

[for Cyber-Physical systems (CPS)]

B. BEN HEDIA, *E.* HAMELIN, *C.* MRAIDHA *& S.* TUCCI-PIERGIOVANNI

CEA, LIST, Gif-sur-Yvette, France

{belgacem.ben-hedia, etienne.hamelin, sara.tucci@cea.fr, chokri.mraidha}@cea.fr

Abstract

There are several "disconnects" which need to be addressed to provide effective means for engineering Cyber Physical Systems (CPS). One of them is how to construct an optimized application starting from high-level specifications taking into account exacerbated interactions with the physical environment and in the same time addressing new paradigms like mixed criticality, and distributed multi/many-cores platforms. We introduce in this position paper a new methodology called **model-compilation** for Cyber-Physical systems. This methodology introduces new concepts and process that can be seen as a specification that tool editors and CPS application developers can integrate (instantiate) in their tool chain or development process.

Keywords: Cyber Physical Systems (CPS), model-compilation, real-time and embedded system, design space exploration, correct-by-design, mix-criticality

1 Introduction

Cyber Physical Systems (CPS) [25] can be considered as the next level of embedded systems [27]. Where an embedded system was mainly concerned with the computational elements for controlling systems, CPS are a network of complex interacting elements which require different physical inputs and outputs and therefore the stress is put on the interaction between separate elements rather than on the computation of a standalone device. However, most of the added value comes from this complex interaction between several devices and the physical environment[26]. The European consumer electronics industry is among one of the strongest players in the fierce global competition for leadership in the embedded systems domain and is now leading the way in CPS.

In CyPhERS project [9] report, for instance, we read: *the principal barrier to developing the field of Cyber-Physical Systems (CPS) is the lack of a theory and of application best practices that comprehend cyber and physical resources in a single unified framework. There are several "disconnects" which need to be addressed to provide effective means for engineering CPS*.

One need that became apparent while building several European project proposals on the development of CPS is to reduce the gap between high-level modeling tools (for instance: Simulink for control and Modelica for physical modeling) aiming at describing functional behaviors and structure, and the actual software implementation on the target platform.

Given the new CPS design challenges [26, 22, 39], the CPS domain needs to move forward from traditional approaches [22], where transition from model to software relies either on fully manual rewriting (a highly error-prone activity), or on code generation tools that can only perform a simple, unoptimized transformation, hard to fine-tune to a given platform architecture. These tools do not address distributed multi/many-cores platforms. Only few research tentatives aim at generating correct-by-design multi-task code from high-level modeling languages, e.g. [40] which addresses the modular generation of multi-rate solvers for Modelica models, and [20] which focuses on reliability aspects.

In our opinion these traditional approaches are not asking the right questions, and limits rapid application prototyping. During the ITEA2 project OPENPROD [19], we have worked on how to construct a software architecture and generate source code starting from a Modelica [3] model for a time-triggered execution platform. At the end of this project, we came to the conclusion that, before starting to build a software architecture and subsequently generate an optimized source code, there are many questions that need to be answered:

- How to generate an optimized software architecture and source code well suited for a given target platform?
- How to integrate the specificity of a given execution platform model (HW platform, communication and runtime support, mix-criticality) during this process, especially with distributed multi/many-cores platforms?
- How to construct the CPS software application structure that will take into account the structural elements available in the high-level model?
- How to construct a CPS process application that respects all requirements described in the high-level model, especially with mixed degree of dependability for different functions and data, through all construction steps, without systematic a posterior V&V methods. In other words how to make the CPS application correct-by-design?

To sum up, we need to construct an approach or methodology that allows the integration of existing tools and techniques and that meets two main concerns:

1. Design space exploration and optimization of software architectures for CPS applications.
2. Correct-by-design methods for CPS applications construction, from high-level model to binary code into execution platform.

This position paper aims at presenting the first rationale of this methodology and at defining key concepts in order to develop an emerging domain of research and technology that we call **model-compilation**: *this term denotes the semi-automated, multi-criteria optimized synthesis of dependable and efficient software implementations distributed on a network of multi/many-cores embedded computers, from abstract system-level models (including multi-physics hybrid control models).*

Integrated into a model-driven system development methodology, the **model-compilation** methodology will help system, control, software and safety engineers work together to generate a software implementation respecting all their concerns at once, in a holistic, iterative and de-verticalized way. By automating the optimization and trade-offs selection during the software architecture creation, the **model-compilation** methodology will also enable faster development cycles from concept to realization and validation, up to rapid-prototyping of complex, safety-critical applications and hence reduce the development costs and enhance the qualities of cyber physical systems in many industrial domains.

The **model-compilation** methodology can then be seen as a specification that tool editors and developers or CPS application design and development teams can integrate (instantiate) in their toolchain or adopt as a development process with the goal to construct their CPS application. Throughout this paper we illustrate the **model-compilation** methodology concepts on a use-case represented in figure 1.

This position paper is organized as follows: section 2 will introduce the new challenges of software architecture for CPS applications in terms of design space exploration and optimization; section 3 will detail the **model-compilation** methodology and how to integrate its different steps into existing toolchains or how to adopt it as a development process; in section 4 we present how the **model-compilation** methodology answers the challenges expressed in section 2 and its applicability to a use case; section 5 gives an overview of related works in the state of the art; and finally we conclude the paper in section 6

2 CPS challenges

The challenge of implementing a cyber-physical system, as many engineering challenges, can be described in the form of an optimization problem. Some parameters of this optimization problem will be specific to one application domain, however many parameters can be seen as generic problems, only to be weighted differently among use-cases. The optimization problem can be written in the form of:

$$\begin{cases} \text{Min}_{sys}\ Cost(sys) \\ Feasible(sys) \end{cases}$$

In other words: *"find a system implementation* (sys) *which minimizes the cost function, such that the system is feasible"*. In a traditional mathematical optimization formulation, the $Cost()$ function has as output a single scalar value, whereas the Feasible function is a vector of several feasibility constraints of the form $Feasibile_i(sys) \leqslant 0$

The $Cost()$ and $Feasible()$ functions are specific for each CPS problem; however they are built using generic building blocks, with estimations/weights specific to the problem. Within the Cost function, most CPS domains will enlist the aspects of the system that should be minimized or optimized:

- Cost aspects (accounted positive, to be minimized):
  - Recurring or per-unit costs: e.g. most hardware costs, cabling requirement, devices.
  - Non-recurring costs: e.g. specific HW and SW engineering and development costs, certification costs.
  - System usage costs: e.g. power consumption, maintenance costs.
- Quality and performance aspects (accounted positive, to be maximized):
  - Functional performance of the implemented functionality.
  - Reliability of components.

At this point it is important to note that, since the output of *Cost* function is a scalar evaluation of a proposed system implementation, a weight must be given to all cost components enlisted above: this weight distribution will drive the resolution of necessary trade-offs between performance and cost aspects (e.g. a "low-cost" vs. "premium" strategies). Within the feasibility function many CPS domains will enlist all their domain-specific engineering constraints:

- System minimal viability requirement:

$$Performance \geqslant required$$
(threshold for functionality to be granted)

- System resource constraints:

$$\begin{cases} SW\ memory\ usage \leqslant HW\ memory\ available \\ SW\ proc.\ \&\ comm.\ usage \leqslant HW\ capability \end{cases}$$

- Certification/qualification constraints:

$$Evaluated\ safety\ level \geqslant required\ level$$

The canonical method for minimizing $Cost(sys)$ such that $Feasible(sys)$ is satisfied is to explore the domain of feasible system implementations sys, while keeping a track of the best one. Most optimization solvers choose smart strategies to avoid exploring the whole feasible domain, through backtracking and iteration, and split the optimization process into several successive approximations.

The theoretical difficulty is that an accurate cost/performance and feasibility evaluation can only be performed on a concrete system implementation -which makes this optimization-driven engineering method just useless. Moreover, the engineering time spent on finding the optimum of this problem is also part of the non-recurring cost valuation, and may become crucial when time-to-market is a stringent requirement.

It thus becomes necessary to perform "approximate" cost & feasibility evaluations. However an implementation optimal wrt those approximations may consequently not be optimal to the exact cost & feasibility evaluation, should it be performed. It becomes useless looking for an optimal solution, one should rather look for an implementation "good enough" wrt. estimated cost/feasibility. Moreover estimations can be performed at intermediate steps in the design process, i.e. during phases in which the system implementation is only partially known or specified: on system design models. These estimations should however be checked again a posteriori on the concrete system, once implemented. A large effort is dedicated, in the CPS community, to developing tools and design patterns that help design more robust and simple model-based approximations of certain system costs/feasibility evaluations.

As an intermediate conclusion, we assume that CPS systems design is an optimization process, where design constraints and costs are rarely well specified and hard to evaluate accurately, and in which we only look for good-enough, but practical, implementations.

One classical way to speed up approximate optimization problems is to partition variables into mostly-orthogonal sets. An example is usually to split up CPS design into separate dimensions, e.g. with one team exploring the HW trade-offs (against estimated SW requirements), and another team exploring the SW implementation choices. However recent evolution in the CPS industry show that cross-domain optimizations are highly demanded:

- Embedded systems design used to involve both hardware and software engineers to explore cross-domain trade-offs for higher performance control and communication systems,
- Mechatronics systems design used to mix previous embedded systems teams with mechanical engineers to explore tighter system integration,
- Now Cyber-Physical Systems implies to cooperate with multi-physics modelling, instrumentation and control engineers, for flexible, distributed systems interacting with humans and physical systems.

The whole **model-compilation** challenge relies in:

- identifying how much of the approximate-optimization problem solving can be assisted by automatic computer reasoning;
- defining cross-domain interfaces that allow control, software, and hardware engineers cooperate on coherent models, to allow for multi-domain trade-off selection;
- setting-up practical tools through which multi-domain experts can cooperate with computer-aided design choices, in order to develop feasible, good-enough system implementations.

We call this process **model-compilation**, as a tribute to the software compilation process, in which a high(er)-level language is automatically transformed into a lower-level (assembly, then binary) implementation, through a roughly-optimizing compilation process.

When compilers were introduced, an assembly expert could certainly write a fine-tuned, high-performance implementation of a given algorithm, it would take so much time that the system performance drawback of using a compiled language is undoubtedly outweighed by the productivity of the software engineer. Today assembly is used only in very specific situations, and the performance drawback is moreover significantly reduced thanks to compiler and high-level languages improvements. We expect similarly that model-driven engineering will take over, thanks to general adoption of both model-driven engineering and stepwise improvements in the implementations generated by model-compilers, so that within a couple decades most of a CPS's implementation will be semi-automatically generated, and CPS software engineers only have to implement specific aspects such as low-level glue between HW and the generated SW part.

The **model-compilation** process can be seen as a multi-domain toolbox that enables control engineers, embedded software architects, and hardware designers share their domain-specific requirements and iterate on multi-domain trade-offs. The control engineer in particular expects to:

- express the CPSystem's functionality, often in the form of a block chain involving sensing, filtering and prediction, communication and actuation;
- validate system functionality, stability, robustness and reactivity in various scenarios, usually through MiL (model-in-the-loop) simulations involving interconnecting the controller model with a model of the physical plant under control;
- assess, at each iteration of the domain-space exploration process, that a given software/hardware implementation is functionally correct w.r.t. the original control model, e.g. through semantically-correct-by-construction software generation, or MiL or SiL (Software-in-the-Loop) validation.

The hardware engineer expects to:

- define a hardware architecture made of ECUs interconnected through networks, each node having specific computing structure (e.g. a specific SoC with single/multi/many-core architecture), and specific limitations (e.g. memory/computing/bandwidth resource);
- perform preliminary safety assessment of the effects of common HW failure modes:
- negotiate HW/SW trade-offs with SW engineer, e.g. upgrade a CPU computing power, or downgrade SW functionality.

The software architect expects to:

- assess software architectures, at several abstraction levels (functional component, task, runnable, or source-code block), for feasibility, performance, and safety metrics;
- explore software architectures with computer assistance towards a nearly-optimal choice;
- automatically generate a correct source code implementation of the chosen software architecture, that suits a specific RTOS API, and if necessary re-write only the performance-critical part of it;
- iterate quickly with hardware or control engineers on SW/control or SW/HW trade-offs.

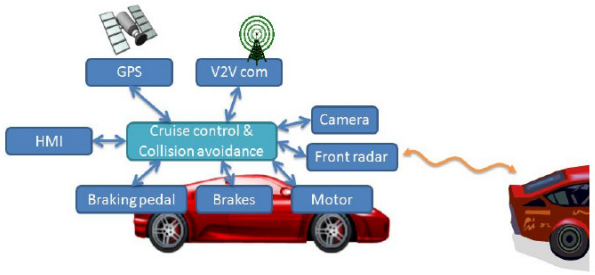

Fig. 1: Adaptive cruise control and collision avoidance system

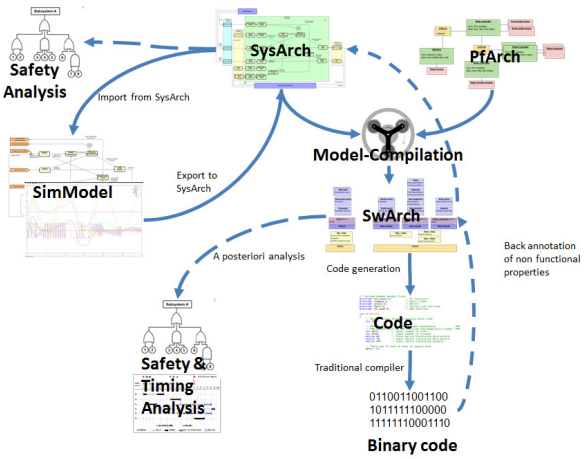

Fig. 2: Global **model-compilation** methodology

3 Model-compilation (methodology & approach)

In order to best present the complete end-to-end approach, and how it is integrated within a model-driven development process, this section highlights the application of the **model-compilation** methodology on an adaptive cruise control and collision avoidance system. This system is sketched by figure 1.

Figure 2 gives an overview of the **model-compilation** process. Like a third generation programming language compiler, the model-compiler is composed by a front-end, middle-end and back-end parts.

3.1 Front-end: from multiple heterogeneous high-level models

The system is first designed as a conceptual systems architecture, using a general purpose or domain specific architecture modelling tool, where main system constituents and their relations are identified as illustrated on the following Unified Modelling Language (UML) diagram. The Cruise control and collision avoidance system is composed of a set of sensors, actuators as well as a human-machine interface (HMI) and a controller.

At this level, a first safety analysis and risk assessment can be conducted to estimate the level of risk associated with specific items (elements of the architecture). Since we

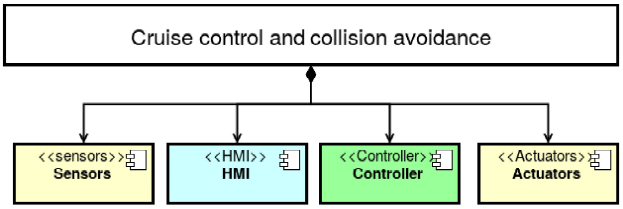

Fig. 3: Conceptual systems architecture

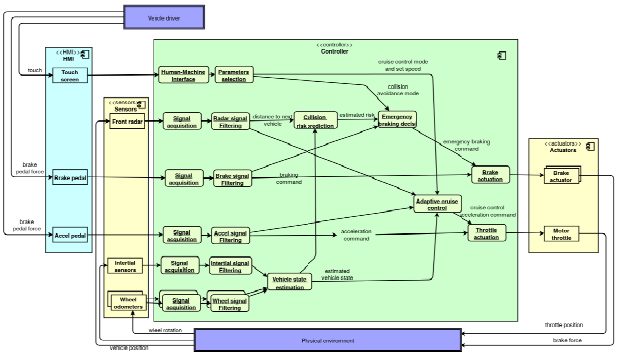

Fig. 4: SysArch: **Sys**tems **Arch**itecture model

are considering an automotive use case, for risk assessment we will use the concepts coming from the functional safety standard for automotive equipment, the ISO26262 standard. Accordingly to ISO26262, the risk level is specified through an Automotive Safety Integrity Level (ASIL). A Safety Goal is then defined for each hazardous event identified (e.g. "when the brake pedal is pushed, a braking torque must be applied to the wheels") as well as a set of essential safety requirements.

SysArch: systems architecture model The component-based design is then detailed as an abstract data flow diagram, illustrated below. This diagram is usually hierarchic, for sake of representation and understanding. This diagram will later be referred to as the functional architecture, or systems architecture: in brief SysArch.

This SysArch can efficiently be used to perform several design activities relevant at the systems abstraction level, as required by the ISO26262 standard. This activity includes the definition of functional safety requirements and their allocation to this preliminary functional architecture. At this phase the System Hazard Analysis (SHA) is conducted to study the propagation of failures across the system architecture. A preliminary safety assessment is conducted as well through Fault Tree generation and qualitative Analysis (FTA), Failure Mode Effects Analysis (FMEA) and Common Cause Analysis (CCA). These methods aim at analyzing fault propagation through the system and help in the definition of safety goals and ASIL criticality level. For instance, a functional chain that supports a ASIL C safety goal should rely on functional blocks assigned a criticality of ASIL C or higher.

This functional system architecture model can also be used for the specification of timing requirements. Some chains of functions can be identified and used to define global end-to-end deadlines (e.g. "latency from brake pedal

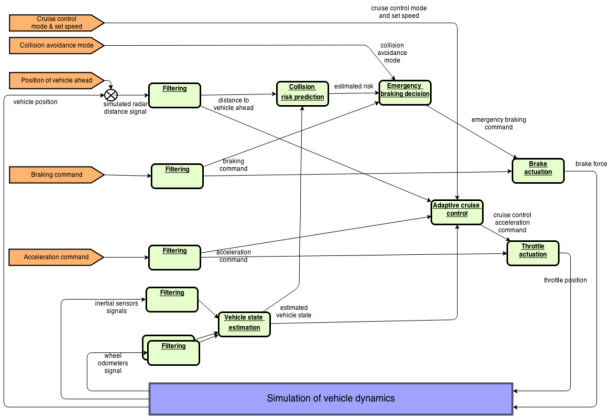

Fig. 5: SimModel: **Sim**ulation **Model**

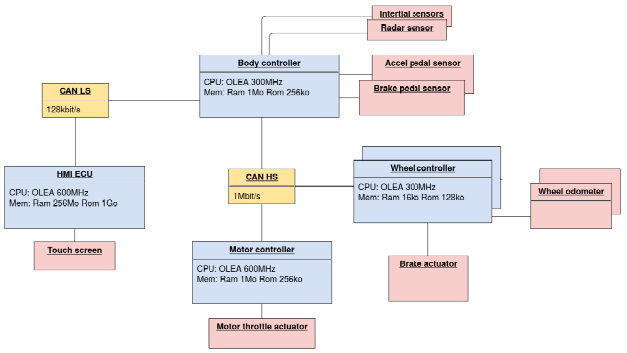

Fig. 6: PfArch: **P**lat**f**orm **Arch**itecture

signal acquisition to the corresponding brakes actuation should always be under 10ms“).

SimModel: the Simulation Model Then, for the control engineering team to work out the regulation principles, the data flow model is turned into a simulation model that will be used in particular to design and validate the controller regulation parameters, and assess their stability and robustness. This simulation is built from the same architecture, except that it includes a simulation of the environment (e.g. of the vehicle dynamics). The Simulation model is specified in a multi-physics modelling language.

Some blocks of the original systems diagram have been abstracted away: mainly the controller's interfaces to the external world are replaced with simulation stubs. For instance, some sensor blocks have been replaced by input signals (the orange blocks) that connect to external signal scenarios (e.g. a list of cruise control set speeds, brake pedal positions, etc. at given simulation time instants). Some other blocks (like actuators and sensors) are replaced with transparent connections to signals from/to the vehicle dynamics simulation or a simulation of the sensor or actuator's internals. The “human-machine interface” and “parameter selection” blocks are also abstracted away by the “cruise control mode & set speed” input signal block, because these blocks represent a graphical user-interface and menu handling functions, therefore are not suitable for multi-physics simulation.

Although the final software implementation of the whole cruise control and collision avoidance system will be realized by purely discrete computations, at this point both the vehicle dynamics and the controller may be specified with either discrete or continuous (i.e. time-differential) equations. Typically the “vehicle state estimation” block in the controller will integrate differential equations. Many parameters evaluated during this simulation will be used to drive the **model-compilation**.

PfArch: the Platform Architecture In parallel to the simulation modelling activity, a hardware platform model, here named PfArch, is defined. It can be represented by a graph of interconnected computing, communication, sensing/actuating resources as illustrated below. Each computation resource should be annotated with its own specific resource constraints, e.g. CPU speed, RAM and ROM memory limits, hardware peripherals available, maximum criticality allowed (i.e. ASIL qualification level); and similarly buses may be annotated with a bandwidth or any other required by later feasibility analyses.

In many cases, it can be allowed to change this architecture to some extent in order to meet the needs of the final software implementation. In this case, some production rules may be defined, together with an associated cost value, that allow the computer-assisted optimization to create a new platform architecture from the initial one if the application does not meet the constraints of the initial platform architecture (e.g. “another Electronics Component Unit (ECU) can be added on a bus, at some cost; a bus can be replaced with one with higher bandwidth, at some cost, etc.”).

3.2 Middle-end: model-compilation into software architecture

The middle-end part is at the heart of the **model-compilation** approach and lies in the optimized and efficient synthesis, from a system architecture model (SysArch), of a software architecture (SwArch) mapped onto the platform architecture (PfArch), that simultaneously satisfies safety, timing, resources and behaviour requirements. The **model-compilation** is implemented by a combination of optimization algorithms and heuristics for transforming, in an optimized manner, a system-level model into a software implementation mapped onto a platform architecture model:

1. Optimisation algorithms aim at finding a solution correct-by-construction, i.e. the transformation itself is proven to guarantee that the SwArch produced respects all the constraints related to timing, safety, behaviour, resources, etc. Such approach called multi-staged optimization approach for multi-objective optimization has been already investigated in our previous works [30, 42]. The approach was able to find correct solutions which respect a wide range of constraints including causality and schedulability, while identifying design trade-offs. Note that producing correct solutions means that the analysis supposed to verify correctness is included in the optimization algorithm as a set of constraints.
2. Heuristics fall in a trial-and-error paradigm: here the SwArch obtained with the heuristics must be anal-

ysed a-posteriori (i) to establish if all the constraints are respected and (ii) if the required trade-offs are met. Heuristics will be applied whenever the overall complexity of the problem hinders the application of deterministic optimisation algorithms. One example of efficient trial-and-error strategy is to use coarse estimates of parameters to produce several nearly-optimal feasible solutions (e.g. use only a coarse estimate of CPU utilization factor for task creation step), then apply fine-grain analysis only to those few solutions (e.g. validate schedulability with a detailed period & WCET analysis).

For the automated transformation to be possible, either as correct-by-construction generation or as analysis-driven trial-and-error, all concerns must be first formalized and quantified. Some requirements will be quantified as quantitative relationships that must be verified, whereas some other requirements will be defined as the maximization of some quantified objective. To enable multi-concern trade-offs, a single objective (usually a weighted sum of all sub-objectives) shall be defined. Different weights could be selected depending on the application. [13, 14] aimed to integrate the specificity of an execution paradigm (RTOS) during the **model-compilation** process by integrating some characteristics of the execution platform when the SwArch is generated. Since then, the generation process became more accurate, since it now integrates information about the paradigm of the execution platform.

It should be noted that both optimization algorithms and heuristics accept some degree of parameter uncertainty. For instance, worst case execution time estimates could be extracted from in-situ measures (when reusing a function or task), or safe over-approximations (when using static analysis tools on the actual software), or only coarse estimates when the function has not yet been implemented/generated.

SwArch: Software Architecture intermediate representation Concretely at the end of the **model-compilation** process, all functional blocks from the SysArch model are transformed into some software code mapped onto one or several tasks or runnables, each allocated to a CPU, and all data or events (the edges of the SysArch diagram) are mapped to variables or inter-task communication messages or to bus messages; all tasks and signals have scheduling parameters assigned, in such a way that on each computation unit all tasks are schedulable, messages are schedulable on each bus, all end-to-end latency constraints are met, no CPU resource is exhausted, items of different criticality levels are separated and monitored by safety elements qualified at the appropriate criticality level, etc. Several reports may be produced together with the SwArch, for instance: a schedulability report for each CPU, an analysis of end-to-end latency constraints, a safety analysis at the software architecture level demonstrating the respect of criticality-related constraints, etc.

In the following illustration, functions of different criticality levels are mapped onto tasks. Tasks of different ASIL levels are separated either physically or in software by a SEooC RTOS providing sufficient partitioning mechanisms. A first work addressing the co-simulation of the SimModel and SwArch is provided in [32].

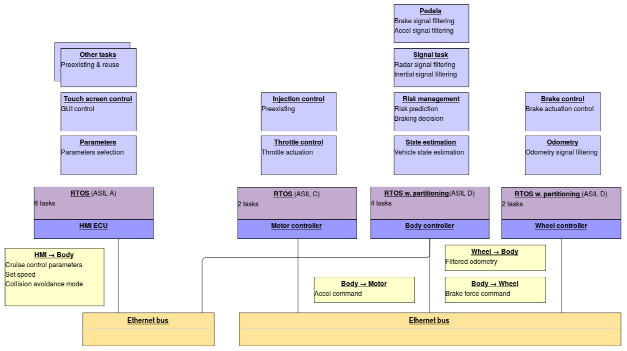

Fig. 7: SwArch: **S**oft**w**are **Arch**itecture output

3.3 Back-end: transformation into concrete target platforms

A rather simple model-to-code transformation back-end translates the SwArch model, into a set of files that actually implement in a software language e.g. C, all these tasks, and properly configure each CPU and RTOS instance. A first work [13, 14] was carried out to integrate a generic services of an execution paradigm (RTOS) during a **model-compilation** process and integrate some information about execution plate-form in the generated SwArch, with the intention of making easier the Back-end stage. The back-end of the **model-compilation** process then relies on off-the-shelf C compilers to generate the binary file that will be loaded on each ECU. This translation back-end is specific to the RTOS since it relies on the specific RTOS application programming interface (API) and services for the use of temporal control flow and inter-task messaging for instance. For some blocks of the SysArch where only a wrapper/placeholder was given in the simulation model, an empty task or runnable is generated together with the inter-task communication infrastructure, so that it still remains easy to manually add the manual implementation part. The wrapper & placeholder method facilitates the integration of existing tasks into the **model-compilation** process.

3.4 Design iterations

For each artifact of the SwArch model (task, runnable, message, etc.), a traceability annotation ensures that a straightforward link identifies the SysArch functional block or signal it was generated from and the design constraints that apply.

When the generated code is run on the actual platform properly instrumented, it is possible to measure relevant metrics at actual runtime: for instance measure actual execution times with real-time tracing/debugging tools. These runtime metrics can then, thanks to the traceability feature, be retro-annotated onto the SwArch model, then on the SysArch model. In many cases these actual metrics are much more accurate than the estimations given at first (if any), and can be used to perform another run of the **model-compilation** for a more optimized or more accurate synthe-

sis. This iterative design approach also allows an expert engineer to identify on actual execution abnormalities, violations of unexpressed or soft requirements, bottlenecks, sources of possible optimizations unseen before, etc. In this case, the engineer can annotate the SysArch model with different constraints that drive some steps of the **model-compilation** towards his preferred solution, applying the **model-compilation** process again.

4 Model-compilation methodology assessment

Here we address the question whether, and how far, the challenges defined in sections 1 and 2 can be resolved by implementing the approach illustrated in section 3.

Applicability of model-compilation approach The **model-compilation** methodology would be applied by first implementing all (semi-)automated analysis and optimization algorithms into already-existing tools, and then applying the development method to realistic use-cases.

1. **Integration/implementation by existing tools.** To effectively support the model-compilation process, analysis/optimization plug-ins and model transformations must be integrated to existing model-based development (MBD) tools:

   - A set of model analysis/optimization techniques will refine different aspects of the SwArch model. Most model-based development (MBD) tools allow to define specific languages and therefore support the SwArch models; moreover they support plug-ins and extensions mechanisms, within which feasibility, performance evaluation and optimization techniques can be implemented. Our previous work [13, 19, 14, 42, 30, 33] address different analysis & optimizations techniques as a proof of concept of this methodology.
   - The development process defined here certainly needs to rely on several engineering tools, therefore some bridges are required to support a seamless workflow along several iterations of individual steps of the **model-compilation** methodology. This generic toolbox can then be tailored by each engineering team into a product-specific development method, in which eventually a set of process guidelines can also be enforced through tool usage rules. Each process step should moreover be formalized with a set of hypotheses and guarantees, so that it could be eventually possible to verify the overall coherency of each specific workflow – this point will be further detailed in forthcoming publications.

2. **Realistic use-case application.** The illustrative use-case detailed in section 3 proves that the approach is applicable, through several analysis & transformation steps, on a virtual use-case representative of a realistic, non-trivial, industrial system.

Productivity enhancements The CPS development productivity will be illustrated on three main axes:

1. **(semi-)automation of individual steps.** Several tasks are automated or computer-aided, especially those most labor-intensive, with low systems-engineering added value, or most error-prone. Common MBD tools usually generate hardly-readable source-code in only one iteration, are poor at engineer interaction, with limited (if any) analysis or user-guided optimization facilities.
2. **Overall workflow acceleration.** With **model-compilation**, the CPS development workflow enables an exploration and optimization of SwArch. This allows to quickly evaluate several configurations of the software before choosing which one to apply for the final source-code generation. Existing tools generally generate code that is poorly structured, therefore adaptation to a specific execution platform (RTOS) or HW platform is very costly. Architectural exploration is subsequently only manual, and very long. Within the **model-compilation** methodology, all necessary information for the SwArch exploration phase are integrated, so that exploration steps are quickly iterated.
3. **Early feasibility assessment.** The feasibility and cost/performance evaluations performed at different phases during architecture exploration allow to early focus only on feasible alternatives.

5 Related works

This section presents the current state-of-art of some research areas relevant for CPS development, which in our opinion need further consideration.

Model-based methodologies for safety and timing Considering the whole body of work on development methodologies for systems and software is of course unfeasible. Nevertheless we are interested in giving an overview particularly focused on safety, timing and methodological management based on the use of modelling languages, which narrows the scope of this literary review.

Methodologies for safety management have been mainly proposed by T.Kelly and his group [18, 38, 17, 16]. These methodologies focus on how to build a safety case both at system and software level. Safety case patterns are proposed to ease this task [16], however, these approaches propose only an abstract methodology, i.e. they are not focusing on the use of either particular language (e.g. SysML, UML, etc) or models. This lack of concretization also implies a lack of methodology automation. A model-based approach has been proposed in the FP7 MAENAD project, where the GMP pattern methodology developed in TIMMO and TIMMO-2-USE projects has been instantiated on EAST-ADL models. This concretization allowed the automation of some safety activities such as FMEA and static and temporal FTA; still the automation support is partial with respect to the whole safety management, and safety case patterns were not integrated.

On the timing side, while a vast number of model-based approaches have been proposed for performance prediction [29], methodologies enabling analysis of timing constraints are more recent and are gaining a growing interest with the increasing complexity of embedded real-time systems [8, 4]. A UML based methodology based on the Y-chart principle has been proposed18 which then envisages the use of mapping algorithms (**model-compilation**) but not specifically focusing on timing analysis. A UML-based methodology for the analysis of objected oriented models but without proposing an automated mapping step has also been proposed [21]. Mraidha et al. [33] proposed a MARTE based methodology based on a scenario-based mapping and enabling schedulability analysis of mono-processor systems.

Note that the only methodological attempt, we are aware of, about the integration of safety and timing has been pursued in TIMMO and TIMMO-2-USE projects with the GMP pattern methodology. However this methodology is very general and do not specify how safety and timing management interact during the development process, this specification is left to the system engineer. New methodologies proposing a tighter integration of safety and timing activities throughout the development processes are needed to better support the development of CPS with mixed-critical constraints.

Model-compilation In the development of cyber-physical systems, abstraction levels must be used to manage complexity [24]. Industrial standards (like the automotive AUTOSAR [1] and the Model-Driven Architecture from the OMG [24]) and academic frameworks (including the Platform-Based Design [37]) recommend system development along the lines of the Y-chart approach [23]: a functional model representing the system functions and the signals exchanged among them (model often represented by Simulink-like models) is deployed onto an execution platform model consisting of nodes, buses, tasks and messages. Throughout this paper we call **model-compilation** the refinement of a functional model in its "execution counterpart": functions concretised by software modules (code) deployed across sofware (middleware, OS, etc...) and hardware (CPUs, memories, buffers, etc..) execution architectures, mostly distributed (as for CPS). Correctness of **model-compilation** implies that this functional execution counterpart must respect timing and safety constraint when running in the target platform.

On this topic, the current state of the art has been developed by two rather separated communities: the formal methods community focusing on functional model scheduling [28, 7, 36, 34] (with implicit assumption about a deployment on uni-processor platforms) and the real-time community focusing on allocation and scheduling of tasks models (with a large use of optimization and design exploration techniques), leaving the function-to-software transformation to the designer [15, 35, 11, 31, 10].

Providing a holistic optimization approach for **model-compilation**, i.e. optimization applied to the deployment of a functional model over a multi-processors architecture (multi-core and/or distributed). In this direction recent works, such as Mehiaoui et al. [30] and Wozniak et al.56 proposed a two-staged multi-objective optimisation technique able to output schedulable solutions for functions to be deployed in a distributed architecture. These approaches must be extended to specifically address safety constraints and mixed-criticality. Another important research topic about **model-compilation** concerns model transformations for Matlab Simulink models. While semantics preserving translations of Simulink models into tasks exist for discrete models, the translation of continuous blocks is an open research issue.

Physical modelling Historically, physical modelling and discrete control have mostly been treated as separate engineering activities. However, over the years, due to an increasing demand in tools covering a wider range of product lifecycle, the need for more integrated approaches has emerged [41].

Today, industrial applications of physical modelling are handled by tools providing informal semantic models mostly inspired from results in the field of continuous system simulation. Discrete aspects, despite being essential even in pure physical applications, are generally poorly supported by modelling tools, sometimes leading to important issues as soon as discrete aspects have to tightly interact with continuous ones [12, 5, 6].

Currently, tools essentially resort to their own semantic models, believed to be compatible with the operational requirements of some emerging standards such as Modelica [3] and FMI [2]. "High-level" proposals such as Modelica, VHDL-AMS and Verilog-AMS, and "low-level" proposals such as FMI attempt to provide a common basis for different users using different tools to describe models in such a way that tool interoperability and model exchange are hopefully possible. However, to the best of our knowledge, none of these proposals currently attempt to fully formalize the operational semantics required to reach the desired portability of hybrid models, especially their continuous-time part [6]. This means that a sound operational semantics for physical models is yet to be defined so that models coming from various sources (so including physical models) have to be combined in a sound way.

On the other hand, the situation is more comfortable in pure discrete-time applications since many semantically sound approaches have been proposed over the years to cope with them, one of the most prominent among them being the synchronous approach. Many tools targeting embedded applications (among which CEA's tools) support a form of synchronous approach. This approach allows one to reason about logical timing properties of models at a high level of abstraction, even before target code has actually been generated. This makes the synchronous level of abstraction an attractive "common denominator" for models that need to be checked for semantic compatibility and for qualification for embedded purposes. However, no tool allows physical models to be rigorously transformed into this intermediate form currently [6], which clearly constitutes an important challenge to be addressed.

6 Conclusion

In this position paper, we have defined a roadmap for a method of development for cyber-physical systems, to address the new challenges of rapidly designing multi-

viewpoint optimized implementations, that we call model-compilation. The proposed approach addresses two main concerns: design-space exploration and optimization of a software architecture, and correct-by-design construction from high-level models to source code and binary for a given execution platform. The model-compilation methodology can then be seen as a general specification that tool editors and CPS design teams can integrate (instantiate) in their toolchain, or adopt as a development process.

This roadmap is deemed feasible through several previous work and publications by the authors, but many aspects will be further detailed to confirm that the whole model-compilation approach can be efficiently implemented in industrial practice.

References

[1] Autosar 4.0 specifications. `http://www.autosar.org/`.

[2] Fmi standars. `http://www.fmi-standard.org`.

[3] Modelica. `http://www.modelica.org`.

[4] Cesare Bartolini, Antonia Bertolino, Guglielmo De Angelis, and Giuseppe Lipari. A uml profile and a methodology for real-time systems design. In *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on*, pages 108–117. IEEE, 2006.

[5] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78(3):877–910, 2012.

[6] Simon Bliudze and Sébastien Furic. An operational semantics for hybrid systems involving behavioral abstraction. In *Proceedings of the 10th International ModelicaConference*, number EPFL-CONF-199085, pages 693–706. Linköping University Electronic Press, Linköpings universitet, 2014.

[7] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, and Peter Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. In *ACM Sigplan Notices*, volume 38, pages 153–162. ACM, 2003.

[8] Rong Chen, Marco Sgroi, Luciano Lavagno, Grant Martin, Alberto Sangiovanni-Vincentelli, and Jan Rabaey. Uml and platform-based design. In *UML for Real*, pages 107–126. Springer, 2003.

[9] CyPhERS. Cyber-physical european roadmap & strategy, 2014.

[10] Werner Damm, Alexander Metzner, Friedrich Eisenbrand, Gennady Shmonin, Reinhard Wilhelm, and Sebastian Winkel. Mapping task-graphs on distributed ecu networks: efficient algorithms for feasibility and optimality. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 87–90. IEEE, 2006.

[11] Cagkan Erbas. *System-level modelling and design space exploration for multiprocessor embedded system-on-chip architectures*, volume 132. Amsterdam University Press, 2007.

[12] Sébastien Furic and LMS Imagine. Enforcing reliability of discrete-time models in modelica. In *Proceedings of the 8th International Modelica Conference*, 2011.

[13] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, Saddek Bensalem, and Jacques Combaz. Towards time-triggered component-based system models. In *The Tenth International Conference on Software Engineering Advances (ICSEA)*, 2015.

[14] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, and Saddek Bensalem. Externalisation of time-triggered communication system in bip high level models. *JRWRTC 2014*, page 41, 2014.

[15] Arne Hamann, Razvan Racu, and Rolf Ernst. Formal methods for automotive platform analysis and optimization. In *In Proc. Future Trends in Automotive Electronics and Tool Integration Workshop (DATE Conference), Munich*. Citeseer, 2006.

[16] Richard Hawkins, Kester Clegg, Rob Alexander, and Tim Kelly. Using a software safety argument pattern catalogue: Two case studies. In *Computer Safety, Reliability, and Security*, pages 185–198. Springer, 2011.

[17] Richard Hawkins, Ibrahim Habli, and Tim Kelly. Principled construction of software safety cases. In *SAFECOMP 2013-Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, page NA, 2013.

[18] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In *Advances in systems safety*, pages 3–23. Springer London, 2011.

[19] B. Ben Hedia and E. Hamelin. Model to embedded real-time real time embedded code transformation, 2012.

[20] Jia Huang, Simon Barner, Andreas Raabe, Christian Buckl, and Alois Knoll. A framework for reliability-aware embedded system design on multiprocessor platforms. *Microprocessors and Microsystems*, 38(6):539 – 551, 2014.

[21] Dongxi Jin and David C Levy. An approach to schedulability analysis of uml-based real-time systems design. In *Proceedings of the 3rd international workshop on Software and performance*, pages 243–250. ACM, 2002.

[22] S.K. Khaitan and J.D. McCalley. Design techniques and applications of cyberphysical systems: A survey. *Systems Journal, IEEE*, 9(2):350–365, June 2015.

[23] Bart Kienhuis, Ed F Deprettere, Pieter Van Der Wolf, and Kees Vissers. A methodology to design programmable embedded systems. In *Embedded processor design challenges*, pages 18–37. Springer, 2002.

[24] Stefan Kugele, Wolfgang Haberl, Michael Tautschnig, and Martin Wechs. Optimizing automatic deployment using non-functional requirement annotations. In *Leveraging Applications of Formal Methods, Verification and Validation*, pages 400–414. Springer, 2008.

[25] Edward A. Lee. Cyber-physical systems - are computing foundations adequate? In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, October 2006.

[26] Edward A. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[27] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Lee and Seshia, 1 edition, 2010.

[28] Roberto Lublinerman and Stavros Tripakis. Modular code generation from triggered and timed block diagrams. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 147–158. IEEE, 2008.

[29] Jack E Matson, Bruce E Barrett, and Joseph M Mellichamp. Software development cost estimation using function points. *Software Engineering, IEEE Transactions on*, 20(4):275–287, 1994.

[30] Asma Mehiaoui, Ernest Wozniak, Sara Tucci-Piergiovanni, Chokri Mraidha, Marco Di Natale, Haibo Zeng, Jean-Philippe Babau, Laurent Lemarchand, and Sébastien Gerard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. *ACM SIGPLAN Notices*, 48(5):121–132, 2013.

[31] Alexander Metzner and Christian Herde. Rtsat–an optimal and efficient approach to the task allocation problem in distributed architectures. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 147–158. IEEE, 2006.

[32] Matteo Morelli, Yasmina Seddik, Marco Di Natale, Chokri Mraidha, and Sara Tucci-Piergiovanni. Simulation-driven optimization of real-time control tasks. In *Proceedings of International Conference on Embedded Software and Systems (ICESS)*, 2015.

[33] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum: a marte-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, 2011.

[34] Thomas M Parks, Edward Lee, et al. Non-preemptive real-time scheduling of dataflow systems. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 5, pages 3235–3238. IEEE, 1995.

[35] Traian Pop, Paul Pop, Petru Eles, and Zebo Peng. Analysis and optimisation of hierarchically scheduled multiprocessor embedded systems. *International Journal of Parallel Programming*, 36(1):37–67, 2008.

[36] Marc Pouzet and Pascal Raymond. Modular static scheduling of synchronous data-flow networks. *Design Automation for Embedded Systems*, 14(3):165–192, 2010.

[37] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, (6):23–33, 2001.

[38] Mark A Sujan, Floor Koornneef, Nick Chozos, Simone Pozzi, and Tim Kelly. Safety cases for medical devices and health information technology: Involving healthcare organisations in the assurance of safety. *Health informatics journal*, 19(3):165–182, 2013.

[39] Paulo Tabuada. Cyber-physical systems: Position paper. In *NSF Workshop on Cyber-Physical Systems*, 2006.

[40] Bernhard Thiele, Martin Otter, and Sven Erik Mattsson. Modular Multi-Rate and Multi-Method Real-Time Simulation. In Hubertus Tummescheit and Karl-Erik Årzén, editors, 10^{th} *Int. Modelica Conference*, Lund, Sweden, May 2014.

[41] Vincent Berthoux Sébastien Furic Loïc Wagner and LMS Imagine SA. Statecharts as a means to control plant models in lms imagine. lab amesim.

[42] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci Piergiovanni, and Sebastien Gerard. An optimization approach for the synthesis of AUTOSAR architectures. In Carla Seatzu, editor, *Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA 2013, Cagliari, Italy, September 10-13, 2013*, pages 1–10. IEEE, 2013.