The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems

Fernando Herrera [a,*], Héctor Posadas [a], Pablo Peñil [a], Eugenio Villar [a], Francisco Ferrero [b], Raúl Valencia [b], Gianluca Palermo [c]

[a] University of Cantabria, ETSIIT, TEISA Dpt., Av. Castros sn, 39005 Santander, Spain
[b] GMV, Aerospace and Defence S.A.U., C/Isaac Newton, 11 P.T.M. Tres Cantos, 28760 Madrid, Spain
[c] Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy

1. Introduction

The design of embedded systems is in a highly competitive context. The translation of an efficient design into a successful product highly depends on becoming the first product in the market with new complex functionalities fulfilling tight performance constraints, and at an affordable price. In this scenario, the task of system engineers becomes challenging. They have to do an early assessment of the design alternatives since about 90% of the overall cost is determined at the first stages of the design [1]. At the same time, a right assessment becomes difficult due to the complexity of applications and platforms. Performance depends on a diverse set of factors, such as the architecture of the software application, the architecture of the hardware platform, how the application functionalities are executed by the processing resources of the platform, and many parameters such as cache sizes, memory sizes, etc. This makes Design Space Exploration (DSE) a key design activity [2] in ESL design for enabling such an early assessment.

A DSE framework has three main requirements: (1) a specification methodology suitable for Design Space Exploration (DSE), as well as for other ESL design activities (i.e., verification and SW synthesis); (2) techniques and tools able to produce fast and sufficiently accurate performance metrics; and finally (3) exploration strategies able to prune a potentially huge design space.

An important number of methodologies relying on different techniques for performance estimation and specific exploration strategies have been proposed. Some of them have also enabled a high-level input, based on a model which captures the system architecture and main system parameters. Specifically, there has been an effort on starting from models based on the Unified Modeling Language (UML) [3], supported by specialized profiles, such as SysML and MARTE [4]. These approaches effectively joined Model Based Design (MBD) with Electronic System Level (ESL) design. However, these methodologies present a number of drawbacks yet. UML/MARTE methodologies do not present all the features required for DSE. Specifically UML/MARTE models have to be edited along DSE iterations, which slows down the exploration, because they do not provide mechanisms to model and define the design space in terms of the parameters, architectures and architectural mappings that the user wants to explore, and in terms of the performance metrics which need to be constrained. Second, traditional simulation-based performance estimation technologies are too slow for exploring DSE systems, while analytical approaches

* Corresponding author. Tel.: +34 942 200878; fax: +34 942 201873.
E-mail address: fherrera@teisa.unican.es (F. Herrera).

have to sacrifice much accuracy. Third, the exploration strategies propose implementations tightly coupled to the solution of a specific optimization problem, which makes those DSE frameworks specific and difficult to extend with enhancements on both, performance estimation and exploration techniques. Out of the UML/MARTE context, and even from the model-based design context, other works, explained later on in Section 2, have provided advanced features for DSE including some of the aforementioned ones. This work has combined, extended and applied many of them to an UML/MARTE modelling context.

Two main approaches to DSE can be distinguished attending the way a design point is evaluated, those based on analytical techniques and those based on simulation [5]. Analytical approaches relying on worst-case workloads and predictable architectures are suitable for applications which require a high degree of confidence on constraint fulfilment, e.g. time critical or safety critical applications. However, these approaches often lead to more inefficient design solutions and their applicability is limited by modelling constraints required for the application of the analytical model.

In contrast, a simulation-based DSE approach, like the one proposed in this paper, enables more accurate estimations, and makes feasible the assessment of performance of complex application models and advanced architectures, where the development of an analytical model gets too complex. Therefore, simulation-based DSE is a suitable solution in domains such as consumer electronics, where it is necessary to find efficient designs in an affordable design time for applications with QoS requirements and time-critical (but not safety critical) constraints, on top of devices with complex architectures (multiprocessor, memory and communication hierarchies, etc.).

This paper presents the UML/MARTE Modeling and Design Space Exploration (DSE) framework for embedded systems, developed in the context of the COMPLEX project [6][7]. In short, we will refer to this framework as COMPLEX UML/MARTE DSE framework, to distinguish it from other concepts, techniques and tools developed in the COMPLEX project, e.g., which include for instance a Matlab/Simulink front-end, or the support of power management, and which are not part of the work presented in this paper.

Distinctive aspects of the COMPLEX UML/MARTE methodology are the following:

- It fits the general MBD-based DSE flow sketched in Fig. 1. In this approach the user input is a system model supporting component-based modelling and separation of concerns, and containing all the required information for the DSE activity.
- It supports a MBD model of the system working environment. This way, more efficient solutions are found by tuning them to the specific working environment. The integration of the MBD environment model on an accurate simulation-based performance model is automated.

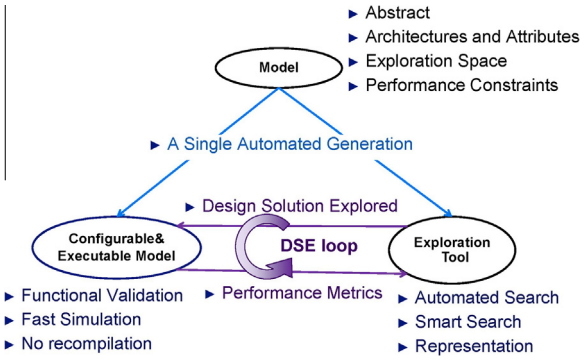

Fig. 1. Generic solution of simulation-based DSE flow implemented.

- The model is captured once and it can be reused for other ESL design activities, e.g. SW synthesis (*single source* approach). In this sense, the model serves also as a *specification*.
- From the specification, an executable and configurable performance model is automatically generated.
- The estimated performance associated to each explored design point is obtained by relying on native simulation, much faster than virtualization or Instruction Set Simulators (ISS).
- The technology supports SW and HW estimation; it considers the impact of different communication types (SW–SW, SW–HW, etc.,); and supports a generic, RTOS independent API, which enables the analysis of different architectural mappings without the need to generate the complete SW stack in each executing node or to synthesize the HW.
- The performance estimation model and the exploration tool are kept separated and integrated through a XML-base interface which enables a modular DSE framework.

In previous work the UML/MARTE methodology for modelling the system [9] and the stimuli environment [10] has been presented. In [11], the automatic generation from the UML/MARTE specification of a performance executable and configurable model was explained. This performance model relies on the SCoPE technology [14–16]. The integration of the IP-XACT format in the code generation [12], and its support as part of the SCoPE front-end was explained in [13]. In [20] M3SCoPE, which connected the performance estimation tools (SCoPE) and the exploration tool (M3Explorer) [21], was explained. Performance estimation technology has been improved, so SCoPE+ now supports the performance estimation of different architectural mappings from the same Platform-Independent model (PIM) without the need to generate the SW stack in each node [17]. Moreover, a fast performance estimation technique for the estimation of performance of the functionality mapped to HW was developed in [19].

In this paper, a complete overview of the COMPLEX UML/MARTE-based methodology for DSE is given. Moreover, the paper contributes new aspects not addressed in previous work, specifically:

- The MOST exploration tool is shown as a main component of the flow which is smoothly integrated through an XML based interface. MOST enables the analysis of a huge design space, which can include different scenarios, without requiring the edition of the environment model along the DSE loop.
- An extended example (vocoder instead of coder), showing more significant features and capabilities of the methodology, such as the capability to explore distinct mappings and considering different scenarios.
- The integration of the HW estimation library in SCoPE+.The co-simulation of the system with the SystemC environment.

The rest of the paper is structured as follows. In Section 2, the state of art is reviewed. Section 3 explains the UML/MARTE modeling:modelling methodology. Then Section 4 explains the generation of the SCoPE+ executable and configurable performance estimation model and introduces MOST and its integration in the flow. Section 5 provides experimental results. Section 6 provides the main conclusions of this work.

2. State of Art

2.1. Modeling methodologies in UML/MARTE and model-based DSE

Despite the relative recent development of the MARTE profile, several works have proposed UML/MARTE based methodologies. The MARTE profile is an OMG standard that offers a rich set of

extensions specifically suited for the specification of embedded real-time systems. MARTE enables building models containing detailed information about the application, about the platform attributes and its architecture, for enabling performance analysis.

Gaspard2 [22,23] is a design environment for data-intensive applications which enables a MARTE description of both, the application and the hardware platform, including MPSoC and regular structures. Gaspard2 uses composite diagrams and the MARTE profile for capturing both, application and platform architectures. Gaspard2 tooling supports the chaining of different model to model (M2M) transformation tools. This facilitates the generation of synthesis flows, and also of performance models. Specifically, Gaspard2 supports the generation of SystemC TLM models at the Programmers View Time (PVT) level. It enables fast simulations, which speeds up exploration.

MoPCoM [24] is another design methodology for the design of real-time embedded systems which supports UML and the MARTE profile for system Modeling. Specifically, MoPCoM uses the NFP MARTE profile for the description of real-time properties; the HRM MARTE profile for platform description; and the Alloc MARTE profile for architectural mapping. Moreover, MoPCoM defines three levels of generation. The second level, called Execution Modeling Level (EML), targets the generation of models for performance analysis, and it is suitable for obtaining performance figures used in DSE iterations. However, work reported in [24] mostly focuses on the Detailed Modeling Level DML level, intended for implementation, by enabling VHDL code generation. In [25] a solution for automatically synthesizing models combining new communication semantics with standard UML/MARTE real-time management features is provided. This approach provides a flexible and easy-to-use way to specify and explore the system's concurrent architecture.

Co-Fluent methodology [26] captures application and hardware architecture by means of composite diagrams and SysML blocks. UML activity diagrams are used to specify application execution flows. The MARTE HRM profile is used for capturing the HW platform. This methodology uses the <<assign>> stereotype for expressing allocations. However, they are used for Modeling a single allocation, thus a single implementation alternative.

The main limitation of the previous methodologies is that the exploration of architectural alternatives requires the edition of the UML/MARTE model and a re-generation of the executable performance model.

In [27], a UML/MARTE based methodology relying on *activity threads* is proposed in order to reduce the effort required to capture the set of architectural mappings. An activity thread is a UML activity diagram where each path reflects a design alternative, that is, an architectural mapping.

In [28], a methodology for supporting designers on the evaluation of the HW/SW partitioning solutions, specifically, to identify design points fulfilling the timing constraints is shown. It proposes a way to depict in one set of diagrams all possible combinations of system configurations. By means of annotation of MARTE non-functional properties and of the application of schedulability analysis, the design space is restricted to the design points fulfilling timing requirements. However this methodology neither reports optimum solutions, nor it relies on automated technologies for the estimation of performance metrics.

The proposed MARTE based specification methodologies are still limited for DSE purposes. In these methodologies, the exploration of different platform architectures, of different architectural mappings, and even a small change in a design parameter (e.g., a cache size) requires a manual change of the model. Moreover, when the model is used for producing an executable virtual system for a simulation-based performance analysis, a regeneration of the executable model is typically required. Model edition and regeneration of the executable performance model are added to the simulation time, thus increasing the iteration time of the DSE loop and having a significant impact in the exploration time, up to a point which can make the exploration of a sufficiently wide design space unaffordable. Another desirable feature is to enable the capture of the output performance metrics to be used by the objective function(s) of the DSE process within the model. Enabling their capture in the model in a tool independent manner enables the direct relation of such metrics with the performance constraints also captured in the model. Performance constraints are mandatory, since they determine the frontier for acceptable solutions.

Moreover, building a modelling methodology suitable for DSE should not prevent features which have been proven to be useful for system specification, e.g. model driven engineering (MDE) [29] principles in the development of HW/SW embedded systems. Predominant role of SW leverages to target *software centric* methodologies [30] where the description of a platform independent model (PIM) can be fully allocated on default to a SW implementation, and thus it can be considered as an application model. A component oriented approach [31] is convenient. In Component-based Software Engineering (CBSE) [32], the system is built as a composition of application components interacting with each other only through well-defined interfaces. According this approach, components are software units that exhibit their interfaces (provided or required). On this way, the application can be split into clearly separable and reusable blocks, improving the organization of the product as well as its reusability and modularity. This CBSE approach also allows the designers to distribute the application blocks on the different processing nodes thanks to the container and connector concepts defined in [33], which makes the allocation of components on the hardware system transparent to the user and improves the design space exploration and performance estimation.

There are a set of model-based approaches which have tackled at least part of the aforementioned issues or requirements for DSE. MILAN [34] enables a model-based approach to capture the application and the platform (called *resource model*). MILAN also introduced the idea of constraint model, distinguishing between composability rules and constraints on performance, called *semantic constraints* and *design constraints* respectively in MILAN. The framework shows how a model-based approach facilitates the integration, not only of several simulation tools, but also of several of several levels of abstraction in the estimation of the performance of the design point by relying on a *Generic Modeling Environment* (GME) [35].

Koski [37] provides an UML-based interface to describe, as well as the application and the platform, the design constraints, including mapping constraints. The framework also enables the capture of the cost function by relying on default generic metrics. It also includes a code generation phase which produces an executable model for functional validation, as long as functionality is available.

DESERT [36] is a domain independent tool chain which enables the abstraction of the structural characteristics of the components available for building up a platform, and enables the construction of *platform templates*. Platform templates represent a set of solutions, that is, a design space in structural terms. DESERT enables the description of structural and compatibility constraints, which enable to prune the design space and synthesize a fully specified model fulfilling the aforementioned constraints.

Recent approaches [38,39] have related exploration to the MDE fundamental concept of model transformation. In [38], a declarative, relational approach is proposed where typical activities of the development process (functional partition or the application, refactoring of the platform architecture or mapping to the platform) are bounded by constraints. In [38], these constraints are formalized and implemented as model transformation rules to

support what is called *mechanized exploration*. The main aim of mechanized exploration is to support an interactive and incremental DSE process, capable to present feasible solutions to the user, which can make decisions and control a gradual refinement of the model

MODES [39] is a recent proposal which exploits model-to-model transformation in order to generate an internal representation model which is not only used for DSE, but also for formal verification and co-synthesis. This way, model-based design serves for the integration of DSE in the design flow. The internal representation model comprises structural aspects (similar to DESERT) of an application model and behavioral aspects through a Control and Data Flow Graph (CDFG). Moreover, MODES uses UML/MARTE models for some transformation rules.

In [40], a recent survey of DSE frameworks is used to state a set of requirements and research challenges in the development of a meta-framework for design space exploration. Based on them, an incipient proposal of such a meta-framework, implemented on GME, is presented. The aim is to facilitate the coding of domain specific DSE environments. Interesting features refer the support of an abstract DSE language (ADSEL) [41], or the automatic translation of the design space model into a mathematical model for mono-objective optimization, which can be exported to a language such as MiniZinc [42], which decouples the DSE problem from the solving solution.

As will be explained in Section 3, the proposed modelling approach adheres to [28–33] since it is component-based and software-centric, and supports MDA features for enabling concurrent development and industrial development. Moreover, the proposed modelling methodology, is specifically suited for DSE, since it breaks the limitations of previous methodologies [22–27] to reflect in a single model a design space comprising a variety of parameters and implementation alternatives. Moreover, the proposed methodology enables an efficient search of optimum solutions through accurate and fast simulation of each design solution, rather than just a filtering of the solutions compliant with the time constraints based on schedulability analysis. The most remarkable differences at the modelling level are that our approach relies on MARTE, and that it is component-based and supports a very general application model. Previous approaches, such as MILAN [34], Artemis [53], or Koski [37] rely on streaming based models, adhering to models of computation such as SDF or KPN. There are other lacks on features pursued by our approach. For instance, MILAN requires model refinement for architectural mapping exploration. DESERT [36] objective is to perform a prune of the design space in structural terms, before the application of behavioral, thus simulation-based, analysis methods. MODES [39] currently targets a static analysis tool for performance assessment, although authors say that a SystemC simulation target is also in plan.

2.2. Performance estimation technologies

Currently available simulation technologies present limitations for fast, sufficiently accurate performance analysis before architectural mapping. Pure functional simulation does not provide performance metrics. Thus, no performance information can be obtained from the direct execution of the C or C++ application code. Matlab-Simulink models enable the evaluation of different application architectures, based on components consuming and producing data at certain rates. This information obtained can serve to infer constraints on the implementations of these components. However, it does not reflect any specific implementation [43].

Modular and compositional performance analysis techniques [44,45] are promising approaches that can provide global timing and resource utilization properties of the system. These approaches abstract applications and platform resources to different formalisms, such as network calculus [45], or workload models using formal scheduling analysis techniques and symbolic simulation (Sym/Ta) [46].

Simulation-based performance analysis is still required for accurate estimation of complex architectures which cannot be easily abstracted into analytical models. A simulation-based approach is capable of considering the dynamism of the system at a fine level of granularity, and it can yield accurate estimations suited to the most likely conditions of the working environment provided that an accurate model of such an environment is provided.

However, not all simulation-based techniques are suitable for DSE. There is a trade-off between the level of abstraction and the accuracy obtained. Several simulation-based performance analysis techniques can be distinguished attending the different accuracy-simulation speed trade-off.

In *workload models*, the application is conceived as a network of concurrent activities such as threads or actors connected through communication media such as channels, shared variables, etc. In this approach, performance annotations (workloads) are associated to concurrent activities and channels. The main advantage of this kind of models is that they can be applied before code development [47]. Workloads can be implementation agnostic and facilitate a first quantitative approach to the consideration of different targets, e.g. through SW-like and HW-like workloads. The accuracy can be improved once the code in each concurrent activity is known as then the execution times and even the power consumption can be estimated from the source code [48] or from a binary counterpart [49]. Although these approaches can provide very accurate global performance figures of the system behavior under different architectural mappings, they miss details which regard to the execution of the application code in a specific processor architecture, compiler optimizations, and of the interaction of the processor with other elements of the platform, e.g. instruction and data caches.

A classical approach to achieve more accuracy has been the use of instruction set simulators (ISS) for modelling the behavior of processor models, as core element of a virtual platform [50]. This virtual platform can contain other elements, such as cache simulators, and bus and HW accelerator models. For SW performance estimation, specific performance annotations can be associated to each executed instruction. In order to consider the aforementioned processor architecture details, cycle-accurate ISS have been developed. However, they do not provide and acceptable simulation-speed for a DSE process requiring thousand or million simulations. Moreover, the portability of these approaches is reduced since ISS are instruction architecture specific and cycle-accurate ISS processor architecture specific. Virtualization and source-level models are more acceptable and promising approaches.

Process virtualization is based on binary translation technology, e.g. QEMU [51]. Virtualization achieves higher simulation speed than traditional ISS, for functional simulation. Virtual platforms based on process virtualization are able to provide enough accuracy to ensure the functional and non-functional correctness of the design. The main drawback of this approach for DSE is that it requires the development of the complete SW stacks to be executed by each processing node. However, such SW stack is often not available at the beginning of the design process. Moreover, when the virtualization model is adapted for providing accurate execution times and power consumption, the simulation speed is drastically reduced and so the number of simulation runs that can be afforded at this level of abstraction.

Source-level models can provide more accuracy with short execution times for design-space exploration. Accuracy is important to make the design points distinguishable and to validate the decisions taken on then. In a source-level approach, the source code of a functional model of the system application is instrumented with

a set of annotations. A model of the platform and of the mapping of the application to the platform enables to determine annotation values and thus a precise assessment of the impact of each design alternative. The application model can be as simple as a network of actors described as Finite-State Machines with several execution modes [52], or a complete application [53]. There are two main variants of the source-level approach, the ones based on traces and native simulation.

A trace-based approach consists of two steps. In a first step, the source code is analyzed and annotated with a set of commands which encode the activity of the processor, e.g. executing, accessing memory, etc. Then the simulation serves to obtain a sequence of such commands which is named *trace*. Traces depend on the input stimuli, but are independent from the specific design solution. Therefore, traces are extracted only once for a specific stimuli. In a second phase, an off-line processing of the trace shows the effect on performance of each design solution. The difference comes because trace commands are annotated with specific times and power consumptions depending on the specific architectural mapping and platform configuration. Moreover, traces are re-scheduled, considering also the specific scheduling policy of the design solution. When an abstract model of the OS is used, additional traces have to be considered [54]. Re-scheduling is avoided when a deterministic model of computation (MoC) is used [52,53]. Trace-based simulation has been proposed as an alternative to virtualization in order to construct accurate virtual platforms for complex, heterogeneous many-core systems supporting DSE. To achieve this goal, multiple atomic traces per basic block allowing an accurate reconstruction of the processor's behavior have to be used [55]. Higher accuracy and flexibility in the architectural mapping alternatives on a heterogeneous platform comes at the cost of a more complex analysis of a higher number of traces.

Native simulation technologies have been proposed to generate virtual platforms at the beginning of the design process [56,57]. The methodology is very similar to trace-based simulation. An executable performance model of the system is produced by annotating the source code of the application model. The fundamental difference with trace-based simulation is that the code is instrumented directly with back-annotated information able to provide the estimated performance figures, e.g. execution times and power consumption, during and immediately after the simulation. The performance estimation and code annotation can be made directly from the source-code by considering weights associated to source-level operations. A more precise method is to calculate the annotated weights after cross-compilation. This method is trickier since it requires a correlation of the original source code with the object code produced, whose structure can be significantly changed. It also requires the availability of the cross-compiler. However, last advances have enabled the application of native estimation to binary code [58], which has widened its applicability for simulation-based DSE.

While a trace-based approach can save simulations, native simulation is more general in the support of application models such as the one proposed in this work. Trace-based approaches, e.g. [37,53], require relying on a computational model, such as KPN which ensures the validity of the reused trace when the different design solutions are explored.

2.3. Design exploration frameworks

Analytical DSE approaches abstract the DSE problem into a mathematical or formal model, which enables the application of known solving and optimization techniques and tools. They have tackled the mapping and scheduling problem in separated phases [60] due to its NP-complex nature. Later work [61] has proposed constraint-based programming (CP) to achieve completeness, and to separate the design problem (constraint modelling) from the solver engine (constraint solving). Later approaches have enabled the finding of several mappings to be exploited by a run time scheduler, like in [62], where each mapping reflects a Pareto optimum considering a throughput-energy trade-off.

Analytical approaches have traditionally relied on simplified modelling approaches, e.g. the synchronous dataflow (SDF) or the homogeneous SDF MoCs. The scenario-aware or SA-SDF based approach in [63] has enabled a better consideration of application dynamism for analytical approaches. However, it still relies on worst-case workload estimation techniques, e.g. WCETs [59], and difficulties are found when advanced architectures which improve average optimization, but spoil predictability have to be taken into account. Moreover, analytical DSE techniques have focused on a limited set of performance metrics, typically throughput, due to the challenge of finding precise analytical models for them.

Several advanced simulation-based DSE frameworks have been recently proposed. *Sesame* [64] is an environment, integrated in the *Daedalus* [65] system-level design flow. *Sesame* selects candidate architectures using analytical modelling and multi-objective optimization relying on a tool called SPEA2, while it uses simulation for evaluating the candidate architectures. The interface between problem-specific and generic parts of the exploration framework is made explicit by defining an interface called PISA [65].

Multicube [66] and NASA [67] have proposed generic infrastructures enabling the coupling and combination of different simulation-based performance estimation and exploration tools for system-level MPSoC DSE, which avoids recompilation of the executable model for each architectural alternative explored. Moreover, NASA proposes a *dimension oriented approach,* which consists in the possibility of enabling a concurrent exploration (co-exploration) of various design dimensions, which potentially can employ different exploration algorithms. NASA simulation framework relies on MoCs which avoid deadlock by construction for facilitating the automation of the exploration. NASA does not support a MBD front-end.

The exploration algorithm or exploration tool is important in order not only to avoid the exhaustive exploration of a huge design space in a limited and feasible time, but also to make it as effective as possible. Exploration algorithms need some intelligence to decide the experiments to perform in the most efficient way. Traditional techniques have proposed Design of Experiments (DoEs) [68], Response Surface Modeling (RSMs), e.g. DPSO [69], and multi-objective optimization heuristics, such as MOSA [69], NGSA-II [71] and MOE/AD [72].

The approach shown in this paper has evolved Multicube-Explorer [20], developed in the context of the Multicube project [66], towards SCoPE+, a multi-level performance exploration tool, supporting SW and HW exploration, and implementation agnostic and component-based application models, directly derived from the UML/MARTE model. This model is more general than the KPN front-end of Sesame [64] or NASA [67]. By relying on SCoPE+, thus on native simulation, the accuracy of the method is ensured with regard to the trace-based performance estimation methods. As NASA [67], the proposed approach is in the context of a framework based on XML-based interfaces, which facilitate the integration of other performance estimation and exploration tools, and which avoid the regeneration of the executable model.

3. UML/MARTE Modeling methodology

3.1. Basic features

The system Modeling methodology described in this paper covers the main demands addressed at the end of Section 2.1. It follows a *component-oriented* approach; it is software centric;

and it follows *Model Driven Architecture* (MDA) concepts. The design exploration and implementation activities are developed around the model, which is also amenable to be used for static analysis and for generating documentation. The methodology also enables modelling and estimation of implementing application components in HW (HW mapping). Another remarkable feature is that the COMPLEX methodology supports the separation of concerns paradigm, keeping the functional and non-functional concerns well-differentiated. This separation is achieved by providing distinct model viewpoints to the designer, in the shape of UML packages, each one focused on a relevant aspect of the system. In particular, the views supported are the following:

- *Data View:* captures the data model that is used at system level, i.e., types of data exchanged at interface level.

- *Functional View:* defines the functional entities of the system, as a set of classes implementing and using a group of interfaces.
- *Communication and Concurrency (CC) View:* copes with the definition of the different components and their assembly to model the application architecture. Components wrap instances of the classes defined in the functional view, delegating the operations of their provided interfaces on those interfaces. It also captures the non-functional aspects of system functionality related to the application behavior, such as concurrency and real-time constraints.
- *Platform Description View:* describes both software and hardware resources of the platform.
- *Architectural View:* describes the platform architecture and the architectural mapping of the application components onto platform processing resources. It is also the view where the DSE parameters, rules and constraints that will enable the exploration of the different architectural solutions are captured.
- *Verification View:* is user for modelling the system stimuli environment.

3.2. System Modeling: A single design solution

3.2.1. Description of the Platform Independent Model (PIM)

The initial steps in the methodology consist of identifying the system functions, modeled through the Data Model view and the Functional view.

The Data Model view is captured as a UML package with the COMPLEX <<Dataview>> stereotype applied. A similar procedure is applicable for the rest of model views. All the information corresponding to a view is captured within a UML package decorated with a COMPLEX specific stereotype which identifies the view.

The Data Model view contains all the non-primitive data structures which will be exchanged by model components. Capturing this information in the model prevents incompatibilities. Moreover, it also helps to provided information about complex data structures, e.g. size, which can be later exploited by the performance assessment methodology. UML classes are used to declare new data types in the model. The MARTE *<<CollectionType>>* stereotype is used to capture regular structures, i.e. arrays, as shown in Fig. 2.

System functions are captured by means of the use of UML use cases and the relations between them. The UML use cases will allow the designer to identify the UML interfaces that model the system functions and create the UML classes that would implement them. System functions are modeled by means of UML interfaces stereotyped with the MARTE <<ClientServerSpecification>> stereotype. The operations of <<ClientServerSpecification>> interfaces might have parameters whose types should have been defined in the Data Model.

Fig. 3 shows the functional view of an enhanced full rate (EFR) vocoder system [76]. An EFR vocoder is a compression/decompression system widely employed in (GSM) mobile telecommunication to save bandwidth. A vocoder is composed of an encoding branch and a decoding branch. The encoder is in charge of transforming raw audio frames into a compressed format, according to a model of the human voice production, which considers parameters such as the pitch. The decoder branch performs the inverse operation. Four classes enclosing Vocoder functionality are shown: "Audiocontroler", *"Coder"*, "Decoder" and "VoiceActivity". These classes realize services pointed by the operations of the *<<ClientServerSpecification>>* interfaces, captured by means of UML interface realizations. For instance, the VoiceActivity class provides a functional implementation for *detect* and *reset* methods of the VADIF interface. These classes can also use services provided by other classes, which is modeled by means of UML "use" relationships. For instance, the "Coder" component uses voice activity detection service, by calling the *detect* method. As shown in Fig. 3, interface methods declare input and output parameter, whose type is declared in the data view.

Moreover, classes may require interfaces that are provided by the surrounding environment (i.e. devices connected to the system). The COMPLEX *<<ExternalInterface>>* elements in the bottom of Fig. 3 model the services that the vocoder requires from the environment through the expression of UML usage relationships between the vocoder classes and those interfaces. Additionally, it is possible to declare services provided to the environment through the declaration of UML interface realizations of external interfaces. This is the case of the *CoderCtrlIF* interface, which is provided by the vocoder and thus can be required and used by the environment.

The concurrency and communications (CC) view defines both the application components and the application seen as an assembly of components communicating through well-defined interfaces.

In the first step, the application components must be captured. The CC view contains two kinds of components stereotyped with either MARTE <<RtUnit>> or MARTE <<PpUnit>> stereotypes. The former stereotype identifies a component that has its own execution flow, providing/requiring services to/from others components by means of its provided and required interfaces. The latter represents a passive component that provides concurrent or guarded access to its services to the former kind of components. The functional behavior of those components is defined by instantiating the functional classes as properties of the component and delegating the component interfaces on those properties. Components expose UML Ports decorated with the MARTE *<<ClientServerPort>>* and *<<RtFeature>>* stereotypes for defining the provided or required interfaces that serve for communicating with other components, and for specifying real-time non-functional attributes for each operation of the interface.

In a second step, the architecture of the platform independent model (PIM) is captured. As the proposed approach is software centric, a default implementation is a SW implementation, and the PIM reflects an application. Thus the PIM architecture is the application architecture. First, the user has to add to the CC view an additional UML component decorated with the COMPLEX <<system>> stereotype which represents the PIM or application. The application architecture is captured by means of a composite diagram, where application components, such as the one shown in Fig. 4, are instanced and interconnected.

As an example, to capture the EFR vocoder PIM, first a <<system>> component named "EFRVocoder" is declared in the UML package corresponding to the CC view. The internal structure of the "EFRVocoder" component is then described through the UML composite diagram shown in Fig. 5, which reflects the architecture of the EFR vocoder application. It consist of four component instances: "controllerComp", "decoderComp", "coderComp", and "VADComp". Each instance, e.g. "coderComp", is an UML property

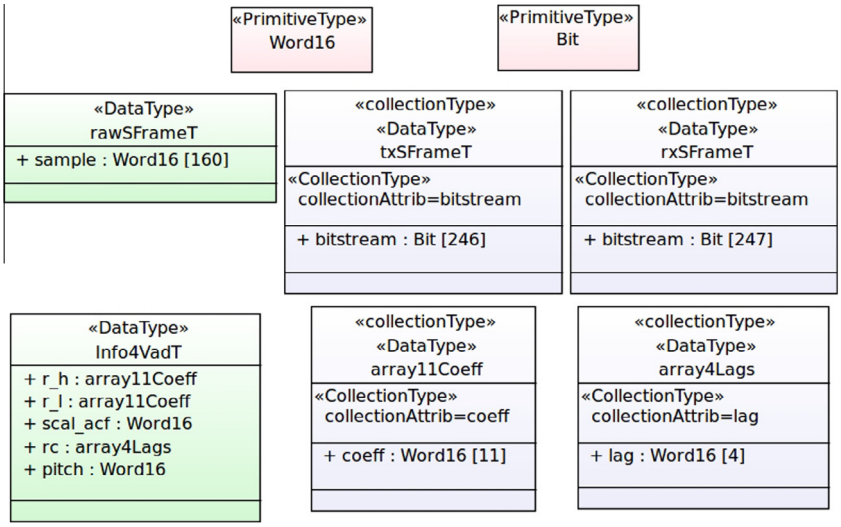

Fig. 2. Data model of the EFR vocoder.

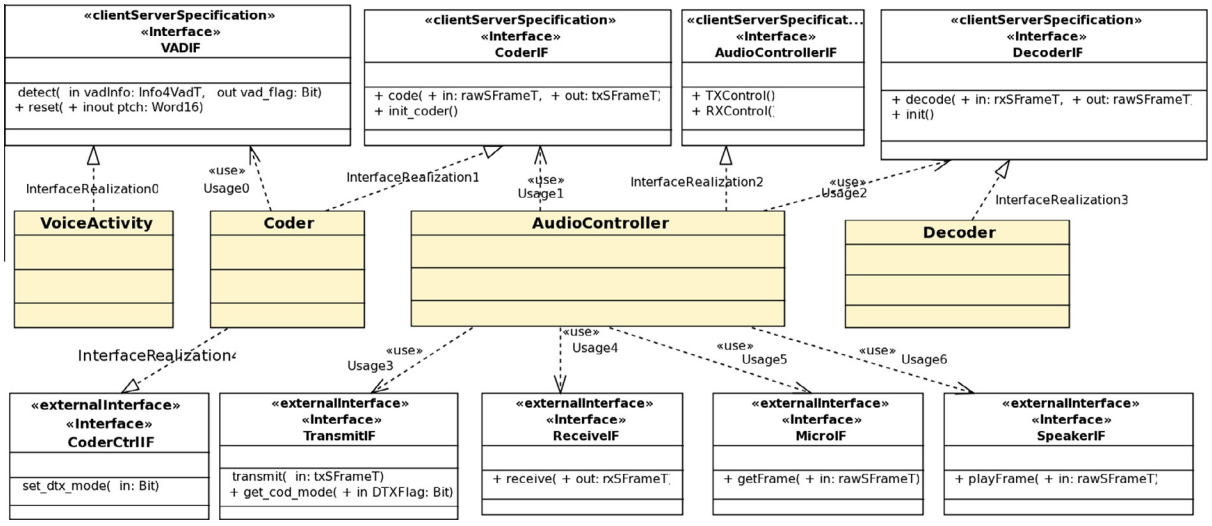

Fig. 3. Functional View of the EFR vocoder showing interfaces and classes.

typed as a component previously defined in the CC view, e.g. "CoderComp". Component instances are assembled through UML port to port connectors. The <<system>> component may also define external ports to interact with the system environment. Those UML ports are not stereotyped but typed with the interfaces which have been declared as COMPLEX <<ExternalInterface>> in the Functional view.

3.2.2. Platform Model and Architectural Mapping (PSM)

The proposed modelling methodology supports the description of the HW/SW platform within two views, the Platform view and the Architectural view.

The Platform view contains the declaration of the SW and HW components which will appear in the platform. These components are declared by means of UML classes (thus in a UML class diagram) decorated by MARTE sterotypes. An RTOS is declared by relying on the MARTE <<Scheduler>> stereotype. Declaration of HW components relies on the MARTE Hardware Resource Modeling (HRM) subprofile (i.e. <<HwProcessor>>, <<HwBus>>, <<HwCache>>, <<HwRAM>>, etc.).

The architecture of the platform is captured within the Architectural view, specifically within a UML component ("archi_system" in Fig. 6), decorated with the COMPLEX <<system>> stereotype again. This <<system>> component inherits from the COMPLEX <<system>> component defined in the CC view, and therefore it can refer application component instances and ports (show with the arrow symbol in Fig. 6). The platform architecture, as reflected in Fig. 6, contains instances of HW and SW components declared in the Platform view (i.e. RTOS, processors, memories, etc.), and port to port connections among them. Here, instance specific values for component attributes (e.g., a processor instance can be assigned a frequency different from other processor, regardless the process type is the same) can be also captured.

The architectural view describes also the architectural mapping, that is, the mapping of application component instances to platform component instances. For it, UML abstraction relationships decorated with the MARTE <<Allocate>> stereotype are used. It is noted that the external ports defined in the PIM are now delegated on specific HW instances which model <<HwIO>> devices.

3.3. Features for DSE: Modeling a design space

The diagram of Fig. 6 models a single implementation since it shows a fixed architecture, a single architectural mapping, and

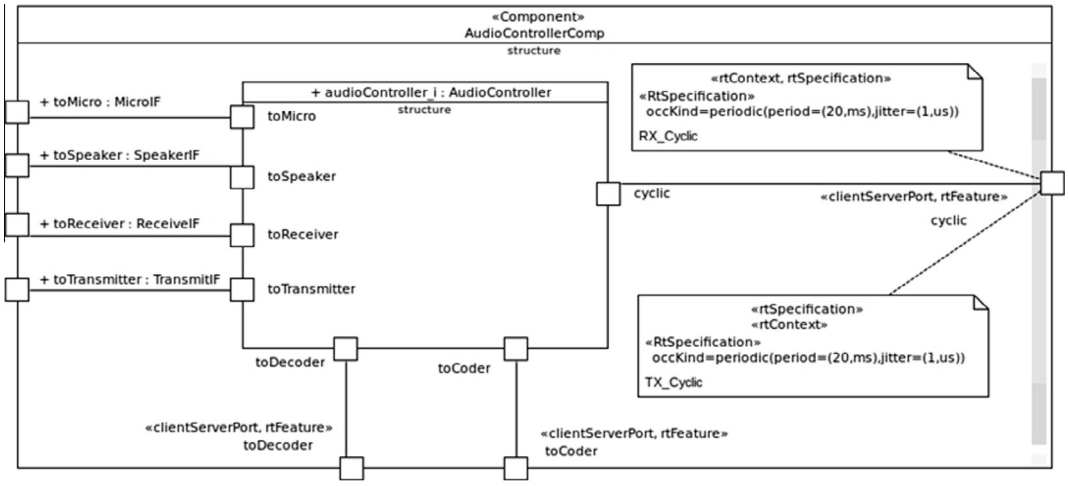

Fig. 4. Declaration of the *AudioController* component in the CC view.

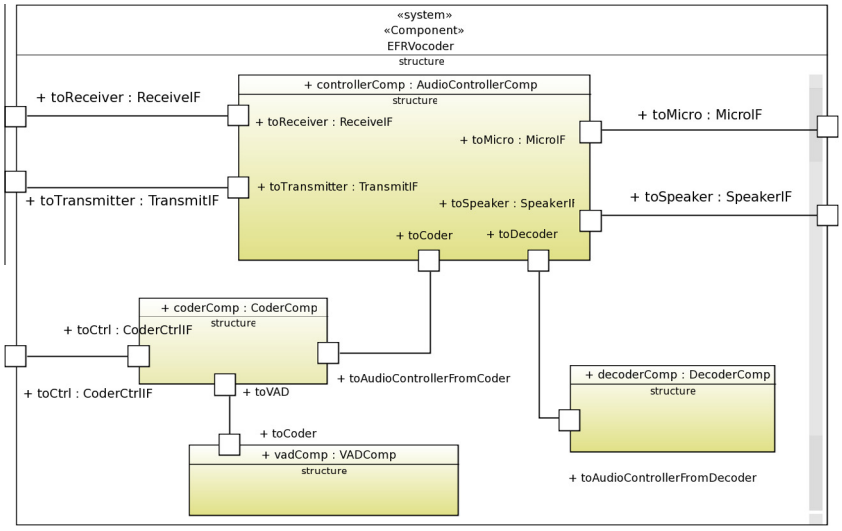

Fig. 5. Architecture of the EFR vocoder Application.

fixed attributes for platform components (e.g. processor frequency, memory size, etc.). The COMPLEX UML/MARTE modelling methodology enables the specification of a design space instead of a fixed solution.

3.3.1. Specification for Design Space Exploration

The COMPLEX UML/MARTE Modeling methodology enables the specification of a design space which can consist of (a) a set of possible architectural mappings (allocation space); (b) a range of values for platform attributes (attribute values space); and (c) a set of platform architectures (architecture space). Enabling a parameterization of the model for DSE purposes on these three aspects enables the description of a wide set of implementation alternatives, which should cover almost any exploration need.

An approach where any application component could be mapped to any targetable platform component, or where each attribute could take any feasible value would easily lead to an intractable design space size. In contrast, the proposed methodology enables a controlled building of the design space. The idea is that the user has to explicitly add the constructs that build up the design space on top of a "base model" like the one presented in the previous section. Such constructs override fixed attributes which could clash with them. This way, in an implementation context, the COMPLEX UML/MARTE model can reflect "one solution" through the "base model", which eventually, can be updated with the values resulting from the DSE process. The same COMPLEX UML/MARTE model can also reflect a design space in a DSE context (the one we are focusing in this paper). In general, for DSE purposes, the "base model" does not need to be complete. However, the "base model" plus the design space captured has to reflect one or more solutions, which configure the design space.

An explicit description of the design space considering the aforementioned aspects can still easily lead to a huge design space. Because of this, the modelling methodology enables shaping and constraining the design space through the definition of *DSE rules* and *DSE constraints*. Finally, the methodology also enables the definition of the output metrics to be considered in the building of the goal functions employed by exploration tool in the DSE loop. A dedicated UML profile, the COMPLEX profile, has been created to add the necessary semantics that are missing in the MARTE profile with regard to the aforementioned features. Four entities have been created to represent the necessary concepts: DSE exploration parameters, of three possible types (allocation parameters, scalar parameters and vector parameters), DSE Rules, Constraints and Estimation parameters (for defining output metrics).

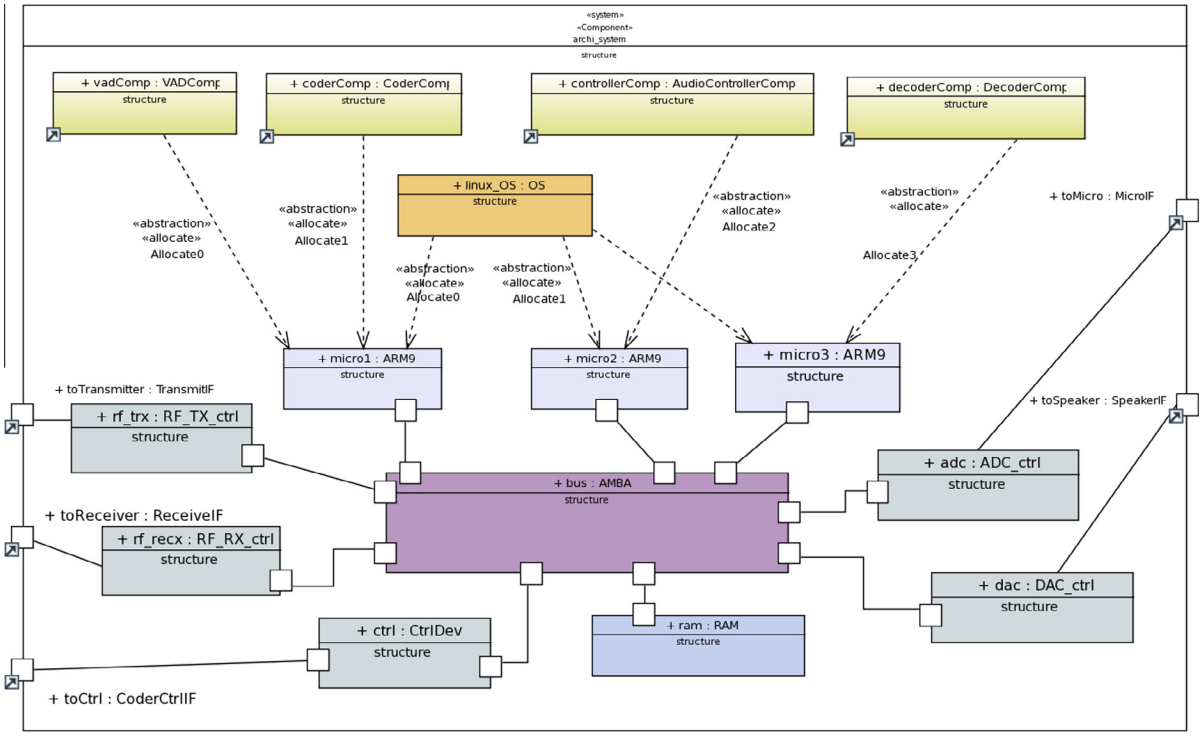

Fig. 6. Architecture of the EFR Vocoder system.

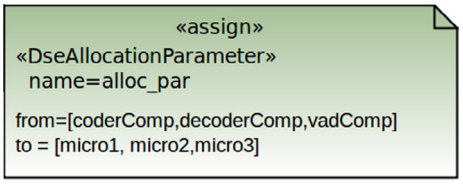

Fig. 7. Specification of a set of architectural mappings.

3.3.2. Definition of the design space

3.3.2.1. Specification of a space of allocations. Architectural mapping is a factor with a big impact on performance. This methodology enables the description of a set of architectural mappings. This set is captured through one or more UML comments placed in the Architectural view, and decorated with the MARTE <<Assign>> stereotype, to specify the allocation itself, and the COMPLEX <<DseAllocationParameter>>, which provides a name attribute to the allocation. The name of the configurable allocation can be later used by the DSE rules to bind the exploration space (see Section 3.3.3).

Fig. 7 shows an example for the EFR vocoder model which models 27 possible allocations (resulting from the combination of mapping application component instances on *micro#* processing nodes). Specifically, the combinations arising from considering the allocation of "coderComp" to any of the processors "micro1", "micro2", and "micro3" and similarly for the component instances "decoderComp" and "VADComp". As shown, it is expressed in a compact way through a single construct. The assign construct defined a space of allocation solutions which can be explored. Each of this mapping solutions override any clashing allocation defined with the <<allocate>> association in the based model.

3.3.2.2. Specification of an space of attribute values. Many platform attributes can have a significant impact on system performance (cache sizes, core and bus frequencies, etc.). The proposed methodology enables that the user states a subset of the attributes of the model as DSE parameters. It is done through UML comments which are associated to the components which contain the attributes which are being declared as a DSE parameters and thus, which will be explored. For capturing a DSE parameter, either a <<DseScalarParameter>> or a <<DseVectorParameter>> COMPLEX stereotype is applied to an UML comment. These stereotypes enable the association of a range of values the attribute they refer to, and which will override at exploration time the default values at the base model. That is, the exploration tool will assign one of the values stated by DSE parameter at exploration time. A variety of attributes can be defined as DSE parameters, including working frequencies, sizes, bus widths, etc.

DSE scalar parameters specify a sequential progression associated to a specific non-functional property. The designer can specify either minimum, maximum and step values to define possible values or annotate a specific sequence of values. DSE vector parameters enable modelling vector parameters with constraints on the possible combinations of the elements.

Fig. 8 shows the Modeling of the platform parameters space for the EFR Vocoder model. In the example, this space is defined by the possible values of data and instruction cache memories (1K, 2K, 4 K, 8K and 16K); and by the consideration of different core frequencies, which is specified through other three DSE scalar parameters. Thus, the size of the space of platform attribute values in Fig. 8 example is 5 (core frequencies) $\times$ 5 (data cache sizes) $\times$ 5 (instruction cache sizes) = 125 design alternatives only for the first processor, augmented by the frequencies combinations for the other processors.

3.3.2.3. Specification of several platform architectures. DSE parameters and <<Assign>> comments widen the design space without changing the platform architecture. However, the user might be interested in exploring the impact on performance of different platform architectures. The COMPLEX UML/MARTE modelling methodology supports the specification of different platform architectures

in the same model. It is done by enabling the specification of several architectural views, in a similar way as several architectures can be associated to an entity in VHDL. The designer can include in the DSE loop solutions where the platform architecture is very different and thus the alternatives cannot be represented by means of a parameterized template. Additionally, it is also possible to capture a cluster of processors through a single component instance, whose multiplicity is associated to a DSE parameter.

3.3.3. DSE rules

DSE rules reflect additional *design constraints* which enables the reduction of the size of the design space. They can be used to eliminate solutions that might not be feasible because of practical reasons, e.g., certain physical platforms are not available.

DSE rules are logical expressions referring one or more DSE parameter included in the model. DSE rules create a logical condition that in case of not being fulfilled discards the design point from the set of solutions to be explored. DSE rules are specified as comments with the COMPLEX <<DseRule>> stereotype.

A straightforward application of the DSE rules is over the scalar DSE parameters. For instance, the user can specify a rule to state that the instruction and data caches have to take always the same size in the exploration space, as shown on Fig. 9a.

However, DSE rules can be applied in a more general way, to state which stimuli environment to explore, or to limit the allocation space, since the assign constraint can reflect many allocations. As a simple rule, the allocation space defined by the rule of Fig. 7 can be reduced by the set of rules shown in Fig. 9b stating that the "coderComp" and "decoderComp" components cannot be allocated to the same processor. This would reduce the allocation space from 27 to 18 mappings.

3.3.4. Estimation parameters

The methodology allows stating in the model which output metrics have to feed the exploration loop. They are called estimation parameters in the sense that, in a DSE context, performance metrics are obtained by means of a performance estimation technology. There are two types of DSE estimations. DSE *Application Estimations* refer to non-functional attributes corresponding to the application components, e.g. the minimal inter-arrival time of a sporadic service. DSE Platform Estimations refer to the performance metrics on the platform resources, e.g. load of CPU, power consumption, etc. The estimation parameters are defined in the Architectural View by means of UML comments stereotyped with the COMPLEX <<DseAppEstimation>> and <<DsePlatformEstimation>> stereotypes.

Fig. 10 shows how the model states the reporting of two application performance metrics of the EFR vocoder to the exploration loop: the mean and the maximum execution time of the TXControl and of the RXControl operations (associated to the "controllerComp" component). These metrics can be used for the definition of the objective function(s). The definition of the objective functions is tool-dependent, and thus delegated to the COMPLEX toolset. Estimation parameters can be later referenced to define DSE rules and constraints.

3.3.5. DSE constraints

In the proposed methodologies DSE constraints refer to *performance constraints*. DSE constraints are logic expressions referring to one or more estimation parameters. That is, DSE constraints support the reduction of the set of feasible design alternatives by relying on output metrics. DSE rules directly bound the design space. DSE rules lead to discard design points, which required no evaluation, that is, no simulation. However, DSE constraints do not prevent the evaluation of design points. They are relevant after each evaluation (simulation). Specifically, the COMPLEX toolkit, before completing the DSE loop iteration, reads the output performance metric and evaluates the logic expression to check the validity of the design point. If the logic expression is proved to be false the framework discards such design point from set of feasible solutions reported. For instance, the DSE constraint of Fig. 11 states that the system power consumption, a platform performance metric, has to be reported and that any design solution involving power consumption over 2 W has to be discarded.

3.4. Modeling of the stimuli environment

In the proposed methodology, the environment is captured through UML elements and by relying on the UTP standard profile. As shown in the following subsections, the methodology provides a structured way to capture all the information of the stimuli environment model, clearly separating it from the system model. Relevant information of the environment model includes the declaration of environment components; how the environment components are interconnected with the system; and how the system and environment services are called, that is, if there is any specific pre-stated order among those service calls. Completing all this information enables to complete the description of an *environment scenario,* which model the input stimuli for a specific use case.

Furthermore, one novel and DSE specific feature of the proposed methodology is that the same environment model can comprise several *scenarios*. In correspondence with this modelling capability, the COMPLEX UML/MARTE tooling supports the generation of an executable model where the scenarios to be simulated can be configured. This way, the exploration tool can simulate the performance of the system for one scenario, a specific set of scenarios, or all the scenarios captured in the environment model. This becomes only a matter of the explorer configuration script, which can be easily configured through the COMPLEX UML/MARTE tooling.

The proposed methodology delegates the inclusion of functional code to the user after the code generation phase. The injection of functional code is more practical later on at the SystemC level,

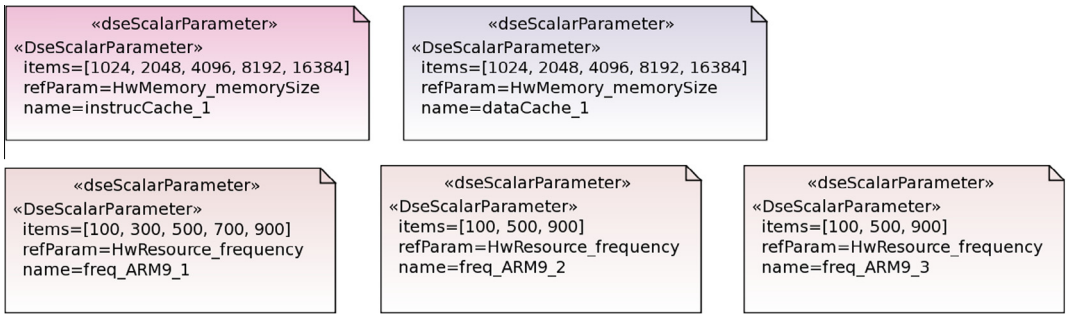

Fig. 8. Attribute value space for the EFR Vocoder model.

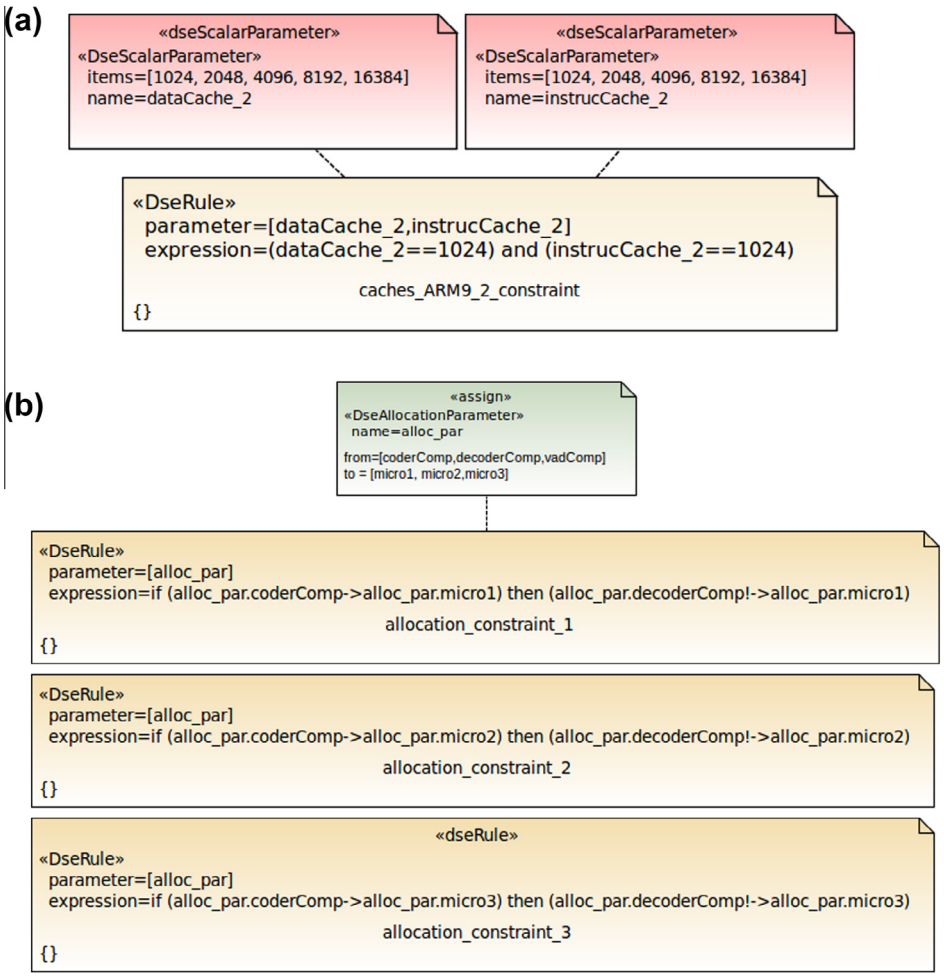

Fig. 9. DSE Rules.

where the user can simply identify hook functions produced by the generator of the SystemC stimuli code, and fill them with user-specific SystemC environment functionality.

3.4.1. Environment structure

The environment model is enclosed within a specific view of the COMPLEX UML/MARTE model, the *Verification view*. The verification view is modeled as a UML package with the COMPLEX <<VerificationView>> stereotype applied. This way, a tool-independent separation of system and verification elements in the model is achieved.

The verification view declares the whole set of environment actors as a set of UML components with the UTP <<testComponent>> stereotype applied (bottom part of Fig. 12).

For instance, in the EFR vocoder environment model:

- The *Microphone* component provides a fixed number of frames (N) corresponding from a standard test to the system to be coded.

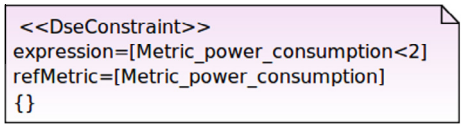

Fig. 11. DSE constraints.

- The *Transmitter* component receives the frames that have been coded by the coder from the inputs generated by the microphone.
- The *Receiver* component emulates the reception of coded frames to deliver them to the system for decoding.
- The *Speaker* environment component receives the output of the decoder.
- The *Configurator* environment component calls the *set_dtx_mode* configuration service of the vocoder to state if the vocoder works in continuous or discontinuous mode. In the latter case,

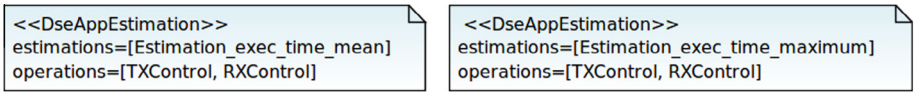

Fig. 10. Estimation parameters.

the voice activity detection functionality is enabled and the coder can get a better compression if not voice, but just noise is detected.

An additional UML component with the UTP <<testContext>> stereotype is used for the declaration of a *top* component (*StimuliCompo* in Fig. 12) which will be used to model the structure of the verification environment, and moreover its interconnection with the system through an associated UML Composite structure diagram. Fig. 13 reflects the structure of the EFR vocoder verification environment and its interconnection with the EFR vocoder system. It is composed of instances (UML properties) of several environment components (*<<TestComponent>>* components).

The system component is captured as a reference to the system component of the Architectural view. Using a reference to the system component of the architectural view keeps the model compact and saves coherence checks between the verification view and the system views. The application of the UTP <<SUT>> (System Under Test) stereotype to the system component instance, facilitates its identifyication it among environment component instances by the code generation tool, which does not need to navigate the whole model.

Port to port interconnections among the system component and environment components fix which environment components will implement the services called by the system, and which environment components will call services provided by the system to the environment. For instance, in the EFR vocoder model, the *configurator* environment component is linked through a port to the system *toCtrl* port, to state that the *configurator* component will call (thus will use) the *set_dtx_mode* service provided by the *CodeCtrlIF* interface.

3.4.2. Modeling one scenario

In order to complete the description of one scenario, an explicit scenario declaration has to be done. A scenario is captured as a UML package belonging to the verification view package, and stereotyped with the COMPLEX <<scenario>> stereotype. Then a new <<TestContext>> component has to be declared within the *scenario package*. Moreover, the new <<TestContext>> component must be declared as a specialization of the *top* <<TestContext>> component used to capture the structure of the environment and of the system-environment interconnection (Figs. 12 and 13). The UML generalization association is use for it, as shown in Fig. 14.

The advantage of this construct is that one description of the environment structure and its interconnection with the system can be easily reused for describing several scenarios. Then, in order to complete the description of the recently created scenario, its *behavior* has to be captured.

3.4.3. Modeling the scenario behavior

The scenario behavior refers to the modelling of the interactions between each environment component and the system component.

Each interaction is a totally ordered sequence of service calls between the system and the environment component. Service calls can be done from the system or from the environment; and can be synchronous or asynchronous.

The interactions are captured by means of UML sequential diagrams. The sequential diagram reflects an ordered sequence of UML messages. In the proposed methodology, such messages represent function calls, that is, service calls, owned by <<ExternalInterface>> elements which are provided either by the system or by the environment. The sense of the messages states whether the system calls a function provided by the environment or, to the contrary, the environment requires a service provided by the system.

Fig. 15 shows an example of sequence diagram capturing the interaction between an environment component and the system component. A UML lifeline references the instance of the system component (the EFR vocoder), while the other lifeline references an instance of the environment component (the *Microphone*). The *getFrame* UML message reflects a call from the EFR vocoder to obtain a new raw voice frame. Fig. 15 also shows the possibility for capturing in a compact way a loop through a UML *loop combined fragment*, which in this example bounds the number of frames send to the ones available by the test (10 frames).

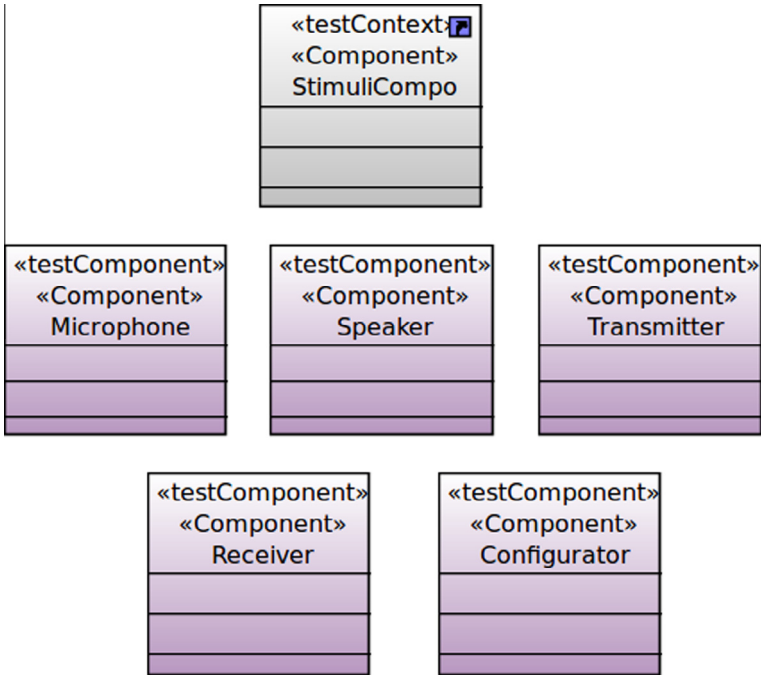

Fig. 12. Declaration of Environment and test components in the Verification view.

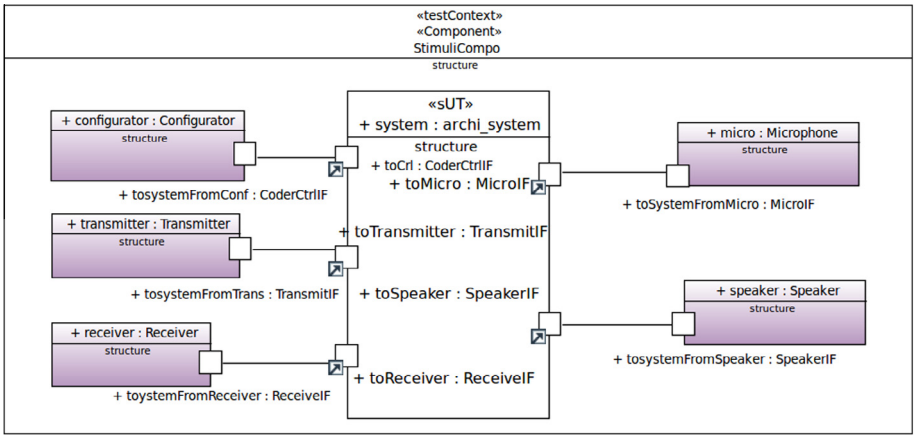

Fig. 13. Structure for the EFR vocoder environment.

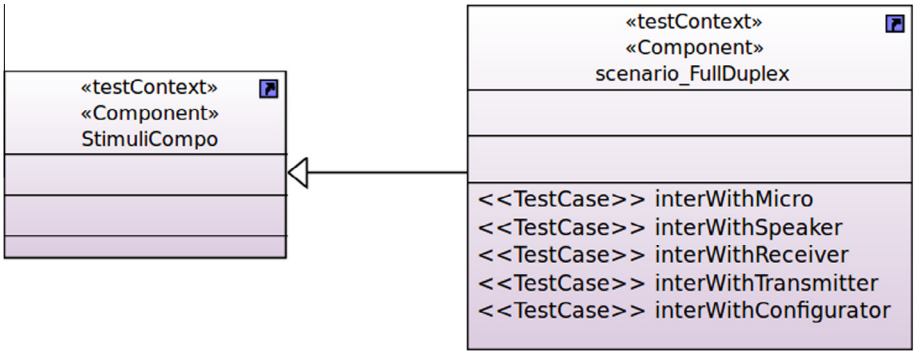

Fig. 14. Declaration of an scenario through a generalization.

Two different types of UML messages are used: synchronous messages and asynchronous messages. They enable the specification of synchronous and asynchronous services. Synchronous services require the return of the function call before continuing the execution, e.g. because the client expects some output information from the service call. An UML synchronous message is represented by a filled arrow head (as the one used in Fig. 15). UML asynchronous messages are represented through an open arrow head.

The sequence diagram graphically represents the order in the exchange of messages. However this ordering information is not contained in the XMI file containing the information of the UML/MARTE model, and which is used as input by code generation tools. Therefore, in order to keep message ordering information in the model, a unique order identifier ("i:") can prefix the names of the messages of the sequential diagram. This index is sufficient when the sequential diagram express the interaction of the system with a single environment component.

The methodology supports a compact specification of the interactions. Specifically, the example of Fig. 16 shows a sequential diagram that reflects the specification of the interaction between the EFR vocoder system and the transmitter and receiver environment components in a scenario which reflects a closed loop configuration, where the transmitter output is directly wired to the receiver input. In such a case, a single sequential diagram (instead of two) serves to capture the interaction of the system with the two environment components.

The methodology also enables to fix order relationships among messages exchanged by different environment components with the system. Specifically Fig. 16 states that the *Receiver* environment component will not serve a receiver frame until a previous frame has been sent by the *Transmitter* environment component. This way, the order semantics of the closed loop scenario is modeled. Since this condition refers to the interaction of the system with two environment components, a UML *combined fragment* has been used.

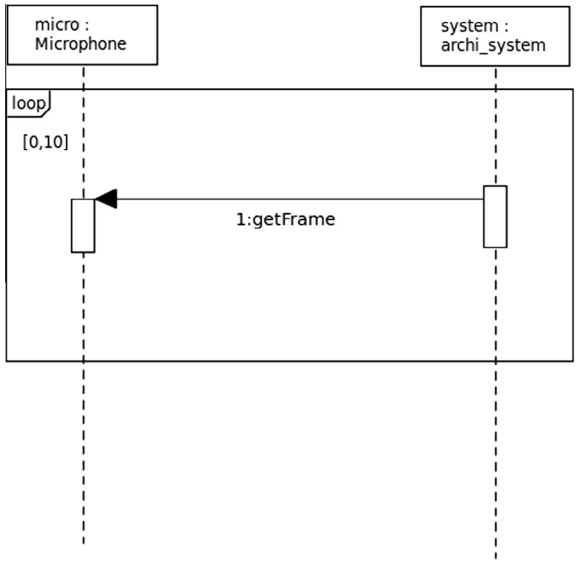

Fig. 15. Sequence diagram for the interaction between the EFR Vocoder system component and the Microphone environment component.

3.4.4. Modeling several scenarios

The proposed methodology supports the modelling of several scenarios. That is, a single environment model covers the stimuli associated with different use cases. As a result, it is possible to perform a DSE exploration for a specific use case or a set of use cases, and to tailor the design for this use case(s). Complementarily, a wide range of scenarios can be used to validate a single design them. Each scenario is modeled by means of a scenario package, which in turn contains the test case component specializing the top test case component. This test case component contains the environment structure with the specific interactions which describe the new scenario.

In Fig. 17 an example of two scenarios for the EFR vocoder case is shown. The first scenario, "*scenario_FullFuplex*", reflects a full duplex case, where the "*Transmitter*" and "*Receiver*" environment components run in parallel. Thus the codification and decodification branches of the vocoder will be able to run in parallel if they are mapped to different processors, or at least in concurrency if they are mapped to a single processor with a round-robin scheduling. A second scenario, "*scenario_local_closed_loop*", reuses the same structure of environment and connection with the system, but defines a different interaction (the one shown in Fig. 16) which emulates the closed loop configuration.

3.4.5. Modeling physical time information

The features presented up to here enable the specification of a partial order of service calls in the environment. Formally speaking, this is the most abstract way to specify time constrains in the environment model. Furthermore, the proposed methodology enables the association of physical time information to the environment model. Specifically, the initiation of each service call can be placed in a specific physical time stamp. In order to specify it, the MARTE <<timedProcessing>> stereotype is used. This stereotype is applied to the UML message resembling the service call placed in physical time. The stereotype provides the attribute "start", which denotes an UML Time Event, which in turn is placed in physical time through an UML Time Expression.

4. Generation of the executable model and exploration

4.1. Complex toolset

The COMPLEX modelling methodology is supported by the COMPLEX Eclipse Application (CEA) tool which provides an integrated framework for the HW/SW co-design and design exploration. This tool is integrated within the Eclipse framework and makes extensive use of the Eclipse Modeling Framework (EMF) [77] and Model-to-Text facilities [78]. This section aims at providing a flavour of the most important features of the CEA tool in supporting the COMPLEX modelling methodology.

The CEA tool (shown in Fig. 18) is based on open-source standards and tools so it guarantees its compatibility with other existing development environments in the market. Interoperability, configurability and extendibility are the main features that drove the development of the tool.

The tool is built around the PapyrusMDT [79] modelling environment, which serves mainly as a UML (graphical) model editor with a set of interesting features supporting the model development:

- Support for editing any model compatible with the Eclipse Modeling Framework (EMF) model, in particular UML2 models and SysML.
- Support for UML profiling by providing extensions to register UML-based profiles, like the MARTE profile or the COMPLEX profile.
- High customizability, in particular the modelling palette, feature highly appreciated by users.

The COMPLEX Eclipse Application provides a set of features to support the COMPLEX methodology in the development of the model. They can be listed in the following bullets:

- Automatic generation of the COMPLEX project structure (i.e. creating COMPLEX views).
- Support to the configuration of the DSE loop and automatic generation of the necessary inputs to the exploration tool.
- Implementation of set of model analysers for ensuring that the model is consistent with the defined modelling rules described in the COMPLEX methodology, and which enforce the separation of concerns by verifying that each model viewpoint only specifies the right modelling entities, e.g. the Functional view only contains classes and interfaces.
- Support for the generation of the performance executable model:
  - o which produces a set of text files (i.e. source code, text files, XML files) with all the information for building the performance model, which can be used by different performance estimation back-ends;
  - o which structures automatically produced source code, separating the ancillary one from the code amenable for user edition;

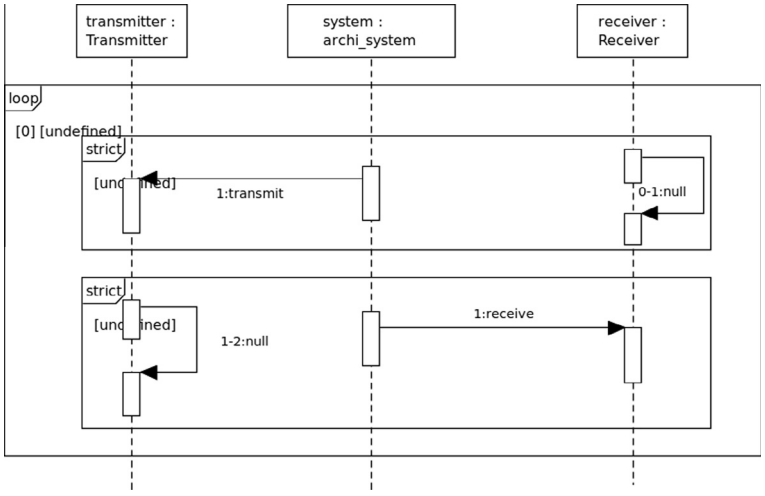

Fig. 16. Sequence diagram for the interaction between the EFR Vocoder and the transmitter and receiver components in a closed loop scenario.

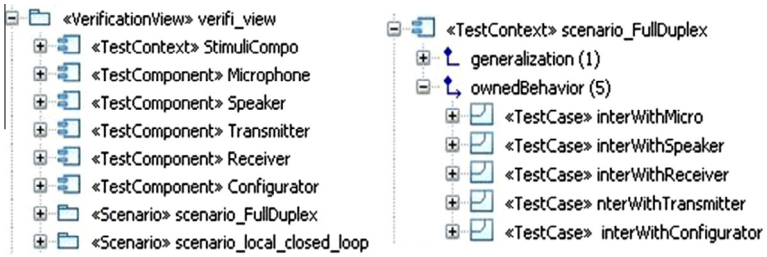

Fig. 17. Several scenarios are supported.

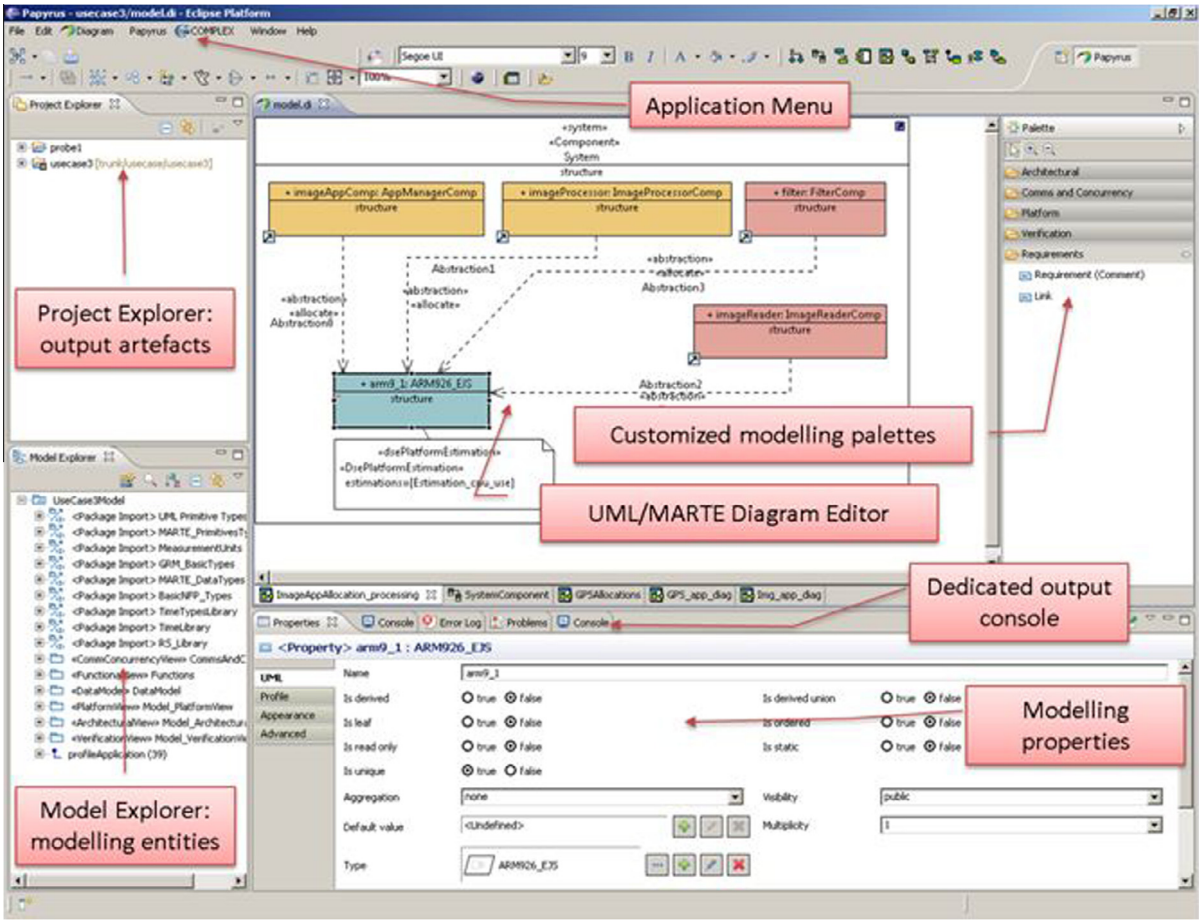

Fig. 18. Snapshot of COMPLEX Eclipse Application.

- which triggers the compilation of the SCoPE+ model, produces the configuration scripts of the exploration tool, and in summary enables a smooth integration with the analysis and exploration tools (i.e. SCOPE+ and MOST tools, respectively), so that the generation of the performance model executable and execution of the exploration tool is fully automated
- Implementation of a customized modelling palette that limits the set of the UML/MARTE design entities to a sub-set for a given model viewpoint and UML diagram type, as defined by the posed methodology

Additionally, the tool affords analysis and transformation extension points by using the existing Eclipse features so that third party developers may add new analysis or transformation engines to the existing ones. This feature improves the extensibility of the tool by allowing adding new functions to interface other tools or methodologies.

4.1.1. UML/MARTE model editor and code generation

Fig. 19 depicts the design flow where the CEA tool is inscribed. The UML/MARTE model is produced by the PapyrusMDT editor, which comprises the PIM, the platform, the architectural mapping, the description of the design space, and the description of the stimuli environment. Additional source code might serve as input to the application and stimuli environment in the UML/MARTE model.

If the analysis and/or transformation engines installed in the tool did not trigger any error, the tool generates the following output artefacts:

- The application source code skeletons, to be implemented by the user with the actual code. It is noted that the tool generates the necessary stub code to simulate the actual code with the non-functional information included in the model. The tool can also automatically compiler, reference and link source code complementing the model with functionality (and overriding the default workloads). It can be also done afterwards, after code generation.
- The application component containers and connectors (i.e. supported by the CFAM front-end of SCoPE+) that wrap the application functional code into components and implements the non-functional concerns declared in the model (i.e. cyclic execution of the application functions, service dispatching).
- The XML files that configure the analysis tool (i.e. SCoPE+ tool).
- The source code and components of the stimuli environment.
- The configuration script that defines the objective functions, constraints and analysis methods for the exploration tools (i.e. MOST tool).
- The IP-XACT specification of the hardware platform description that will serve as input to generate the virtual platform on which the application will be executed.

4.1.2. Model analysers

The COMPLEX methodology makes use of a sub-set of the UML modelling entities and MARTE stereotypes for specifying the system and the stimuli environment. The UML meta-model implemented using the EMF only validates the UML model against the OMG UML standard, so it is necessary to enforce that the given UML/MARTE model follows the modelling rules and constraints specified in the COMPLEX modelling.

By means of the CEA extension points for analysis, the CEA tool implements a set of analysis engines that focus their model checks on specific aspects of the system:

- Compliance of the UML/MARTE model with the COMPLEX component model
- Enforcement of the separation of concerns concept by verifying that all views define the right design entities
- Verification of the model consistency
- Identification of potential issues while model-to-text transformation

Prior to the generation of the different output artefacts produced by the application in Fig. 19, designers should execute each of the existing analysis engines. In case of any error triggered by the tool, the framework reports errors and disable the code generation.

4.2. SCoPE+

SCoPE+ is the simulation infrastructure supporting the production of the performance model from the text-based representation of the system, the environment and the design spaced produced by CEA application. It supports the component-oriented structure of the COMPLEX UML/MARTE model. In that context, the generation of the executable model combines four different inputs: the source code of each component, the integration wrappers (called CFACM wrappers) that connect the functional components and the simulation infrastructure, a set of XML files describing the HW platform, the allocation of functional components to HW resources and the design space; the compilation scripts; and finally, the SystemC code modelling the environment.

With those inputs, the executable model can be generated, enabling the simulation of the system and obtaining different metrics such as time delays, power consumption or CPU utilization that can help the designer to evaluate the different configurations selecting the optimal ones.

A new simulation technology called SCoPE+ has been developed to give support to the features covered by the modelling methodology presented along the paper and for DSE. This simulation technology extends the previous SCoPE infrastructure ([14–20]) and provides a new solution oriented to design space exploration.

A set of features made SCoPE suitable for being selected as a base for the simulation performance estimation framework required, which led to SCoPE+. SCoPE relies on native simulation, which enables fast performance assessment of the system. A set of specific plugins [16,17] made SCoPE a more suitable tool for DSE. In previous work, we referred to this toolset as M3SCoPE, to distinguish it from the core of the tool. M3SCoPE enabled the generation of a highly configurable model supporting an XML interface compatible with several exploration tools, e.g. MExplorer [21] or ESTECO [80]. Through those interfaces, the exploration tool can first configure a new solution and later launch the simulation, to finally receive the estimated metrics to evaluate the solution.

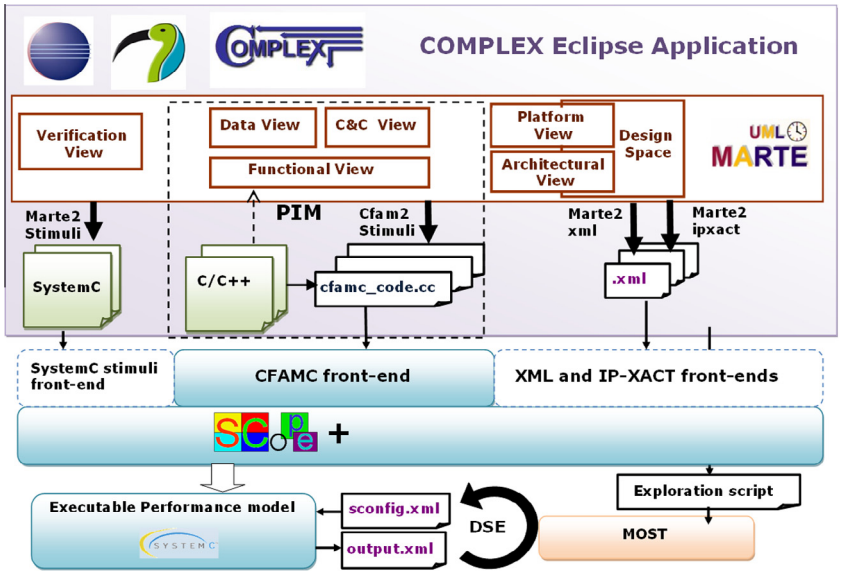

Fig. 19. COMPLEX Eclipse Application in the COMPLEX UML/MARTE flow.

M3SCoPE supports *run-time generation* of the model. That is, the compiled model is a platform template, which requires a set of parameters to be passed for defining the specific solution. This way, the configurable and executable performance model is compiled only once and each new model generation and simulation does not require a re-compilation. This is exploited in the UML/MARTE COMPLEX flow to fit the general schema of Fig. 1. In the proposed flow, the UML/MARTE model becomes an executable performance model after the application of a code generation step, and then a compilation step (as shown in Fig. 19). These two steps are applied only once, while the generated configurable and executable performance model can be simulated multiple times. Therefore, code generation and compilation overhead is minimized in the exploration loop. M3SCoPE also added a set of advanced features for accuracy. Specifically, instruction cache effects, compiler optimizations, the effects of the operative system, and processor allocation were considered. Precise low-level modelling was also enabled, since it was also possible the modelling of drivers, interrupts, and direct access to HW registers through pointers.

SCoPE+ has been developed as an evolution of the SCoPE core, and M3SCoPE. SCoPE+ supports an additional set of features required by the COMPLEX UML/MARTE flow and which configures an advanced performance estimation toolset. Specifically, SCoPE has provided support for:

- A component-based and implementation agnostic modelling API (CFACM API). The CFACM API enables to reflect at a code-level the component-based COMPLEX UML/MARTE model. Being implementation agnostic enables the assessment before the code development phase.
- Multi-level modelling, since CFAMCM implementation agnostic parts can be merged with RTOS dependent (legacy) code parts in the same model.
- Modeling of impact on performance of HW/SW communication dependent on the architectural mapping.
- Data-cache modelling.
- Architectural mapping is supported as configurable parameter.
- Integration of SystemC environment models (compatible with the ones automatically produced from the COMPLEX UML/MARTE model).
- Integration of custom HW with performance estimation models.

We address the latter two aspects in the following sections. COMPLEX project reports provide more information on the CFACM API. We address the remaining aspects in the following, but data cache modelling which is out of the scope of this paper.

4.2.1. Integration of SW and HW performance estimation methodologies

In order to fulfil complete design exploration, both the parameters of the HW platform and the resource allocation have to be explored. While considering different values on the HW platform parameters is simple, since it only requires numeric modifications in the XML files, the exploration of different allocations of the functional components presents a greater challenge. Both the performance results and the communication mechanisms must be modified to evaluate these different allocations.

To solve the generation of different metrics depending on the resource allocation, a solution based on the use of dynamic libraries is applied. Using the compilation scripts generated from the UML model, multiple libraries are automatically built, combining the functional source codes and the integration wrappers for each component. More exactly, one library is generated per functional component and possible allocation of this component to a different resource to be explored. During the generation of these libraries, the original code is annotated with additional information that enables performing a complete modelling of the code, including time modelling, cache access modelling and power modelling. These annotations are performed both for SW [18] and HW [19] possible mappings of the functional components, applying different techniques to generate the libraries for SW and HW performance modelling. As a result of these annotations, metrics about time and power consumption are obtained for each block of source code executed. This metrics are applied to the simulation to obtain complete results for the operation of the entire system.

Once all the required libraries have been generated, the simulation can start. The simulation engine starts reading the XML files describing the HW platform and component allocations. As a result, the simulation engine generates the virtual platform and loads the adequate libraries depending on the selected allocation of the functional components.

Additionally, the simulation infrastructure has been developed to support the modelling of different allocations, including mapping to HW and SW resources. This support is divided in two parts: one providing the services required by the functional code to execute the functionality, and a second part managing the access to the HW computing resources and therefore the performance assessments.

For the former part, SCoPE+ has incorporated a new global OS modelling infrastructure. This infrastructure enables a consistent execution of the original, implementation agnostic source code regardless from the allocation or design solution explored. Internally, the global OS enables shared memory spaces for keeping an efficient and fast simulation of the model.

For the second part, similar executable models for HW and SW resources have been created in the simulation engine. These resource models provide a common interface to control the execution of the functional codes, which are simulated in a different way depending on the nature of the computing resource. For instance, while in SW processors the resource model only allows the execution of one task at a time, HW resources supports the execution of multiple tasks in parallel. Therefore, the allocation has an impact of the execution semantics and as a consequence on the time performance.

4.2.2. Cosimulation of the system with the SystemC environment

SCoPE operates on top of a SystemC engine, which provides services for time modelling, and SystemC threads for HW platform modelling. However, application task modelling directly relies on native thread support. In SCoPE, such "native" application model synchronizes with the SystemC platform model for performance modelling. Therefore, direct integration of a SCoPE system model with the high-level and functional SystemC environments generated from the UML/MARTE model was not possible.

SCoPE+ has solve that gap by providing a set of *system-environment wrappers* which translate the functional calls requiring and providing services from/to the environment into the SystemC channel accessing the environment model. These system-environment wrappers are also automatically generated from the UML/MARTE model at the same time that the component wrappers corresponding to the system model are. System-environment wrappers perform three different operations to provide correct system-environment communication:

- Multiplexing/Demultiplexing of the information transferred as function arguments into SystemC channel buffers and viceversa.
- Enforcing the execution order specified by UML/MARTE model.
- Performing the different operations required to model the impact on performance of the high-level system-environment communication as data transfers in the I/O peripherals associated to the high-level I/O, and on the corresponding buses.

Notice that, in the UML/MARTE model, this information is present in the Architectural view, as connections between ''high-level ports'' and I/O HW component instances (Fig. 6).

The object code of the SystemC environment generated, which includes the ancillary code generated by CEA, and the functionality contributed by the user is packaged as a dynamic libraries, in a similar was as it was done for the application components (for each mapping alternative). This way, the regeneration of the executable environment models (which also involves code generation and compilation) is avoided and performed only once. Moreover, the the simulation infrastructure can perform simulations automatically with different environments, and the set of environments to sweep in the exploration be easily configured from the exploration tool. However, notice that although the environment will not considered as a parameter to be optimized in general, this framework would let study, for instance, which environment conditions are optimize certain metric.

4.3. Exploration tool and interface with the performance model

4.3.1. Most

The Multi-Objective System Tune (MOST) tool enables discrete optimization specifically designed for enabling design space exploration of hardware/software architectures. MOST is the tool integrated within the COMPLEX UML/MARTE framework for the automatic design space exploration phase.

MOST drives the designer towards near-optimal solutions to the architectural exploration problem, by covering DoE, RSM and multi-objective optimization heuristics introduced in Section 2.2. The final output of the framework is a Pareto set of configurations within the design evaluation space of the given architecture and a large set analysis on the effects of design space parameters onto the objective functions.

MOST integrates a rich set of advanced and automated exploration strategies [73–75]. MOST provides a command line interactive interface which enables to construct the exploration strategies. Moreover, the strategy can be captured as a command script which is interpreted by MOST, eliminating the need of user intervention and enabling the batch execution of complex strategies. This has been exploited in the COMPLEX UML/MARTE flow, to enable an interface in the CEA tool fast and convenient for novel users. Specifically, CEA produces a default MOST script performing exhaustive exploration and which relies on a set of performance metrics which can be selected by the user. The user can select metrics among a default set of default model independent metrics (e.g., total power consumption), and from the estimations parameters captured in the model (Section 3.3.4). For each selected metric, a basic minimization or maximization goal function is be constructed.

A distinctive aspect of MOST is that it enables to build a global search strategy considering all the parameterizations exposed by the COMPLEX UML/MARTE model and explained in Section 3.3.2. CEA also enables a custom and thus direct edition of the MOST script, which enables a flexible usage for users which want to describe a customized search strategy.

4.3.2. Interface with the performance model

In the COMPLEX UML/MARTE framework, to easily interface the exploration tool with the automatically generated system model, a modular approach is followed by decomposing the problem in two main blocks: (i) the exploration tool which is mainly problem independent (ii) the performance model, that is completely problem dependent and exposes to the exploration tool the knobs that can be used for the optimization.The specific back-ends used in the implementation presented are MOST, as an exploration tool, that is an ''optimizer'' that tries to find the solution of a specific problem specified in the MOST script (produced from CEA and the UML/MARTE model) by triggering the SCOPE+ performance model.

The exploration is automatic since, once the script is configured and MOST is launched, MOST can autonomously select each new design point, and because MOST can transfer the configuration data to the performance estimation tool (SCOPE+), launch it, and collect the resulting performance metrics.

In the COMPLEX UML/MARTE framework, the exploration tool and the performance model have adopted and improved the standard XML interface defined in [21]. In particular, the exploration tool receives as input an XML description of the design space. Such XML description is automatically generated from the UML/MARTE description of the design space (explained in Section 3.3). With such an information, MOST explores the possible solutions by using an iterative technique. For each iteration, the executable performance model receives as input a system configuration to be evaluated. Each configuration assigns specific values for all the parameters of the model (e.g. core frequency, cache configuration). The evaluation of the configuration, that is, in this framework, the simulation of a specific system configuration for a specific scenario, produces as output the list of system metrics related to the requested configuration (e.g. power, area and delay). The list of metrics produced by the system model can be system wide and also local to a specific component, exposing values such as resource usage or task execution time.

5. Experimental results

In order to show the capabilities and advantages of the proposed methodology, a design space exploration of the EFR vocoder model introduced in the previous sections has been done. Indeed, the model was modified to show a larger and more interesting design space. It was decided to fix the mapping of some components to avoid the exploration of equivalent design points and to explore working frequencies independently, per processor code. The reason for this was that, in one initial trial, it was observed that the computational load of the components was not quite balanced and that the coder took easily ten times more time than the decoder to compute.

The UML/MARTE model was captured and the SCoPE+ executable and configurable model was generated. A manual adaptation of the GSM standard source ANSI-C code of the EFR vocoder was done just to enable a division of the original code into three functions (one for the coding functionality, one for the decoding functionality and one for the activity detection functionality). This code was directly linked to the executable code generated from the UML/MARTE model.

The GSM standard source code reused in the experiments relies on a C model of 16-bits finite precision data types, thus it has a modelling overhead which would not be present in a C implementation. However, it ment that on the reports the solutions did not get below 20 ms (the time constraint associated to transmission and reception times for this system in reality). The optimization of such C code, e.g. substitute 16-bits finite precision data types by *float* types, would modify EFR vocoder functionality at a bit-level, and would oblige to develop a more elaborated functional automatic validation of each design solution (e.g. comparing the signal to noise ratio of the simulated output with regard to the one of the reference output). However that costly effort is uninteresting to show the validity of the proposed framework. Thus, instead of that, by reusing the ANSI-C reference code, we implemented a simpler automatic validation of each design solution through a bit-level comparison of the simulated outputs against a standard reference output.

The exploration space analyzed has covered different architectural mappings considering the execution of the 4 components of the vocoder (as the ones presented in Fig. 5) in up to three ARM processors (as the ones reflected in the architecture of Fig. 6). The mapping of the Controller and Coder components was restricted to the first ARM core, to explore the effects on performance of mapping the other two components to any of the other three processors of the platform. To fix Controller and Coder components to the first core no specific DSE constraint was required, but it was enough with two <<allocate>> associations from the Controller and Coder components to the first ARM code. The mapping alternatives were reflected through an <<assign>> comment similar to the one shown in Fig. 7, but removing the coder component from the "from" attribute. Notice that the <<assign>> construct overrode the fix <<allocate>> associations from the decoder and from the voice activity detection components. Additionally, as the first core has a potential higher load, 5 possible frequencies and 5 cache sizes have been explored for this core. For the other cores, fixed cache sizes have been defined and three configurable frequencies explored. Therefore, the number of solutions of the design space was:

$$\begin{aligned} N_{\text{SOLUTIONS EXPLORED}} &= 5(\text{freq.ARM1}) \cdot 5(\text{cache sizes ARM1}) \\ &\quad \cdot 3(\text{freq.ARM2}) \cdot 3(\text{freq.ARM3}) \cdot 6(\text{mappings}) \\ &= 1350\text{solutions}. \end{aligned}$$

Moreover, each of these solutions was checked for two different environments, a full duplex and a closed loop configuration. Therefore, the virtual executable and configurable executable generated with the COMPLEX tooling enabled 2700 different configurations. A full search (that is, an exhaustive search) was performed, which involved 2700 SCoPE+ simulations. The full search was totally automatic, and required no change on the model or user intervention. The automatic check validated the functionality of all design solutions.

The time required for the full search in an Intel(R) Core(TM) 2 Duo CPU E6550 @ 2.33 GHz, CPU frequency 2.000 MHz, 4 MB cache, and 2 GB of RAM was measured with the Linux *time* command and reflected in Table 1. One second of physical time was simulated for every iteration. The time required for each simulation was variable from 1.6 to 4.5 s depending on the test bench, frequency and number of misses modeled during the simulation.

In order to estimate the average time spent in preparing the next configuration and launching it, a specific experiment was done in two steps. First, a specific design solution was simulated six times independently (without MOST). The result is reported as t_{dp} in the bottom raw of Table 2. Then, a DSE exploration where MOST launched 25 simulations was done. In order to ensure that each simulation of the iteration took the same host time as the simulation measured in the first step, the XML configuration files were manipulated to simulate always the solution measured in the first step and to ignore the configuration generated by MOST. However, the transfer of such data from MOST to SCoPE + and the launch of the simulation was measured. The time for simulating the 25 evaluations is reported as t_{DSE} in the first raw of Table 2.

The average time spent by the COMPLEX framework on preparing and launching (t_{pl}) the simulation of a design solution was calculated then through the formula $t_{pl} = (t_{DSE} - 25^{*} t_{dp})/25$. The result is 0.4 ms, a negligible time (below the tick timer precision) per iteration.

A very conservative guess of the time saved in the design exploration through the proposed framework with regard an approach with performance model re-compilation in the DSE loop can be given. If the user managed to reconfigure the performance model, e.g., manually changing a parameter file, and launch the simulation in only 2s on average (what can be considered a fast user performance) then the full search would have taken around 1 h and a half more time, thus making the exploration to last more than double time than with the proposed framework, engaging the engineer with such a boring and error prone task.

Moreover, if it is compared against an approach supporting a MBD front-end which requires the model editions in at least on some iteration, involving code generation and recompilation, the gain is much more apparent. If, for instance, the user managed to edit the UML model, regenerate the executable performance model and pass the new arguments in 10 s on average, the full search would take 7.5 h more. Therefore, a conservative guess let state that the proposed framework easily reaches DSE speed ups of 1 order of magnitude.

In addition, the proposed framework benefits from the detailed and user friendly reports generated by MOST, which allows fast and detailed analysis. As a demonstrative example, we present some of the reports provided to the designer for the EFR vocoder by using the proposed DSE framework. Particularly, maximum and mean times required by receiver and transmitter functions have been compared, relating them with instructions executed and power consumed for the different configurations defined in the design space.

Fig. 20a and b. show the result of the exploration within two objective spaces: Power-TX_mean_time (power consumption-mean transmission time) and Power-RX_mean_time (power consumption-mean reception time) respectively. Looking at the figure, it is possible to see a clear trade-off between power consumption and transmission (or reception) times. Additionally, in both cases the solutions result to be clustered into two main sets: one on the top left corner characterized by a lower power consumption and higher latency, and one on the bottom right corner characterized by a higher power consumption and lower latency. We will see in the following analysis that this behavior is mainly due to the ARM1 frequency.

To better clarify the source of the clustered behavior reported before, Fig. 21 shows the box plot analysis on the POWER objective for the frequencies of the three ARM cores. These reports make easy to see a not so evident fact. The trend and sensitivity on power consumption when increasing the core frequency is similar for ARM 2 (micro1) and ARM 3 (micro2) cores (Fig. 21b and c). However, when it comes to the ARM 1 core (Fig. 21.a), the power increment while passing from the lowest frequency (100 MHz) to the next frequency (300 MHz) is significant, while further increments of frequency produce a lighter increment. This is consistent with the larger amount of functionality mapped to ARM 1 (micro0).

The same step-like behavior in the ARM1 frequency space can be seen for both TX_max/mean_time and RX_max/mean_time. Fig. 22 reports the data only for the max metric. The results on

Table 1
Time for the full search.

Time	Real	usr	sys	usr+sys
Full search	101 m 42.909s	84 m 0.137s	11 m 19.332s	95 m 19.469s

Table 2
Experimental data for calculating the time required by the framework for preparing a configuration and launching a simulation in a full search mode.

Time for	# Meas.	usr + sys (average)	usr + sys (std.dev.)
t_{DSE}: 25 eval. of the same dp	3	40,99s	0.261s
t_{dp}: Simulation of a design point	6	1,64s	0,010s

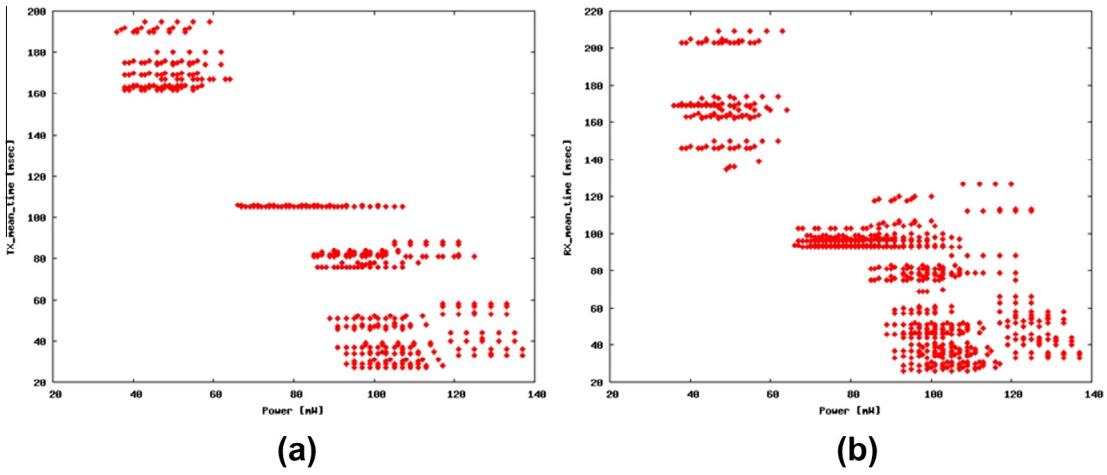

Fig. 20. Exploration results in the (a) Power-TX_mean_time and (b) Power-RX_mean_time spaces.

the mean metrics are the similar (but down shifted). Thus, since frequency reduction highly affects computing times, the combination of that figures with maximum and mean delays can enable the designer to identify the adequate design solutions.

Looking into the allocation parameters, Fig. 23 shows how the allocation of the decoder component either to the "micro1" or to "micro2" does not change the average behavior but the distribution of the RX_max_time (the same effect was observed for the RX_mean_time) across the different configurations. In fact, mapping the decoder into the "micro1" seems to present a more robust behavior than mapping it on "micro2", since most of the values for maximum time are lower than the mean and the maximum value is smaller than the maximum for the allocation to "micro2".

Additionally, information about impact of cache sizes can be obtained. For example, in Fig. 24, the impact of cache sizes on the maximum transmission time can be shown. Analyzing the figure it is possible to see that caches smaller than 2 KB produce a dramatic increment on execution times, while caches larger than 4 KB have almost no impact on the result.

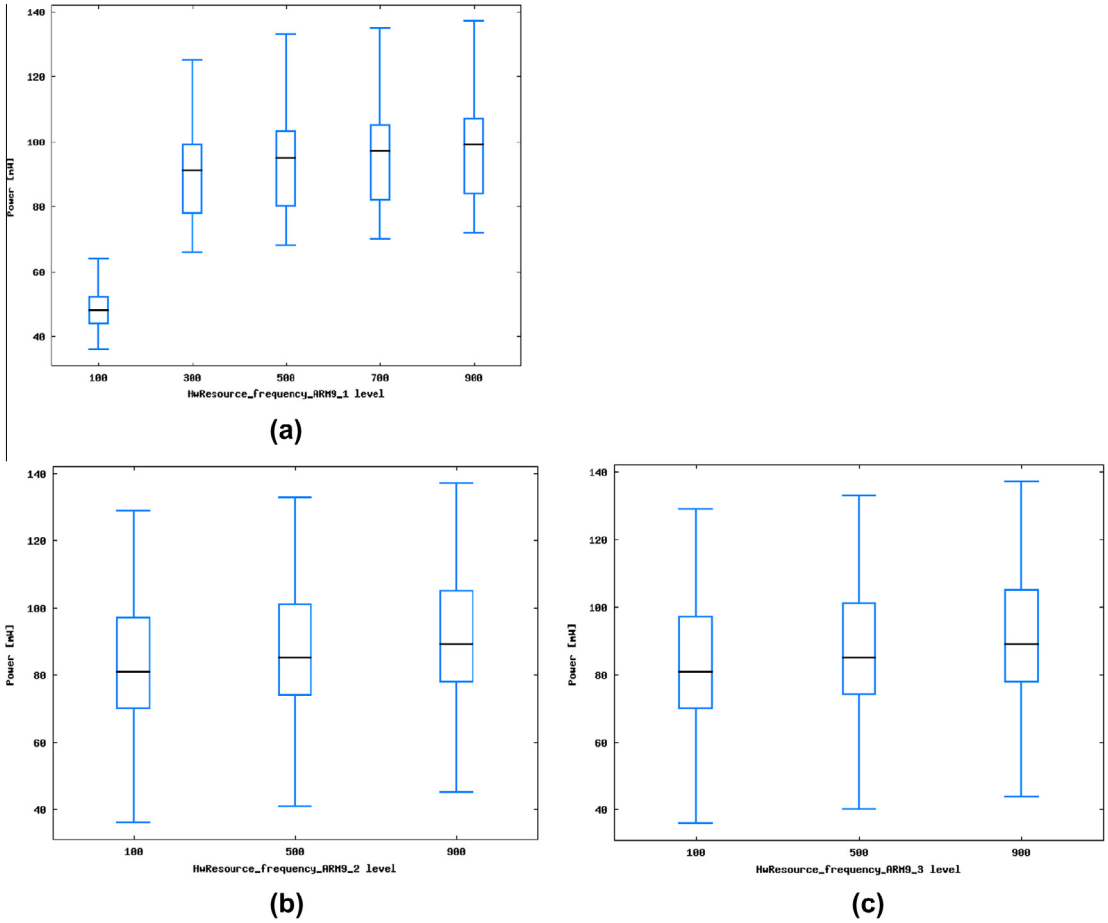

Fig. 21. Box plot analysis for the Power objective on the (a) ARM1 frequency, (b) ARM2 frequency and (c) ARM3 frequency parameters.

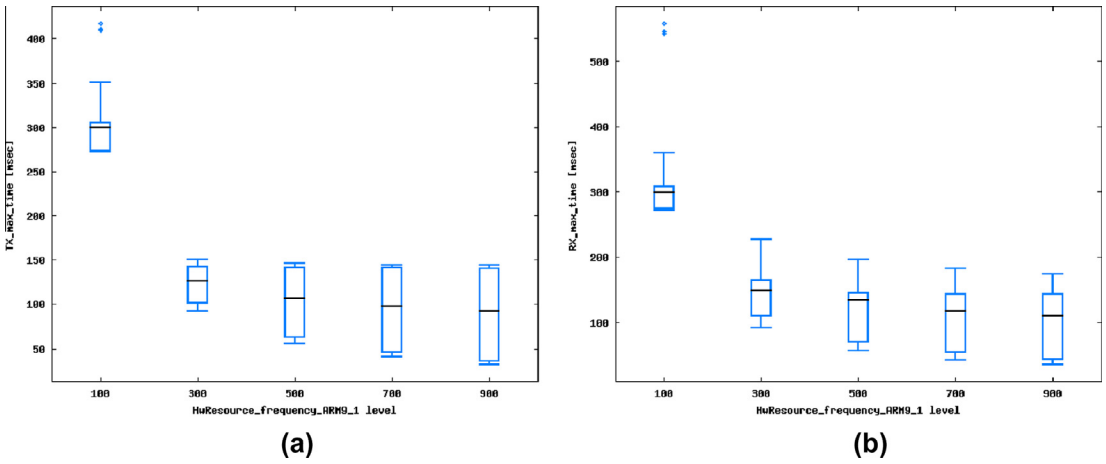

Fig. 22. Box plot analysis for the (a) TX_max_time and (b) RX_max_time objectives on the ARM1 frequency parameter.

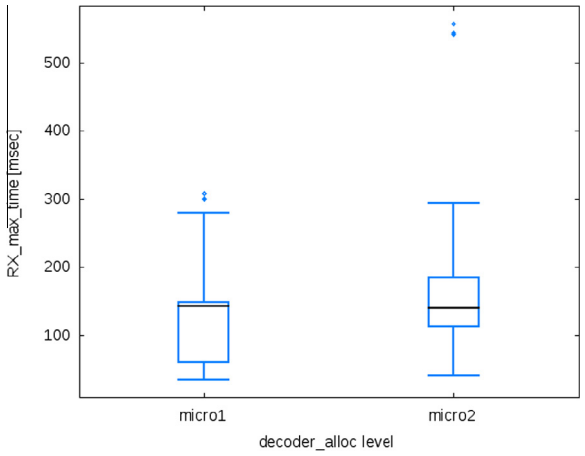

Fig. 23. Box plot analysis for the RX_max_time objective on the decoder_alloc parameter.

6. Conclusions

This paper has presented the COMPLEX UML/MARTE based methodology for Design Space Exploration of Embedded systems, a key activity for the design of complex embedded systems. The methodology integrates key and mature technologies in modelling, performance estimation and exploration.

A UML/MARTE modelling methodology, which integrates advanced features specifically suited for DSE has been presented. The methodology consolidates key aspects for industrial modelling-based design, i.e. component-based design, separation of concerns and concurrent development, and at the same time it incorporates features for enabling an efficient DSE flow. In order to efficiently conjugate model-based design with DSE, some capabilities supported by the methodology, such as modelling of a design space including architectural mappings, explicit design space description, and the generic approach presented in Fig. 1, which avoids the regeneration of the executable performance model are crucial to avoid exploration speed degradation of orders of magnitude, even for small examples. The work has also show that it is possible to integrate the environment model in the flow, from the model-base level down to the automatically generated performance model. The possibility to consider and explore easily several environment scenarios facilitates the customization of the design to a specific set of use cases.

The COMPLEX UML/MARTE framework has practically demonstrated the generic DSE methodology by relying on the novel CEA framework, and on advanced tools for performance estimation (SCoPE+) and exploration (MOST), completing a fully automated DSE loop which produces an extensive and user-friendly information which facilitates user analysis and decisions. Moreover, the framework proposed is modular and extensible, in order to enable the reuse of the UML/MARTE model on top of potentially new configurations relying on other back-end performance estimation and exploration tools.

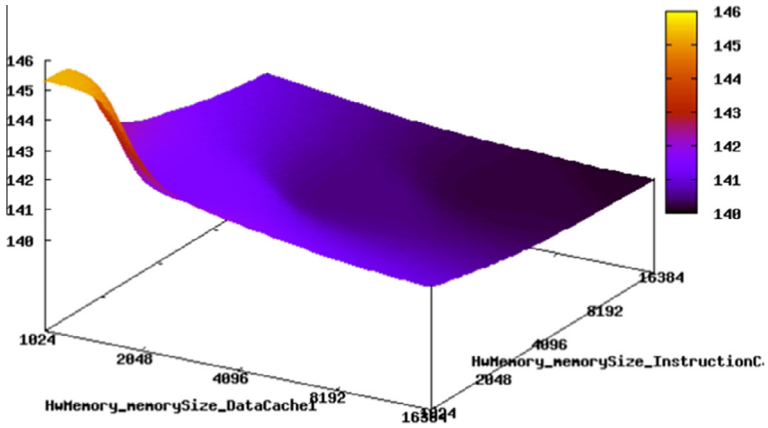

Fig. 24. Surface plot for 'TX_max_time' with respect to DataCache1 and_InstructionCache1 parameters.

Future work will aim a cooperative DSE framework which connects the proposed framework with an analytical DSE framework to target efficient designs of real-time and mixed-criticality systems. The support of the domain-oriented co-operative searching proposed by NASA [67] is also interesting. It would likely require changes on the exploration part of the proposed framework. The integration of the proposed DSE framework with a UML/MARTE-based synthesis framework under development in the context of the FP7 PHARAON project [81] will also provide a synergistic UML/MARTE-based design framework.

References

[1] M. Holzer, Design space exploration for the development of embedded systems, Thesis Dissertation in the TU Vienna, Vienna, Austria, April 2008.

[2] H. Chang, L. Cooke, M. Hunt, G. Martin, A.J. McNelly, L. Todd, Surviving the SOC Revolution: A Guide to Platform-Based Design, Kluwer Academic Publishers, 1999.

[3] OMG, Unified Modelling Language™. Available from: <http://www.omg.org/spec/UML/>.

[4] OMG, UML Profile for MARTE: Modelling and Analysis of Real-Time Embedded Systems, Version 1.1, Dec., 2012. Available from: <http://www.omgmarte.org>.

[5] T. Kempf, G. Ascheid, R. Leupers, Multiprocessor Systems on Chip, Design Space Exploration, Springer, 2011.

[6] The COMPLEX project (247999), Codesign and power management in platform-based design space exploration, Last visited, 2013. Available from: <http://complex.offis.de>.

[7] K. Gruettner, et al., COMPLEX – COdesign and power Management in PLatform-based design space EXploration, in: Proc. of the 15th Euromicro Conference on Digital System Design (DSD'2012), Cesme-Izmir, Turkey, 2012.

[9] F. Herrera, P. Peñil, E. Villar, F. Ferrero, R.Valencia, An embedded system Modelling methodology for design space exploration, in: III Jornadas de Computacion Empotrada, I Jornadas SARTECO, 2012.

[10] F. Herrera, P. Peñil, H. Posadas, E. Villar, A model-driven methodology for the development of SystemC executable environments, in: Proceedings of Forum of Design Languages, FDL'2012, 2012.

[11] F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia, A MDD Methodology for specification of embedded systems and automatic generation of fast configurable and executable performance models, in: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System, Synthesis, CODES+ISSS'12, 2012, pp. 529–538.

[12] F. Herrera, E. Villar, A Framework for the generation from UML/MARTE Models of IP-XACT HW platform descriptions for multi-level performance estimation, in: Proceedings of the Forum of Design and Specification Languages 2011 (FDL'2011), 2011.

[13] F. Herrera, P. Peñil, E. Villar, D. Calvo, Enhanced IP-XACT platform descriptions for automatic generation from UML/MARTE of fast performance models for DSE, in: 15th Euromicro Conference on Digital System Design, DSD'2012, 2012.

[14] SCoPE website, 2012. Available from: <www.teisa.unican.es/scope>.

[15] P. Botella, P. Sánchez, H. Posadas, Automatic Generation of SystemC SMP Models for HW/SW Co-Simulation, in: Proc. of XXV Conf. on Design of Circuits and Integrated Systems, DCIS'10, 2010.

[16] H. Posadas, G. de Miguel, E. Villar, Automatic generation of modifiable platform models in SystemC for Automatic System Architecture Exploration, in: Proc. of Design of Circuits and Integrated Systems, DCIS' 2009, Zaragoza, Spain, 2009–11.

[17] H. Posadas, E. Villar, Automatic Communication Modelling for early exploration of HW/SW allocation based on native co-simulation, in: Proc. of XXVI Conf. on Design of Circuits and Integrated Systems, DCIS'11, 2011.

[18] J. Castillo, H. Posadas, E. Villar, M. Martínez, fast instruction cache modeling for approximate timed HW/SW co-simulation, in: 20th Great Lakes Symposium on VLSI (GLSVLSI'10), Providence, USA, 2010–05.

[19] P. González, P. Sánchez, J. González, Hardware performance estimation by dynamic scheduling, in: Proc. of the Forum on specification and Design Languages 2011, Oldenburg, Germany, 2011.

[20] H. Posadas, S. Real, E. Villar, M3-SCoPE: Performance Modelling of Multi-Processor Embedded Systems for fast design space exploration, in: C. Silvano, W. Fornaciari, E. Villar (Eds.), Multi-objective Design Space Exploration of Multiprocessor SoC Architectures: the MULTICUBE Approach, Springer, 2011.

[21] Multicube Explorer site. Available from: <http://home.dei.polimi.it/zaccaria/multicube_explorer_v1/Home.html>.

[22] E. Piel, R.B. Atitallah, P. Marquet, S.Meftali, S. Niar, A. Etien, J.L.Dekeyser, P. Boulet, Gaspard2: from MARTE to SystemC Simulation, in: Proceedings of the Design, Automation and Test in Europe (DATE'08), Munich, Germany, 2008.

[23] J.L. Dekeyser, A. Gamatié, A. Etien, R.B.Atitallah, P. Boulet, Using the UML Profile for MARTE to MPSoC Co-Design, in: First International Conference on Embedded Systems & Critical Applications (ICESCA'08), Tunis, Tunisia, 2008.

[24] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, J.P. Diguet, A Code-design approach for Embedded System Modelling and code generation with UML and MARTE, in: Proceedings of Design Automation and Test in Europe, (DATE'09), Dresden, 2009.

[25] P. Peñil, H. Posadas, A. Nicolas, E. Villar, Automatic synthesis from UML/MARTE models using channel semantics, in: Proc. of 5th International Workshop on Model Based Architecting and Construction of Embedded System ACES-MB workshop, Models, Innsbruck, Austria, 2012.

[26] T. Robert, V. Perrier, COFLUENT Methodology for UML: UML SysML MARTE Flow for CoFluent Studio, White paper, 2012. Available from: <http://www.cofluentdesign.com/index.php/solutions/uml-sysml-marte>.

[27] A.W. Liehr, H.S. Rolfs, K.J. Buchenrieder, U. Nageldinger, Generating MARTE allocation models from activity threads, in: Languages for Embedded Systems and their Applications, Lecture Notes in Electrical Engineering, vol. 39, 2009, pp. 43–56 (Part I).

[28] M. Mura, L.G. Murillo, M. Prevostini, Model-based design space exploration for RTES with SysML and MARTE, in: Proceedings of the Forum on Specification and Design Languages 2008, FDL'2008. Stuttgart, Germany, 2008.

[29] S. Kent, Model driven engineering, in: Proceedings of the Third International Conference on Integrated Formal Methods, ser. IFM '02, London, UK, Springer–Verlag, 2002, pp. 286–298.

[30] K. Yamashita, Possibility of ESL: a software centric system design for multicore SoC in the upstream phase, in: Proceedings of the Asian Pacific Design Automation Conference (ASP-DAC), Taipei, Taiwan, 2010, pp. 805–808.

[31] C. Szyperski, Component Software: Beyond Object-Oriented Programming, second ed., Addison-Wesley Professional, 2002.

[32] Panunzio M. Vardanega, Tullio, On Component-based development and high-integrity real-time systems, in: Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, ERTCŚ09, Beijing, China, 2009.

[33] M. Panunzio, Vardanega Tullio, A component model for on-board software applications, in: Proc. of the 36th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA), Lille, France, 2010.

[34] A. Bakshi, V.K. Prasanna, A. Ledeczi, MILAN: a Model Based integrated simulation framework for design of embedded systems, in: Proceeding LCTES '01 Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems, 2001, pp 82–93.

[35] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, P. Volgyesi, The generic modeling environment, in: Workshop on Intelligent Signal Processing, vol. 17, Budapest, Hungary, 2001.

[36] S. Neema, J. Sztipanovits, G. Karsai, K. Butts, Constraint-based Design-Space exploration and model synthesis, in: R. Alur, I. Lee (Eds.), Embedded Software, Volume 2855 of Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, Berlin/Heidelberg, 2003, pp. 290–305 (Chapter 19).

[37] T. Kangas et al., UML-based multiprocessor SoC design framework, Journal ACM Transactions on Embedded Computing Systems (TECS) 5 (2) (May 2006) 281–320.

[38] B. Schatz, F. Holzl, T. Lundkvist, Design-space exploration through constraintbased model-transformation, in: Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on, IEEE, 2010, pp. 173–182 (ISBN 978-1-4244-6537-8).

[39] F.A. Nascimento, M.F.S. Oliveira, F.R. Wagner, A Model-driven engineering framework for embedded systems design, in: Innovations in Systems and Software Engineering 8(1), 2012.

[40] T. Saxena, G. Karsai, A meta-framework for design space exploration, in: Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on. IEEE, 2011, pp. 71–80. ISBN 978-1-4577-0065-1.

[41] T. Saxena, G. Karsai, Mde-based approach for generalizing design space exploration, Model Driven Engineering Languages and Systems Lecture Notes in Computer Science 6394 (2010) 46–60.

[42] MiniZinc website, Last visited 2013. Available from: <http://www.minizinc.org/>.

[43] J. Ou, V.K. Prasanna, MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors, International Parallel and Distributed Processing, Symposium (IPDPS'05).

[44] S. Künzli, A. Hamann, R. Ernst, L. Thiele, Combined approach to system level performance analysis of embedded systems, in: Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on, vol., no., pp. 63,68, Sept. 30 2007-Oct. 3 2007.

[45] S. Perathoner, K. Lampka, L. Thiele, Composing heterogeneous components for system-wide performance analysis, in: Proceedings of the Design Automation and Test (DATÉ11) Conference, Grenoble, France, 2011.

[46] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, System level performance analysis – the SymTA/s approach, Computers and Digital Techniques, IEE Proceedings 152 (2) (2005) 148–166.

[47] J. Kreku, K. Tiensyrjä, System exploration, in: D. Soudris, A. Jantsch (Eds.), Scalable Multi-core Architectures: Design Methodologies and Tools, Springer, 2012.

[48] J. Kreku, K. Tiensyrjä, G. Vanmeerbeeck, Automatic workload generation for system-level exploration based on modified GCC compiler, in: Proceedings of the Design Automation and Test in Europe (DATÉ10). Dresden, Germany, 2010.

[49] S. Jaddoe, M. Thompson, A.D. Pimentel, Signature-based calibration of analytical performance models for system-level design space exploration, in: P. Stenström (Ed.), Transactions on High-Performance Embedded Architectures and Compilers IV, Springer–Verlag, 2011.

[50] L. Benini, et al., MPARM: Exploring the Multi-Processor SoC Design Space with SystemC, Journal of Signal Processing Systems, 2005.

[51] F. Bellard, QEMU, a Fast and Portable Dynamic Translator, USENIX 2005 Annual Technical Conference, 2005.

[52] M. Streubühr, R. Rosales, R. Hasholzner, C. Haubelt, J. Teich, ESL power and performance estimation for heterogeneous MPSoCs using SystemC, in: Proc. of Forum of specification and Design Languages (FDĹ11), Oldenburg, Germany, 2011.
[53] A.D. Pimentel, The Artemis workbench for system-level performance evaluation of embedded systems, International Journal of Embedded Systems, 3(3), 2008.
[54] R. Plyaskin, A. Masrur, M. Geier, S. Chakraborty, A. Herkersdorf, High-level timing analysis of concurrent applications on MPSoC platforms using memory-aware trace-driven simulations, in: Proceedings of International Conference on VLSI and System-on-Chip, IEEE, 2010.
[55] R. Leupers, G. Martin, R. Plyaskin, A. Herkersdorf, F. Schirrmeister, T. Kogel, M. Vaupel, Virtual Platforms: Breaking new grounds, in: Proceedings of the Design Automation and Test in Europe (DATÉ12), Grenoble, France, 2012.
[56] J. Schnerr, O. Bringmann, A. Viehl, W. Rosenstiel, High-performance timing simulation of embedded software, in: Proceedings of the Design Automation Conference (DAĆ08), Anaheim, CA, USA, 2008.
[57] H. Shen, M.-M. Hamayun, F. Pétrot, Native Simulation of MPSoC using Hardware-Assisted Virtualization, in: IEEE Trans. on Computer-Aided design of Integrated Circuits and Systems, V. 31, N.7, 2012.
[58] Z. Wang, J. Henkel, HyCoS: hybrid compiled simulation of embedded software with target dependent code, in: Proc. of 8th IEEE/ACM/IFIP Int. Con. on HW/SW Codesign and System, Synthesis CODES-ISSŚ12, 2012, pp. 133–142.
[59] R. Wilhelm, et al., The worst-case execution-time problem overview of methods and survey of tools, in: ACM Transactions on Embedded, Computer Systems, vol. 7(3), 2008, pp. 36:1–36:53.
[60] S. Stuijk, Predictable mapping of streaming applications on multiprocessors, Phd thesis, 2007.
[61] A. Bonfietti, M. Lombardi, M. Milano, L. Benini, Throughput constraint for synchronous data flow graphs, in: Proc. of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOŔ09), Berlin, Germany, 2009, pp. 26–40.
[62] A. Kumar, T. Srikanthan, Accelerating throughput-aware run-time mapping for heterogeneous MPSoCs, in: ACM Transactions on Design Automation of Electronic Systems (TODAES), 2012.
[63] S. Stuijk, M. Geilen, T. Basten, A predictable multiprocessor design flow for streaming applications with dynamic behaviour, in: Proc. of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD '10, Lille, France, 2010, pp. 548–555.
[64] A. Pimentel, C. Erbas, S. Polstra, A systematic approach to exploring embedded system architectures at multiple abstraction levels, Computers, IEEE Transactions on Computers 55 (2) (2006) 99–112.
[65] Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, E. Deprettere, Daedalus: toward composable multimedia MP-SoC design, in: DAC '08: Proceedings of the 45th annual Design Automation Conference, ACM, New York, NY, USA, 2008, pp. 574–579.
[66] C. Silvano, et al., Multicube: Multi-objective design space exploration of multi-core architectures, in: VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on, 2010, pp. 488–493.
[67] Z.J. Jia, A.D. Pimentel, M. Thompson, T. Bautista, A. Nuñez-, NASA: A generic infrastructure for system-level MP-SoC design space exploration, in: Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on, vol., no., pp. 41,50, 28–29, 2010.
[68] D. Sheldon, F. Vahid, S. Lonardi, Soft-core processor customization using the design of experiments paradigm, in: Proceedings of Design Automation and Test in Europe, DATE, 2007, Nice, France, 2007, pp. 821–826.
[69] Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, Discrete Particle Swarm Optimization for Multi-objective Design Space Exploration, in: Euromicro Proceedings of DSD'08 – Conference on Digital System Design, Parma, Italy, 2008, pp. 641–644.
[70] K.I. Smith, R.M. Everson, J.E. Fieldsend, C. Murphy, R. Misra, Dominance-based multiobjective simulated annealing, IEEE Transactions on Evolutionary Computation 12 (3) (Jun. 2008) 323–342.

Further reading

[8] E. Alana, F. Ferrero, A.I. Rodriguez, R. Valencia, E. Conquet, J. Puente, J. Zamorano, F. Herrera, R. Varona, Component-based technologies for HW/SW co-design, in: Embedded Real Time Software and Systems- ERTS2, Toulouse, France, 2012.

[71] K. Deb, S. Agrawal, A. Pratab, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II in Evolutionary Computation, IEEE Transactions on 6(2) (2000) 849–858.
[72] Qingfu Zhang; Hui Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, Evolutionary Computation, IEEE Transactions on, 11(6) (2007) 712,731.
[73] Giovanni. Mariani, Gianluca. Palermo, Cristina. Silvano, Vittorio. Zaccaria, OSCAR: an Optimization Methodology Exploiting Spatial Correlation in Multi-core Design Spaces, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 21 (5) (May 2012) 740–753.
[74] Gianluca. Palermo, Cristina. Silvano, Vittorio. Zaccaria, ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28 (12) (December 2009) 1816–1829.
[75] Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria, An efficient design space exploration methodology for multiprocessor SoC architectures based on response surface methods, in: Proceedings of IEEE IC-SAMOS'08 – International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation, Samos, Greece, 2008, pp. 150–157.
[76] ETSI/EN, ETSI/EN 301 245, Digital cellular telecommunications system (phase 2); enhanced full rate (EFR) speech transcoding, 1998a. Available from: <http://www.etsi.com>.
[77] Eclipse Modelling Framework. Available from: <http://www.eclipse.org/modeling/emf/>.
[78] Acceleo. Available from: <http://www.eclipse.org/acceleo/>.
[79] Papyrus MDT. Available from: <http://www.eclipse.org/modeling/mdt/papyrus/>.
[80] ESTECO. Available from: <http://www.esteco.com/home/mode_frontier/Optimization/DOE.html>.
[81] PHARAON project. Available from: <http://cordis.europa.eu/projects/rcn/99825_en.html>.