

Chapter 2

Reusable DSE Methodology for Scalable and Near-Optimal Frameworks

2.1 Introduction

Our goal is to provide a way to solve complex, dependent and large DSE problems in a near-optimal and scalable way. In this target domain, as shown in Chap. 1, the conventional DSE methodologies are less appropriate, because they are inherently based on bottom-up approaches and ad-hoc splits that are not driven by constraint propagation. As a result, they either require too much exploration time to find near-optimal designs due to the high number of design parameters and the correlations between them or they end up with a less efficient result in order to find a design within the acceptable exploration time.

In order to achieve near-optimal designs in reasonable exploration time, a methodology which divides the problem into smaller sub-problems (to reduce the exploration time) and which maintains the full functionality of the sub-problems (to guarantee the near-optimality of the design) is required. When the design problem has been divided into small enough sub-problems, then the conventional design techniques can achieve near-optimal results in reasonable exploration time per quite restricted sub-problem. The division of the design problem into sub-problems could be performed by enumerating all the possible valid division options and their combinations. However, this enumeration process will lead to a huge explosion of the possible combinations and thus it is not practically applicable in large design problems. The alternative option, which is the proposed reusable DSE methodology, is to apply a “gray-box” top-down approach, where the different divisions of sub-problems are explored and refined in hierarchical abstraction layers and in a nested way. In each layer, each sub-problem groups similar options of the lower layers and has known interface (what aspect) but unknown internal part (how aspect). The sub-problems of each layer are not independent and the constraints between them are explicitly derived. The constraints show how the sub-problems affect each other in a uni-directional way allowing their efficient combination. Examples of the unidirectional constraint propagation exist in the sub-steps of the unified design meta-flow of [52], where the “gray-box” top-down approach has been applied

in the context of the abstract layers of design flow, as described in Sect. 2.5. In this book, we systematically define the “gray-box” top-down approach and apply it for DSE and mapping in the background memory and the processing part of the embedded systems for the designing in the processor abstraction level of the unified embedded system design meta-flow [52]. With this “gray-box” top-down process, the near-optimality of the obtained design is maintained, since constraints are not overlooked during the combination of the solutions of the sub-problems. In addition, the sub-optimal combinations are pruned by the uni-directional constraint propagation. The proposed “gray-box” top-down process is iteratively applied: we start from the description of the problem to be solved and we refine it in a complete set of sub-problems with the unidirectional constraint propagation between them. The refinement process is repeated until the sub-problems are small enough to be near-optimally and within reasonable exploration time solved by conventional techniques. Then, the first sub-problem is solved and the results are propagated following the uni-directional constraint propagation to the next sub-problem.

In the rest of this chapter, we describe the principles of the reusable DSE methodology to compose scalable and near-optimal DSE frameworks in Sect. 2.2. Section 2.3 presents how the proposed principles are applied to create a complete DSE framework with the partitioning of all the available options into sub-problems. Section 2.4 describes how the DSE framework is used to select the near-optimal and scalable solutions for the design problem of a target domain in a reasonable time. In the remaining chapters of this book, we present the DSE frameworks derived by applying the reusable DSE methodology. In this way we obtain scalable and near-optimal DSE frameworks in several contexts of large and complex design problems and we use the developed frameworks to determine effective solutions for the target domains under study.

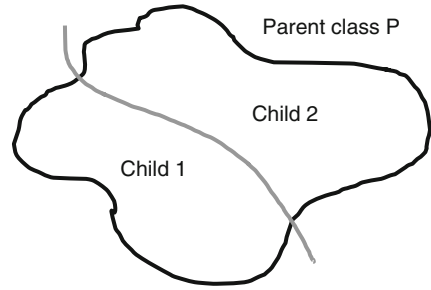
2.2 Principles

The reusable DSE methodology to develop scalable and near-optimal frameworks is applied to a specific context in order to create a projected DSE framework with the division of all the available design space options into cases that correspond to the sub-problems. The DSE framework derives by systematically applying a set of principles, which guarantee completeness during the partitioning of the design space into cases, as no option is left out, and provides uni-directional efficient ordering between the cases based on constraint propagation.

The first principle of the reusable DSE methodology is the “gray-box” top-down division principle which defines the process of partitioning the design exploration space into cases.

Principle 1. *The “gray-box” top-down division principle rigorously applies top-down splits.*

Fig. 2.1 The design space is described by the area inside the *black curve* (Parent class P). The top-down split is described by the *gray line*, which divides the space into two parts, i.e. Child 1 and Child 2



Definition 2.1. A top-down split divides the main characteristics of a parent class P into two children subclasses $S1$ and $S2$, i.e. $S1 \subset P$ and $S2 \subset P$, by detecting an asymmetry in the characteristics of the parent class. The subclasses have the following requirements:

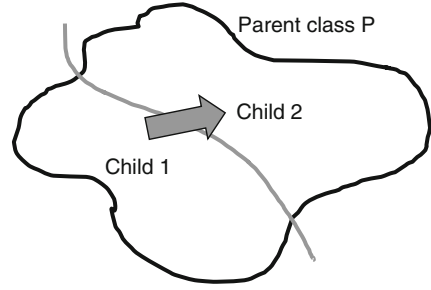
1. Complementary. The parent class P is split into a subclass describing the “X” part and a subclass describing the “not X” part of the detected asymmetry, i.e. $\overline{X} = P - X$.
2. Positive. The “not X” subclass is reformulated into a positive description in order for both subclass to be positive, i.e. $S1 = X$ and $S2 = \overline{X}$.
3. Non-overlapping. The intersection of the subclass is zero, i.e. $S1 \cap S2 = X \cap \overline{X} = \emptyset$.
4. Complete. The subclasses together still compose all feasible options and cover the design space of the parent class P , i.e. $S1 \cup S2 = X \cup \overline{X} = P$.
5. Balanced. The size of the space described by the subclasses is similar, i.e. $S1 \approx S2$.

Depending on the properties of the class, the detected asymmetry may describe the different instantiations of the parent class (top-down split of what type) or the required steps to achieve the parent class (top-down split of how type). Notice that this division into what and how type of top-down splits also describes an asymmetry of a top-down split of a what type! In the top-down split of the what type only one case is enough to instantiate the parent class, while in the top-down split of the how type more than one subclasses may be required to instantiate the parent class. A schematic description of the partitioning of the design space of a parent class through a top-down split is depicted in Fig. 2.1.

The second principle of the reusable DSE methodology is the horizontal uni-directional propagation of constraints in a top-down split, which defines the ordering of the subclasses based on the asymmetry of the top-down split.

Principle 2. *The correlation of the subclasses of a top-down split is described by the horizontal propagation of constraints in a uni-directional way derived by the asymmetry of the top-down split.*

Fig. 2.2 The arrow of the unidirectional constraint propagation illustrates the direction of propagating constraints between the subclasses, i.e. Child 1 and Child 2



In a top-down split, the source subclass is solved first and the solution and the decisions made in the source subclass are propagated following the horizontal uni-directional propagation of constraints to the destination subclass. The uni-directional propagation of constraints is schematically depicted by an arrow between the subclasses, as depicted in Fig. 2.2. The uni-directional ordering of the subclasses is decided based on a set of constraint propagation rules, which are instantiated in the context of the problem under study and with the asymmetry of the top-down split. The constraint propagation rules are:

1. The source subclass provides parameters to the destination subclass, which are required in order to apply the solution of the destination subclass.
2. The source subclass provides parameters to the destination subclass, which are required in order to select how to implement the solution of the destination subclass.
3. The characteristics and nature of the source subclass (heavily) prune the unrealistic or sub-optimal combination of the subclasses.
4. The result and the decisions of the source subclass do not remove potential promising options in the destination subclass.

If no constraints exist between the subclasses, they are considered as independent and thus no arrow is imposed between the subclasses. However, the subclasses independence is a rare case in practice.

The third principle of the reusable DSE methodology is the reusability principle, which is applied during the creation of the DSE framework. When top-down splits are iteratively applied, the more reusable splits are applied first and the more general subclasses are at higher layers during the design problem partitioning. In this way, the higher layers of the DSE framework can partially reused in similar contexts. In the lower layers of the DSE framework, the top-down splits refine more the subclasses leading to more and more concrete cases.

Principle 3. *The reusability principle favors the more reusable top-down splits first among the existing top-down splits.*

When the DSE principles are used into a context of a design problem in order to create the DSE framework, top-down splits following the horizontal uni-directional propagation of constraints are iteratively applied. The result is an ordered partition

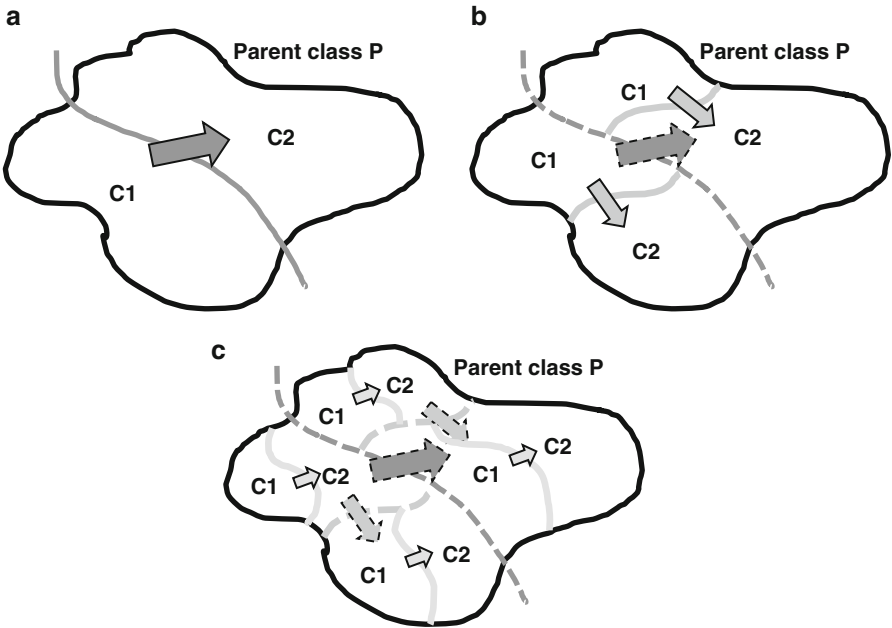


Fig. 2.3 The result of the iteration process of applying the “gray-box” top-down division and horizontal propagation principles of the reusable DSE methodology, which partition the complete space and provide the ordering of the subclasses. (a) First iteration. (b) Second iteration. (c) Third iteration

of the subclasses, as depicted in Fig. 2.3. The first top-down split partitions the complete design space into two subclasses, as depicted by the dark gray line of Fig. 2.3a and the horizontal propagation of constraints defines the direction of the dark gray arrow between the subclasses. In the next iteration, top-down splits are applied in each subclass, as depicted by the gray lines in Fig. 2.3b, which divide again each subclass into two complete and non-overlapping subclasses. The ordering is defined by the gray arrows. The process is repeated per subclass to create further partitions in the space, e.g. as illustrated by the light gray lines and arrows in Fig. 2.3c.

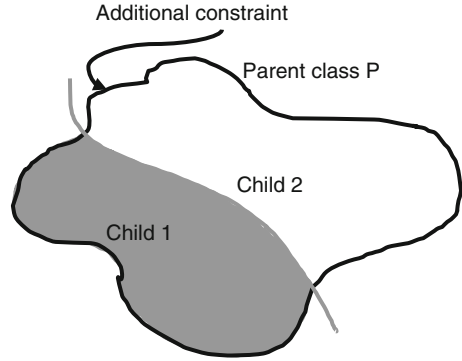
The next set of proposed DSE principles is used after the creation of the DSE framework and drives the use of the DSE framework to derive scalable and near-optimal DSE methodologies.

The fourth principle is the vertical propagation of constraints.

Principle 4. *The subclasses of a top-down split should satisfy the properties of the parent class.*

The vertical propagation principle is responsible for the consistency of the subclasses in a top-down split, when the parent class is partially instantiated. The partial instantiation of the parent class derived from further restricting the initial

Fig. 2.4 The pruning of a subclass due to an additional constraint vertically propagated from the initial parent class through top-down splits



parent class by inserting additional constraints. For instance, scheduling techniques are divided into stochastic and deterministic approaches. A partial instantiation of the parent class, i.e. scheduling techniques, is the scheduling techniques intended specifically for dealing with small graphs. With the vertical propagation principle, the additional constraints are propagated to the subclasses. If a subclass is not compatible with a propagated constraint, i.e. the characteristics and the properties of the options of the subclass are not valid under the propagated constraint, the subclass is pruned. For instance, in this example the additional constraint of small graphs is vertically propagated to the stochastic and the deterministic subclasses and prunes the stochastic techniques. The pruning of the subclass is schematically depicted in Fig. 2.4, where the gray subclass (Child 1) is pruned due to the constraint propagated from the partially instantiated parent class.

The fifth principle is the combination principle.

Principle 5. *When subclasses are combined, the destination class should satisfy the properties and the solutions of the source class, which are propagated as design constraints by the arrows of the horizontal constraint propagation.*

The horizontal propagation of constraints describes the direction of how the constraints should be propagated between two subclasses in order not to prune optimal solutions. Hence, when the solutions of the subclasses are applied, the solutions of the subclass in the arrow source are propagated to the subclass in the arrow destination. Then, the solutions of the destination subclass which are not compatible with the design constraints propagated by the solutions of the source subclass should be pruned, as they do not satisfy the constraints described by the horizontal uni-directional arrow. In this way, the subclass is partially instantiated into a more specific subclass.

In the next section we apply the principles to describe the proposed DSE methodology which creates DSE frameworks for a specific goal and context and the DSE methodology which uses the frameworks to derive near-optimal and scalable solutions for partially instantiated target domains, inside the context of the DSE framework.

2.3 DSE Framework Creation

In order to create the DSE framework with the partitioning and the ordering of all the available options into subclasses, the initial problem is defined and the DSE principles of the Sect. 2.2 are systematically applied.

2.3.1 Methodology Steps

The steps of the reusable DSE methodology to create the DSE framework with the partitions of the complete design exploration space are summarized in the pseudo-code of Algorithm 1. The first step in the DSE framework creation process is to carefully and unambiguously formulate the goal and the problem under study. The problem formulation and the goal are analyzed to identify the most important constraints, objective axes and properties. This step is very crucial since it describes the context of the problem under study and thus the context of the framework. The root of the framework is described by this context and thus the obtained DSE framework should meet the constraints of this context. Then, the refinement process is applied. The root is divided into two smaller subclasses by applying the DSE principles. The top-down split searches in the properties of the context of the problem under study to detect an asymmetry in the space. The reusability principle favors the asymmetries that are more reusable and general in the initial layers of the top-down refinements. Then, the “gray-box” top-down division principle is applied

ALGORITHM 1: Pseudo-code of the proposed reusable DSE methodology to develop the DSE framework with the partitioning of the design space by applying the proposed set of principles

Input: Context of the problem under study
Output: DSE Framework
 Define properties of context(v);
 $v \leftarrow \text{root}$;
 Q : list of parent classes;
 Enqueue(Q, v);
while ($Q \neq \emptyset$) **do**
 Select a parent P from Q ;
 Identify relevant top-down splits based on asymmetries in the parent characteristics;
 Select based on re-usability of candidate top-down splits;
 Prefer more balanced splits, $|S1| \approx |S2|$;
 Apply split by creating $X = S1$ and $\bar{X}$ subclasses;
 Positively reformulate $\bar{X}$ to $S2$ subclass;
 Define direction of constraint propagation based on asymmetry and uni-directional constraint propagation rules;
 if (*further refinement is required*) **then**
 Enqueue($Q, S1, S2$);

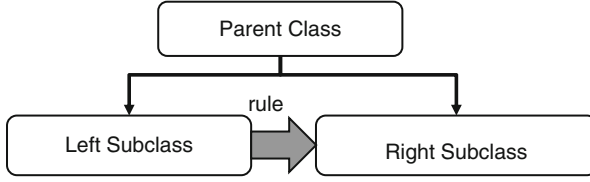


Fig. 2.5 Schematic representation of the general step of the propose methodology to develop DSE frameworks, where DSE principles are applied in the parent class to refine it into two subclasses connected through unidirectional constraint propagation

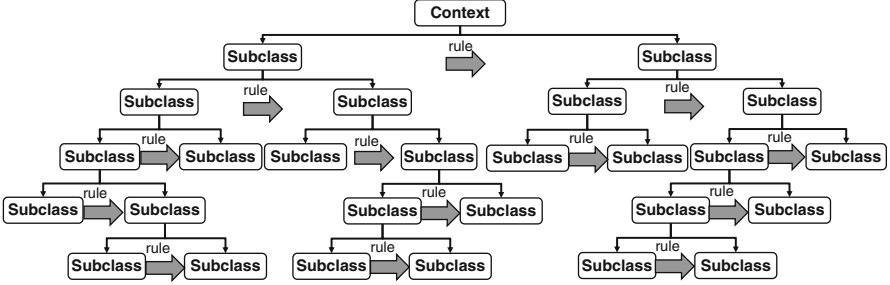


Fig. 2.6 DSE framework with the subclasses of the complete space of the problem under study after iteratively applying the general step of the proposed methodology to develop DSE frameworks

based on the detected and selected asymmetry. The result is two subclasses, which describe a smaller set of options that the root and satisfy the properties of the root. The size of the described space, i.e. the options, is similar between the subclasses due to the balanced property. Each subclass describes non-overlapping parts of the space and they, together, describe the complete space of the root. The horizontal uni-directional constraint propagation principle is applied in the top-down split to define the ordering of the subclasses. Based on the detected asymmetry, which is used in the top-down split, the corresponding constraint rules are applied. The result is the ordering of the two subclasses connected with a uni-directional arrow, which describes the constraint propagation between them. The negative subclass is reformulated into a positive description, e.g. non-stochastic scheduling approaches are reformulated into the deterministic approaches. The top-down split with the uni-directional arrow is schematically depicted in Fig. 2.5, where the parent class is split into the left subclass, which propagates constraints into the right subclass, and the rule above the uni-directional arrow describes which rule is applied by the horizontal uni-directional constraint propagation principle to derive the corresponding arrow.

The process is iteratively applied for each of the subclasses creating a tree structure T with refined subclasses and their unidirectional arrows, as schematically depicted in Fig. 2.6. The uni-directional arrows of higher layers are valid also in the branches of the subclasses. The refinement process terminates, when the subclasses are small enough to be solved by in a scalable and near-optimal way by bottom up approaches, e.g. specific conventional techniques, or generalized techniques described by parametric templates.

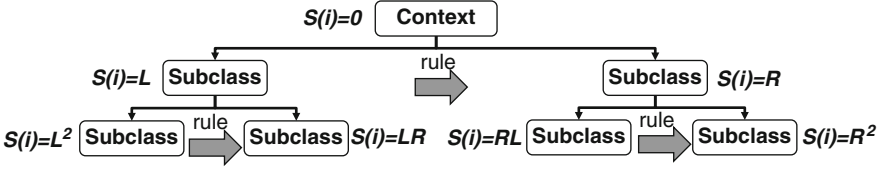


Fig. 2.7 Tree with two levels of depth and the corresponding labels per subclass

2.3.2 DSE Framework Properties

2.3.2.1 Framework Completeness

Each subclass i has a label $s(i)$ to indicate its position in T and describes the path from the root to subclass i . The label of the root is 0. The elements of the label of the subclasses are L and R , where the L describes the Left subclass and the R the Right subclass in a top-down split in T , as depicted in Fig. 2.7. The length of the label $length(s(i))$ indicates the depth of the subclass in T (subclass layer).

Definition 2.2. Two subclasses i and j are called brother subclasses when they derive from a split applied in the parent class p . They belong to the same layer of T , i.e. $length(s(i)) = length(s(j)) = length(s(p)) + 1$. The brother labels $s(i)$ and $s(j)$ differ in the last element. They derive from concatenating the parent label $s(p)$ with a L or a R element depending on the position of the child subclass in T , i.e. $s(i) = s(p)|L$ or $s(i) = s(p)|R$.

Definition 2.3. A couple of subclasses m and n is: $m - n$.

Based on the “gray-box” top-down division principle we remedy the fact that missing points exist, leading to the Theorem 2.1. The subclasses of the DSE framework still cover all the possible options of the design space and also describe non-overlapping areas of the search space, as characteristics and options loss cannot occur during the DSE framework creation process. This completeness is crucial to guarantee a global DSE framework with all the available feasible options and to guarantee near-optimality.

Theorem 2.1. *The derived DSE framework provides a complete and consistent partitioning of the available options of the context of the problem under study into subclasses.*

Proof. The process starts from the root, which describes the context of the problem under study, i.e. the problem formulation, the goal and the properties. In each layer, the process applies top-down splits which lead to complementary, non-overlapping, positive and complete subclasses. This iterative partitioning is enabled due to the positive reformulation of the subclasses, which allows the splitting process to continue. Let’s assume that the derived DSE framework by the proposed methodology is not complete. Then, an option exists which meets the context properties and

constraints, but it is excluded from the framework, i.e. the characteristics of this option are not included in any of the framework subclasses. The latter means that a split without the top-down split properties exists in the proposed DSE framework. In order the latter to be possible, the proposed methodology to derive the DSE framework does not strictly apply the “gray-box” top-down division principle, which contradicts the Principle 1.

2.3.2.2 Framework Compactness

The options of the subclasses can be shared following the constraint propagation arrows. This reuse of subclasses allows the description of all options with a reduced number of subclasses leading to a quite compact DSE framework, which is essential for an effective DSE methodology.

Theorem 2.2. *The DSE framework consists of a limited set of the unique primitive subclasses.*

Proof. Assuming that the DSE framework consists of more than the required subclasses, then it is redundant. In this case, two subclasses N and M both describe the options of the same area of the exploration space. If these redundant subclasses N and M are children of the same parent, it contradicts the properties of the top-down split, as the brother subclasses describe not complementary parts. If they are children of different parents, redundancy exists in their ancestors. Following the DSE framework tree in a backward way from the subclasses N and M to their ancestors, we can identify which split is responsible for these redundant subclasses. Due to the tree structure of the DSE framework, the redundant subclasses will be children of the same ancestor in a higher level. Then, the split in the ancestor creates two subclasses which describe non-complementary set of characteristics, which again contradicts the properties of the top-down split.

Theorem 2.3. *The DSE framework is a compact scheme.*

Proof. By Theorem 2.1 the DSE framework is a complete tree. By principle 4 the subclasses are restricted by the characteristics of the parent class. The Theorem 2.2 does not allow redundancy to exist. Hence, all options can be described by the sharing through consistent combination the unique subclasses, which indicates a compact DSE framework scheme.

2.3.3 DSE Framework Types

Depending on the properties of the context of the problem under study, the resulting DSE framework may describe the different ways in which the goal can be achieved (what type framework) or may partitioning the goal into smaller sub-goals that will lead to the achievement of the initial goal (how type framework) or a combination

2.4 DSE Framework Usage

The DSE framework describes the partitioning and the ordering of the subclasses, which completely describe the options of the design exploration space of the problem under study. When the initial context is restricted, several of the subclasses should be pruned due to incompatibility reasons with the additional constraints.

2.4.1 *Insight of the DSE Options*

The DSE framework has well-defined subclasses which describe non-overlapping areas and are reused and shared in a systematic way. Hence, the DSE framework offers useful insight into the subclasses, as it efficiently supports their analysis. The insight is achieved by decomposing the properties of a quite restricted context into its primitive components, i.e. the classes of the DSE framework. The decomposition enables the understanding of the similarities, the differences, the main characteristics and the interrelationships of the different components, which is essential for the efficient use and the further improvement and optimization of the techniques described by the subclasses. The DSE framework can precisely identify the similarities in every subclass, as each subclass is unique with at least one non-overlapping characteristics compared with the remaining subclasses. In addition, the DSE framework identifies the most relevant and unambiguous differences between the subclasses due to the asymmetries used to create the top-down splits. The similarities and differences do not form each other's direct complement. This process is essential for the in-depth understanding of the DSE options and thus for their efficient use and improvement with a minimized time and effort compared with previous frameworks or with individual ad-hoc study of all the existing DSE options.

2.4.2 *Framework Projection*

The vertical propagation principle is responsible for the consistency of the splits in a DSE framework, when the DSE framework is used to select near-optimal solutions for a partially instantiated target domain, i.e. a partial instantiation of the context of the DSE framework. For instance, we assume a framework which describes the scheduling techniques which are applied at design-time and are able to search the entire solution space to find the near-optimal solutions. This DSE framework can be used for several partially instantiated target domains, such as the scheduling of applications described by small graphs or of applications with large subgraphs and hidden hierarchy etc. The partially instantiated target domain is more restricted than the initial context and thus inserts additional constraints to the root class. Depending

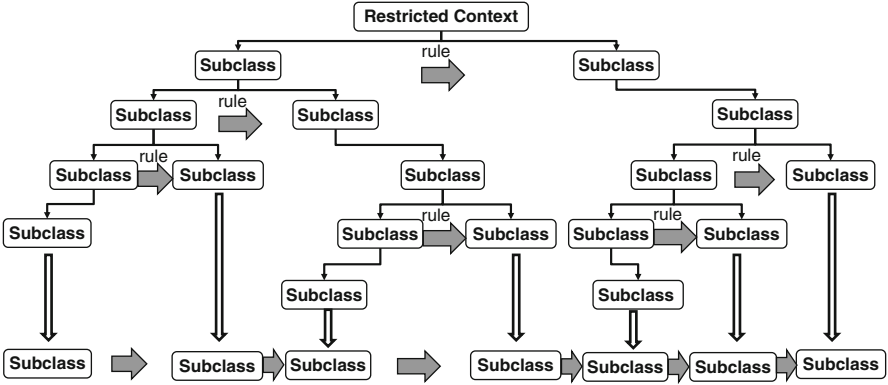


Fig. 2.9 Pruned tree after applying the vertical constraint propagation principle. The characteristics of the subclasses from the root to each leaf are merged and flattened into one subclass

on the type of the DSE framework, the result of the DSE framework projection can be either the steps of a DSE methodology, the different required properties of a solution of the problem under study or the different cases that may occur during solving a design problem.

The additional constraints of the restricted target domain are combined with the properties of the root of the DSE leading to a restricted context. The additional constraints are vertically propagated to prune the initial DSE framework. The vertical constraint propagation pushes the constraints of the restricted context down to the two subclasses. First, for the subclass, which is the source of the uni-directional horizontal constraint propagation arrow, it is verified whether it meets the vertically propagated constraints. If it meets them, then the subclass is not pruned. The vertical constraint propagation continues with the branch below the source subclass. The subclasses are compared with the vertically propagated constraints by verifying first the source subclass of the unidirectional arrow etc. If the source subclass is incompatible with the vertically propagated constraints, it is pruned and the branch below the pruned source subclass is not further explored. Then, the destination subclasses are explored for compatibility with the vertically propagated constraints. After the process of the vertical constraint propagation, the DSE framework has been pruned and the subclasses that have been left are the only valid and compatible with the constraints. In Fig. 2.6, the initial DSE framework for the problem under study is depicted. In Fig. 2.9 the pruned tree after the vertical constraint propagation is illustrated. After flattening of the valid subclasses following the uni-directional arrows, the result is an ordered chain of the valid subclasses. Flattening is performed by merging the characteristics of each leaf with the characteristics of the subclasses of the path from the root to this leaf. After the flattening process, each subclass can be filled with near-optimal and scalable conventional techniques or by parametric templates (see below) which describe in a generic way the different options of each subclass. The result of the chain of subclasses is connected through uni-directional arrows.

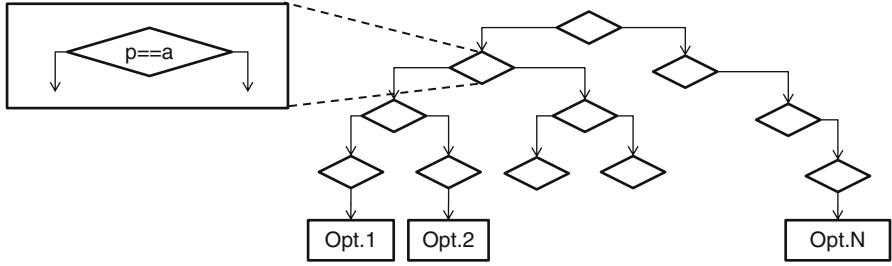


Fig. 2.10 Schematic description of a parametric template of a subclass. The boxes describe the design options and the rhombus the condition expressions which lead to the different options

2.4.3 Trade-Off Exploration

During the DSE framework creation process only the constraints of the problem under study have been taken into account. During the use of the DSE framework, the trade-offs have to also be considered through trade-offs exploration in order to guide the decisions towards the most promising solution of the design problem under study.

For each subclass of the flattened DSE framework, the different options can be defined by either a parametric template, which describes a generalized solution with parameters, algorithms and closed form equations, or by conventional existing approaches, which now can be near-optimal and scalable because of the limited scope and size of each subproblem described by the subclasses. In Fig. 2.10 we schematically depict a parametric template that describes the different design options in an efficient what-if structure. To have a Pareto curve for the overall trade-offs of the design, the relevant trade-off axes have to be explored in each subclass. Then, the combining principle is used to merge the trade-offs of the subclasses into the final Pareto curve. Initially, we estimate the value of the relevant trade-offs axes for the different design options in the first subclass of the chain of the flattened DSE framework. The result is a partial Pareto trade-off curve with the near-optimal design options for the first subclass. Based on the combining principle, the results of the first subclass in the chain are propagated to the next subclass following the horizontal uni-directional constraint propagation. The design options of the next subclass are explored taking into account the propagated results of the first subclass. In this way, the design options of the parametric template of the next subclass, which are incompatible with the decisions propagated from the first subclass, are pruned and only the valid design options in the parametric template of the second subclass are considered. In Fig. 2.11, the gray design options are pruned due to incompatibility with the propagated design constraints of the first parametric template. In this way the design options of the second subclass are explored under the constraints created by the valid options of the first subclass. The pruning of the second subclass does not remove promising solutions. The latter

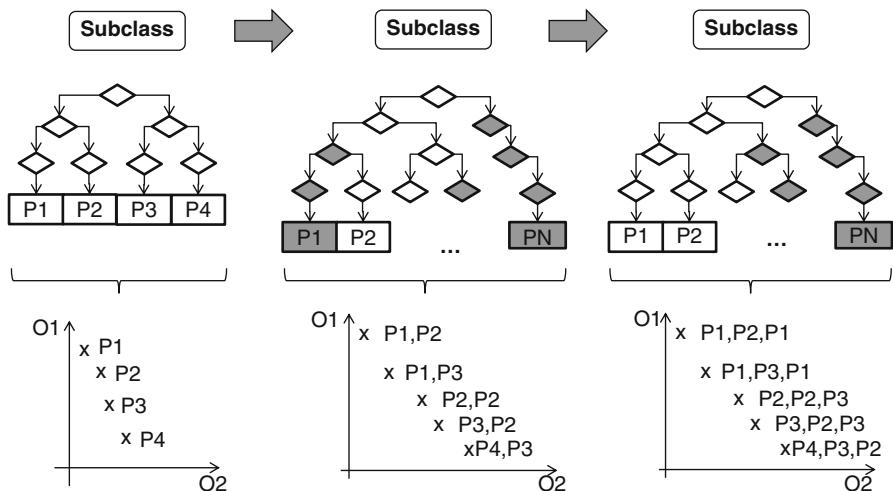


Fig. 2.11 Three subclasses with the exploration of the parametric templates. The parametric templates and the corresponding partial Pareto curves are combined following the uni-directional constraint propagation

has been taken care by the horizontal propagation principle which decided the uni-directional arrows based on rules during the DSE framework creation. The result is a merged partial Pareto curve, where the points describe the near-optimal valid points for both subclasses. The process is repeated for the remaining subclasses of the chain. In the last subclass, the result is the final overall Pareto curve, with points that describe near-optimal solutions for the problem under study.

The process is schematically depicted in Fig. 2.11, where we assume three subclasses connected through uni-directional propagation of constraints. The options of each subclass are described by a parametric template. The partial Pareto curve of the first subclass is created (P1, P2 etc. points in Fig. 2.11) by giving valid values to the parameters of the first parametric template. The results are propagated to the second subclass, where the valid points of the first subclass prune incompatible design options in the second parametric template. The result is a new partial Pareto curve, which describes the valid options of both subclasses, i.e. points P1, P2 and P1, P3 etc. in Fig. 2.11. The P1 point is the point propagated from first subclass and P2 and P3 are the only valid points of parametric template of second subclass for the propagated values of P1. In this way, two Pareto points are created. The final subclass is explored under the propagated Pareto curve. The combined result is the overall Pareto curve of the problem under study. For instance, for the P1 valid point of the first subclass and the P2 and P3 valid points of the second subclass (under the constraints of the P1 of the first subclass), the points of the final Pareto curve are P1, P2, P1 and P1, P3, P1 in Fig. 2.11. The final Pareto curve can be used to select the most promising point based on the specification of the problem under study.

The use of the DSE frameworks to perform scalable and near-optimal DSE methodologies is reapplied in several contexts in this book. The most representative examples are in Chaps. 5 and 6, where the intra-signal in-place optimization methodology uses the unidirectional constraint propagation to remove suboptimal options of the parametric templates of the steps for the non-overlapping and the overlapping case, Chap. 8, where a methodology to develop scalable and near-optimal parametric templates for target domains in the context of near-optimal design-time scheduling techniques is presented and Chap. 9, where we present a design-space exploration methodology based on partial Pareto curves and horizontal propagation of constraints for the context of mapping loop dominated applications onto a microprocessor and several hardware accelerators.

2.5 Unified System Design Meta-flow

The principles of the DSE framework creation have been applied in the higher layers of the unified embedded system design meta-flow in [21, 22] to develop an abstract but unified design flow, which is depicted in Fig. 2.12.

Due to the properties of the reusable DSE methodology of Chap. 2, the unified design flow for global mapping DSE does not include overlapping design steps or redundant design iterations. Hence, it is divided into different abstraction layers, which are connected through uni-directional propagation of constraints. The layers of the unified system design meta-flow derive from applying top-down splits in the context of system design. The context of the problem under study is divided into the algorithm level and the DSE. The algorithm level provides a complete and formal description of the system, which should efficiently deal with the temporal correctness and the refinement of the data types. The result of the algorithm level is a set of optimized system specifications with a set of concurrent communication tasks with timing requirements. The DSE is divided into the subclass that is dedicated to

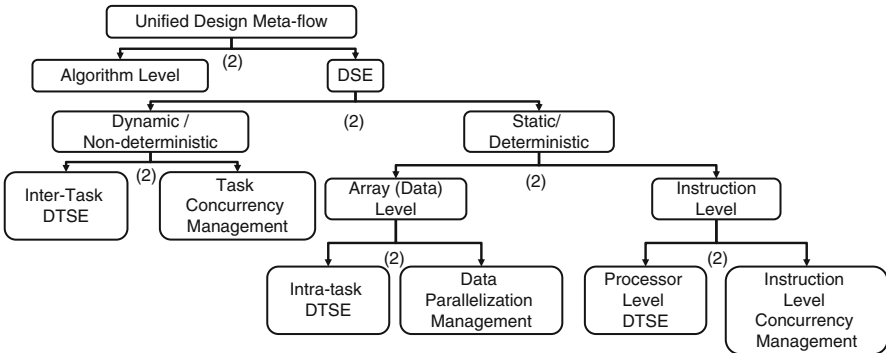


Fig. 2.12 Unified system design meta-flow of [21, 22]

the mapping of the dynamic and non-deterministic parts of the algorithm and the subclass which is dedicated to the mapping of the static/deterministic parts of the algorithm. The dynamic/non-deterministic subclass is dealt by the mapping stages of the task (or thread frame) level.

The Thread Frame Level (TF) mapping deals with the dynamism of the application which is expressed in a non-deterministic way. It mainly focuses on the stochastic events which dynamically trigger the creation of application thread frames. The Thread Frames are heterogeneous in nature. The use of the system scenario concept among thread frames, i.e. the inter-thread frame system scenario approach, allows to modify the thread frame behavior in order to have the overall application graph become more deterministic. However, the complete thread frame behavior cannot be fully transformed to a deterministic equivalent and thus a part of the thread frame remains non-deterministic. Since the thread frames are dynamic and heterogeneous, the hardware platform resources should also be dynamic and heterogeneous in order to support an efficient mapping of these thread frames to the hardware platform. The heterogeneous hardware platform resources should be described by a virtual layer to support the required dynamism, which results in a view based on abstract processors. In this way it is possible to dynamically assign parts of the existing heterogeneous resources to the concurrent thread frames. The task level is divided into the storage mapping stage, i.e. inter-task Data Transfer and Storage Exploration (TF DTSE) stage and the processing related mapping stage, i.e. the Task Concurrency Management (TCM) stage. The TF DTSE stage decides for the assignment of the dynamically created and accessed data on the virtual run-time platform layer. The TCM stage performs the management of the concurrency between the thread frames; it explores the parallelization opportunities in the mapping of the dynamically created thread frames, i.e. the allocation of the dynamically created tasks to the virtual run-time layer heterogeneous processors; and it decides on the run-time management of the communication among the abstract processors, and up to the virtual memory hierarchy.

When the application has deterministic characteristics, the Array (or Data) Level (DL) and the Processor Level (PL) or Instruction Level (IL) mapping are efficiently mapping the application. The deterministic characteristics can be manifest and static or data-dependent, which inserts dynamism into the application. The deterministic dynamism is inserted due to the data dependencies in the conditions and the loops. It can be removed by using system scenarios inside the thread frames, i.e. the intra thread frame system scenario approach. The instantiations of the data dependent applications with similar behavior are clustered in one intra-thread frame system scenario. Then, the worst-case instance inside each individual scenario is used as the representative case for further manifest analysis. In this way, for each scenario, the dynamism both among the thread frames and inside each thread frame has been converted to a manifest/static projection. Inside the resulting graph projection no dynamic task creation, event handling and synchronization or data-dependent conditions remains. In this way, the obtained graphs (one per scenario) can be efficiently dealt by the Data Level and the Processor Level mapping.

The Data Level mapping is dedicated to homogeneous concurrency characteristics inside the thread frames of the heterogeneous Thread Frame Level mapping.

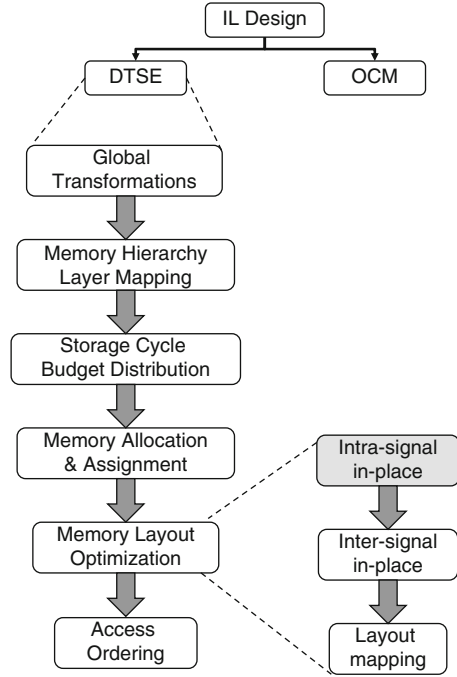
A Thread Frame can be described by functions, which are clusters of non-uniform operations, and uniform operations among the clusters. The homogeneous software characteristics are defined by the operations which have uniform behavior, i.e. they are operations of the same type, they use the same type of operands (e.g. same word length) and the dependencies allow parallelism to exist. The uniform operations are mapped to an array of homogeneous super-processors with internal data memory hierarchy. The Data level is split into the Data Level DTSE (DL DTSE) and the Data Parallelization Management (DPM) stages. The Data Level DTSE stage decides the background memory management of the regularly accessed data of uniform operations inside a thread frame. The DPM is dedicated to the homogeneous mapping of the operations on the arrays of super-processors and the communication of the homogeneous super-processors.

The Processor Level mapping is dedicated to the remaining heterogeneous characteristics of the application, i.e. the functions with the operations which have non-uniform behavior. The non-uniform operations are mapped to heterogeneous resources, i.e. a super-processor with a data memory hierarchy, which include all the required resources to execute the functions of the processor level mapping. The instruction layer is split into the Processor Level DTSE (PL DTSE) and the Operations Concurrency Management (OCM) stages. The Processor Level DTSE is applied first to decide the background memory management of the accessed data of the irregular internal thread frame behavior, i.e. non-uniform operations. The Processor Level DTSE decision is propagated to the next steps of the OCM. The propagation is in this direction since the result of the Processor Level DTSE, i.e. the assignment and the access ordering of the arrays, is required for the address optimization, the communication and the intra-processor mapping. Moreover the background memory related decisions have a higher cost impact than the OCM decisions and they leave sufficient freedom for the OCM decisions to come up with near-optimal results (see [21,22]). Finally, the circuit level stage maps the design on the target technology platform, i.e. custom IC or reconfigurable FPGA.

The unified embedded system design meta-flow is not a specific detailed design flow for a specific application domain and large research effort is needed to derive a customize flow for a specific domain [52]. For a specific domain, not all steps of the unified design meta-flow are crucial, as the characteristics and the constraints of the domain may be irrelevant to some steps. Hence, the irrelevant steps should be pruned when the customized design flow is created. For the remaining steps, appropriate techniques need to be selected or developed to near-optimally and in a scalable way implement these steps. The selection or development of the required techniques is domain dependent.

The scope of this book is to further refine the steps of the abstract unified system design meta-flow in [21, 22], by applying the principles of the reusable DSE methodology of Chap. 2 to create scalable and near-optimal DSE frameworks. In particular, we focus on the Processor Level of the unified system design meta-flow and apply the principles of the creation of scalable and near-optimal DSE frameworks to develop scalable and near-optimal DSE methodologies both for the background memory (PL DTSE) and the processing part (OCM). The steps of the

Fig. 2.13 Design steps of instruction layer abstraction for DTSE



PL DTSE have been presented in [20] and are summarized in Sect. 2.5.1. The steps of the OCM stage are described in [21] and are summarized in Sect. 2.5.2. The focus of the book is to develop a complete DSE methodology for near-optimal and scalable intra-signal in-place optimization design step of PL DTSE part, which is applicable to both regular and highly irregular access scheme, in contrast to polyhedral approaches of existing DTSE methodology, which are applicable up to quite regular memory accesses. We also apply the principles to the OCM part to develop DSE methodologies for the target domain of a microprocessor with several HW accelerators and scheduling and assignment techniques for foreground memory management and data path mapping steps.

2.5.1 Processor Level DTSE

The Processor Level DTSE is dedicated to the management of data inside the application functions, which consist of non-uniform operations. The sub-steps of PL DTSE are described by the DTSE methodology in [20] and they are depicted in Fig. 2.13.

The DTSE is split to the platform independent transformations and platform dependent mapping step. The platform independent transformations have as purpose

to increase the data reuse and to improve the objectives, e.g. the access speed, the area, the power etc. They include data flow, loop and data reuse transformations. The platform dependent mapping step is further refined into the step that performs the memory hierarchy mapping and the memory layer mapping. The memory hierarchy mapping is applied in a heterogeneous way, since the memories in the different layers have different characteristics, allowing an efficient use of the memory hierarchy. The memory hierarchy can be further divided into the platform-independent transformations to improve the initial access behavior and the actual mapping to the memory hierarchy platform. The latter is divided into the logical ordering and the assignment of the data to the memory hierarchy. The memory layer mapping is divided into the mapping of a few heterogeneous resources and into the mapping onto an increased number of homogeneous resources. This split derives from propagating the constraints of the real environment, where a high number of heterogeneous hardware platform resources cannot exist in reality and the mapping to a few homogeneous resources is trivial. The Processor Level mapping is dedicated to the non-uniform operations, which can have both irregular and regular accesses to the data. The irregular array accesses are mapped in the few heterogeneous resources and the regular array accesses are mapped to the homogeneous resources. The opposite option of mapping the regular accesses to heterogeneous resources is suboptimal and the mapping of irregular accesses to homogeneous resources is not possible.

In the case of the few heterogeneous resources a transformation step and a mapping step can be applied. The transformation step applies only the transformations relevant to the characteristics of the few heterogeneous resources. The mapping step describes the techniques to apply ordering in time and then assignment in space. In the PL DTSE, the ordering of the processor DTSE with few heterogeneous resources is described as Storage Cycle Budget Distribution (SCBD) and the corresponding assignment as Memory Allocation & Assignment (MAA). The ordering and assignment decisions are essential since they propagate constraints to the OCM steps.

In the case of several homogeneous resources, a similar partitioning is applied. However, the transformations, the assignment and the ordering are dedicated to spatial characteristics in this case. Hence, a set of spatial transformations is applied to increase the spatial locality. Then, a spatial assignment of the data to the homogeneous resources takes place, i.e. data layout. The data layout step applies especially the in-place mapping to reduce the required storage space, both for the intra-signal in-place optimization, i.e. accesses to the same array, and the inter-signal, i.e. accesses to different arrays. Then, the virtual memory data layout takes place where the data are assignment to a set of virtual memories. Finally, the physical memory data layout is composed by mapping the virtual memory data layout in the physical homogeneous memory elements of the platform. When a hardware-controlled cache is present, also a second data layout step is added which is related to conflict miss reduction. But that step will not be addressed here further. The result of the memory data layout is propagated to the special ordering to finalize the way the data are accessed.

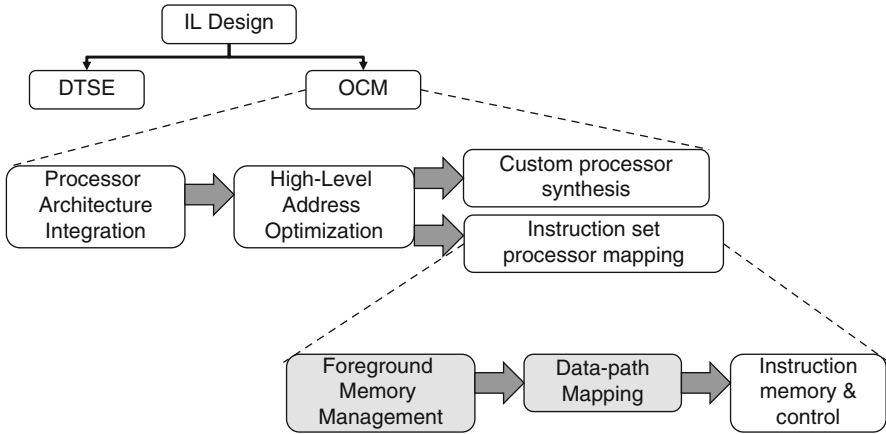


Fig. 2.14 Design steps of instruction layer abstraction for OCM

2.5.2 Operations Concurrency Management

The sub-stages of the OCM stage described in [21] are depicted in Fig. 2.14. The first sub-stage of the OCM, namely the Processor Architecture Integration (PAI), is dedicated to the internal structure of the heterogeneous super-processor, which is decided in terms of individual homogeneous processor cores, e.g. instruction-set processor or custom processor cores, and the threads to be executed in the different processor styles are statically partitioned including decisions related to their communication and synchronization.

Then, the optimization of the addressing of the memory accesses to the array elements based on the result of the Processor Level DTSE is performed. The derived custom memory data layout includes complex addresses, which increase the overhead in the address generation unit and the memory accesses. The address optimization sub-stage is dedicated to the reduction of the complexity of the addresses and provides the final address sequence for the array data to the processor mapping, which is responsible for the actual hardware mapping for the address generation.

Then, the hardware mapping for all the application graph behavior is performed for the different processor styles. The processor mapping step should provide as a result a processor structure that is capable of executing both the address generation operations and the application arithmetic and logical operations. We will work this out in particular for the instruction-set processor style. Then, the instruction-set processor mapping sub-stage takes as an input the decision of the processor style and decides over the foreground memory organization, the data path execution mapping and the Instruction Memory/Control bits.

The foreground memory management step is dedicated to the local memory used to store the scalars or the individual array elements accessed from the data path.

In the foreground memory management step, if the assignment of the scalars is over-constraint, then either spill to background memory is applied and the excess data are stored in the free space of the background data memory, or the foreground memory size is increased or the available cycles are increased up to the point where the constraints (e.g. timing deadlines) are met. In the foreground memory management step, platform independent and platform dependent transformations are applied, which can still modify the Control and Data Flow Graph (CDFG) graph of the application. Then, the ordering and the assignment of the scalars, which can be decided in this step, are performed on the heterogeneous memory resources of the foreground memory. If memory spilling is applied, several scalars have been decided to be stored in the background memory. Then, the background memory data layout and access ordering of the array data in the DTSE stage is updated to the final background memory access and data layout of both the arrays and the scalars. Since the array data are the most dominant part of the background memory and the communication overhead, the result of the DTSE, the address optimization and the Processor Architecture Integration (PAI) communication (sub)stages are not modified. Hence, the result of the foreground memory management step regarding the background memory and the communication is augmented with the results of the previous steps. The data path mapping is dedicated to the allocation of the primitive operators, the scheduling and the assignment of the operations over the primitive operators, the potential pipelining and parallelization of the data-path. The foreground memory management step and the data-path mapping step will be discussed in more detail in Chap. 9.

2.6 Conclusions

In this chapter, we have presented the principles of the reusable DSE methodology to develop scalable and near-optimal DSE frameworks for large and complex design problems. The problem under study is divided into a complete set of sub-problems connected with uni-directional propagation of constraints, i.e. a framework. The completeness and constraint propagation of the framework achieves scalability during exploration and near-optimality. We have described the process of applying the DSE principles to create the corresponding DSE framework with the partitioning and the ordering of the subclasses. Then, we have described how the derived DSE framework can be used to provide an efficient exploration of the solutions for (partial) instantiation of the framework based on the restricted context. Finally, we have summarized the unified system design meta-flow which is based on the uni-directional constraint propagation and we have defined the steps which form the starting point and context of the work in the book, i.e. the intra-signal in-place step and the instruction-set processor mapping step.

Scalable and Near-Optimal Design Space Exploration
for Embedded Systems

Kritikakou, A.; Catthoor, F.; Goutis, C.

2014, XVII, 277 p. 80 illus., 2 illus. in color., Hardcover

ISBN: 978-3-319-04941-0