Karsten Albers
Matthias Büker
Laurent Dieudonné
Robert Hilbrich
Georgeta Igna
Stefan Kugele
Thomas Kuhn
Maike Rosinger
Raphael Weber

9

Optimal Deployment

The SPES XT modeling framework combines multiple modeling viewpoints and offers an effective means for modeling and exploring various embedded systems designs. One crucial design step is to deploy the logical architecture to the technical representation of the physical system constituting the embedded system architecture. Since various system requirements and design goals impact the development, we describe several methods and techniques for mapping, analyzing, and validating different deployment solutions. Finding an optimal deployment is a difficult process due to a large number of often conflicting requirements that should be satisfied. The requirements address resource, timing, safety, deployment, economical, and regulatory aspects. This chapter presents design space exploration methods, including their application to an automotive example.

K. Pohl et al. (eds.), *Advanced Model-Based Engineering of Embedded Systems*,
DOI 10.1007/978-3-319-48003-9_9

9.1 Introduction

Deployment

The quality of today's embedded systems — for example, in vehicles, airplanes, or automation plants — is heavily influenced by their overall architecture. This architecture subsumes the logical and technical structures of an embedded system as well as the mapping of the logical to the technical structure, which we call deployment. In the scope of this chapter we address challenges such as: On which target execution unit do we execute *which* software component as a task? *Which* sensor/actuator shall be connected to *which* distributed I/O unit? Finding a feasible deployment (meeting all constraints) may be difficult; designing an optimal embedded system is an even bigger challenge.

Optimality

Deployment is merely the mapping of a logical architecture to a hardware architecture that satisfies all constraints (e.g., resources, timing, safety, etc.). *Optimal* deployment considers the minimization/maximization of certain objectives such as costs, weight, etc. In order to solve the optimal deployment problem adequately, we have to define a notion of optimality (i.e., optimization objectives). In practice, there are usually many objectives, some of which are conflicting.

Design space exploration

Typically, during system development — regardless of the application domain — engineers face many potentially conflicting requirements. For example: the goal to develop an airplane with a better ratio of kerosene consumed per aircraft passenger is more expensive due to modern materials or jet-propulsion engines. The set of *possible* solutions is called the design space, whereas the set of feasible solutions is called the solution space. The goal of *design space exploration* (DSE) is to find the best compromise (trade-off) amongst the different goals (optimization objectives) in the solution space [Eisenring et al. 2000]. This results in a multi-objective optimization problem [Hamann et al. 2006].

Optimization methods

Traditionally, this problem was approached with a lot of experience and gut feeling [Diebold et al. 2014]. Of course, on the one hand, the influence of engineering experience should not be underestimated; on the other hand, however, these solutions are — if at all — optimal by accident. Experience has taught us that it is necessary to recheck all constraints (e.g., robustness, safety, maintainability, and so on) whenever we change a design. Therefore, methods leading to an optimal architectural design are desirable.

9.1.1 Challenges regarding Optimal Deployment

Prerequisites

For an optimal deployment, the main prerequisite tasks include capturing and formalizing the following: the needs of the system functions to be deployed, the capabilities of the execution units that can potentially be used for the targeted architecture, and the optimization objectives which help to decide which design is *better*. These properties drive the deployment problem. They have to be expressed in a way that allows them to be compared, and they should be capable of being processed by machines in order to automate some design decisions.

Understanding engineering experience

Another challenging aspect is to understand, capture, and again formalize the ideas and principles employed by the experienced system engineers to design valid system architectures. In particular, the rationale behind certain design decisions should be extracted and converted into computationally solvable problems. For example: use the same hardware architecture as a blueprint for the next generation.

Trade-offs and budgets

Trade-offs are necessary to decide which solution is better in a specific context. Optimization objectives are measured with a metric and are usually expressed with phrases such as "the more, the better", or the opposite "the less, the better", for example, when considering costs. A dedicated threshold is not essential for an objective, but often there is one — for example, a maximum weight that must not be exceeded but the further below that weight "budget", the better. As stated above, the objectives often conflict, which means that improving one objective worsens another (e.g., weight and costs).

Deployment constraints

In contrast to optimization objectives, constraints describe properties that have to be fulfilled. Here it is sufficient to just reach the required constraint — there is no need to do better than that. Constraints can also be optimization objectives and act as a threshold that marks the border from an invalid solution to a valid one and towards an even better one. Other constraints can be given to define possible target architectures consisting of execution resources that may be consumed if some functionality is deployed to them. There are also mapping constraints derived from requirements such as segregation, dissimilarity, or other safety properties.

Challenge: exploring the design space

All of these properties, along with a sound understanding of how they influence each other, constitute the design space. The main challenge addressed in this chapter is how to explore this vast design space in an efficient and effective way to find feasible and

optimized solutions. This results in two DSE challenges which we address in this chapter:

1. Find an optimal design
2. Find a feasible design

No ideal solution

Since the problem of finding feasible and optimal designs itself is too difficult to solve in an ideal way, there are several approaches which support the designer in finding an optimized deployment as well as a feasible design.

9.1.2 Methodological Framework

Generic DSE framework

To address the prerequisite and efficiency challenges from Section 9.1.1, and in order to offer a common interface for concrete DSE techniques, here we integrate and apply the design space exploration approach in the typical development process for software-intensive embedded systems. We define a *generic DSE framework* (see Fig. 9-1) related to the SPES XT modeling framework described in Chapter 3 and Chapters 6, 8, and 10.

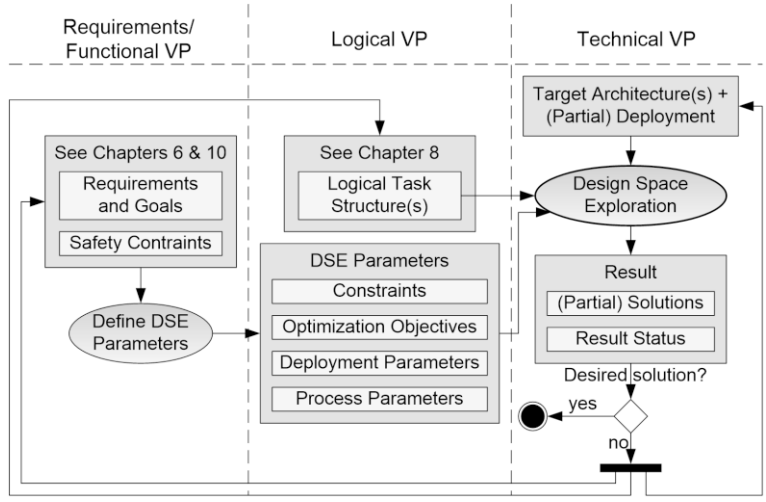

Fig. 9-1 *Overview of the generic DSE framework*

DSE context

The goal of this framework is to allow the integration of a set of specific DSE techniques (presented in Section 9.3) by offering a common interface to address the two main DSE challenges (find an optimized solution and find a feasible solution). Fig. 9-1 also gives an overview of the relationships to other challenges.

Relationship to the requirements viewpoint

The requirements viewpoint specifies goals, requirements, and constraints which need to be refined into DSE constraints, optimization objectives, deployment parameters, and process parameters during the design phase if they are relevant for the DSE process. This usually also includes a more concrete specification and formalization of requirements such that the requirements can be processed by an automatic DSE method. For example, a safety requirement might lead to a DSE constraint stating that certain tasks are not allowed to be allocated to the same computing resource to avoid a single point of failure (dislocality). In addition, DSE constraints may be derived from requirements that address different aspects such as timing, safety, security, power, and also more abstract goals such as reusability, verifiability, and testability. For the latter in particular, it is important to note that these goals have to be specified formally to enable direct support by an automatic DSE method. For example, this may be achieved by defining a respective metric. Typical goals such as "minimize system costs", which refer to one or several concrete optimization functions, utilize cost metrics whose values shall be minimized as, for example, costs of electronic control units and costs of cables.

Contributing methods

The modular safety assessment (see Chapter 10) may contribute to the process of defining DSE parameters by deriving safety constraints from requirements based on the results of safety analysis techniques such as *fault tree analysis* (FTA) or *failure mode and effects analysis* (FMEA). The early validation of engineering artifacts (see Chapter 6) may also play a role in this process step — for example, by offering techniques for formalizing textual requirements, which is an important step in deriving constraints needed by automatic DSE methods.

Process parameters

Process parameters may be defined in order to control the automated parts of the DSE process. These parameters may also be derived from a high-level requirement or be defined in this process step. An example of a process parameter is a timeout for the DSE process which restricts the time spent searching for an optimal solution to a maximum of 5 hours after which the optimization process shall be aborted.

User support

The process step *Define DSE Parameters* covers all steps needed to decide which requirements influence the deployment process. This also includes refining the affected requirements to a sufficient level of detail to be able to apply the DSE process. This may be done either manually by the user or supported by tools, such as wizards. The consideration of user support methods is not covered

in this chapter but there are some works that address this topic — for example, [Rosinger et al. 2015] and [Rosinger et al. 2014].

Logical components and their communication

In order to apply the DSE methodology, a logical architecture is necessary to identify the deployable entities in terms of logical tasks and signals (logical task structure). In addition, the communication (signals between tasks) has to be modeled explicitly for the DSE methods because those signals have to be allocated to communication resources such as buses if the respective tasks are not allocated to the same computing resource. In addition to the task structure, a set of non-functional properties also has to be defined — for example, concerning the activation behavior and resource demand of tasks and signals. This is important for the second DSE challenge of finding a feasible solution.

Task structure

Generally, the logical task structure is derived from a functional model. In this transition step from the functional to the logical viewpoint, the designer has to decide which functions should be bundled into one logical component and which are the atomic logical components (tasks). See Chapter 8 for more details.

User-guided process iterations

The execution of a single DSE process run does not necessarily lead to a valid and complete solution that is also satisfactory for the user. In practice, therefore, such a process will typically be repeated several times until a solution is found that fits the user's needs. In Fig. 9-1, we outlined some typical process iterations that detail how the DSE process may be repeated. Another important step is the evaluation of the results of one process iteration to assist the user in assessing the proposed solutions and deciding how to continue the process. Again, there are methods and wizards to support the user in this step (see [Weber et al. 2014] and [Rosinger et al. 2015]), although this issue is not addressed in this chapter.

Finishing and restarting the DSE process

Based on the evaluation results, the user may choose one of the proposed solutions best fitting his needs. If there is such a solution, the process is finished and the chosen deployment is returned as the result. If there is no solution, the user may modify the technical target architecture(s) including the deployment (e.g., based on the intermediate solutions and evaluation results), adjust the requirements and goals, or directly modify the DSE parameters and then start the DSE process again.

Refinement of input artifacts and DSE parameters

There are several possible intervention points. For example, the user may decide (based on an intermediate solution of the previous DSE process run) that the specification of goals was not complete and that another goal needs to be added — for example, a requirement to minimize the cable length of the DSE system. As a next

step, the user may also add or refine constraints and process parameters or adjust the optimization objective. Here, for example, the concrete characteristic of the optimization function may be adjusted or the prioritization or rating of different optimization objectives may be changed. In addition, the task structure and target architecture may be modified. For example, the allocation of a certain subset of tasks that have been successfully deployed in a previous solution may serve as the initial deployment for the next process iteration. Only the remaining tasks not yet deployed are then considered by the DSE.

Relationship to the SPES XT tool platform

The SPES XT tool platform (see Chapter 14) comes into play once the DSE methods described in this framework have been implemented and the subjects of tool interoperability and exchange of model artifacts become relevant. The generic DSE framework already defines an interface for input and output artifacts of DSE methods, which is put in more specific terms in the method descriptions of Section 9.3 by using a common template. These templates provide the basis for defining specific service descriptions for the SPES XT tool platform.

9.2 Extensions to the SPES Modeling Framework

Abstract interface of input and output artifacts

This chapter details how the SPES modeling framework has been extended to foster optimal deployment in the SPES XT modeling framework. In the center of Fig. 9-1, the abstract interface in terms of input and output artifacts of the generic DSE process is depicted denoted as *DSE Parameters*, *Logical Task Structure(s)* and the technical *Target Architecture(s)*, possibly including a *(Partial) Deployment*. The output of this process is a result that contains a set of *(Partial) Solutions* and a *Result Status*. In the following, we will explain the artifacts and describe their role in the process of finding an optimal solution.

Logical task structure

A mandatory input for the DSE is a logical task structure consisting of communicating atomic components denoted as tasks that are to be executed on the available hardware resources. In general, the task structure is derived from a functional model by deciding which functions should be bundled into one task. This task structure also contains information about the communication between each task — how often communication takes place and how much information is exchanged. The logical task structure is completely reflected by the logical viewpoint of the SPES modeling framework.

Target architecture

Another mandatory input artifact for the DSE is a technical hardware architecture (denoted as the target architecture) which models the resources to which tasks should be deployed. A target architecture may contain a (partial) deployment relationship referring to tasks already deployed. This is because experience shows that in some domains, embedded systems are often developed incrementally. This means that systems are built on top of an existing model. In order to reuse such existing models, a target architecture may consist not only of a hardware architecture which offers certain services and interfaces, but also of tasks already deployed which represent the functionality of the existing design. The technical hardware architecture can also be completely represented with existing SPES modeling framework artifacts.

DSE parameters

The remaining inputs are summarized as *DSE parameters*. They represent constraints, optimization objectives, deployment parameters, and process parameters; there may also be additional classes of parameters not explicitly listed here. *Constraints* specify which DSE system configurations are considered to be valid, thus restricting the design space. *Optimization objectives* define the attributes that should be considered during the DSE to rate solutions in order to find optimal deployments. In contrast, *deployment parameters* do not define what should be optimized but rather describe the parameters that may be set or changed during the optimization process and thus the degree of freedom for the DSE method. For most DSE techniques, this is primarily the deployment itself. We also define *process parameters* which capture all parameters that influence and control the automated parts of the DSE process itself without relating to the system under development.

DSE results

The output of the DSE process consists of a set of valid (but potentially incomplete) solutions and a result status. Solutions are valid (correct) if and only if all given constraints are satisfied. Completeness means that all tasks of the given task structure are allocated to the technical architecture. If there are no valid (partial or complete) solutions at all, the DSE may either return an empty solution set or the unmodified technical architecture as specified at the start of the previous DSE step. An incomplete solution may serve as an intermediate result and the DSE process may iterate with adjusted inputs and parameters to obtain a better and (possibly) complete result. If valid (partial or complete) solutions exist, the DSE returns the best solutions for the optimal deployment problem w.r.t. the defined optimization objectives.

Additional DSE artifacts

In addition to the artifacts visible at the interfaces of the general

DSE framework, there are some intermediate artifacts which relate constraints and optimization objectives to the corresponding architectures (logical and technical). In relation to the SPES modeling framework, constraints and optimization objectives are a specialization of requirements with an explicit relationship to metrics which in turn refer to parameters. Fig. 9-2 gives an overview of the additional DSE artifacts and their relationships to SPES modeling framework artifacts.

Metrics

A metric defines a function which combines multiple properties of a system (referred to as the relationship to abstract parameters) into a characterization. An example would be the combination of multiple cost values of subcomponents, where the sum of all costs denotes the overall system costs. Other metrics may be more complex. Extensions may therefore be defined to denote more complex metrics.

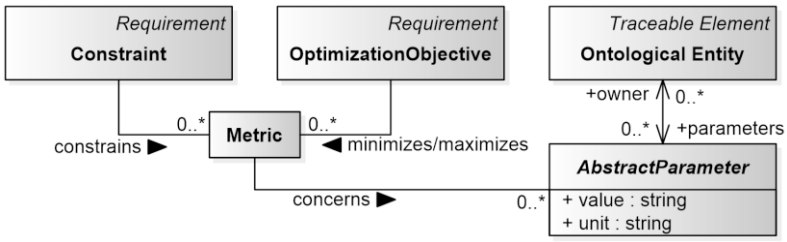

Fig. 9-2 *DSE-specific SPES modeling framework extensions*

An abstract parameter is supposed to be a superset of all parameter artifacts. It defines the relationship to ontological entities but leaves the actual property or parameter definition undefined. The *value* and *unit* properties, however, must be present so that a metric may directly refer to them. Abstract parameters may be refined into system parameters referring to system properties or measures. An abstract parameter may also represent a process parameter which constrains the scope or runtime of a design space exploration process as mentioned above. With the abstract parameter and the metric artifacts, the transitive relationship between constraints and optimization objectives and ontological entities (which may represent the logical or technical architecture) is complete. The following section will detail methodological building blocks for addressing the optimal deployment challenges.

9.3 Methodological Process Building Blocks

In the following, the two main DSE methods and suitable backend techniques are described. These methods and techniques constitute the building blocks for our DSE methodology ("method toolbox").

Two DSE steps

To understand the techniques and how they contribute to the optimization of deployments, it is important to explicitly define the two steps which directly correspond to the two DSE challenges above. There are different ways to approach this. Usually, a DSE method tries to find a "good" starting point that satisfies all constraints (is feasible) and then optimizes it in subsequent steps. Some DSE methods do not try to find this starting point and instead require the engineer to provide it. In either case, due to the non-linear shape of the solution space within the design space, it is impossible to assess how close to the optimum these starting points are. Therefore, in general, there are two steps:

1. Synthesize optimized solution
2. Find feasible solution/validate solution

Different techniques

Every DSE technique implements these steps in a different way. Because each technique is required to be very efficient (the goal is to test as many solutions as possible for feasibility and optimality), these two steps are closely integrated. Fig. 9-3 gives an outline.

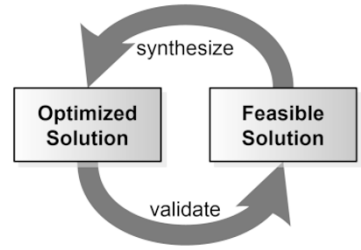

Fig. 9-3 *Two typical iterative DSE steps*

Diverse constraints and optimization objectives

The set of possible constraints and optimization objectives along with the metrics are very diverse. Hence, each technique performs differently depending on the complexity of these steps. For example: synthesizing a feasible solution for a priority-based scheduling policy may be more difficult than for a time slot-based scheduling policy. In the following subsections we describe the above-mentioned two steps in more detail along with the techniques that address them.

9.3.1 Finding an Optimized Spatial Deployment

Optimization without schedulability

The goal of finding an optimized spatial deployment is to explore the design space to find an optimized and feasible (to some degree) solution. The feasibility explicitly excludes timing consideration since checking the feasibility of timing parameters requires a schedulability analysis which itself is too complex to be part of this method. At most, we can cover abstract time resource budgets to guide the DSE. This is also why the optimization may have to abstract certain objectives in order to find a solution.

Optimized mapping respecting simple constraints

As stated above, the DSE is a very difficult challenge and it is so complex that for each additional optimization objective, the size of the design space grows exponentially. This is why we first look for a spatial deployment and then find out whether the solution is completely feasible. A spatial deployment denotes a mapping of tasks to a target architecture where each required property of a task is satisfied by a provided property. For example, if a task T_1 requires a direct input from a sensor S, all execution units with a sensor of type S are valid deployment targets for T_1. Another example: if another task T_2 requires a certain execution unit-specific extension E, it may only be deployed to execution units with an extension of type E. Depending on the complexity of these constraints, it may be easy to find an exact solution (optimal with respect to the accuracy of input properties). However, if the constraints generate a rather large solution space, it may be a good idea to use heuristics which only deliver some parts of the solution space. Ideally, these are only parts for which we know (from engineering experience) that they contain mostly optimized solutions.

9.3.2 Finding a Feasible Schedule

Validating deployment-dependent constraints

After finding a deployment that meets optimization objectives and simple constraints, the correct timing behavior has to be guaranteed. There are a number of techniques to ensure a feasible timing behavior. This is important in particular for safety-critical systems that only work properly if the right information is provided at the right time. Control algorithms are an example of periodically updated sensor data which has to be processed within a certain time limit in order to provide actuators with the necessary data to control environment properties. Since these systems rely on some feedback from their surroundings, this results in distinct timing requirements for the processing part of the chain. Finding a feasible schedule which meets these timing requirements is thus very im-

portant. In some cases, proof that these time limits are met is required. Depending on the degree of distribution of the whole system and on the scheduling policy, it may be difficult to integrate the complete schedulability analysis into a deployment optimization method. Hence, we distinguish between finding an *optimized spatial* deployment and a *feasible schedule.*

Scheduling strategies

In the context of a feasibility check there are different classes of techniques that cover different scheduling strategies:

a) Utilization-based (e.g., estimation)
b) Static scheduling (e.g., timetable-driven)
c) Dynamic scheduling (e.g., fixed priority)

Comparison of strategies

Since techniques of class a) perform a more abstract and approximate estimation of timing properties, their preconditions are typically not as strict as the preconditions of techniques from classes b) and c). Thus, it is possible to perform these techniques before applying techniques from classes b) and c). In those cases, the techniques from classes b) and c) may benefit from the previous application of a technique from class a) because certain design decisions were already taken, resulting, for example, in additional constraints. The techniques from classes b) and c) cannot typically be combined with each other consecutively because they address different scheduling policies. In addition to the consecutive combination of techniques, where the compatibility of interfaces has to be considered, it is also possible to repeat complete technique sequences. However, this only makes sense if new information was derived from the previous iteration. There are a number of techniques that implement these methods; they are explained in the following.

9.3.3 DSE Techniques

Automatic Deployment

Purpose/goals

The automatic deployment technique is based on an *integer linear program* (ILP) optimization approach. The problem is transformed into a representation as a system of mixed integer linear equations. This equation system is solved by an ILP solver taking defined optimization objectives into account.

Preconditions

The ILP optimization approach requires a conforming input model that defines logical tasks and communication dependencies. Furthermore, the target architecture consists of runtime platforms and communication links. To retain flexibility and enable rapid

adaptation to new network types, link capacities are specified using an additive model. Additional project-specific resources such as memory and CPU capacities may be specified for target platforms.

Supported constraints and optimization objectives

Explicit constraint specifications permit developers to control the output of the optimization algorithm. Currently, basic Boolean constraints (and/or/not/implications) as well as set-based constraints (forAll and atLeastOne) are supported. Recursive use of these constraints enables the definition of more relevant deployment constraints that limit the solution space for the ILP algorithm. Examples of constraints include the following:

- Function A must be deployed to computer X
- Function B must be deployed to a computer with attribute Y=Z
- Functions A and B must be deployed to different types of computers

Constraint definitions are usually provided by system designers as part of ILP definition files. Optimization criteria request that particular variables take maximum or minimum values. By default, only one variable is optimized. If multiple variables are to be optimized at the same time, a target variable that combines those variables using constant weight factors has to be provided by developers.

Outputs

The ILP optimization algorithms return a deployment matrix that defines whether a function is deployed to a particular target environment or not. This technique also states whether a physical entity is used or not.

Method description

The mapping of a deployment problem to an ILP description is supported by a generic ontology that captures all relevant properties of the input models. It serves as the target for a tailored model-to-model transformation, for example, from domain-specific languages.

The transformation of conforming model elements into an ILP format enables the definition of optimization criteria and constraints using the native language of the CPlex Optimizer[3]. In this way, basic domain knowledge that is invariant to deployment projects is provided. Furthermore, it is possible to add tailored transformations from the input model that generate further, project-

[3] http://www.ibm.com/software/integration/optimization/cplex-optimizer/

specific constraints that enable detailed control of the deployment process. The overall process of this technique is depicted in Fig. 9-4.

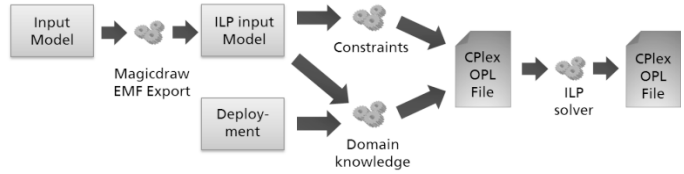

Fig. 9-4 *ILP-based deployment process*

Tool support

Currently, the ILP-based deployment algorithm is realized as an Eclipse-based tool that transforms input models into ILP description files which are read by the commercially available CPlex Optimizer.

Deployment under Consideration of Timing Optimization

Purpose/goals

The optimization of the timing behavior is important for a robust and reactive system. Even with a fixed deployment, parameters such as priorities, execution orders, or offsets can require an optimization to keep the necessary timing constraints. For the optimization, meaningful feasibility evaluation methods are required.

Preconditions

A precondition for optimizing the scheduling parameters is a fixed deployment, otherwise the logical tasks and the planned architecture are required. The task model can, for example, consist of AUTOSAR software components with their executable entities which should be distributed to an architecture consisting of single or multi-core processors.

Supported constraints and optimization objectives

Timing constraints such as end-to-end worst-case response times (for timing paths), reaction times, local response times, utilization limits, constraints on the execution rate, or the data consistency (e.g., exclusion of data loss) and more are supported. Optimization objectives can be, for example, the reduction of the average response time or a balanced utilization of resources.

Outputs

An output of the optimization is one or more complete models, including the deployment and scheduling parameters which were selected. In addition to the model, detailed simulation and analysis results of the behavior of the optimized model are available, including the evaluation of the timing constraints.

Postconditions

The postconditions are that the resulting model is consistent and can be simulated or analyzed and that the timing requirements are satisfied (depending on the concrete problem).

Method description

The timing optimization consists of a set of extendable optimization methods supported by methods for the timing simulation and

schedulability analysis. The scheduling parameters must be optimized to achieve meaningful and precise information on the quality of a candidate system in terms of the timing behavior. The real achievable response times and other timing objectives can only be calculated for systems with optimized scheduling.

Timing optimization

The proposed timing optimization can use round-based metaheuristic optimization approaches such as simulated annealing [Kirkpatrick et al. 1983], tabu search, and others. To allow fast and reliable execution, the optimization uses the detailed evaluation results from the simulation and schedulability analysis of one candidate solution to suggest further optimizations, promising modifications, or candidate solutions. The optimization is guided by detailed evaluation results.

One possible optimization method is the distribution of functionality to a target architecture (deployment). Several very different problems can be summarized under this method depending on the specific models and goals — for example, the distribution of AUTOSAR software components or the static distribution of tasks on a multi-core processor.

Candidate solutions based on a fixed deployment still have several degrees of freedom in terms of scheduling parameters such as priorities, offsets, orders of executable entities, etc. Good values are required here to allow a meaningful evaluation of the candidate solutions, a realistic statement on the fulfillment of timing requirements, and to realize a reliable and robust system in terms of timing.

Tool support

For the timing optimization technique described in this section, an implementation framework is available which allows quick adaptation of the optimization process to specific challenges and the addition, for example, of other constraints. It is realized in the tools chronOPT (see *http://www.inchron.com/tool-suite/tool-suite.html*) from INCHRON which utilizes chronSIM (for simulation) and chronVAL (for schedulability analysis) for evaluation purposes. The goal of this tool suite is the development, investigation, and optimization of the timing behavior of embedded systems on several levels of the design process.

Deployment Based on Correctness by Construction

Purpose/goals

This engineering technique is intended to support system and/or software architects in achieving spatial and temporal deployments. The method is based on the concept of achieving correctness by applying formalized approaches for the construction of artifacts

using *constraint programming* [Chapman 2006], [Hall and Chapman 2002].

Preconditions

Constraint programming requires a precise and formal problem specification. The input distinguishes between logical and technical components in order to separate the problem specification from the solution description. Software applications are instances of logical components and hardware resources are structured into technical components. The function network is expressed as a logical task structure with a representation of the requirements of each software application (hardware resources, safety, timing, etc.). Each software application consists of at least one thread. All threads of the same software application have identical hardware requirements. Relationships (e.g., dislocality) always apply to all parts in a parallelized software application. A hardware description must adhere to a tree topology, where the cores are the leaves of the tree; the processors are one level higher; above these are the boards, etc.

Supported constraints and optimization objectives

The optimization objectives focused on in this technique are the number of execution units and an optimal scheduling of the software application's threads on the selected hardware resources. The technique deals with quantitative and qualitative constraints, such as capacity (memory and CPU), safety and criticality level, dissimilarity, dislocality, communication proximity, I/O type, and timing requirements (period, jitter, delay, timing synchronization).

Outputs

The technique produces different valid spatial and temporal deployment solutions. A valid spatial deployment is represented by a mapping of software applications to hardware resources (on cores, but also as occupation and/or consumption of I/Os, memory, communication and bus bandwidth, etc.). A temporal deployment is a scheduling of each application thread according to the timing and performance requirements. It includes (amongst other things) time slices, computation of a hyper-period, definition of tolerable jitter, additional delays for initialization tasks, execution, etc. The different spatial deployment variants are submitted to the system/software architect who compares and sorts them by applying different evaluation criteria based on metrics. Several metrics are predefined, such as *uniform cCore load distribution*, *max. freecore capacity*, etc., and new ones can be also defined by the architects. The temporal deployment is calculated only on the chosen spatial deployments.

Method description

In order to facilitate and simplify the input of a mapping specification for the user, textual domain-specific languages were developed. The grammars of these languages describe the correct syntax

for mapping and scheduling specifications. These languages constitute a "bridge" between the domain-specific context of a mapping/scheduling challenge and the technical context of a constraint solver. The specification for a mapping/scheduling problem is subsequently transformed from the internal data model into a constraint satisfaction problem (CSP). If there are solutions for the specified CSP, these solutions will be used to enrich the internal data model with allocations of software components and utilizations of hardware resources, such as processing cores, memories, or IO adapters. More details are presented in [Hilbrich and Dieudonné 2013].

Tool support

The tool PRECISION PRO was developed by Fraunhofer FOKUS to evaluate and illustrate the *correctness by construction* technique. It was developed on an Eclipse 4 Rich Client Platform. It is based on the Eclipse Modeling Framework (EMF) and the OSGi Implementation Equinox. The constraint solver *firstCS* from Fraunhofer as a form of a Java library has been used to perform constraint programming [Hofstedt and Wolf 2007], [Wolf 2006].

Deployment and Scheduling Synthesis Using SMT Solving

Purpose/goals

The search for solutions within the DSE parameter space can be automated using highly efficient satisfiability solvers. In this section, we present a DSE technique based on *satisfiability modulo theories* (SMT) [Barret et al. 2009] which, depending on the DSE constraints, can be used to synthesize valid schedules for a given deployment or valid deployments including schedules for a given target architecture, or to identify a suitable platform and the corresponding deployments and schedules.

Preconditions

The input for this technique consists of a set of tasks, a target architecture, a set of optimization goals, and, optionally, a partial deployment. The set of tasks is annotated with precedence constraints and further attributes such as their individual resource claims or safety classifications. The technical architecture consists of a topology of computing units (e.g., cores of a multi-core platform) connected by communication and storage resources (e.g., buses, shared memory, etc.). Design parameters can also be included — for example, safety levels or energy constraints. In addition, the designer may add constraints obtained from previous system designs regarding permitted or forbidden mappings of tasks onto computing units. The method uses a time-driven approach — that is, the user should specify timing properties that represent the worst-case execution time (WCET) for each task. A WCET of a task represents the

worst/longest time in which the task should be executed. Each computing unit can process only one task at a time (single core). Moreover, a task cannot be interrupted during execution, which means that task pre-emption is not allowed. Communication between tasks deployed on separate computing resources is ensured by signals sent via a shared bus. The user can also specify a worst-case transmission time for each signal.

Supported constraints and optimization objectives

The technique supports different optimization objectives for increasing overall system performance in terms of time and resource usage (e.g., minimize the end-to-end latency, minimize the number of computing resources). In general, this enables a Pareto-optimal solution of the system design.

Outputs

The technique generates valid and complete deployments which respect the precedence order between tasks and fulfill all timing and resource constraints. Each deployment includes a set of feasible schedules. Furthermore, optimal solutions are obtained when the search terminates within the time limits defined by the user. Each schedule provided for a deployment is unique for the allocation; if two schedules have equal end-to-end latencies, there is at least one task or signal that is scheduled differently in the two schedules.

Method description

The inputs of the deployment problem are formalized as logical and arithmetic formulas encoded using the SMT-LIB2 language [Barret et al. 2010]. An SMT solver is then used to synthesize valid and complete solutions. To find a deployment based on SMT solving, we generate a set of formulas over variables that represent the start time and end time of tasks and signals. The precedence order between tasks and signals is transformed into constraints between these variables. In addition, we specify constraints between these variables to prevent time overlaps for tasks allocated on the same resources. Furthermore, we also generate formulas over variables representing the mapping of each task or signal to the platform resources. If the mapping is known at the beginning of the deployment process, the corresponding variable is bound to a fixed value, ensuring that any valid solution will preserve it. Otherwise, the SMT solver chooses a valid value for this variable.

DSE evaluation options

To find one solution, the SMT solver is called only once. However, to find optimal solutions, we apply either a binary search algorithm or a guided-search algorithm. The latter algorithm requires that, during each round, the SMT solver is queried for a shorter schedule than the one obtained in the previous round.

With this technique, the DSE phase can result in refining the deployments between tasks and the computing units, or in adjusting

the target architecture or the task structure. The overall process is implemented in an architecture wizard enabling provision of a step-by-step refinement of the system design.

Tool support

AutoFOCUS3 [Kondeva et al. 2013] implements this method. Moreover, each viewpoint of the SPES XT modeling framework is represented as a separate module in the tool. AutoFOCUS3 is implemented on the Eclipse Rich Client Platform and employs Z3 [de Moura and Bjørner 2008] for SMT solving.

Two-Tier Iterative Design Space Exploration

Purpose/goals

This technique finds valid and near-optimal deployments of a set of currently undeployed software tasks onto a given base target architecture to which other tasks might have already been allocated.

Preconditions

As input, this technique requires a target architecture that consists of a global TDMA scheduled backbone bus (e.g., FlexRay) which connects several clusters, each of which contain processing units which in turn are connected by a local priority-based bus (e.g., CAN). Each cluster thereby contains one processing unit which acts as a gateway to other clusters. The rest of the input artifacts are the same as defined in the general DSE framework.

Supported constraints and optimization objectives

Supported constraints include timing constraints (end-to-end and local deadlines) and deployment constraints. Supported optimization objectives include monetary hardware costs, power consumption, the number of processing units used, the weight of hardware resources, and the volume/size of hardware resources.

Outputs

One output is a model which consists of a hardware architecture resulting from applying a (possibly empty) set of modifications of the set of allowed modifications to the base hardware architecture. Furthermore, the model consists of the task structure that was input into the technique and a (partial or total) deployment relationship. Another output artifact of this process is a result status detailing the status of the DSE process, that is, whether a timeout occurred, if the solution is partial or complete, if an error occurred, etc.

Postconditions

The deployment relationship is valid and is a refinement of the input deployment relationship, meaning that the allocation of tasks that were already deployed in the existing system is identical. Furthermore, the system is consistent and can be scheduled.

Method description

To cope with the complexity, the DSE process is divided into two steps (global/local analysis). The first optimizes the system level and the second optimizes each hardware subsystem separately.

Global analysis

The goal of the global analysis is to distribute the set of unallocated tasks among the different subsystems of the hardware archi-

tecture of the existing system. The computational capacity of each subsystem is calculated by aggregating the computational capacities of its processing units. On the global analysis tier, as a notion for computational capacity we use the maximum fraction (share) of processor time a task would need from the whole processor time (it is assumed that each processor can provide 100% computational capacity) when running on a processor. Based on this metric, the costs needed to place a set of tasks on a subsystem is estimated. The resulting global pre-allocation serves as input for the local analysis.

Local analysis

The local analysis is performed separately for each hardware subsystem. It allocates the tasks that were mapped to the subsystem to the processing units of that subsystem but without exceeding the subsystem's hardware cost limit that was predicted by the global analysis. In this step, an exact characteristic of computation capacity is used which takes into account that each allocation has to satisfy a full-blown schedulability analysis to be considered feasible. If there are tasks that could not be deployed to a subsystem, they remain undeployed and a so-called "backtracking" phase is started. The result of the local analysis step is a local deployment of tasks to processing units, which may be incomplete, denoted as partial deployment.

Backtracking

In the backtracking step, tasks that could not be deployed to processing units within the proposed subsystem under the given cost limit are distributed to the hardware subsystems again by re-applying the global analysis. Here, only the undeployed tasks are considered, while the allocation of already deployed tasks remains unchanged.

Tool support

There is an Eclipse/Papyrus-based UML/SysML implementation for the specification of the DSE-specific modeling elements and it provides an input for the DSE backend tool called Zerg [Thaden 2013]. Zerg itself translates the global and local analysis parts into different backend formats (e.g., an MILP backend) which are solved by a respective solver (e.g., Gurobi or CPlex). Depending on the results and the configuration of Zerg, the backtracking step triggers another DSE execution. More details on the global/local analysis and the backtracking can be found in [Büker et al. 2011] and [Büker et al. 2013].

Flexible Deployment Respecting User Objectives and Constraints

Purpose/goals

The goal of this approach is to give the user of the deployment tool as much freedom as possible in terms of *what* to optimize and

which constraints to respect. Therefore, rather than specifying how to obtain an *optimal* solution, the approach specifies *what* it should look like. This is achieved by providing a flexible solver infrastructure in combination with the domain-specific language *SAOL* (system architecture optimization language) [Kugele and Pucea 2014]. It is intended to easily but yet concisely specify both objectives and constraints, whereby special focus is placed on simplicity and usability by engineers rather than mathematicians.

Preconditions

Therefore, the engineer has to specify goals, constraints, and orders using SAOL. Orders are used to rank solutions according to the user's priority and to specify compatibility relationships (e.g., DAL classification). Moreover, an enriched model with deployable entities, be they software components (tasks, applications, etc.) or hardware components (e.g., field sensors, actuators, pumps, etc.) and a technical model of the deployment targets (e.g., ECUs, DIOs, inverters, etc.) has to be given. This means that at least the attributes of the modeling artifacts that are referred to within objectives and constraints have to be given.

Outputs

The method returns a number (depending on the selected algorithm) of valid deployments — that is, only those that satisfy all the given constraints.

Postconditions

The methods provided guarantee that only valid solutions are returned. The solutions can be persisted within the model for further processing.

Method description

Through its generic architecture, the *system architect* provides a flexible extension of the solver infrastructure. Currently, several solvers and combinations of them are supported:

- *MOEA (multi-objective evolutionary algorithm)*: the NSGA-II algorithm used returns "optimized" solution(s).
- *IDE (iterative deployment enumeration)*: an SMT-based approach iteratively enumerates (not optimized) deployment solutions.
- *MGIA (modified guided improvement algorithm)*: MGIA computes Pareto-optimal solution(s) for a multi-objective problem.
- *MOEA + IDE*: MOEA uses a valid IDE-generated initial population.
- *MOEA + MGIA*: MOEA uses an optimal MGIA-generated initial population.

Tool support

The Eclipse-based tool *System Architect* provides powerful modeling support: a text-based editor with syntax highlighting and intelli-

gent text completion is accompanied by modeling support for all viewpoints of the SPES XT modeling framework.

9.4 Application to the Automotive Example

In the exterior lighting system and the speed control system described in Chapter 2, optimization is advantageous for several aspects. The available task structure has to be deployed to the available hardware resources. There are several variations for the hardware structure with some freedom in the selection and replacement of hardware components. Here, the method described in Section 9.3.1 can be used to select the appropriate hardware. For the deployment of tasks to the hardware architecture and the feasibility analysis, the second method explained in Section 9.3.2 may be used.

Ensuring certain timing constraints plays a crucial role for the optimization. Constraints on deadlines, activation periods, and execution demands force the distribution of the task structure to several hardware components. In the exterior lighting example, there are several timing requirements for the system which need to be validated.

All of the techniques described in Section 9.3.3 are suitable for considering timing properties, some of them using abstraction metrics on the utilization. Even if there is only a fixed hardware structure and deployment of software tasks to the hardware components, optimization may still prove beneficial. Most of the components have more than one task deployed to them. The scheduling of tasks on each resource can require optimization to achieve responsiveness and meet the (end-to-end) timing constraints of the system. Some techniques support static scheduling with others also supporting dynamic scheduling. For multi-core systems, this scheduling optimization might include a static assignment of tasks to different cores. One technique also allows separate optimization of scheduling parameters such as priorities.

9.5 Summary

This chapter focused on the SPES XT modeling frameworks methods to foster optimal deployment. During the design of software-intensive embedded systems optimizations of different natures occur. To support engineers during this design process, a methodological framework was presented. Different methodological building blocks along with their implemented techniques fit well into this generic framework, facilitating a seamless but yet flexible design of software-intensive embedded systems. To take optimization into account, several extensions to the SPES modeling framework were presented. The building blocks were classified and the application of the DSE method was illustrated using a case study from the automotive domain.

9.6 References

[Barret et al. 2009] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli: Satisfiability Modulo Theories. In: Handbook of Satisfiability, IOS Press, 2009, pp. 825-885.

[Barret et al. 2010] C. Barrett, A. Stump, C. Tinelli: The SMT-LIB Standard: Version 2.0. In: Technical Report, Department of Computer Science, The University of Iowa, 2010.

[Büker et al. 2011] M. Büker, W. Damm, G. Ehmen, A. Metzner, E. Thaden, and I. Stierand: Automating the Design Flow for Distributed Embedded Automotive Applications: Keeping Your Time Promises, and Optimizing Costs, Too. In Proceedings of the International Symposium on Industrial Embedded Systems (SIES), 2011, pp. 156-165.

[Büker et al. 2013] M. Büker, W. Damm, G. Ehmen, S. Henkler, D. Janssen, I. Stierand, and E. Thaden: From Specification Models to Distributed Embedded Applications: A Holistic User-Guided Approach. SAE International Journal of Passenger Cars- Electronic and Electrical Systems, 2013, pp. 194-212.

[Chapman 2006] R. Chapman: Correctness by construction: a manifesto for high integrity software. In: Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software, Vol. 55, 2006, pp. 43–46.

[de Moura and Bjørner 2008] L. de Moura and N. Bjørner: Z3: An Efficient SMT Solver. TACAS, LNCS. Springer, Berlin Heidelberg, Vol. 4963, No. 24, 2008, pp. 337–340.

[Diebold et al. 2014] P. Diebold, C. Lampasona, S. Zverlov, S. Voss: Practitioners' and Researchers' Expectations on Design Space Exploration for Multicore Systems in the Automotive and Avionics Domains: A Survey. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM, New York, 2014.

[Eisenring et al. 2000] M. Eisenring, L. Thiele, E. Zitzler: Conflicting Criteria in Embedded System Design. In: Design & Test of Computers, Vol. 17, No. 2, 2000, pp. 51-59.

[Hall and Chapman 2002] A. Hall and R. Chapman: Correctness by construction: developing a commercial secure system. Software IEEE, Vol. 19, No. 1, 2002, pp. 18–25.

[Hamann et al. 2006] A. Hamann, M. Jersak, K. Richter, R. Ernst: A Framework for Modular Analysis and Exploration of Heterogeneous Embedded Systems. Real-Time Systems, Vol. 33, No. 1-3, 2006, pp. 101-137.

[Hilbrich and Dieudonné 2013] R. Hilbrich, L. Dieudonné: Deploying Safety-Critical Applications on Complex Avionics Hardware Architectures. Journal of Software Engineering & Applications, Vol. 6, No. 5, 2013.

[Hofstedt and Wolf 2007] P. Hofstedt, A. Wolf: Einführung in die Constraint-Programmierung Grundlagen, Methoden, Sprachen, Anwendungen. Springer, Berlin Heidelberg, 2007.

[Kirkpatrick et al. 1983] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi: Optimization by Simulated Annealing. Science, Vol. 220, No. 4598, 1983, pp. 671-680.

[Kondeva et al. 2013] A. Kondeva, D. Ratiu, B. Schatz, S. Voss: Seamless model-based development of embedded systems with AF3 Phoenix. ECBS, IEEE, 2013, pp. 212.

[Kugele and Pucea 2014] S. Kugele, G. Pucea: Model-Based Optimization of Automotive E/E Architectures. In: 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, 2014, pp. 18-29.

[Rosinger et al. 2014] M. Rosinger, M. Büker, R. Weber: A User-Supported Approach to Determine the Importance of Optimization Criteria for Design Space Exploration. In: Proceedings of IDEAL'14 Workshop, IFIP Springer Series, Springer, Berlin Heidelberg, 2014.

[Rosinger et al. 2015] M. Rosinger, M. Büker, R. Weber: An Approach to Guide the System Engineer during the Design Space Exploration Process. In: W. Zimmermann, W. Böhm, C. Grelck, R. Heinrich, R. Jung, M. Konersmann, A. Schlaefer, E. Schmieders, S. Schupp, B. T. Widemann, T. Weyer (Eds.): Software Engineering Workshops 2015 (SE-WS 2015) - Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015, Vol. 1337, CEUR-WS.org, 2015, pp. 81-90.

[Thaden 2013] E. Thaden: Semi-Automatic Optimization of Hardware Architectures in Embedded Systems. PhD thesis, Carl von Ossietzky University of Oldenburg, 2013.

[Weber et al. 2014] R. Weber, S. Henkler, A. Rettberg: Multi-Objective Design Space Exploration for Cyber-Physical Systems Satisfying Hard Real-Time and Reliability Constraints. In: Proceedings of IDEAL'14 Workshop, IFIP Springer Series, Springer, Berlin Heidelberg, 2014.

[Wolf 2006] A. Wolf: Object-Oriented Constraint Programming in Java Using the Library firstCS. WLP, 2006, pp. 21–32.