Integration of Design Space Exploration into System-Level Specification exemplified in the Domain of Embedded System Design

Falko Guderian and Gerhard Fettweis
Vodafone Chair Mobile Communications Systems
Technische Universität Dresden, 01062 Dresden, Germany
Email:{falko.guderian, fettweis}@ifn.et.tu-dresden.de

***Abstract*—The specification of system functionality and design space exploration (DSE) are becoming very challenging in embedded systems due to an increasing number of design parameters and system specifications during the design cycle. An executable system-level specification (SLS), proposed in this paper, reduces design complexity. The SLS represents an executable DSE methodology and encapsulates system specifications. The aim is to formalize and automate design flows in order to scale to larger and more complex embedded systems. SLSs should not be limited to certain embedded system types. Hence, SLSs need to be standardized across tools, designers, and domains. A meta-methodology, as well as a meta-model are proposed to define a domain-independent SLS. Moreover, an electronic design automation environment is presented allowing to graphically create, automatically execute and validate embedded domain-specific SLSs. Finally, a design flow case study demonstrates multiple SLSs for the heterogeneous multicluster architecture.**

Keywords–embedded system design; system-level design; executable specification; design space exploration

I. INTRODUCTION

Over the past decades, embedded design kept up with an increasing technology scaling through a continuous improvement and integration of computer aided design (CAD) tools. CAD tools evolved from the layout level to the logic level and later to the behavioral synthesis. Consequently, the next step was to develop system-level design tools, including the specification and exploration of complete systems. These advancements in CAD are closely coupled with the development of electronic design automation (EDA) flows. Early EDA flows were dominated by capturing and simulating incomplete specifications. Later, logic and register-transfer synthesis allowed to describe a design only from its behavior. But, a system gap between software (SW) and hardware (HW) designs exists since SW designers still provide HW designers with incomplete specifications [1].

An executable specification, such as a SystemC model [2], closes the system gap by describing the system functionality and enabling design space exploration (DSE) of various design alternatives. Design reuse and documentation are improved through executable specifications [1]. Nevertheless, the design complexity of future embedded systems with thousands of cores increases the number of available design parameters and system specifications during the design cycle [3]. In this work, system specifications include input / output models consumed / produced in the design steps, such as executable specifications, descriptions of application, architecture, application mapping, validation result, tool configuration, etc. So far, the challenging tasks of systems specification and defining a DSE methodology are decoupled. But the combined specification and the reuse of DSE methodologies promise for a reduced design complexity and design time, respectively. Hence, we believe that a system-level specification (SLS) needs to consider both the specification of systems and DSE, as exemplary seen in Figure 1. The specified DSE methodology includes two design steps. First, dimensioning creates a HW architecture from an executable specification, the HW unit options and application description. Then, DSE results are obtained from scheduling the application on the HW architecture.

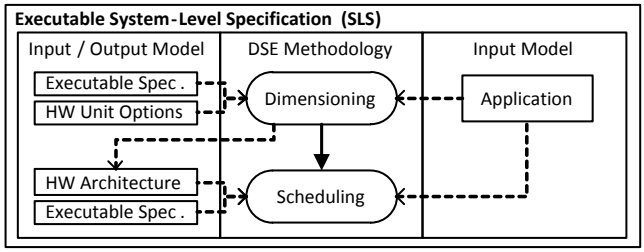

Fig. 1: Example of an executable system-level specification.

This paper introduces an executable SLS which represents an executable DSE methodology and encapsulates system specifications. In other words, our work relates to a higher abstraction level of executable specifications. An SLS realizes a formalization and automation of design flows allowing to scale to larger and more complex embedded systems. In order to be not limited to certain embedded system types, SLSs will be standardized across tools, designers, and domains. Therefore, a meta-methodology, as well as a meta-model are proposed to define a domain-independent SLS.

In the remainder of the paper, Section II gives an overview about specification languages, related DSE environments, and meta-modeling activities. Section III introduces a conceptual framework generalizing SLS at a meta-level and a domain-level. At meta-level, a domain-independent SLS is proposed enabling interoperability across tools, designers, and domains. This SLS is described using a methodology about design methodologies and a model about design models. At domain-level, domain-specific SLSs are created following the proposed meta-methodology and meta-model. This allows to model various design flows applicable for em-

 ISBN: 978-1-61208-213-4

bedded systems with different characteristics, such as real-time, safety-critical, secure, fault-tolerant, robust, etc. In the section, a domain-specific SLS is illustrated via the λ-chart model [4]. In Section IV, an EDA environment is introduced realizing CAD support to build embedded domain-specific SLSs based on the λ-chart. DSE is automatically executed and validated as defined in the SLS. Then, Section V presents a design flow case study for the heterogeneous multicluster architecture built up from SLSs. Finally, the conclusions and open topics are discussed in Section VI.

II. Related Work

The related work focuses on specification and DSE in embedded system design. First, selected specification languages and representative DSE environments are presented. Then, related studies on meta-modeling are discussed.

Specification Languages and DSE Environments

There is a variety of graphical and textual specification languages and frameworks. They can be used to realize DSE methodologies. Nevertheless, this is done in a less formal and less generic manner compared to our SLS approach. Hence, the reuse and interoperability across tools, designers, and domains are limited. An example is the specification and description language (SDL) [5] allowing for formal and graphical system specification and their implementation. In [6], HW/SW co-design of embedded systems is presented using SDL-based application descriptions and HW-emulating virtual prototypes. Moreover, SystemC [2] and SpecC [7] are system-level design languages (SLDL) which model executable specifications of HW/SW systems at multiple levels of abstraction. These simulation models support SW development. For example, SystemCoDesigner [8] enables an automatic DSE and rapid prototyping of behavioral SystemC models. In [9], a comprehensive design framework for heterogeneous MPSoC is presented. Based on the SpecC language and methodology, it supports an automatic model generation, estimation, and verification enabling rapid DSE. Another example is the specification in a synchronous language, e.g., via Matlab/Simulink. Instead, Ptolemy [10] supports various models of computation to realize executable specifications including synchronous concurrency models.

In addition, the MultiCube project [11] and the NASA framework [12] address the need of a generic infrastructure for system-level DSE mainly enabled by modularization. Nevertheless, the works present neither a domain-independent SLS nor a domain-specific SLS.

Meta-modeling

Our paper differs to existing work since it is the first using meta-modeling in order to describe a domain-independent SLS. In the embedded domain, meta-modeling has been studied to transform from the unified markup language (UML) to SystemC at the meta-model level [13]. This guarantees reuse of models and unifies a definition of the transformation rules. In [14], meta-modeling enables heterogeneous models of computations during modeling. In [15], meta-modeling is used to improve the model semantics and to enable type-checking and inference-based facilities.

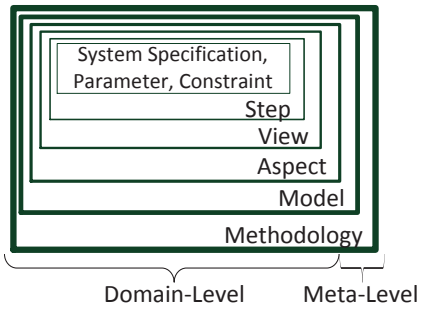

Fig. 2: System-level specification hierarchy.

III. Conceptual Framework

As mentioned before, SLSs aim at reducing the design complexity. The SLS hierarchy, illustrated in Figure 2, gives a hierarchical understanding of SLSs. The abstraction is used as starting point of a formal description. This is realized by separating into a meta-level and a domain-level. The meta-methodology and meta-model allow for developing and testing a methodology and model for a specific design purpose. At the domain-level, specific design aspects, views, steps, system specifications, parameters, and constraints are chosen depending on the domain. That means, certain design tasks are realized in a design aspect using a domain-specific design methodology and design model. Each design aspect includes one or multiple design views modeling orthogonal design functionalities, such as communication, computation and administration infrastructure. Moreover, each design view follows a design process with several steps. Various system specifications, design parameters, and constraints are considered in the steps in order to realize the DSE methodology. Focusing on embedded system design, both levels will be explained in more detail.

A. Meta-Level

At the meta-level, a domain-independent SLS is described to be able to develop and evaluate domain-specific SLSs. Hence, the transfer of design skills gets independent on a design domain and can reach a larger audience. In addition, design concepts and formalisms will be reusable across different tools, designers, and domains. Figure 3-4 illustrate the proposed meta-methodology and meta-model.

In Figure 3, the meta-methodology represents a guiding procedure in order to transform the domain-independent SLS into a domain-specific SLS. It starts to create a separation of the design space into design aspects and a separation of the design aspects into steps. Design aspects divide the design space at a higher abstraction level, as seen in Figure 2. In contrast, a step, system specification, parameter and constraint represent a lower abstraction level. As mentioned before, the specification of design views allows to model orthogonal design functionalities. Referring to Figure 3, an executable DSE methodology is built through an algorithmic ordering of the design aspects and steps. That means, dependencies, loops, branches, etc. realize

 ISBN: 978-1-61208-213-4

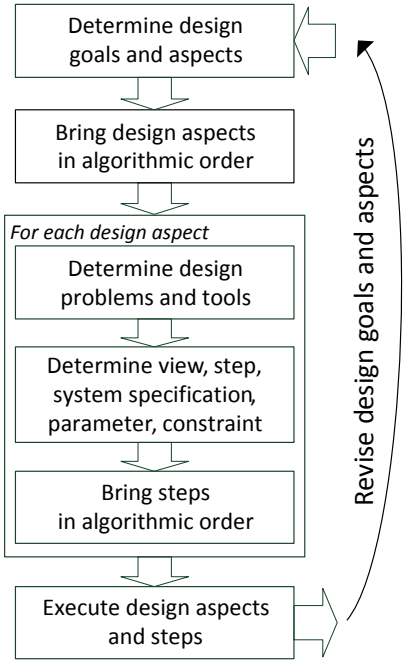

Fig. 3: Meta-methodology for the proposed SLS.

an execution order of aspects and steps in an algorithmic manner. Moreover, design tools are determined in all steps solving the relevant design problems. A design parameter represents a possible description of the structure, behavior, and physical realization of a system. Aiming at improved tool results, suitable design tool parameters are also considered. In each step, the design tools are parameterized and executed using the system specifications. From the DSE results, design goals and aspects can be revised. Finally, the design space is explored by varying the design parameters based on the algorithmic order and DSE strategy, such as exhaustive or heuristic search.

In Figure 4, the proposed meta-model, described via the UML class diagram, represents a model to build domain-specific SLSs. The meta-model forms a fundament or kernel of an EDA environment presented in Section IV. Hence, it includes the definition of the modeling language described via meta classes. Referring to Figure 4, an Element contains Properties and Transitions from/to Elements. A Transition between two Elements is used to model a unidirectional dependency and a Property represents a system specification, design parameter, design constraint, or additional information added to an Element. Moreover, an Aspect and Node inherit from Element. An Aspect includes one or several Nodes. Aspects can be nested to be resolved recursively. This allows to reduce model complexity and to improve the reuse of already modeled aspects. Finally, a Node represents an executable Element, such as a step, loop and branch node, which are necessary to build an algorithmic order of aspects and steps.

B. Domain-Level

In the following, an instantiation of a domain-specific SLS is illustrated with the help of the λ-chart [4] model, as

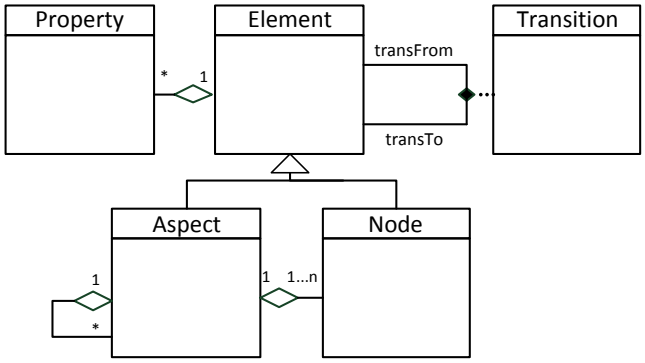

Fig. 4: Meta-model for the proposed SLS.

depicted in Figure 5 (left). The λ-chart models system-level design and exploration in the embedded system domain. As mentioned before, the proposed meta-methodology is used to select appropriate design aspects, views, steps, etc. In addition, an algorithmic order of the aspects and steps must be defined. The λ-chart model is an instance of the proposed meta-model. Referring to Figure 5 (left), a design aspect is represented by a λ-chart instance allowing to define steps in three design views. The administration view considers tasks for planning, monitoring, and control. Computation relates to code execution. Communication includes the design of data storage and data exchange between components. Furthermore, concentric bands underline the five steps of a unified design process. We refer to [4] for a detailed explanation of the λ-chart.

Referring to Figure 5 (left), the exemplary SLS starts with modeling and partitioning the design limited to the communication view. After scheduling and allocation, the DSE results are validated. The allocation and validation steps are iteratively traversed aiming at improved DSE. Similar to [16], the derived network-on-chip (NoC) aspect focuses on finding suitable NoC topology parameters, such as number of rows, columns, and modules per router. Furthermore, meta-model instantiation examples of the domain-specific SLS are illustrated in Figure 5 (right). The allocation step and loop node correspond to a node element in the meta-model. In allocation, exemplary properties are a “Rows” parameter and the communication view. Moreover, a transition from loop to allocation implies an algorithmic order realizing a part of the DSE methodology. The instantiation of an aspect is also shown.

C. Integration in Specification Languages

Specification languages, such as SDL and SystemC, do not currently support the proposed SLS. By doing so, an advantage would be to keep the system designers more aware of the design space in early design stages. The organization into design aspects helps the designer to cope with a complex system-level DSE. In addition, system designers need to structure and arrange their designs into design views. This brings greater attention to orthogonal system functionality, such as computation, communication, and administration. Given the design goals and constraints, it will be more evident that a systematic variation of design parameters is

 ISBN: 978-1-61208-213-4

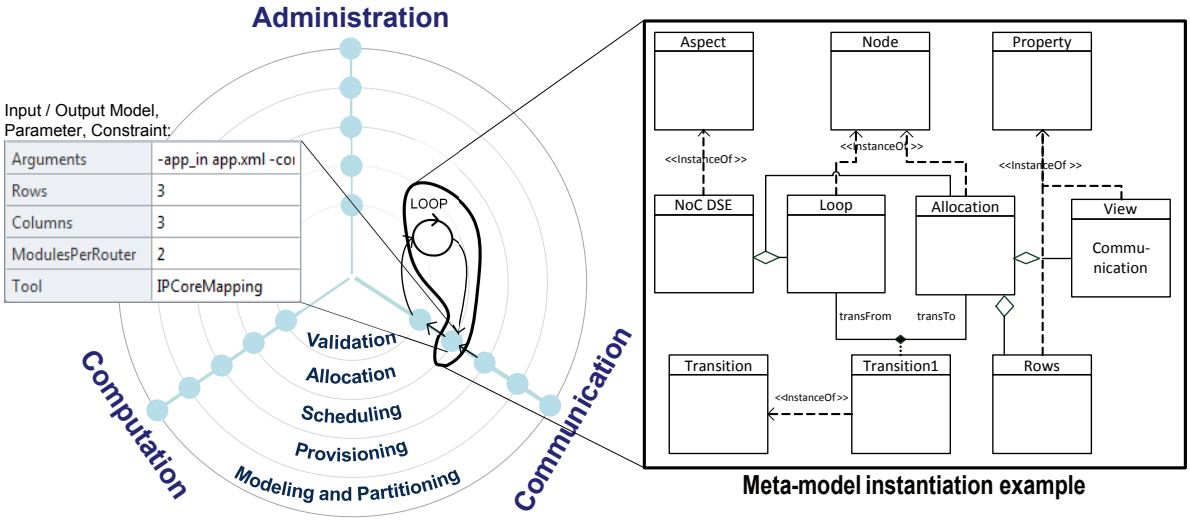

Fig. 5: Example of a domain-specific SLS modeled via the λ-chart [4] (left). Meta-model instantiation examples of the domain-specific SLS (right).

necessary to reach optimal parameter combinations. Hence, the problem of selecting effective search strategies is getting into the focus. Furthermore, an SLS realizes a comprehensive view on available design parameters. Hence, it becomes easier to improve design time and quality by detecting insignificant and interfering parameters. This helps system designers to focus on relevant design and tool parameters.

The integration aims at a coexistence or merger of the proposed SLS and existing specification languages. For example, SDL and SystemC contain module concepts that help to embed system specifications into an executable node of the proposed meta-model. In SDL, systems include a hierarchy of agents called processes and blocks. In SystemC, the Main is the starting point of a SystemC specification. A Main contains several modules and signals to model communications between modules. In the proposed SLS, a node encapsulates an execution of design tools solving design problems, such as scheduling or allocation tasks. These tools produce new or modified system specifications further used as tool input. Depending on the design tool, different system specifications, such as of applications, architectures, application mappings, validation results, tool configurations, etc. are produced and consumed. Hence, a node element can enclose multiple system specifications.

IV. Electronic Design Automation Environment

In the following, an EDA environment is briefly introduced allowing to model and execute domain-specific SLSs based on the λ-chart and presented in Section III. In Figure 6, the tripartite structure consisting of front-, middle- and back-end is depicted. In the EDA environment, an SLS is graphically defined and automatically processed by running the executable node elements. The nodes communicate via parameters and XML-based input/output formats representing system specifications. The specifications address very early system-level design by using coarse-grained representations, such as fixed execution time and high-level task graphs. In the graphical front-end, designers are able to create SLS via CAD. Dynamic search and exploration strategies are

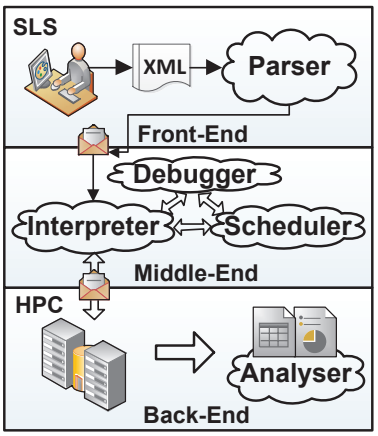

Fig. 6: EDA environment.

considered by including node elements, such as loop, branch, abort, etc. Referring to Figure 6, a parser is responsible for a syntactic analysis of an XML file representing the SLS. In the middle-end, the SLS is interpreted and an execution order is determined via as soon as possible scheduling. A correct execution of each node can be checked through the debugger. In the current implementation, each node executes a command line tool solving specific design problems. Hence, parameters, constraints, and system specifications are assigned via command line arguments. In the back-end, the nodes are executed in a distributed computing environment (High Performance Cluster, HPC). Depending on the purpose, different output formats are created during node execution. DSE results are automatically analyzed and validated as defined in the SLS. Further details of the EDA environment are out of the scope of the paper.

V. Design Flow Case Study

The following case study demonstrates the usage of executable SLSs in order to realize a design flow for the heterogeneous multicluster architecture [17]. The SLSs are executable on the EDA environment presented in Section IV.

 ISBN: 978-1-61208-213-4

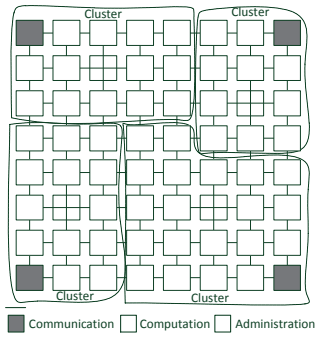

Fig. 7: The heterogeneous multicluster architecture model.

In the paper, the detailed explanations will be limited to one design aspect and the variation of a single design parameter.

A. Application and Architecture Model

The models consider functionalities of the communication, computation and administration view defined in the λ-chart [4]. The application model includes multiple, concurrently running applications and threads, respectively. A thread is represented by a high-level task graph and it sequentially executes tasks. Threads are only synchronized before or after execution. Then, a task is an atomic kernel exclusively executing on an intellectual property (IP) core, e.g., processor core, memory interface, controller interface, etc. Tasks produce and consume chunks of data accessed via shared memory. Side effects are excluded by preventing access to external data during computation.

As shown in Figure 7, the architecture model is a heterogeneous set of multiprocessor system-on-chips (MPSoCs) and clusters, respectively. The administrative unit (AU) represents an application processor and includes a load balancer aiming at equally distributing thread loads amongst the clusters. Moreover, an MPSoC contains heterogeneous types and numbers of IP cores. In the model, each MPSoC contains a NoC connecting the IP cores. Moreover, each cluster includes a control processor (CP) responsible to dynamically schedule arriving tasks to the available IP cores. The CPs are directly connected to the AU.

B. The Design Flow

The design flow, shown in Figure 8, lists several design aspects and executable SLSs, respectively, to create a heterogeneous multicluster architecture. According to Section III, the five domain-specific SLSs are modeled via the λ-chart. They are arranged by the explored parameter types, more specifically with the help of parameters of the design tool, structural, behavioral and physical design. Each SLS realizes a part of the design flow organized as follows:

1) The genetic algorithm (GA) sensitivity aims at finding the best tool parameters for two GAs each solving the multicluster dimensioning and IP core mapping problems;
2) The design aspect “Multicluster Dimensioning” creates a heterogeneous multicluster architecture by distributing estimated application mappings among clusters and solving the optimization problem via a GA [18];
3) The design aspect “IP Core Mapping” places IP cores in an 1-ary n-mesh NoC constrained by the number of modules at each router. The optimization problem is solved via a GA [16];
4) Both design aspects aim at finding suitable behavioral schemes from a selection via simulation. “NoC arbitration” compares a locally fair with a globally fair arbitration scheme. In addition, flit-based and packet-based switching are considered [19]. “Multicluster load balancing” compares different estimators of cluster load, such as response time and queue size, used in the load balancing scheme of the AU.

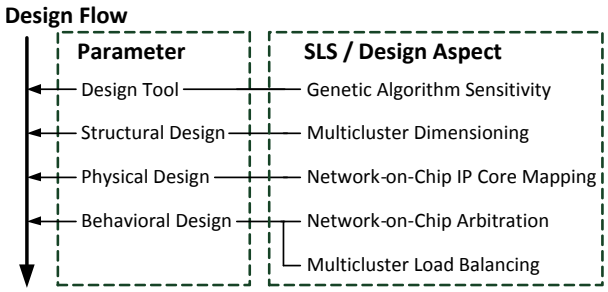

Fig. 8: Design flow case study for the heterogeneous multicluster architecture.

C. Exemplary SLS and Test Results

In the following, multicluster dimensioning has been selected as exemplary SLS. Detailed explanation of the approach and benchmark results can be found in [18]. Referring to Figure 9, the example focuses on the design aspect of creating a suitable computation infrastructure for the heterogeneous multicluster architecture. Hence, DSE is limited to the computation view of the λ-chart. The same benchmark and simulation setup given in [18] has been chosen. The “modeling and partitioning” step serves as a starting point without any further purpose. In the “provisioning” step, the specifications of the target application and the optional IP cores are used as input for the parallelism analysis [20]. The estimated parallelism values represent the numbers and types of IP cores necessary to execute the applications. Since, multiple applications are concurrently running on the target architecture, the parallelism values are used to dimension the processing elements (PEs) in the clusters and MPSoCs, respectively. All clusters are constrained to a maximum number of PEs per cluster (maxPEs) used as a design parameter in the SLS. Further parameters are out of the scope of the paper. As mentioned before, the optimization problem is solved via a GA. The subsequent “scheduling” step performs application mapping via simulation. The scheduling results are analyzed in the “validation” step. Referring to Figure 9, a loop node is inserted to increment the design parameter from two to six. The selected parameter interval depends on the available numbers and types of PEs defined in the chosen benchmark setup. Figure 10 shows the validation results in terms of number of clusters / PEs and the thread response time. The response time is defined as time from the request

 ISBN: 978-1-61208-213-4

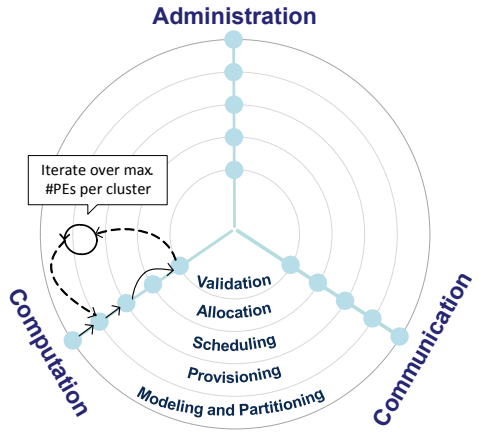

Fig. 9: Multicluster dimensioning SLS and design aspect, respectively.

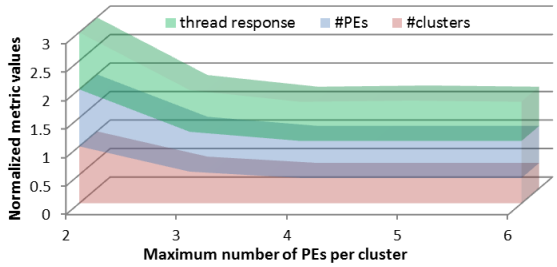

Fig. 10: Test results via parameter exploration for the "Multicluster Dimensioning" SLS.

of a thread until its end. All values have been normalized. In the setup, the best tradeoff is reached for $maxPEs = 4$ resulting in four clusters and 13 PEs. The slightly varying cluster configurations for $maxPEs = \{4, 5, 6\}$, not shown in Figure 10, are due to the heuristic nature of a GA.

VI. Conclusions and Open Topics

In the paper, an executable SLS is proposed in order to cope with an increasing number of design parameters and system specifications during the design cycle. An SLS represents an executable DSE methodology and encapsulates system specifications. The aim is to formalize and automate design flows in order to scale to larger and more complex embedded systems. SLSs should not be limited to certain embedded system types. Hence, SLSs need to be standardized across tools, designers, and domains. Therefore, a meta-methodology, as well as a meta-model are developed defining a domain-independent SLS. Hence, an EDA environment is presented allowing to graphically create and automatically execute embedded domain-specific SLSs. Finally, a case study shows the feasibility of the proposed SLS. Therein, several SLSs demonstrate a realization of a design flow for the heterogeneous multicluster architecture.

In the rest of the paper, a discussion of open topics outlines the future work. So far, an executable SLS can be graphically defined in the introduced EDA environment. Nevertheless, many designers prefer textual programming languages, but according tool support, such as a parser, interpreter, and debugger, is a challenging task. This is left out for future work. Another open topic is to embed industry-relevant specification languages, such as SDL and SystemC, into the executable SLS. Furthermore, an integrated development environment (IDE) should be provided implementing the proposed meta-modeling and integrating the tool chain. The purpose is to create and execute domain-specific SLSs with IDE support. Additional research is necessary to introduce executable SLSs in other domains, such as in communications and sensor networks.

References

[1] D. D. Gajski, J. Peng, A. Gerstlauer, H. Yu, and D. Shin, "System design methodology and tools," *CECS, UC Irvine, Technical Report CECS-TR-03-02*, January 2003.

[2] Systemc, osci. [Online]. Available: http://www.systemc.org/

[3] S. Borkar, "Thousand core chips: a technology perspective," pp. 746–749, 2007.

[4] F. Guderian and G. Fettweis, "The lambda chart: A model of design abstraction and exploration at system-level," in *Proc. of SIMUL*, 2011, pp. 7–12.

[5] ITU-T, *Recommendation Z.100 (08/02) Specification and Description Language (SDL)*, International Telecommunication Union (2002).

[6] S. Traboulsi, F. Bruns, A. Showk, D. Szczesny, S. Hessel, E. Gonzalez, and A. Bilgic, "Sdl/virtual prototype co-design for rapid architectural exploration of a mobile phone platform," in *Proc. of SDL*, 2009, pp. 239–255.

[7] D. D. Gajski, R. Zhu, J. Dmer, A. Gerstlauer, and S. Zhao, *SpecC Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[8] C. Haubelt, T. Schlichter, J. Keinert, and M. Meredith, "System-codesigner: automatic design space exploration and rapid prototyping from behavioral models," in *Proceedings of the 45th annual Design Automation Conference*, ser. Proc. of DAC, 2008, pp. 580–585.

[9] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-chip environment: a specc-based framework for heterogeneous mpsoc design," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 5:1–5:13, Jan. 2008.

[10] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, jan 2003.

[11] C. Silvano *et al.*, "Multicube: Multi-objective design space exploration of multi-core architectures," in *Proc. of ISVLSI*, July 2010, pp. 488 –493.

[12] Z. J. Jia, A. Pimentel, M. Thompson, T. Bautista, and A. Nunez, "Nasa: A generic infrastructure for system-level mp-soc design space exploration," in *Proc. of ESTIMedia*, Oct 2010, pp. 41 –50.

[13] L. Bonde, C. Dumoulin, and J.-L. Dekeyser, "Metamodels and mda transformations for embedded systems." in *FDL*, 2004, pp. 240–252.

[14] D. Mathaikutty, H. Patel, S. Shukla, and A. Jantsch, "Ewd: A meta-modeling driven customizable multi-moc system modeling framework," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 33:1–33:43, May 2008.

[15] D. Mathaikutty and S. Shukla, "Mcf: A metamodeling-based component composition framework–composing systemc ips for executable system models," *IEEE Transactions on VLSI Systems*, vol. 16, no. 7, pp. 792 –805, july 2008.

[16] F. Guderian, R. Schaffer, and G. Fettweis, "Administration- and communication-aware ip core mapping in scalable multiprocessor system-on-chips via evolutionary computing," in *Proc. of CEC*, 2012, accepted for publication.

[17] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The multicluster architecture: reducing cycle time through partitioning," in *Proc. of Micro*, 1997, pp. 149–159.

[18] F. Guderian and R. Schaffer and G. Fettweis, "Dimensioning the heterogeneous multicluster architecture via parallelism analysis and evolutionary computing," in *Proc. of CEC*, 2012, accepted for publication.

[19] F. Guderian, E. Fischer, M. Winter, and G. Fettweis, "Fair rate packet arbitration in network-on-chip," in *Proc. of SOCC*, sept. 2011, pp. 278 –283.

[20] B. Ristau, T. Limberg, O. Arnold, and G. Fettweis, "Dimensioning heterogeneous MPSoCs via parallelism analysis," in *Proc. of DATE*, 2009, pp. 554–557.

 ISBN: 978-1-61208-213-4