Formulation of Design Space Exploration Problems by Composable Design Space Identification

1st Rodolfo Jordão
KTH Royal Institute of Technology
Stockholm, Sweden
jordao@kth.se

2nd Ingo Sander
KTH Royal Institute of Technology
Stockholm, Sweden
ingo@kth.se

3rd Matthias Becker
KTH Royal Institute of Technology
Stockholm, Sweden
mabecker@kth.se

***Abstract*—Design space exploration (DSE) is a key activity in embedded system design methodologies and can be supported by well-defined models of computation (MoCs) and predictable platform architectures. The original design model, covering the application models, platform models and design constraints needs to be converted into a form analyzable by computer-aided decision procedures such as mathematical programming or genetic algorithms. This conversion is the process of design space identification (DSI), which becomes very challenging if the design domain comprises several MoCs and platforms. For a systematic solution to this problem, separation of concerns between the design domain and decision domain is of key importance. We propose in this paper a systematic DSI scheme that is (a) composable, as it enables the stepwise and simultaneous extension of both design and decision domain, and (b) tuneable, because it also enables different DSE solving techniques given the same design model. We exemplify this DSI scheme by an illustrative example that demonstrates the mechanisms for composition and tuning. Additionally, we show how different compositions can lead to the same decision model as an important property of this DSI scheme.**

I. Introduction

The design space exploration (DSE) activity [1] is an integral element of embedded system design methodologies, such as HepSYCode [2], SDF^3 [3], Daedalus [4], and AutoFOCUS3 [5]. Its main purpose in such methodologies is to aid the designer in refining a specification into a more concrete form through exploration of viable refinement alternatives. For instance, DSE in said examples is used to find software-to-core mappings, execution and communication schedules for the target hardware platform, memory allocation etc.

In these methodologies, the designer specifies the embedded system with formal models that define applications, platforms and extra-functional properties. We refer to these models as *design models*. To enable a precise definition and clean semantics of the design models, we advocate the usage of models of computation (MoCs) [6] for the applications, models of architecture (MoAs) [7] for the platforms, and object constraint language (OCL) [8] for extra-functional constraints that need to be satisfied. Additionally, there are objectives such as minimization of power consumption. DSE requires transforming these models into other representations; such as genome encoding for genetic algorithms (GAs), constraint programs for constraint programming (CP), or linear mathematical formulas for mixed integer linear programming (MILP). We refer to such DSE-ready models as *decision models*, and to the set of all design and decision models as *design and decision domains*, respectively. Finally, we refer to the activity of converting a design model to a decision model as design space identification (DSI). Fig. 1 illustrates the role of DSI within the design process as described.

This research was partially funded by the ITEA 3 project PANORAMA (17003) and the Swedish Governmental Agency for Innovation Systems NFFP7 project 2017-04892 Correct by construction design methodology (CORRECT).

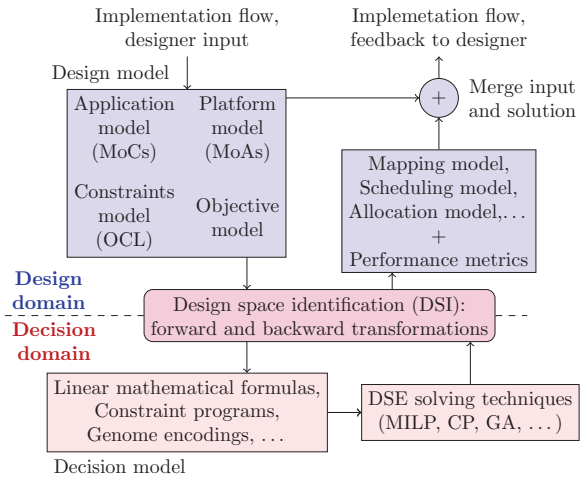

Fig. 1. Position of DSI and DSE in design flows. Blue colors indicate system modeling and design activity elements, red colors indicate decision making elements.

A tuneable but systematic DSI is key to achieve separation-of-concerns between decision domain and design domain, as DSI is the link between system modeling and automatic decision making. An appropriate DSI scheme should be:

- *composable*, in the sense that extensions to the design domain can be stepwise converted to the decision domain as effortlessly as possible;
- *tuneable*, in the sense that the same design model can lead to different decision models, e.g. a MILP and a GA encoding for the same design model.

Our contribution in this paper is a DSI scheme fulfilling these two properties, with the requirement that any design model can be translated to a graph, which for practical design methodologies does not imply any limitations.

II. Identification scheme

This DSI scheme is built around the concept of *composable rules*. These rules take a design model and a list of decision models in order to produce a new decision model. In this way, we can compose rules stepwise by appending the result of one rule to the input list of others repetitively.

The rules produce decision models stepwise by converting *types* of elements in a design model, for example, by converting all synchronous dataflow (SDF) actors in a design model. Therefore, given a design domain and a ruleset with rules for every type of element, any design model in this design domain can be converted to a decision model. If later we directly extend the design domain to accommodate new design models, e.g. an application model described by a MoC not covered yet, then we define new rules for the new type that is built over the existing ruleset, as illustrated in Fig. 2, indirectly extending the decision domain. The definition of new rules is always backwards-compatible: no previously converted decision model is lost. Therefore, this extension process can be repeated indefinitely.

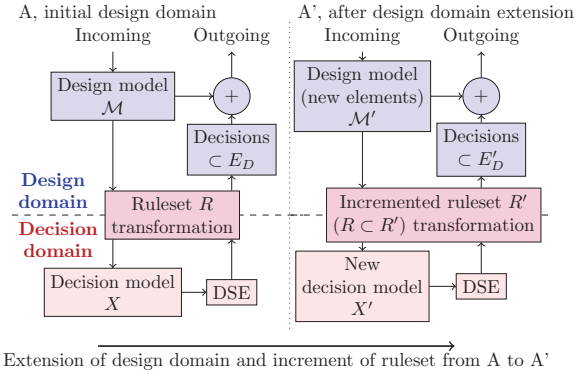

Fig. 2. Overview of a design domain extension step in the DSI scheme, together with addition of new rules to indirectly extend the decision domain.

We assume that all design models are given as a tuple $\mathcal{M} = (G, E_D, J, C)$ as follows. The graph $G = (V, E)$ has nodes V and edges E representing elements from MoCs and MoAs including hardware and software elements such as schedulers. Additionally, E also contains *decision edges* such as "partitions" or "mappings" between nodes. The set E_D of edges represents all the possible decision edges that can further be added between nodes in V as a result of DSE. The design objectives J and the constraints C are given in a language like OCL, using references to G.

Given a design model $\mathcal{M}$, we must find decision edges in E_D to be added in G satisfying the constraints C and fulfilling the objectives J. Accordingly, we must find a decision model X where a DSE solving technique returns a solution that is equivalent to finding the decision edges in E_D satisfying constraints C and fulfilling objectives J. Thus, a decision model X *identifies* $\mathcal{M}$ if there is a function dse that returns $G' = dse(X)$, where the edges of G' are the same of G plus *any* subset of E_D. In other words, all possible decisions can be converted to $\mathcal{M}$ from the solutions of X via dse. Likewise, a decision model X *partially identifies* $\mathcal{M}$ if any solution via dse contains a strict subset of E_D, that is, only a subset of the decisions are captured by X.

Our proposed DSI scheme has three steps. First, we define a ruleset $R = \{r_1, \ldots, r_n\}$, where each rule r_i takes a design model $\mathcal{M}$, a set of decision models $\mathcal{X} = \{X_1, \ldots\}$, and outputs a decision model $X = r(\mathcal{M}, \mathcal{X})$. Second, we define the mechanism that uses R to produce, stepwise, composed decision models out of $\mathcal{M}$,

$$\mathcal{X}_{i+1} = \begin{cases} \{r(\mathcal{M}, \mathcal{X}_i) | r \in R\} \cup \mathcal{X}_i, & i > 0 \\ \{X_\emptyset\}, & \text{otherwise} \end{cases}, \qquad (1)$$

where $X_\emptyset$ is a baseline "empty" decision model. The recursion stops at the fixpoint $\bar{\mathcal{X}}$, when $\bar{\mathcal{X}} = \mathcal{X}_{i+1} = \mathcal{X}_i$. This mechanism relies on the fact that we formally require every rule $r \in R$ to also take a set of decision models, so that r can partially identify more elements of $\mathcal{M}$ by taking previously partially identified $X \in \mathcal{X}$. Third, we choose one decision model from $\bar{\mathcal{X}}$ that identifies $\mathcal{M}$, as there may be several X that identify $\mathcal{M}$, and perform a dse technique on it.

In summary, the scheme consists in a) choosing an initial design domain and defining rules R that compositionally identify any design model $\mathcal{M}$ in it by partial identification, then b) applying the mechanism of Equation (1) to construct these decision models $\bar{\mathcal{X}}$, and lastly c) pick a decision model $X \in \bar{\mathcal{X}}$ that identifies $\mathcal{M}$ and feed it to a dse technique, formalized as a function. Extending the supported design domain involves adding new rules to R as done for the initial design domain.

Two remarks should be made for this rule-based DSI scheme. First, as all decision models are always kept in Equation (1), the order of evaluation for the ruleset R is irrelevant as long as $\mathcal{X}_i$ is computed before $\mathcal{X}_{i+1}$. Second, Equation (1) explains how the scheme is backwards-compatible: if a ruleset R can construct one X that identifies $\mathcal{M}$, then additional rules in R cannot remove X from the existing decision models.

III. Illustrative Example

The design model is based on the use case found in [9]. It consists of a Sobel filter modeled as a SDF application, a predictable time-division multiplexing (TDM) bus-based multi-processor system-on-chip (MPSoC) model, where each tile consists of a processor, memory and a network interface, and abstract cyclic order elements. The order elements describe an arbitrary number of slots, each of arbitrary size. They can represent non-preemptive cyclic execution of programs on a processor, or time slices in a TDM bus that are processed in cyclic order. They also indicate that the Sobel SDF actor firing and communication must be mapped and scheduled order-based in the MPSoC platform. The objective is to optimize the throughput of the Sobel filter. A visual representation of this design model can be seen in Figs. 3 to 5.

A. Ruleset definition and identification mechanism

There are mainly three different *types* of elements in the design model, i.e. SDF graph, order elements, and platform. We define a ruleset $R_{aop} = \{r_a, r_{ao}, r_{aop}\}$, discussed in order of rule composition, and shown in Fig. 3.

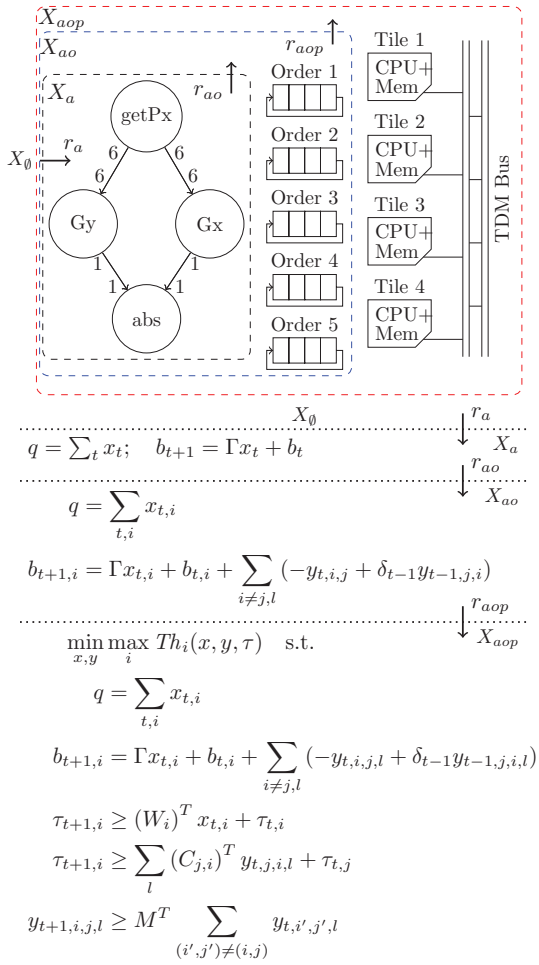

Fig. 3. Graphical and formal display of the stepwise DSI scheme, from empty decision model $X_\emptyset$ to X_{aop}, via application of rules $R_{aop} = \{r_a, r_{ao}, r_{aop}\}$ on the given design model (SDF graph, order elements and platform).

First, r_a takes the empty decision model $X_\emptyset$, and partially identifies the Sobel SDF actors and channels to give X_a, following the semantics of SDF [10]. The new decision variable x_t is the SDF firing vector at index t, b_t is the token vector at t, Γ the topology matrix and q is the repetition vector.

Next, r_{ao} takes X_a and partially identifies the ordering elements to give X_{ao}, introducing the semantics of static cyclic execution for the SDF actors. A new decision variable $y_{t,i,j}$ is introduced for communication that represents the number of tokens sent from tile i to j at t, and δ_t is an indicator function that is zero if $t < 0$ and 1 otherwise. The new i index for x represents each schedule slot i where an actor can fire.

Afterwards, r_{aop} takes X_{ao} and partially identifies the MPSoC to give X_{aop}, introducing execution and communication times as the platform provides these details. Since X_{aop} has the Sobel SDF semantics and platform information, it is an identification. The new index l for y describes which slot l of the TDM bus is being used at t, and M is a matrix so that if TDM slot l is used to communicate from i to j at t, then it must used by i to j on all $t' > t$. The variable $\tau_{t,i}$ is the starting time at t for tile i. The constants W_i is the worst case execution time (WCET) for every actor in tile i, $C_{i,j}$ is the worst case communication time (WCCT) for a token from tile i to j. The function Th_i gives the throughput for tile i based on x, y, τ, WCET and WCCT. In this example, we assume that WCET and WCCT are given as design constraints, and that Th_i has a linear representation.

Last, we could evaluate $milp(X_{aop})$ or $cp(X_{aop})$ (a MILP or a CP solver) to obtain a solution containing order, slot and mapping edges missing in the input design model, as exemplified in Fig. 4. In it, Orders 1 to 4 are used to schedule the SDF actors, and are then mapped to individual tiles. Order 5 is used to schedule the communication and is mapped to individual slots of the TDM bus.

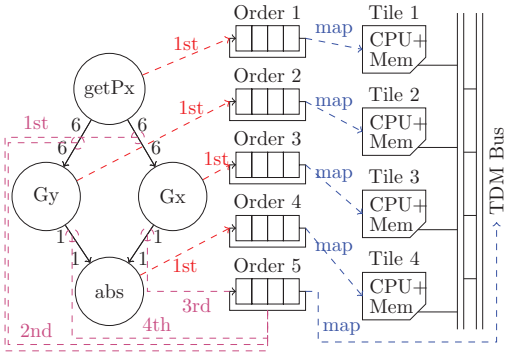

Fig. 4. Possible solution translated back to the design domain. Decision edges represent maps (blue), schedules (red) and TDM slot reservations (purple).

B. Equivalent ruleset definitions

Alternatively, we define a ruleset $R_{poa} = \{r_p, r_{po}, r_{poa}\}$ that starts the scheme from the MPSoC to identify X_{aop}, as seen in Fig. 5. The decision model that partially identifies the platform X_p has no dse function, and serves as a stepping stone for X_{po}. The decision model X_{po} partially identifies the order elements and platform. It has abstract execution and communication slots, u and v, which means that a process is executed or communicated non-preemptively in order, akin to x and y in X_{aop}. The semantics of the SDF Sobel is introduced as r_{poa} takes X_{po} and partially identifies Sobel to give X_{aop}.

Note that both R_{aop} and R_{poa} could be unified, leading to the same decision model X_{aop}. If $R_{both} = R_{aop} \cup R_{poa} = \{r_a, r_{ao}, r_{aop}, r_p, r_{po}, r_{poa}\}$, then the final decision model set would be $\bar{\mathcal{X}} = \{X_\emptyset, X_a, X_{ao}, X_{aop}, X_p, X_{po}\}$ via R_{both}.

C. Definition of new solving functions for tuning

The decision models identified until now are given in a mathematical programming format, amenable to MILP or CP techniques, which may require significant computation time to yield a solution. Without adding a new rule to R_{both} to transform X_{aop}, we can also define other solution techniques over X_{aop}. For example, let us define a $greedy(X_{aop})$ function that is a list-scheduling algorithm, which greedily allocates SDF actors in any available tile (greedily choosing $x_{t,i}$), and

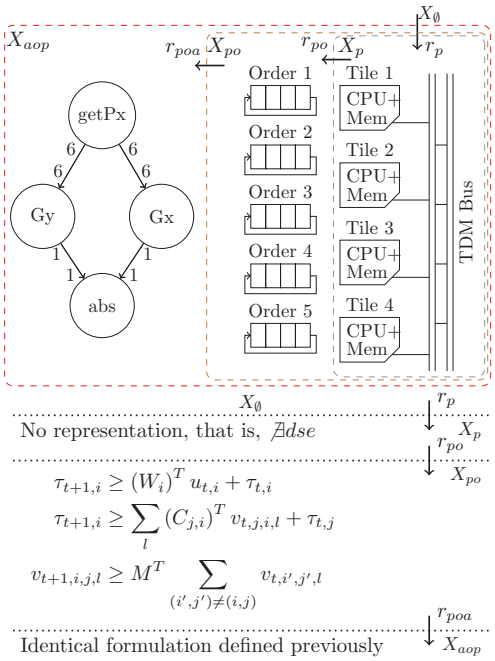

Fig. 5. Graphical and formal display of the alternative stepwise identification, reaching X_{aop} from $X_\emptyset$, via application of rules $R_{poa} = \{r_p, r_{po}, r_{poa}\}$, for the same previously given design model.

allocates TDM slots as necessary whenever communication must occur (greedily choosing $y_{t,i,j,l}$). This would give the designer a choice between an exact solution and a fast solution for this input design model, or any other which is part of the design domain covered by R_{both}.

If instead a GA-based solution is desired, we have to define $R_{ga} = R_{both} \cup \{r_{toga}\}$, with the rule r_{toga} that takes X_{aop} and gives X_{ga} in genome-encoded format. Then $ga(X_{ga})$ is evaluated via GA algorithms.

IV. Related Work

In [11], the authors surveyed a body of DSE methods and identified the need for DSI to increase flexibility and re-usability of embedded systems design flows. Based on this survey, they proposed a DSI meta-framework with base models that can be extended to specific design domains. These extensions can are then translated to new DSE instances. Likewise, in [12], the authors proposed a meta-model that cover distinct embedded system design instances. If extensions to these models satisfy certain meta-model interfaces, the framework generates CPs instances of the DSE problem based on clock constraints. Two alternatives to a systematic DSI are to narrow the design domain, for instance, only SDF graphs on MPSoCs, as in SDF^3 [3], or, to use simulation-based methods such as in COMPLEX [13]. In the latter, the possible design model decisions can be explored directly via meta-heuristics that use simulators and/or performance estimators to test every solution candidate. These methods do not compete with the scheme proposed here, but can be described in it by choosing a descriptive ruleset. For example, in [12], for every type of application and platform element, we can define rules that partially identify the elements to clock constraints and variables in CP format, which another rule later uses to produce the overall CP formulation.

V. Conclusion

We presented a systematic design space identification (DSI) scheme that is composable and tuneable for efficient design space exploration (DSE). The DSI scheme is rule-based and enables the systematic extension of both design and decision domains in tandem, can be tuned to employ different DSE solving techniques, allows arriving at the same decision model by different rule compositions, and is backwards-compatible with respect to rule addition. We demonstrated these properties and the potential of this novel DSI scheme using an illustrative example, where the mechanism of the scheme and different rules were stepwise applied. While this paper provided the basic ideas and mechanisms for the DSI scheme, future work aims to demonstrate how the scheme can cope with heterogeneous design models in form of different underlying models of computation (MoCs) and complex platforms, and to integrate it with existing DSE methods.

References

[1] A. D. Pimentel, "Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration," *IEEE Design Test*, Feb. 2017.

[2] V. Muttillo, G. Valente, and L. Pomante, "Criticality-aware Design Space Exploration for Mixed-Criticality Embedded Systems," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18*, 2018.

[3] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, Jun. 2006.

[4] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, and J. Teich, "Electronic System-Level Synthesis Methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Oct. 2009.

[5] B. Schätz, A. Pretschner, F. Huber, and J. Philipps, "Model-Based Development of Embedded Systems," in *Advances in Object-Oriented Information Systems*, 2002.

[6] E. A. Lee and A. L. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 1998.

[7] M. Pelcat, A. Mercat, K. Desnos, L. Maggiani, Y. Liu, J. Heulot, J.-F. Nezan, W. Hamidouche, D. Menard, and S. S. Bhattacharyya, "Reproducible Evaluation of System Efficiency With a Model of Architecture: From Theory to Practice," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Oct. 2018.

[8] S. Flake and W. Mueller, "An OCL extension for real-time constraints," in *Object Modeling with the OCL, The Rationale behind the Object Constraint Language*, Jan. 2002.

[9] K. Rosvall and I. Sander, "Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms," *ACM Trans. Des. Autom. Electron. Syst.*, 2017.

[10] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, Jan. 1987.

[11] T. Saxena and G. Karsai, "A Meta-Framework for Design Space Exploration," in *2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, Apr. 2011.

[12] S. Attarzadeh-Niaki and I. Sander, "Automatic construction of models for analytic system-level design space exploration problems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2017.

[13] F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia, and G. Palermo, "The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems," *Journal of Systems Architecture*, Jan. 2014.