Translation, Abstraction and Integration for Effective Smart System Design

Translation, Abstraction and Integration for Effective Smart System Design

Michele Lora, *Member, IEEE*, Sara Vinco, *Member, IEEE*, Franco Fummi, *Member, IEEE*

Abstract—Virtual platforms are a powerful support for the development and early validation of embedded SW. However, complex smart devices are built by aggregating heterogeneous components provided by different vendors, thus requiring the development of custom ad-hoc virtual platforms. Even worse, components of the underneath HW platform may belong to different design domains, that are usually expressed using a huge variety of different languages. This high degree of heterogeneity in terms of both design and specification languages must be effectively managed by the design flow so to help engineers in assembling the final system.

This paper proposes a meet-in-the-middle approach to create virtual platforms of heterogeneous systems. The starting point is a set of heterogeneous models, developed by adopting the designer's favorite language and formalism. The methodology envisions the adoption of existing automatic translation and abstraction tools to automatically integrate models of components into a single homogeneous system-level executable description. The approach is supported by an analysis of the typical design flow, that leads to the definition of design domain/abstraction level taxonomies. Such taxonomies are then used to identify what characteristics would allow efficient system-level simulation, and the corresponding transformations to be applied to the starting models to achieve a "holistic" system executable representation.

The benefit of such an approach is particularly evident on any kind of highly heterogeneous systems, such as smart devices. The proposed methodology has been applied to two case studies with different degrees of heterogeneity, with the goal exemplifying its adoption on concrete scenarios and to prove its effectiveness.

1 Introduction

The field of Electronic Design Automation (EDA) has been heavily pushed in the last decades by the desire to stay in the track set by the Moore's Law. However, while the physical limits of Silicon are being approached, new trends started to emerge in computation: smartphones, the Internet of Things and pervasive computing technologies in general force designers to introduce sensors, actuators and communication devices within everyday more and more miniaturized systems: the *smart devices*. This trend forced EDA in the More-than-Moore era, where the main problem is no longer the integration of an increasing number of transistors in a single chip, but rather the introduction (and integration) of different, heterogeneous technologies within a single System on a Chip (SoC) [1].

A mandatory requirement for a device to be considered smart is *self-awareness*, i.e., it must be able of monitoring its own state, as well as its surrounding environment to properly adapt itself and to react to state changes [2]. Consequently, modern devices must be equipped with sensors and actuators to sense and interact with the physical part of system.

M. Lora and F. Fummi are with the Department of Computer Science, University of Verona, Italy, e-mail: name.surname@univr.it.
S. Vinco is with the Department of Control and Computer Engineering, Politecnico di Torino, Italy, e-mail: sara.vinco@polito.it.

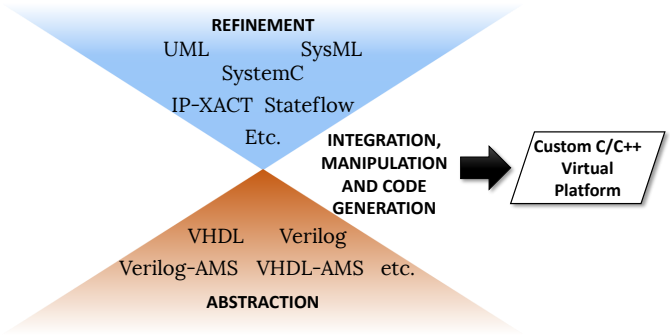

Figure 1: Overview of the proposed approach.

As a result, a variety of heterogeneous design domains are involved in the design process, ranging from digital and analog HW, to embedded SW, networking, system-level integration, *etc.* [3]. Each of these domains is handled by designers with specific expertise and background, by using tools and languages closed to the designer's expertise and to the domain of interest. This heterogeneity leads in fact to a babel of design and specification languages that must coexist in the same design flow, as exemplified in Figure 1.

Models heterogeneity becomes a primary issue when trying to perform overall system simulation, that is on the other hand necessary to allow the development of applications and of control SW running on the embedded device. In such a case, the correctness of SW functionality is not the sole concern of designers: it is also necessary to correctly evaluate the interactions of SW with the physical components of the system, and to take into account exact timing, in particular when dealing with real-time constraints. In general, the more properties have to be guaranteed, the more concerns must be considered while developing the SW [4].

SW development will thus require an accurate and executable model of the HW and physical components composing the underlying platform, *i.e.*, a *Virtual Platform*. To ensure effective design support, the virtual platform must thus provide efficient simulation of the entire smart device, and it must be both accurate and fast: two requirements that usually are in contrast with each other.

Accuracy in holistic system simulation can be achieved by carefully connecting different simulators typical of each domain, by building a co-simulation environment. Each simulator is in

charge of simulating a different part of the system, and the holistic view of the system behavior is given by the *integration* of simulators. However, integrating different environments is a time-consuming and error-prone task. The risk of incurring on integrations errors can only be reduced by using standardized interfaces [5], and standardized interfaces are not available for all tools and design domains [6]. Additionally, the overhead due to interprocess communication drastically slows down simulation.

Simulation speed is usually achieved through *abstraction*, i.e., by reducing the details of each model that must be simulated to the minimum. However, automatic abstraction is not always available, and manual abstraction may lead to unwanted accuracy losses.

Objectives and novelty

The main target of this work is to **organize into a coherent and theoretically sound methodology all partial attempts to integrate heterogenous components of smart device into a homogeneous virtual platform (the so called holistic model)**. The underlying idea is to allow designers to develop each component with his/her favourite language and formalism, to take full advantage of the designer's expertise. Our methodology aims at reconciling the resulting heterogeneity in terms of languages, formalisms and levels of abstraction by building a single monolithic, holistic and homogeneous model of the overall platform, by targeting a single language and abstraction level, as suggested by Figure 1. This homogeneous representation of the system under design can then be the starting point of additional design steps, since refinement, exploration of alternative configurations, and validation and verification challenges can now be solved on a homogeneous description rather than on a babel of abstraction levels and languages.

To achieve these goals, the methodology builds upon an analysis of the current design flows, to identify the main languages, domains and abstraction levels involved. This analysis has been formalized through the definition of design domains/abstraction level taxonomies, that allowed to identify the main ingredients for an efficient system-level simulation.

The taxonomies allow to identify the possible starting points of the heterogeneous system, in terms of typical abstraction level and language of the various components. From here, it is possible to define how the starting heterogeneous descriptions can be integrated and manipulated to obtain a monolithic executable model for holistic system simulation, thus building a custom virtual platform. For the sake of efficiency, we thus aim at building a homogeneous custom virtual platform: the resulting models are specified in a single language, *i.e.*, C++.We indeed advocate that reconciling the heterogeneous descriptions into a single monolithic and homogeneous model will allow us to drastically increase simulation speed, as already partially shown in [3].

To reach this result, the methodology relies on a set of translation, abstraction and integration approaches available at state-of-the-art. In particular, we adopted a set of approaches we defined in the recent years, that reconcile the different initial formalisms into an intermediate format (such as [7]–[11]). However, such transformations should not be considered the core of this work, as the methodology is general and it would still hold when adopting different abstraction/refinement approaches.

This work is organized as follows. Section 2 provides the necessary background and positioning of the proposed flow w.r.t. the current state of the art. Section 3 elaborates on the proposed design domains/abstraction levels taxonomies, to identify the target configuration of the holistic system. Section 4 presents the resulting flow, and references some supporting methodologies. Section 5 exemplifies the proposed approach on two smart system case studies, and Sections 6 and 7 draw our conclusions.

2 State of the Art and Background

2.1 Typical smart systems design flows

Heterogeneity is a well-known problem in the literature of embedded and smart systems design. In the past, many approaches have been proposed, that might be categorized according to the "direction" of their design flow: *top-down* and *bottom-up* [12].

Top-down design starts from a set of high-level requirements and reaches the final implementation through a sequence of refinement steps, that gradually introduce new details of the system implementation. Model-based Design (MBD) is nowadays one of the most accepted top-down approaches [13], [14]. Many different languages have been proposed to support MBD, e.g., UML [15] for SW systems, SysML [16] and Architecture Analysis & Design Language (AADL) [17] for system-level modeling, the specC language for the specification of digital SoCs [18], and Modelica [19] for physical dynamics. However, none of these languages is able to capture all the aspects of a single heterogeneous system.

Nowadays, a number of complete frameworks for the modeling and specification of complex dynamical systems are available. PtolemyII [20] is a framework supporting multiple Models of Computation (MoCs) to model complex heterogeneous system. The same concepts are applied also by commercial tools and frameworks, such as Simulink [21], SystemVue [22], and LabVIEW [23]. However, such frameworks require a combination of graphical, mathematical, and procedural languages. Furthermore, the modeling of a complex system usually relies on a plurality of MoCs, further introducing heterogeneity in the design process.

The main drawback of top-down approaches is the lack of components reuse [12]. Previously designed and verified Intellectual Properties (IPs) can not be easily integrated, and the designer is in fact required to re-model also already available components. This is a major limitation since reuse of IPs is a powerful resource to enhance time-to-market and reduce design costs [12].

The first attempt to build a top-down design flow contemplating also components reuse has been the methodology associated with the SpecC specification language [18]. The design is refined throughout a set of well-defined abstraction levels. Then, at each level the components library of IPs may be used to build a simulation model. Thus, designers may focus on designing the hardware platform communication protocols and functionalities. However, the methodology comes with two drawbacks: components may not be available at each abstraction level, and the connection of IPs requires to produce ad-hoc transducers. Furthermore, IPs involved in a heterogeneous system are usually expressed by a diverse plethora of design languages.

Bottom-up design flows start from a set of previously designed and verified components that are aggregated to implement the required functionality, thus maximizing reuse. Bottom-up approaches heavily suffer the Babel tower of design and specification languages: components belonging to different design domains are usually written in different design languages, and, even within the same domain, each designer usually has her preferences about modeling languages and tools [3]. In order to obtain a holistic view of the system, it is thus necessary to build and employ *co-simulation environments*, where different simulators collaborate with each other to emulate the entire system [24]. The main problem of co-simulation lies on the synchronization

of different tools and languages: communication must often be implemented manually, thus leading to possible integration errors [25], and the introduced overhead of inter-process communication is substantial [26]. Even the current state-of-the-art framework for co-simulation, the Functional Mock-up Interface (FMI) standard, does not overcome these limitations. Despite of supporting both model exchange and co-simulation of dynamic models, FMI does not support the integration of components modeled with different models of computation, and the presence of different synchronization mechanisms heavily impacts on simulation performance [5], [6].

Over time, Platform-Based Design (PBD) emerged as the most prominent solution for combining the advantages of both top-down and bottom-up approaches, through a "meet-in-the-middle" approach [4]. The key concept of PBD is the *platform*, *i.e.*, a library of basic components that can be assembled to implement the functionality of the system [27]. Each element of the library implements a functionality, and it is usually characterized by a set of performance parameters. It is worth noting that not all elements in the library are reusable components, rather they can merely be a specification of the implemented functionality. A *platform instance* is a set of components of the library that are actually instantiated in the system, in order to implement the system functionality.

The process goes both bottom-up and top-down: the top-down flow maps an instance of the functionality of the design into an instance of the platform, while the bottom-up direction defines a platform by choosing its components and an associated performance abstraction. When the two flows meet in the middle, functionality meets the platform, and it is necessary to define a common semantic domain, to enable formalizing and automating the mapping of functionality to platform elements to create an implementation [27]. While the description above gives an abstract definition of *platform*, in the industry many different definitions have been proposed. In this context, the term *Virtual Platform (VP)* indicates an executable model for the simulation of an HW architecture running a SW application. In the last decades, virtual platforms became a popular tool to develop, test and verify embedded SW running on an embedded HW architecture [28]. In this sense, virtual platforms may be considered as an instance of the concept of platform defined by the PBD.

Of course, the bottom-up phase requires characterizing components at higher abstraction levels. In order to support orthogonalization of functionality and architecture, MetroII [29] supports annotation of extra-functional properties into high-level executable models, while Metronomy [30] extends it to support timing annotation. The values to be back-annotated to high-level models may be estimated in different ways: timing estimation of software execution on the concrete platform may be performed by using instruction set simulators [31], exploiting Source Level or Symbolic simulations [32], or by instrumenting the code at compile-time software and analyzing its execution [33]. All of these approaches provide very good estimation. However, the approach presented in this paper bases its timing on the clocked execution of digital components, and the evolution of differential equations in analog and continuous time models preserved by the used abstraction techniques [7], [9]. Other abstraction techniques have been proposed in the area of formal verification [34], [35]. However, these approaches are not well suited for holistic validation of systems, and do not target holistic system simulation.

2.2 Virtual platforms

A virtual platform provides a prototype of a system for simulating its functionality and evaluating its correctness before going to production. virtual platforms are a powerful resource when developing embedded SW: given that the virtual platforms is accurate enough with respect to the actual HW platform,SW can be run and tested on a virtual platform before being actually deployed on the final HW architecture, and simulation allows to gather information about SW correctness, both concerning functional and extra-functional properties, such as timing, memory usage and power consumption [28].

Despite of their potential, virtual platforms are not universally exploited to master the complexity of the embedded SW development process. Given that building a virtual platform for a new custom HW architecture requires to invest a huge amount of time and money [28], virtual platforms are indeed used only whenever the system being designed relies on a standard, fixed HW platform, as the major EDA vendors provide virtual platforms of the main embedded architectures available [36]–[38].

Thus, the state-of-the-art lacks of virtual platforms for custom architectures of smart devices. The Open Virtual Platform (OVP) initiative [39] proposes an approach to virtual platforms that tries to overcome the above limitation. It provides a library of both proprietary and open-source HW IPs that can be composed to build a custom virtual platforms. Different approaches have been proposed in the literature to further customize the models, by integrating custom peripherals and components in OVP [40]. However, none of the approaches tackles the creation of platforms for heterogeneous systems, i.e., systems composed by a combination of digital, analog and mixed-signal components.

In this paper, we present an approach that aims at generating virtual platforms of heterogeneous smart systems, starting from the set of heterogeneous descriptions of the system components. As such, the approach aims at enabling a systematic exploitation of virtual platforms by mastering the heterogeneity of languages to create homogeneous custom virtual platforms of heterogeneous smart systems.

3 Taxonomies for smart systems design

The first step to tackle the heterogeneity of smart system design is to identify the main languages, domains and abstraction levels involved, as only this knowledge allows to derive the typical design space and heterogeneity. We choose to formalize this analysis through the definition of taxonomies, that divide languages and modeling styles with respect to the domain and the abstraction level of each class of components.

3.1 Former efforts

Our former works [3], [41] are our first attempt to describe and formalize the heterogeneity involved in smart systems design through the definition of taxonomies of languages and tools. The taxonomy presented in [3] identifies a set of abstraction levels and design domains typical of smart systems components descriptions. The taxonomy was used to categorize the tools and languages used within the SMAC European Project [42] to model heterogeneous components of smart devices. In [41] we generalized such a taxonomy by considering the tools generally used for the design of heterogeneous systems. Then, we applied the same analysis to the MoCs, thus introducing a novel taxonomy focused on the MoCs supporting the tools.

These taxonomies have been useful to identify the typical starting point for the design of smart devices, that is the abstraction level at which a component belonging to a certain design domain is usually initially provided. Then, the taxonomies allowed us to define the target level of abstraction of each component and of the overall system for our methodology, with the goal of increasing the simulation speed for holistic system simulation.

The main effort of the current work consists of a more sound and general organization of the taxonomies, to abstract from the actual application scenario to cover the whole design space of smart systems. Taxonomies have thus been rationalized, clarified and extended with a new focus on communication and synchronization, and by adding a new level to the classification of abstraction levels. This allows to cover all transformations from the heterogeneity to the holistic level, and to reconcile to a common framework existing transformations.

3.2 Taxonomy of models of computation

Figure 2 reports the taxonomy of the most widespread MoCs for smart systems design: rows show the different levels of abstraction, while columns list the typical design domains. It is important to note that, when dealing with MoCs, it is necessary to consider not only the definition of the components (*i.e.*, what is an actor and how it evolves), but also the concurrency and communication mechanisms (*i.e.*, how actors act together and exchange information). As such, the taxonomy has been enriched with a further dimension (last column of the Table): *Synchronization & Concurrency*, that captures the MoCs governing the interaction between components at a certain abstraction level. From the bottom to the top we can see:

- *Physical level:* all the components are described as a set of physical dynamics. Descriptions of Analog, Micro Electro Mechanical Systems (MEMS) and Physical components come naturally. Network and Communication models describe the properties of the physical communication channel. Digital HW is described with models capturing electrical, thermal and power properties of components. At this abstraction level, any model of the SW would be meaningless. All the models at this level rely on differential equations, and thus rely on the *continuous-time* MoC, that is also used for their synchronization.
- *Structural level:* the system is partitioned in components belonging to different design domains. Only the Analog, MEMS and Physical components are still modeled as a set of physical dynamics through differential equations. Components belonging to the Network and Communiation domain can be described as actors evolving concurrently and reacting to primitives (*i.e.*, *Discrete Events*). Digital HW models are modeled as a set of communicating actors. These models continue to evolve periodically, executing a set of simulation cycles driven by events, thus following the delta-cycle concept used by Hardware Description Language (HDL) simulation kernels. As soon as the composition of the models reaches a *fixed-point*, the simulation moves to the next period. As such, they are usually modeled by using a combination of the *Synchronous-Reactive* and the *Discrete Event* MoCs. The Structural level allows to model also Embedded SW as a set of actors carrying on sequential operations and communicating through messages; thus, Embedded SW can be modeled as a set of *Communicating-Sequential Processes*.
- *Functional level:* at this level all components of the system may be described by defining their functionality. Physical parts of the system are still modeled by differential equations, while each of the other components (*i.e.*, Network and Communication, Digital HW and Embedded SW) can be modeled by using any MoC suitable to describe its functionality. Communication and synchronization are based on data flowing between system components (*i.e.*, *Dataflow*) to reproduce the holistic behavior of the system.
- *Transactional level:* even if this level is very similar to the previous one, here the focus moves from functionality to communication. Components synchronize and communicate only at certain synchronization points, according to precise communication protocols: concurrency is thus modeled according to the *Rendezvous* MoC semantics.
- *Holistic level:* here the system is considered as a "monolithic object". At this level, designers are interested in monitoring a set of well-defined events that may happen in the system: all components are thus aggregated into a unique homogeneous model, that represents system behavior as a sequence of *Discrete Events*. This abstraction level gives an holistic view of the system in the sense that it is built by considering all the parts of the system, as well the interactions between the parts, to create a unique view of the system.

Figure 2 might seem to suggest an order relation among the abstraction levels, that in fact does exist. Figure 3 reports the order relation over the considered abstraction levels, where an arrow from a level to another states that the level is less abstract than the other. From the Figure it is evident that the set of abstraction levels is partially ordered. In fact, no order relation exists between the *Transactional* and the *Functional* levels: they are indeed not comparable, since the Transactional level focuses on modeling the communication between system components, while the Functional level focuses on the sole functionality.

The taxonomy allows us to identify the target abstraction level for efficient system-level holistic simulation. The target model must be able to capture all the input/output events of the system that are relevant for the designer. As such, the entire system can be seen as a sequence of events that are "of interest" for the engineers. Consequently, the *Holistic* level of abstraction is a perfect candidate for system-level simulation.

3.3 Taxonomy of tools and languages

In the practice, each component is described by using different languages, tools and formalisms, depending both on the design domain and the abstraction level. Figure 4 summarizes the most common choices among designers. Figure 2 and Figure 4 are related to each other: the tools in each entry of the latter table are based on the models of computations listed in the corresponding entry of the former. As for the previous taxonomy, we hereby discuss the entries from the lowest level of abstraction up to the most abstract:

- At the *Physical level* simulation relies mostly on continuous time simulators. SPICE-based simulators are used to simulate digital and analog HW components, as well as communication devices' circuitry. Finite Element Methods (FEM) and 3D CAD simulation tools are used to simulate non-electronics physical elements. Extra-functional feautures, such as thermal behavior, power consumption, electromagnetic interferences are evaluated by using ad-hoc simulators.
- At the *Structural level*, analog, MEMS and physical components relies on SPICE-based simulators and Analog-Mixed

Abstraction Levels	Analog, MEMS and Physical	Network and Communication	Digital Hardware	Embedded Software	Synchronization & Concurrency
Holistic	Discrete Events				Discrete Events
Transactional	Continuous Time	Synchronous Data Flow, Discrete Events, State Machine, Rendezvous			Rendezvous
Functional	Continuous Time	Synchronous Data Flow, Discrete Events, State Machine, Rendezvous			Synchronous Data Flow
Structural	Continuous Time	Synchronous-Reactive, Discrete Events		Communicating-Sequential Processes	Synchronous-Reactive, Rendezvous
Physical	Continuous Time			--	Continuous Time

Figure 2: Taxonomy of the main MoCs used for heterogeneous system design.

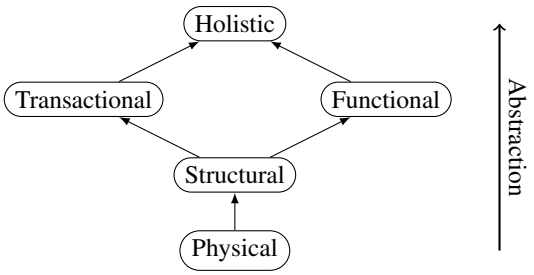

Figure 3: Partial order relations over Abstraction layers.

Signals (AMS) extensions of HDLs. Ad-hoc network simulators are used to model and simulate networking and communication scenarios. Digital HW simulation relies on classic HDLs (*e.g.*, VHDL and Verilog). Embedded SW can be executed by using emulators or Instruction Set Simulators. Figure 4 defines a variation of the Structural level called *Homogeneous Structural level*, that exploits SystemC and its extensions to cover all the design domains. Using different tools and languages forces to use complex co-simulation interfaces to achieve holistic system simulation. The homogeneous structural level allows instead to delegate to the SystemC simulation kernel the entire system simulation synchronization. Another potentially suitable language to implement such an abstraction level would be Verilog, exploiting its Verilog-AMS and SystemVerilog extensions. However, Verilog is not yet equipped with a network extension.

- At the *Functional level*, the functionality of each design domain may rely on tools that implement different models of computation. However, the predominant choice of designers falls on multi-domain modeling and simulation tools, such as (but not restricted to) the ones listed in the Functional abstraction level entry of Figure 4.
- The *Transactional level* mainly relies on SystemC TLM and SystemVerilog, supported by their AMS extensions, that are the most suitable alternatives to create homogeneous models strongly focusing on communication.
- The *Holistic abstraction level* aims at providing a as fast as possible simulation environment, preserving the events of interest for the designer. As such, in this paper we propose to implement this abstraction level by producing *custom C/C++ homogeneous virtual platforms*, that are event-level equivalents to the system under design.

This taxonomy allows to correctly identify when simulation can be used, and when co-simulation is necessary. The taxonomy shows that the lowest levels of abstraction require to exploit co-simulation: the different parts are usually modeled by using tools and languages that are specific for each domain, and it is thus necessary to connect different simulators for the different design domains involved. Vice versa, simulation can be used whenever the abstraction level of the descriptions are raised enough to allow using "generic" system-level simulation technologies.As such, more abstract models will provide two positive effects. On one hand, abstract models allow to simulate fewer details. Furthermore, abstraction allows to remove the resource consuming co-simulation interfaces, thus further speeding up the emulation of system behavior.

The following sections will present how, starting from a set of components described at the lower levels of abstraction, it becomes possible to get automatic generation of custom C/C++ homogeneous virtual platforms of the system for efficient *holistic system simulation*.

4 Proposed Translation, Abstraction and Integration Flow

The taxonomies defined in the former section lead to two main results. On one hand, given a heterogeneous system, it is now possible to categorize each component in terms of its domain and its main characteristics, i.e., abstraction level, model of computation and language. This leads to a clear view of the ingredients composing the system under design. On the other, the analysis of the taxonomies highlighted that the target of the flow must be a holistic simulation of the system: the target virtual platform must indeed capture all relevant events in a homogeneous view of the system, so that all components can be implemented in the same language and that no co-simulation infrastructure is needed. This guarantees effective and efficient simulation of the overall system.

As a consequence, it is necessary to identify flows that allow to move across the different layers of the taxonomy, to enhance the transformation of the system components from their starting description to the holistic level of abstraction. As anticipated by Figure 1, the direction is twofold. High level specifications of the system (top of the Figure) must be concretized, e.g., to derive information about the structure of the system and the interconnection of components, or to derive an implementation of a component from its abstract specification. This direction is thus a typical *top-down flow*. The complementary direction is a *bottom-up flow*, where the existing components are manipulated to bring their starting implementation to the holistic level.

The methodology aims at supporting the different specification languages and tools usually employed in the various design domains involved in smart devices, such as digital and analog HW, embedded SW, network, *etc.*. As such, the methodology shall support a wide range of languages, ranging from Verilog-AMS to UML. To reach this result, the methodology relies on a set

Abstraction Levels	Analog, MEMS and Physical	Network and Communication	Digital Hardware	Embedded Software	Emulation Technique
Holistic	Custom C/C++ Homogeneous Virtual Platforms				**Simulation**
Transactional	SystemC AMS Verilog-AMS	SystemC TLM SystemVerilog			
Functional	Mathworks Simulink, National Instruments LabVIEW, Keysight SystemVue, Modelica distributions, PtolemyII				
Homogeneous Structural	SystemC AMS	SCNSL	SystemC (RTL)	SystemC TLM, C/C++	
Structural	SPICE, VerilogA and VHDL AMS using SPICE-based simulators	NS3, OMNeT++, OPNET, SCSNL, etc.	Verilog & VHDL using HDL Simulators, e.g., Mentor's Modelsim, Cadence Incisive, etc.	QEMU, OVPSim, other ISSs	**Co-simulation**
Physical	CoventorWave, MEMS Pro, SPICE-based simulators, FEM and 3D CAD simulators	Spectre RF, EMPRO, Keysight ADS, SPICE-based simulators	SPICE-based simulators, EMPro, HotSpot, Extra-functional simulators	--	

Figure 4: Taxonomy of the main tools used for heterogeneous system design.

of translation, abstraction and integration approaches available at state-of-the-art. In particular, we adopted a set of approaches we defined in the recent years, that reconcile the different initial formalisms into an intermediate format (such as [7]–[11]). However, such transformations should not be considered the core of this work, as the methodology is general and it would still hold when adopting different abstraction/refinement approaches.

4.1 Bottom-up transformations

In the context of this work, the most crucial direction is bottom-up, *i.e.*, how to abstract existing component implementations across the levels of abstraction. For such transformations we rely on state-of-the-art tools and methodologies, given that defining new ones is out of the scope of this work. Figure 5 reports some of these tools, by showing the starting level of abstraction and the target one. It is interesting to note that no abstraction methodology is defined between the physical and the structural levels, given that the roll-back from a physical description to a more abstract one would be extremely complex. Vice versa, for some steps many efforts have been concretized into tools, e.g., for digital HW and analog descriptions [7]–[9], [43].

An interesting feature of such methodologies and tools is when a number of them share the same intermediate format or model of computation, despite of working between different levels of abstraction or for different domains. This allows indeed to apply transformations before generating the target virtual platform code and to reconcile the starting heterogeneous descriptions to a single semantics. This eases integration and improves the target code. This is the case of all methodologies developed in the recent years based on the HIFSuite framework (such as [7]–[11]). All transformations rely indeed on the *HIF intermediate format*, a XML-based language that represents a system as a tree of objects, each of them describing a specific component or functionality in the system [8]. The HIF language is also provided with a computational model, namely UNIVERCM, that additionally enhances bottom-up transformations [44], [45]. UNIVERCM is indeed an automata-based model of computation, that extends the support for physical and continuous time behaviors, typical of hybrid automata [46], with a better characterization of discrete time components and software. This allows to reconcile heterogeneous descriptions not only to the same intermediate format, but also to the same model of computation, thus further easing the generation of a homogeneous system description.

Note that other transformation flows may be adopted from the literature: the adoption of a single intermediate format and model of computation is just one of the possible alternatives, with advantages in terms of ease of integration.

4.2 Top-down transformations

Top-down transformations can be adopted to extrapolate and refine information contained in system level descriptions (e.g., IP-XACT and SysML) or derive an implementation of a component from its abstract specifications.

In the latter case, the identified target configuration for the virtual platform can be used to guide the corresponding component implementation, i.e., to refine the specifications to a C++ implementation of the component at the holistic level of abstraction [10]. This allows straightforward integration of the new component in the system. Note that the holistic level is the more abstract level, and thus it naturally is the first step in a top-down design flow.

The extraction of system-level information is on the other hand more interesting when the system-level description describes the organization of the target system, e.g., in IP-XACT or SysML [15], [16]. Both UML and SysML provide *structural diagrams*, such as UML Class diagrams, best suited to represent SW structure, and SysML Internal Block and Block Definition diagrams, that work well to describe components interconnection in heterogeneous systems. Concerning digital HW, the XML-based standard IP-XACT [47] emerged as a specification languages for HW-SW platforms, and its extensions allowed to extend support to analog HW and extra-functional models [48], [49].

Both UML Classes, SysML block definitions and IP-XACT components declarations represent the main system components, their interconnections and communication mechanisms, that allow to build the overall system architecture through connections between entities. This information can be used to extrapolate system organization, and to derive a high-level structure of the system, to be filled with the actual implementation of each component (obtained through top-down refinement or via any bottom-up abstraction flow) [10], [11]. Interestingly, knowing the structure of the system allows also to detect any missing "glue" between component, i.e., any necessary interface, that can then be implemented to ensure correct communication between components [50].

In our case, we chose to extrapolate and refine information contained in system-level descriptions with tools that share the same intermediate format or model of computation, and that in

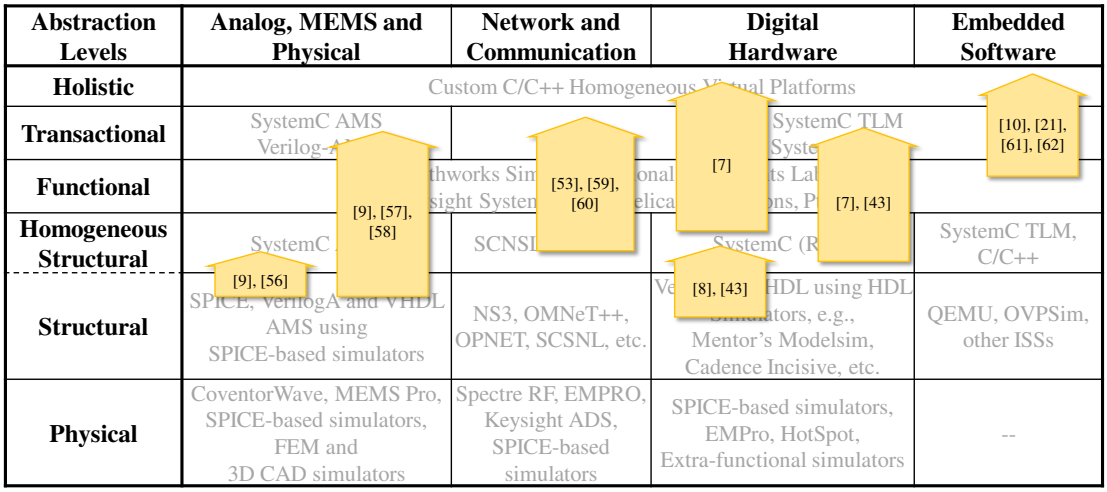

Figure 5: Summary of the major bottom-up approaches available to move across taxonomy layers. Each arrow reports the reference to the methodology or tool available to move from its starting to its target level.

particular rely on the HIFSuite framework, as done for the bottom up tools [10]. This allows indeed to map the constructs of the input languages to a single intermediate format and semantics, thus easing integration of the system components and allowing to generate a monolithic holistic and homogeneous description of the system.

4.3 Software execution

The holistic virtual platforms produced applying the presented flow are meant to be used to evaluate the behavior of software being developed for the system. By integrating a cycle-accurate model of memories and CPUs integrated in the system, the virtual platform is able to interpret, and thus to execute, any binary software being written for the given architecture. The software can be developed in any language, as long as it can be compiled into a binary format interpretable by the CPUs. This enables a wide range of possibilities: e.g., a designer may generate C code from UML models and then cross-compile the generated C code for the target architecture, or he/she may write the software using Assembly and compile it.

After the software has been compiled into a binary executable, the resulting file content must be analyzed, and the opcodes must be dumped into a textual file. The virtual platform provides an initialization routine loading the opcodes into the memory cells. The designer customizes such routine to accommodate the characteristics of the specific memories (and CPUs) architecture. As such, the designer may even express the software by directly specifying its every single opcode.

Once the initialization routine has been executed, the CPUs starts to fetch, decode, and execute the opcodes stored in memories. Thus, the virtual platform emulates the software behavior along with those of the other system components.

5 Methodology Application

In this section we exemplify the proposed flow on two different case studies. Furthermore, a third case study is released as a demonstrator for the the commercial version of HIFSuite[1].

1. For a concrete demonstrator refer to the official website of HIFSuite: `www.hifsuite.com`

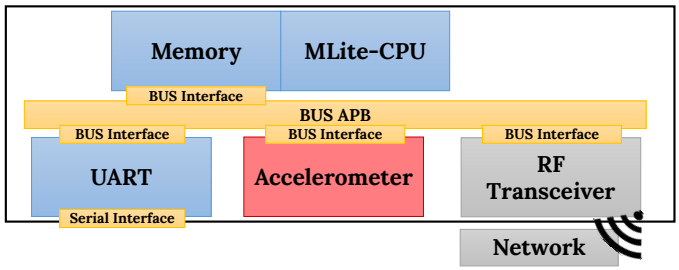

Figure 6: Overview of the S3TC architecture. Colors highlight the different design domains involved: red for analog components, grey for networking, yellow for (system-level) communication and blue for digital HW.

5.1 The SMAC Smart-System Test Case (S3TC)

The proposed flow will be exeplified throughout the next sections on the S3TC case study, developed for the *SMAC European Project* [42].

5.1.1 S3TC specification

The S3TC includes a sensor capturing physical values from the surrounding physical environment. Data are then processed by a computational unit, composed of a general purpose CPU and a RAM. Communication peripherals, both serial and wireless, are provided to allow the device to communicate with other systems. As such, the S3TC is able to perform sensing, computation, communication and actuation, and thus represents a quite typical and generic example of smart device. Figure 6 shows the main components of the smart device:

- Computation is performed by *MLite-CPU*, a CPU implementing the MIPS I architecture. Originally, it is provided as an open-source VHDL RTL model[2].
- The *Memory* supporting the CPU is a 256 KB RAM. It is used to store the SW application as well as all data sensed and computed by the device. It also handles communication with peripherals through Memory-Mapped Input/Output techniques. It is modeled as a Verilog IP.

2. https://opencores.org/project,plasma

- The *Universal Asynchronous Receiver/Transmitter (UART)* is a digital HW peripheral performing parallel-to-serial conversion and vice-versa. It provides the system with a serial interface. Its starting model is an open-source implementation available on OpenCores[3]. It provides both Wishbone and ARM Peripheral Bus interfaces.
- The *Accelerometer* is a MEMS component sensing the motion of the device by measuring the acceleration the system is subjected to. It implements a two-axis accelerometer and its original model is a VerilogA implementation provided by one of the industrial partners of the SMAC European Project.
- Wireless communication is provided by the *RF-Transceiver*, a network peripheral used to transmit data over a packet-based network. The original model is developed as a high-level specification of a network scenario expressed in SystemC Network Simulation Library (SCNSL) [51].
- All the peripherals are connected through an *ARM Peripheral Bus (APB)*. Initially, we were provided only with the high-level specification of the communication protocol [52]. Its interconnections, together with the interface of each component to the bus, are expressed by using IP-XACT. The Bus functionality has been implemented, by synthesizing its UML specification into VHDL [10].

With respect to the taxonomies in Figures 2 and 4, all the components of the S3TC are originally developed at the *Structural level*. As such, holistic system emulation initially can be performed only through co-simulation, as detailed below.

5.1.2 Building the S3TC virtual platform

The different sub-figures of Figure 7 summarize, from bottom to top, the different steps of the presented design flow when applied to build an efficient virtual platform for the S3TC. Table 1 reports the simulation speed achieved at each step.

Initially, (bottom line of Figure 7c) the system components are expressed in their original languages. Thus, it is necessary to build a co-simulation environment to emulate the complete behavior. We build three different co-simulation scenarios at the *Structural abstraction level*: a Ptolemy-based simulation environment (*i.e.*, Keysight's SystemVue) is used to connect different simulators, *i.e.* scenario (1) in Table 1. Mentor's Questa Advanced Simulator environment is used to co-simulate an instance of the Modelsim HDL simulator, and a SPICE-based simulator (*i.e.*, Mentor's ELDO), *i.e.* scenario (2) in Table 1. The scenario (3) in Table 1 uses both Keysight's SystemVue and Mentor's Questa to build a co-simulation environment. The sub-system composed by the MIPS CPU and the memory has been converted into a custom SystemVue block, while the UART is co-simulated by using Modelsim. Due to co-simulation limitations, the networking is troublesome and it requires complex ad-hoc interfaces. The three structural–level scenarios acts as a reference case for our results, as they have been built by using state-of-the-practice co-simulation techniques.

Figure 7c shows the transformation steps applied to the different component models to move from the *Structural* to the *Homogeneous Structural* abstraction level. The automatic translation procedure presented in [9] translates the models of the accelerometer and its interfaces into an equivalent SystemC AMS model. The automatic translation tools for HDLs provided by HIFSuite [8] are exploited translate the digital components of the platform (*i.e.*, CPU, UART, memory and bus) in SystemC at the Register-Transfer Level (RTL). Finally, the top-level of the system, the SystemC RTL implementation of the interconnections among system components are automatically generated by applying the methodology in [50]. After these transformation, all the system components are expressed in SystemC at the RTL: simulation is entirely coordinated by the SystemC simulation kernel, and the device is now modeled at the *Homogeneous Structural* level (*i.e.*, Scenario (4) in Table 1).

Figure 7b reports how to create an executable specification of the S3TC at the *Transactional abstraction level*. The automatic protocol abstraction procedure [7] is exploited to raise the abstraction-level of the digital components models from the SystemC at the RTL to SystemC Transaction-Level Modeling (TLM). The modeling strategy defined in [53] as been applied to (manually) re-describe the network model in SystemC TLM. It is worth noticing that we might have left the network models in SystemC Network Simulation Library (SCNSL), that provides both RTL and TLM SystemC interfaces [51]. The analog components have not been manipulated in this step, however ad-hoc transactors are inserted to let SystemC AMS and SystemC TLM to communicate, as described in [53]. As a result, the device is now modeled at the *Transactional level* (*i.e.*, Scenario (5) in Table 1).

Figure 7a reports the transformation allowing to move the simulation of the S3TC at the *Holistic abstraction level*. The models of the digital HW part of the device are translated in C++ from the equivalent TLM models. This transformation relies on the code optimizations and manipulations defined in [7]. The analog components are abstacted by applying the automatic abstraction methodology we defined in [9]. Finally, the execution relies on the synchronization and orchestration between the digital and analog parts provided by an ad-hoc C++ scheduling routine, automatically generated by applying the methodology in [54]. After these transformations, the S3TC device is monolithically modeled by a C++ executable specification, allowing to efficiently emulate the system-level behavior of the device (*i.e.*, Scenario (6) in Table 1).

Table 1: Execution time needed by the different simulation and co-simulation scenarios considered for the S3TC case study.

Scenario			Simulation Time (s)	Relative Speed-up	Total Speed-up
Co-Simulation	(1)	Structural (SystemVue-based coordination)	278.59	-	-
	(2)	Structural (Modelsim & ELDO SPICE)	215.47	1.29x	1.29x
	(3)	Structural (SystemVue & Modelsim)	153.23	1.37x	1.82x
Simulation	(4)	Homogeneous Structural (SystemC-RTL)	97.59	1.61x	2.85x
	(5)	Transactional (SystemC TLM & AMS)	44.28	2.20x	6.29x
	(6)	Holistic (C++)	21.26	2.08x	13.10x

5.1.3 Results summary

The data reported in Table 1 refers to the execution of each scenario to simulate 100 ms of system execution. The time step for the numerical integration of analog models is 100 ns. Column *Relative Speed-up* reports the speed-up between consecutive entries. Column *Total Speed-up* reports the speed-up of each scenario *w.r.t.* scenario (1).

The main outcomes of the experiment are two:

3. https://opencores.org/download,uart16550

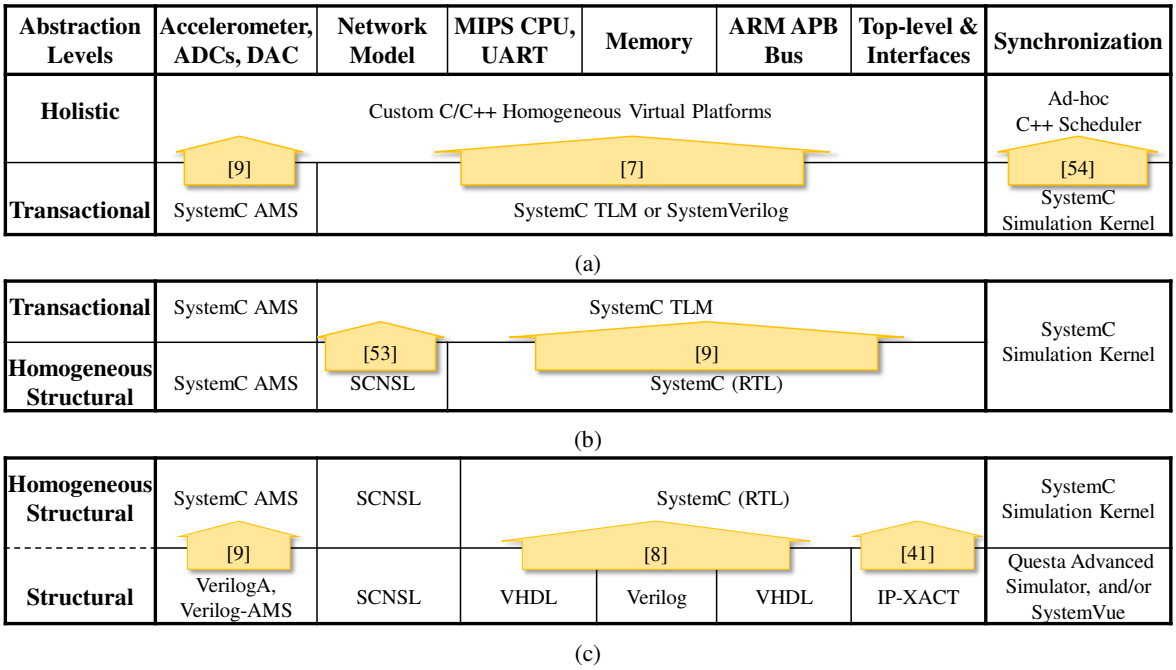

Figure 7: Three main steps of the methodology applied to the S3TC, contextualized within the tools taxonomy. The references in the arrows refer to the methodologies applied for each step.

- *Co-simulation interfaces impact simulation speed:* it is worth noticing that each co-simulation interface (three in the case of the first entry of the table, two in the second and only one in the third), seems to introduce around 60 seconds overhead *w.r.t.* the simulation without any co-simulation interface (fourth scenario). Thus, the impact of interfaces and of conversion layers between different tools seems highly relevant and dependent on the number of used interfaces and external tools. Thus, translating to a unique language positively impacts on simulation time.
- *Abstraction provides more speed-up than integration:* the *maximum relative speed-up* is achieved when abstraction accompanies translation, *i.e.*, scenario (5). As such, the effort required to perform automatic abstraction seems more convenient than the one required by automatic integration.
- The proposed methodology allows to *combine both*, thus obtaining scenario (6), that merges synchronization and behaviors within a unique monolithic executable model, preserving only the events of functional interest for the designer. This provides the most optimized simulation environment for the system and thus the best simulation performance, corresponding to the *maximum total speed-up* (13x).

This experiment proves the efficiency provided by *Holistic model simulation*, and that a methodology to automatically generate custom virtual platforms at the holistic abstraction level is a powerful asset for the design of heterogeneous smart devices.

In all the experiments, digital models are cycle-accurate *w.r.t.* the original components. The analog models, either abstracted and translated, are affected by a negligible numerical error: the worst Root Mean Square Error (RMSE) in our experiments is 10^{-6}.

5.2 The Internet-of-Things (IoT) case study

The proposed methodology has been applied also to a second case study having more complex analog and physical devices. The adopted case study is an HW IoT platform, equipped with a low-power micro-controller and a quite complex MEMS actuator.

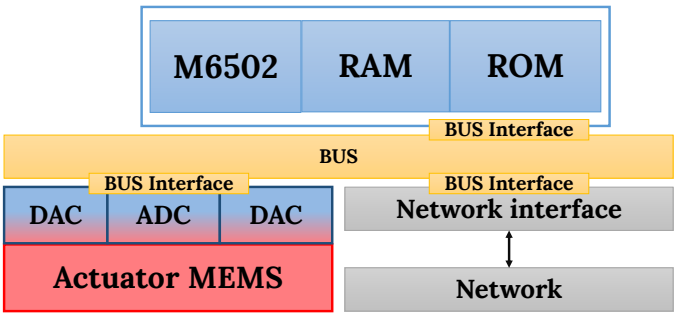

Figure 8: Structure of the IoT device HW platform. Colors highlight the different design domains involved: red for analog components, grey for networking, yellow for (system-level) communication and blue for digital HW.

5.2.1 *IoT device specification*

Figure 8 depicts the structure of the HW platform, a low-power device intended to be used in IoT applications, composed by the following components:

- A *Mos Technologies 6502 (M6502)* micro-controller; it is a 16 bit CPU, providing a RISC instruction set. It executes the SW stored in memory to control the peripherals. The model of the CPU is written in Verilog at RTL.
- A *8KB RAM*, whose model is expressed using Verilog. The RAM is connected to the CPU to perform computation, and to the bus to communicate with the peripherals using a Memory Mapped Input/Output mechanism.
- A *64KB ROM* storing SW and the data necessary for system evolution. Its description is written in Verilog at RTL.

Table 2: Characteristics of the considered platforms, and partition between digital and analog/physical HW components.

	S3TC	IoT device
Total lines of code	6,658	3,735
Lines of code for digital components	6,585	3,283
Lines of code for analog components	73	452
Digital signal assignments	1,483	982
Linear contribution statements	33	55
Non-linear contribution statements	0	13

Table 3: Execution time needed by the different simulation and co-simulation scenarios considered for the IoT case study.

Scenario			Simulation Time (s)	Relative Speed-up	Total Speed-up
Co-Simulation	(1)	Structural (Modelsim & ELDO SPICE)	10,156.27	–	–
Simulation	(2)	Homogeneous Structural (SystemC-RTL)	661.78	15.34x	15.34x
	(3)	Holistic (C++)	276.62	2.39x	36.71x

- A *custom Bus* connects the memory and the CPU to the peripherals. The bus is specified in VHDL.
- A *Network Interface* is used to receive commands from the network. Its model is written in SystemC and interfaced with SCNSL network model.
- The device is equipped with an *Actuator MEMS*, connected to the system through a couple of Digital-to-Analog Converters (DACs) and one Analog-to-Digital Converter (ADC). The device, provided by one of the industrial partners of the SMAC European Project, is described in Verilog-AMS, by using a mixture of different physical disciplines (*i.e.*, electrical, rotational, and logic), and modeling styles (*e.g.*, time- and frequency-domain descriptions). Internally it is composed by four Operational Amplifiers and a Transimpedance Amplifier building a feedback loop between the DACs and the ADC to control the MEMS dynamics. The MEMS presents both linear and non-linear behaviors.

Table 2 highlights the increased relevance of physical sensing and actuation *w.r.t.* to the former experiment. To this extent, it shows for both the case studies the number of lines of code in the original models, also partitioned between analog and digital components; the number of digital signal assignments; the number of analog contribution statements specified in the analog models, partitioned between linear and non-linear statement (since the non-linearities impact more heavily on simulation [55]). The different focus of the two case studies appears immediately clear, since the S3TC has little more than 1% of lines dedicated to modeling linear continuous-time dynamics, while the IoT device model dedicates the 12% of its lines to specify nonlinear continuous behaviors (as an effect of a higher number of linear and non-linear contribution statements).

5.2.2 Building the IoT device virtual platform

The different parts of Figure 9 summarize, from bottom to top, the different steps of the presented design flow applied to build an efficient virtual platform for the IoT device. Table 3 reports the simulation speed achieved by simulating the system at each step.

Initially (bottom line of Figure 9b), the system has been simulated by building a co-simulation scenario. Each component is specified in its original description language, and the execution is managed by Questa Advanced Simulator instantiating both the Modelsim HDL simulator and a SPICE-based simulator (*i.e.*, Mentor's ELDO), *i.e.* scenario (1) in Table 3. This environment provides a structural–level virtual platform of the IoT device that acts as a reference case for our results.

The arrows in Figure 9b show the transformations performed to create a *homogeneous structural* virtual platform for the device. As for the S3TC, the digital components have been translated into their SystemC RTL equivalent models exploiting the tools provided by HIFSuite [8]. The interfaces between the components have been generated by exploiting [50] as for the previous case study. The analog part of the device is partially expressed using Verilog-A (*i.e.*, the actuator) and Verilog-AMS (*i.e.*, the DACs and ADC). Thus, they are treated separately by applying on both cases the automatic translation methodology for analog models we developed in [9], supported by a minor manual refinement of the ADC and the DACs generated models to deal with discontinuities in the Verilog-AMS model. After these transformation, all the system components are expressed in SystemC, at the RTL: the simulation can be managed by the SystemC simulation kernel to provide a *Homogeneous Structural* virtual platform (*i.e.*, Scenario (2) in Table 3).

Figure 9a shows the manipulations for each component allowing to produce a *Holistic abstraction level* virtual platform for the device, *i.e.*, Scenario (3) in Table 3. As for the previous case study, the digital HW has been abstracted and translated into C++ by exploiting the methodology in [7]; the network model has been translated (manually) in C++ by following the modeling approach in [53]; the analog components have been abstracted by exploiting the automatic abstraction methodology [9]. The produced sub-models are then managed by an ad-hoc C++ scheduling routine automatically produced applying the methodology in [54]. The final result is monolithic executable C++ model emulating device behavior.

5.2.3 Results summary

Table 3 summarizes the obtained results, and it refers to the emulation of one second of the execution of the IoT device, by using the three aforementioned scenarios. It is evident that the conclusions of the first experiment still hold.

However, this second set of results gives more information about the scalability of the proposed methodology when the complexity of the platforms increases. Table 2 highlights the higher degree of heterogeneity of the IoT device *w.r.t.* the S3TC. In the case of the IoT device, the relative speed-up achieved by performing abstraction is similar to the one achieved in the S3TC. However, the reconciliation of different design domains into a homogeneous model, through automatic translation and integration, provides a relative speed-up that is one order of magnitude higher for the second case study *w.r.t.* the S3TC (almost 37x). As such, the total speed-up achieved by building the custom virtual platform is one order of magnitude better in the more heterogeneous case.

The results show that custom virtual platforms provide best performance when dealing with highly heterogeneous devices. As the trend on systems design is to integrate more and more heterogeneous components, the proposed methodology is well tailored to address the future challenges in smart systems design.

In all the experiments, digital models are cycle-accurate *w.r.t.* the original components. The analog models, either abstracted and translated, are affected by a negligible numerical error: the worst RMSE in the presented experiments is 10^{-6}.

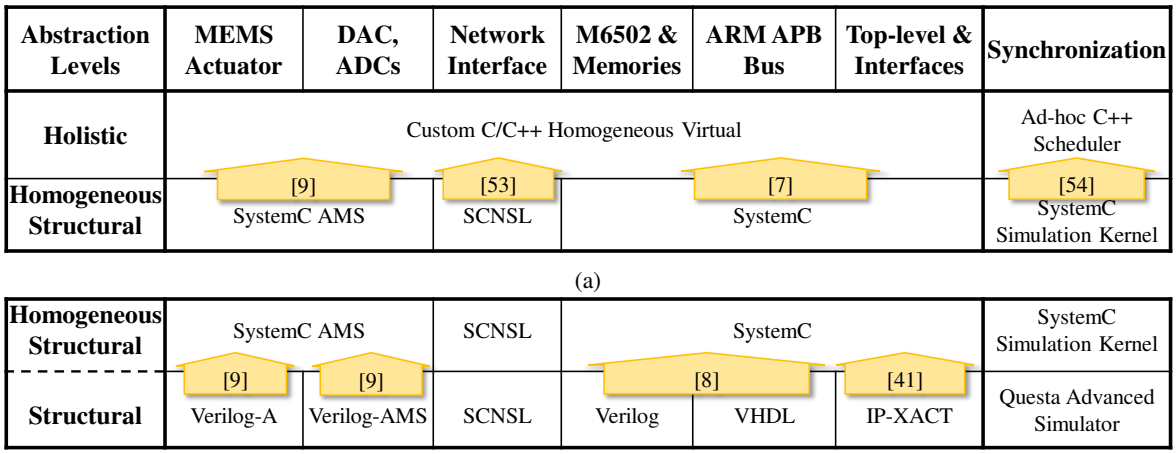

Figure 9: Three main steps of the methodology applied to the IoT device, contextualized within the tools taxonomy. The references in the arrows refer to the methodologies applied for each step.

6 Discussion

In this Section we summarize and discuss the main advantages of the proposed methodology. We will address separately the impact on either *simulation* and *design* of smart devices.

The proposed approach positively impacts simulation:

1) The approach allows to *integrate within the emulated systems any IP*, described at any abstraction level, not forcing designers to either manually translate the models or build complex co-simulation environments. In particular, the approach allows also to create AMS virtual-platforms accurately reproducing continuous-time system behavior.
2) The methodology allows to *avoid co-simulation*. The experimental results, as well as some related work [3], showed the inefficiency of co-simulation technologies. The disadvantages imposed by co-simulation environments are avoided by building "monolithic" models that can be simulated by a single process. Thus, it allows to remove all the interfaces among different tools, that usually rely on Inter-Process Communication mechanisms and, consequently, are the main source of inefficiency.
3) The methodology allows to *further optimize the model* used to simulate the system: the automatic abstraction of the already integrated system allows to maximize the optimization of the mixed-signal scheduler orchestrating system simulation.
4) The proposed approach *increases the accuracy* w.r.t. *state-of-the-art virtual platforms*. Instruction Set Simulators (ISSs) and CPU emulators, commonly used in virtual-platforms, usually approximate the cycle-accurate behavior of models to speed-up the simulation. As such, the accuracy of the simulated SW is usually at *instruction-level*, while we preserve the accuracy *w.r.t.* the clock-cycles of the digital HW models.

Improving simulation performance improves also the overall design flow. Moreover, homogeneous models provide further improvements to the design flows for smart devices:

1) *Any IP can be integrated within a homogeneous model*, not only digital and discrete-time devices. The methodology theoretically allows to consider any description for "incorporation" within the *holistic model* of the system. That is, simply mapping its syntax and semantics in the Heterogeneous Intermediate Format (HIF) representation, any description or specification language can be managed. As such, a custom virtual platform can be built for any system, regardless of its heterogeneity.
2) Inter-process interfaces may become a source of integration errors. The approach provides a *more reliable integration* mechanism by avoiding co-simulation: it relies on the precise semantics of the intermediate representation, while avoiding complex interfaces. Thus, it removes a source of potential integration errors.
3) The *holistic model* focuses on simulating the behavior of the system efficiently, and it does not expose unnecessary details. As such, it allows to perform *design space exploration tasks by focusing only on the high-level system behavior*.
4) Custom virtual platforms produced by applying the proposed approach allow to precisely evaluate the timing of the system. This is very important when designing SW running on heterogeneous platforms that must respect real-time constraints in order to correctly interact with its environment.

As a take home message, we can highlight the potentialities of custom virtual platforms for smart devices: they integrate heterogeneous models into a homogeneous description, allowing to simplify the simulation environment and speeding-up the simulation while preserving the accuracy.

7 Conclusions

In this paper we proposed a meet-in-the-middle approach to create custom virtual platform of heterogeneous system. The main target was to tackle the "Babel Tower" of design and modeling languages for smart devices, while creating efficient custom simulators from heterogeneous components and specifications. The end-to-end design flow presented in this paper "connects the dots" drawn by former works in the latest years. We showed the applicability of the methodology to two complex smart devices, that show how a custom virtual platform outperforms state-of-the-art simulators.

Acknowledgments

We would like to thank all the colleagues that contributed on the different sub-steps of methodology presented in this work. In particular, we would like to thank Valerio Guarnieri for his contribution to the automation of the digital components abstraction; Enrico Fraccaroli for his contributions to the automation

of the analog components abstraction; Francesco Stefanni for constantly maintaining HIFSuite. Finally, we would like to thank STMicroelectronics for their contribution in building the case studies presented in this paper.

REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, feb 2016.

[2] N. Dutt, A. Jantsch, and S. Sarma, "Toward smart embedded systems: A self-aware system-on-chip (SoC) perspective," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, p. 22, 2016.

[3] F. Fummi, M. Lora, F. Stefanni, D. Trachanis, J. Vanhese, and S. Vinco, "Moving from Co-Simulation to Simulation for Effective Smart Systems Design," in *Proceedings of ACM/IEEE Design and Test in Europe (DATE)*, 2014, pp. 1–4.

[4] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, 2000.

[5] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel *et al.*, "Functional mockup interface 2.0: The standard for tool independent exchange of simulation models," in *Proceedings of the MODELICA Conference 2012*, no. 076. Linköping University Electronic Press, 2012, pp. 173–184.

[6] C. Bertsch, E. Ahle, and U. Schulmeister, "The functional mockup interface-seen from an industrial perspective," in *Proceedings of the MODELICA Conference 2014*, no. 096. Linköping University Electronic Press, 2014, pp. 27–33.

[7] S. Vinco, V. Guarnieri, and F. Fummi, "Code Manipulation for Virtual Platform Integration," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2694–2708, Sept 2016.

[8] N. Bombieri, G. D. Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, "HIFSuite: tools for HDL code conversion and manipulation," *EURASIP Journal on Embedded Systems*, vol. 2010, p. 4, 2010.

[9] M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. in press, 2017.

[10] M. Lora, F. Martinelli, and F. Fummi, "Hardware Synthesis from Software-oriented UML Descriptions," in *Proceedings of IEEE Microprocessors Test and Verification (MTV)*, 2014, pp. 33–38.

[11] S. Vinco, M. Lora, E. Macii, and M. Poncino, "IP-XACT for smart systems design: extensions for the integration of functional and extra-functional models," in *Proceedings of ECSI/IEEE Forum on Design and Description Languages*, 2016, pp. 1–8.

[12] G. De Micheli, R. Ernst, and W. Wolf, *Readings in hardware/software co-design*. Morgan Kaufmann, 2002.

[13] J. A. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," *Incose MBSE Focus Group*, vol. 25, pp. 1–70, 2008.

[14] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for system design," 2012.

[15] Open Management Group, "OMG UML: Unified Modeling Language," http://www.uml.org.

[16] Object Management Group, "SysML," URL: http://www.sysml.org.

[17] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2006.

[18] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification language and methodology*. Springer Science & Business Media, 2012.

[19] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[20] C. Ptolemaeus, Ed., *System Design, Modeling, and Simulation: Using Ptolemy II*. Ptolemy. org Berkeley, CA, USA, 2014.

[21] The Mathworks, "Simulink," mathworks.com/products/simulink.

[22] Keysight Technologies, "SystemVue Electronic System-Level Design Software," http://www.keysight.com/.

[23] National Instruments, "LabVIEW," www.ni.com/labview.

[24] W. Li, X. Zhang, and H. Li, "Co-simulation platforms for co-design of networked control systems: An overview," *Control Engineering Practice*, vol. 23, pp. 44–56, 2014.

[25] L. D. Guglielmo, F. Fummi, G. Pravadelli, F. Stefanni, and S. Vinco, "A formal support for homogeneous simulation of heterogeneous embedded systems," in *Proceedings of IEEE Symposium on Industrial Embedded Systems (SIES)*, 2012, pp. 211–219.

[26] D. Becker, R. K. Singh, and S. G. Tell, "An engineering environment for hardware/software co-simulation," in *Proceedings of the 29th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1992, pp. 129–134.

[27] A. Sangiovanni-Vincentelli, "Is a unified methodology for system-level design possible?" *IEEE Design & Test of Computers*, no. 4, pp. 346–357, 2008.

[28] R. Leupers, G. Martin, R. Plyaskin, A. Herkersdorf, F. Schirrmeister, T. Kogel, and M. Vaupel, "Virtual platforms: Breaking new grounds," in *Proceedings of the IEEE/ACM Conference on Design, Automation and Test in Europe (DATE)*, 2012, pp. 685–690.

[29] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu, "A next-generation design framework for platform-based design," in *DVCon 2007*, February 2007. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/228.html

[30] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. Sangiovanni-Vincentelli, and E. A. Lee, "Metronomy: a function-architecture co-simulation framework for timing verification of cyber-physical systems," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2014, p. 24.

[31] P. Giusto, G. Martin, and E. Harcourt, "Reliable estimation of execution time of embedded software," in *Proceedings of the IEEE/ACM conference on Design, Automation and Test in Europe (DATE)*. IEEE Press, 2001, pp. 580–589.

[32] Z. Wang and A. Herkersdorf, "An efficient approach for system-level timing simulation of compiler-optimized embedded software," in *Proceedings of the 46th IEEE/ACM Design Automation Conference (DAC)*. ACM, 2009, pp. 220–225.

[33] A. Bouchhima, P. Gerin, and F. Pétrot, "Automatic instrumentation of embedded software for high level hardware/software co-simulation," in *Proceeding of the IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, 2009, pp. 546–551.

[34] Z. S. Andraus and K. A. Sakallah, "Automatic abstraction and verification of verilog models," in *Proceedings of the 41st IEEE/ACM Design Automation Conference (DAC)*. ACM, 2004, pp. 218–223.

[35] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Theory in practice for system design and verification," *ACM Siglog News*, vol. 2, no. 1, pp. 46–51, 2015.

[36] Cadence Design Systems, "Virtual System Platform," www.cadence.com.

[37] Synopsys, "Platform Architect," www.synopsys.com.

[38] Mentor Graphics, "Mentor Embedded for Intel," https://www.mentor.com/embedded-software/partners/intel.

[39] Imperas Software, "OVP - Open Virtual Platforms," www.ovpworld.org.

[40] F. Cucchetto, A. Lonardi, and G. Pravadelli, "A common architecture for co-simulation of SystemC models in QEMU and OVP virtual platforms," in *Proceedings of IEEE/IFIP Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2014, pp. 1–6.

[41] M. Lora, S. Vinco, and F. Fummi, "A unifying flow to ease smart systems integration," in *Proceedings of 19th IEEE International High-Level Design Validation and Test Workshop (HLDVT)*. IEEE, 2016, pp. 113–120.

[42] R. Gillon, G. Gangemi, M. Grosso, F. Fummi, and M. Poncino, "Multi-domain simulation as a foundation for the engineering of smart systems: Challenges and the SMAC vision," in *Proceedings of the IEEE International Conference on Electronics Circuits and Systems*, 2014, pp. 858–861.

[43] Carbon Design Systems. Carbon Model Studio. http://carbondesignsystems.com/.

[44] L. Di Guglielmo, F. Fummi, G. Pravadelli, F. Stefanni, and S. Vinco, "UNIVERCM: the UNIversal VERsatile Computational Model for heterogeneous system integration," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 225–241, 2013.

[45] L. D. Guglielmo, F. Fummi, G. Pravadelli, F. Stefanni, and S. Vinco, "Univercm: The universal versatile computational model for heterogeneous embedded system design," in *Proceedings of IEEE International High-Level Design Validation and Test Workshop (HLDVT)*, Nov 2011, pp. 33–40.

[46] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.

[47] V. Berman, "Standards: The P1685 IP-XACT IP Metadata Standard," *IEEE Design & Test of Computers*, vol. 23, no. 4, pp. 316–317, apr 2006, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1683721.

[48] W. Kruijtzer, P. van der Wolf, E. de Kock, J. Stuyt, W. Ecker, A. Mayer, S. Hustin, C. Amerijckx, S. de Paoli, and E. Vaumorin, "Industrial IP integration flows based on IP-XACT standards," in *2008 Design, Automation and Test in Europe*. IEEE, mar 2008, pp. 32–37. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4484656

[49] S. Vinco, A. Sassone, F. Fummi, E. Macii, and M. Poncino, "An open-source framework for formal specification and simulation

of electrical energy systems," in *Proceedings of the 2014 international symposium on Low power electronics and design (ISLPED '14)*. ACM Press, aug 2014, pp. 287–290. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627369.2627657
[50] D. Braga, F. Fummi, G. Pravadelli, and S. Vinco, "The strange pair: IP-XACT and univerCM to integrate heterogeneous embedded systems," in *High Level Design Validation and Test Workshop (HLDVT), 2012 IEEE International*. IEEE, 2012, pp. 76–83.
[51] F. Fummi, D. Quaglia, and F. Stefanni, "A SystemC-based framework for modeling and simulation of networked embedded systems," in *Proceedings of IEEE/ECSI Forum on Specification, Verification and Design Languages (FDL)*, 2008, pp. 49–54.
[52] ARM, "AMBA Advanced Peripheral Bus Protocol."
[53] F. Li, E. Dekneuvel, G. Jacquemod, D. Quaglia, M. Lora, F. Pêcheux, and R. Butaud, "Multi-Level Modeling of Wireless Embedded Systems," in *Proceedings of ECSI/IEEE Forum on Specification & Design Languages 2014 (FDL 14)*, 2014, pp. 1–8.
[54] M. Lora, E. Fraccaroli, and F. Fummi, "Virtual prototyping of smart systems through automatic abstraction and mixed-signal scheduling," in *Proceedings of IEEE/ACM Asian and South-Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 232–237.
[55] P. Benner, M. Hinze, and E. J. W. Ter Maten, *Model reduction for circuit simulation*. Springer, 2011.
[56] F. Pecheux, C. Lallement, and A. Vachoux, "VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, feb 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1386377
[57] S. Little and C. Myers, "Abstract modeling and simulation aided verification of analog/mixed-signal circuits," 2008.
[58] M. Damm, C. Grimm, J. Haase, A. Herrholz, and W. Nebel, "Connecting SystemC-AMS Models with OSCI TLM 2.0 Models using Temporal Decoupling," in *Proceedings of the IEEE/ECSI Forum on Specification and Description Languages (FDL)*, 2008, pp. 25–30.
[59] J. Haase, M. Damm, J. Glaser, J. Moreno, and C. Grimm, "Systemc-based power simulation of wireless sensor networks," in *Proceedings of the IEEE/ECSI Forum on Specification & Design Languages*, 2009, pp. 1–4.
[60] M. Damm, J. Moreno, J. Haase, and C. Grimm, "Using transaction level modeling techniques for wireless sensor network simulation," in *Proceedings of the IEEE/ACM Conference on Design, Automation and Test in Europe (DATE)*, 2010, pp. 1047–1052.
[61] L. B. Brisolara, M. F. Oliveira, R. Redin, L. C. Lamb, L. Carro, and F. Wagner, "Using UML as front-end for heterogeneous software code generation strategies," in *Proceedings of the IEEE/ACM conference on Design, Automation and Test in Europe (DATE)*, 2008, pp. 504–509.
[62] G. DeTommasi, R. Vitelli, L. Boncagni, and A. C. Neto, "Modeling of MARTe-based real-time applications with SysML," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2407–2415, 2013.

Michele Lora (M'13) received the Ph.D. degree in computer science from the University of Verona, Italy, in 2016. He currently is a Postdoctoral research fellow at the Singapore University of Technology and Design. His research focuses on automation methodologies for modeling, integration and efficient simulation of heterogeneous embedded systems, contract-based requirement engineering and verification for cyber-physical systems.

Sara Vinco (M'09) received the Ph.D. degree in computer science from the University of Verona, Italy, in 2013. She is currently Assistant Professor at the Department of Control and Computer Engineering, Politecnico di Torino, Italy. Her main research interests are energy efficient electronic design automation and techniques for simulation and validation of heterogeneous embedded systems.

Franco Fummi (M'92) received the Ph.D. degree in electronic engineering from Politecnico di Milano, Italy, in 1995. Since 2000 he is a Full Professor at the University of Verona, Italy, where he became an Associate Professor in computer architecture in 1998. Since 1995, he has been with the Department of Electronics and Information, Politecnico di Milano, as an Assistant Professor. He is a co-founder of EDALab, an EDA company developing tools for the design of networked embedded systems. His current research interests include electronic design automation methodologies for modeling, verification, testing, and optimization of embedded systems.