Exploiting the Model-Driven Engineering Approach to Improve Design Space Exploration of Embedded Systems

Marcio F. da S. Oliveira\textsuperscript{1, 2}, Ronaldo R. Ferreira\textsuperscript{1}, Francisco A. Nascimento\textsuperscript{1}

marcio@c-lab.de, rrrferreira@inf.ufrgs.br, fanascimento@inf.ufrgs.br

Franz J. Rammig\textsuperscript{2}, flavio@inf.ufrgs.br

\textsuperscript{1}Institute of Informatics, Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre, Brazil

\textsuperscript{2}C-Lab, University of Paderborn
Fürstenallee, 11, Paderborn, Germany

ABSTRACT

This paper proposes a design space abstraction, in order to decouple the exploration algorithm from the design space, which allows the application of the design space exploration (DSE) tool in different design scenarios and is appropriate for representing simultaneous and interdependent design alternative. From this new abstraction, the Model-Driven Engineering (MDE) approach is exploited to extract design information and compose the design space to be explored. Our approach uses model-to-model transformation rules as DSE constraints, which prune the available design space. These constraints are automatically generated from the UML model by translating design decisions pre-specified in UML diagrams into model transformation rules. In addition, non-functional requirements specified in UML diagrams are translated into model-to-model transformation rules, which are used to prune and guide the DSE process. Finally, our approach offers an easy way for the designer to extend the set of constraints by using a well-accepted MDE toolset. A real application running on top of an MPSoC is used as case study to illustrate the proposed method.

Categories and Subject Descriptors

C.3 [Special-purpose and application-based system] Real-time and embedded systems
D.2.10 [Software Engineering] Design - methodologies

General Terms

Design, Algorithms, Performance

Keywords

Design space exploration, model-driven engineering, embedded systems

1. INTRODUCTION

Modern embedded systems have increased the functionality they offer by using a large amount and diversity of hardware and software components. During the development of these systems, a wide range of design alternatives arises from different design activities. A complex design space is unveiled by combining the alternative designs and stringent requirements and must be explored under reduced time-to-market. In this scenario, Design Space Exploration (DSE) is performed, looking for an adequate software and hardware design solution that is appropriate for the product in development, according to its non-functional requirements, such as performance, power, size, and cost.

Model-Driven Engineering (MDE) \cite{15} approaches have been proposed to improve the complexity management and also the reusability of previously developed/specifed artifacts. MDE, combined with a standard modeling language, such as UML or Simulink, raises the design abstraction level and provides mechanisms to improve portability, interoperability, maintainability, and reusability of models. In addition, MDE helps abstracting away platform complexity, while also representing different system concerns, by exploiting well-accepted meet-in-the-middle development strategies, following the principles of the Platform-based Design (PBD) \cite{14} approach.

The purpose of this work is to improve the DSE process, providing mechanisms to better represent the design space and to apply constraints that are needed to guide the DSE process. Aiming at this goal, a design space abstraction is firstly provided. The abstraction, based on the categorical graph product, decouples the exploration algorithm from the design space and is well suited to be used in automatic exploration tools, which can exploit MDE transformation rules based on graph grammars. By means of MDE concepts applied in the proposed method, design constraints are automatically generated from the UML model. Non-functional requirements and pre-defined design decisions specified in the UML model are translated into model-to-model transformation rules, which are used to prune and guide the DSE process.

The remaining of this paper is structured as follows. Section 2 reviews related work; Section 3 presents the system specification method used to produce the input to the DSE process; Section 4 introduces the DSE method; Section 5 approaches how MDE concepts are applied to improve the DSE process; a case study based on a Multi-Processor System-on-Chip (MPSoC) platform...
and illustrating the method is provided in Section 6; and, finally, in Section 7 conclusions are drawn and future work is introduced.

2. RELATED WORK

As a complete environment for DSE, the MILAN [1] / DESERT [8] framework is worth mentioning. The focus of MILAN is on the simulation of embedded systems, evaluating pre-selected candidate solutions. The DESERT tool uses models of aggregated system sub-components and constraints to automatically compose the embedded system through Ordered Binary Decision Diagrams and is based on a complete pre-characterization of components.

The ARTEMIS [13] / SESAME [12] framework provides methods for modeling and simulation at different abstraction levels, aiming at an automatic DSE of heterogeneous embedded MPSoCs for multimedia applications. However, this framework does not use MDE, relying instead on lower level languages such as C/C++ and a specific API to represent a Kahn process network. Besides, the designer must specify separated models for implementation and evaluation. These facts increase the design complexity, making difficult the reuse of previous designs and DSE, as each change must be reflected in each model.

The DaRT [3] approach is based on the Model-Driven Architecture (MDA) and has many similarities with our approach. It defines metamodels to specify application, architecture, and software/hardware associations and uses transformations between models to optimize an association model. By doing so, it allows the re-factorizing of an application model in order to better match it to a given architectural model. In DaRT, no DSE strategy based on these transformations is implemented, and the focus is mainly on code generation for simulation at Transaction and Register Transfer levels.

Koski [5] is a UML-based framework supporting MPSoC design, which provides tools for UML system specification, estimation, verification, and system implementation on FPGA. Following the design flow, the application, architecture, and the initial mapping are specified with UML models. An internal representation is generated from the UML models, which is used for architectural exploration. The exploration algorithm uses static and dynamic methods to evaluate design alternatives.

The proposed work extends a previous effort [10], where the H-SPEX (High-level Design Space Exploration) tool was presented. In the current work the H-SPEX approach is improved with a more adequate method for design space representation. In this work, the MDE approach is exploited not only in the context of design representation, which benefits from the orthogonalization proposed by the internal metamodel infrastructure. It also combines the design space abstraction with transformation rules, exploiting the MDE approach in order to prune the design space and speed up the exploration process. Moreover, the previous work uses Simulated Annealing as optimization algorithm to explore the design space. We employ CPACO-MO (see section 3.2) instead.

Compared to our approach, no related work takes advantage of the MDE notion of transformation between models to represent, constrain, and explore the design space. Furthermore, the proposed DSE method exploits the MDE concepts to store and distribute the design information, easily integrating the DSE tool to the development process. Finally, MDE allows the evaluation of possible implementations for a given application on a specified platform at a high abstraction level.

3. DESIGN SPACE EXPLORATION

3.1 Design Space Abstraction

The proposed DSE method uses a categorical graph product to represent the design space and its alternative designs. This approach is appropriate for representing simultaneous and interdependent design options, firstly because graphs are general representations in a common data structure for data processing. Furthermore, graphs allow different views of the same information while preserving the original semantics, such that the design space representation can be used to perform different design activities based on the same design information. Besides, graphs allow the use of the aforementioned data structure containing semantically different information. Therefore, DSE can be performed using different design information, but relying on the same structure and algorithms.

Initially, we define the categorical graph product [16] and then observe its properties, which properly represent the design space. A graph $G = <V, T, \partial_0, \partial_1, \pi>$ is defined as follows: $V$ is the set of all nodes of $G; T$ is the set of all arcs of $G; \partial_0: T \rightarrow V$ is the source function of an arc; $\partial_1: T \rightarrow V$ is the target function of an arc.

Let $G_1 = <V_1, T_1, \partial_{01}, \partial_{11}>$ and $G_2 = <V_2, T_2, \partial_{02}, \partial_{12}>$ be two graphs. The graph product between $G_1$ and $G_2$ is defined as follows: $G_1 \times G_2 = <V_1 \times V_2, T_1 \times T_2, \partial_{01} \times \partial_{02}, \partial_{11} \times \partial_{12}>$ represents the graph product between $G_1$ and $G_2$, where $\partial_{01} \times \partial_{02} \mid k \in \{0, 1\}$ are unambiguously induced by the dot product between nodes and arcs. Two projection functions $\pi_1 = \pi_1$, $\pi_2: G_1 \times G_2 \rightarrow G_1$ and $\pi_2 = \pi_2$, $\pi_2: G_1 \times G_2 \rightarrow G_2$ are defined and return the graphs $G_1$ and $G_2$, respectively.

To illustrate this definition, consider graphs $T$ (Figure 1-a), representing a task graph, and $P$ (Figure 1-b), representing processing units and the communication between them. The graph $T \times P$ in Figure 1-c is the categorical graph product between $T$ and $P$, representing a design space for the task mapping design activity.

![Figure 1. Sample of a candidate design sub-graph using categorical graph product](image-url)
Currently, the activities performed by our prototype DSE tool in the context of an MPSoC design are: i) selecting the number and types of processors; ii) mapping tasks into processors; iii) allocating processors into a communication structure, and; iv) selecting processor voltage operation.

Using the categorical graph product abstraction, the design space exploration problem consists in searching for sub-graphs, which represent candidate designs. Figure 1-d illustrates a selected sub-graph composed by nodes \(<T1, P1>\), \(<T2, P2>\), \(<T3, P2>\), and \(<T4, P2>\). The procedure to select nodes in order to produce the sub-graph is guided by an optimization algorithm, explained in the next section, which searches on the design space graph for candidate sub-graphs. A graph controller guides the iteration over the design space and shapes the sub-graph. In the example illustrated in Figure 1, the sub-graph has the same shape as the task graph \(T\), as all tasks must be mapped into some processor.

The iteration of the design space algorithm exploits the categorical product restriction, where any two vertices \((u, u')\) and \((v, v')\) are adjacent in \(G_1 \times G_2\) if and only if \(u'\) is adjacent with \(v'\) in \(G_2\) and \(u\) is adjacent with \(v\) in \(G_1\). This procedure makes a biased design space exploration, driven by communication dependencies exposed in the design space representation, as the design alternatives on the design space are suggested by following the nodes’ adjacency. Regarding an MPSoC design, this property is particularly interesting as it can reduce communication dependencies at multiple levels (e.g. tasks and processors) and optimize the MPSoC architecture.

Besides communication-driven DSE, the adopted design space representation provides other benefits: i) the multiple optimizations required by different design activities are always seen as the same problem – to find the best set of sub-graphs from the design space graph; ii) as the design space and the solution are represented as graphs, they are uncoupled from the exploration algorithm, and, thus, different optimization algorithms can be used; iii) graphs are easily handled by model-to-model transformation rules (mainly based on graph grammars), which are used to implement the constraints; and, iv) arcs guide the available design alternatives at a specific node, and, hence, constraints can be locally applied between the current node and its neighbors.

### 3.2 Design Space Exploration Algorithm

Since most of DSE activities correspond to a multi-objective (multi-criteria) optimization problem, an algorithm to search for alternative solutions on the design space and provide a Pareto set should be employed. The proposed DSE tool employs the Crowding Population-based Ant Colony Optimization for Multi-objective (CPACO-MO) [2] algorithm. At each iteration, this algorithm generates a set of candidate solutions (sub-graphs), from where the best candidates are selected to compose a population. This population is a set of non-dominated design points (Pareto-optimal), with which the designer can take its decision by considering design trade-offs and system requirements. Since the design space is represented as a graph product and the optimization problem consists always in finding an “optimal” sub-graph (as explained in Section 3.1), the optimization algorithm is not aware of specific DSE information and of the semantic of nodes and arcs on the design space and sub-graphs. This means that the optimization algorithm is detached from the design space and from the specific design exploration problem, such that a specific design optimization approach is not required and any other multi-objective heuristic can be used.

In the MPSoC design context, the DSE tool can be used to optimize the system in terms of energy or power consumption, communication bandwidth, memory footprint, performance (execution time or cycles), or any combination of the above, which can be obtained by different estimation/simulation tools. The values of these objectives used in the optimization depend on the evaluation tool invoked by the DSE. The information obtained is stored in arcs and nodes of candidate sub-graphs, which can be analyzed by any optimization algorithm in its objective function.

### 3.3 High-Level Model-Based Estimation

In order to support a fast DSE process in the early design phases, the SPEU estimation tool [9] is used to evaluate alternative model solutions and to guide the designer’s decisions. SPEU provides analytical estimates on physical system properties (execution cycles, energy/power consumption, volume of communication data, and memory footprint). These properties are used to calculate the solution cost using an objective function, which the optimization algorithm tries to minimize on a best-effort manner.

Using the information extracted from UML application structure/behavior models and stored as a Control and Data Flow Graph (CDFG), estimations are performed with estimation errors as low as 5%, compared to results extracted by cycle-accurate simulation, when the reuse of repository components is largely employed by a platform-based design approach. To achieve accurate estimations, the information specified in the platform repository is used to compute the costs of pre-designed components and added in the final estimation. An Integer Linear Programming (ILP) formulation is used to identify the best and worst-case execution paths in the CDFG. In order to reduce complexity, each ILP formulation is solved in a hierarchical fashion, such that for each task there is only one ILP formulation to be solved. This avoids problems for MPSoCs with a large number of tasks. These estimations allow the DSE tool to quickly evaluate each candidate solution during the DSE process, without depending on costly synthesis-simulation evaluation cycles.

After each evaluation, estimated properties are made available in a repository provided by an MDE infrastructure and used to compute the solution (sub-graph) cost during the multi-objective optimization.

### 4. MODEL-DRIVEN ENGINEERING FOR DESIGN SPACE EXPLORATION

MDE allows a loosely coupled integration of the DSE tool with different specification, simulation, and estimation tools. In order to implement the MDE framework we use the Ecore metamodel from the Eclipse Modeling Framework (EMF) and Open Architecture Ware [11], allowing the environment to define and apply model-to-model transformations in order to translate the design information between these tools, thus building a unified workflow. Furthermore, our approach differs from other ones as additional constraints are orthogonally specified, using a commonly adopted MDE framework toolset, thus allowing easy evolution and reuse of the application specification, as well as a better system evaluation against distinct constraints.

MDE transformations are used in the process to: i) load the UML model into the design repository of the MDE framework; ii) query
the design repository and generate the graphs involved in the graph product; and iii) prune the design space during DSE. The graphs involved in the categorical product are obtained from instances of metamodels from the MDE framework. Different graphs must be generated, depending on which design activities the DSE tool was configured to perform. Currently, the generated graphs are: the task graph, defining the schedulable active behaviors and communications between them; the processor graph, defining the maximum number of processors and the type of each one; and the communication graph, defining the communication structure used to integrate the processors. Although the transformation rules required to build the graphs are dependent on the information contained in the design repository, the final graph product is semantic free. As a result, the exploration algorithm is not aware of the design space exploration semantics when performing optimizations.

4.1 MDE Infrastructure

The model-based DSE is supported by the MODES framework [7], providing the MDE infrastructure to store, transform the models, and make them available to the model-based design tools, which perform DSE, physical estimation, code generation, and hardware synthesis. The MODES framework is based on the Y-chart [6] approach, in order to keep the application specification without architectural/platform implications. Therefore, mapping is required to define how application functions are partitioned among architectural components. Accordingly, four internal metamodels have been proposed to allow the uncoupling of the application modeling from the platform: Internal Application, Internal Platform, Mapping, and Implementation.

Using the Internal Application Metamodel, a system specification captures the functionality of an application in terms of a set of modules that are composed of concurrent communicating processes and/or sub-modules. The module behavior is captured in terms of actions represented by a CDFG that captures the data and control flow between the UML actions in sequence diagrams. The Internal Platform Metamodel stores meta-information about the platform and its components, which are available during the design. This information is derived from a platform pre-characterization, in terms of Quality of Service (e.g. performance, latency, etc.), of services provided by the platform, such as schedulers, APIs (e.g. math and communication operations), algorithms, data structures, and even models at different abstraction levels.

The Mapping Metamodel describes the rules used to transform instances of the Internal Application Metamodel and Internal Platform Metamodel into an instance of the Implementation Metamodel. Thus, the Mapping Model guides the exploration tool to build a candidate Implementation Model. The Implementation Metamodel allows the representation of models that can implement the system specification without violating system requirements. An Implementation Metamodel instance can be described as a list of selected platform and application components, as well as associations between them.

4.2 MDE for Design Space Pruning

In a straightforward way, the categorical graph product is a dot product (product of all vertices of a graph by all vertices of another graph). The set of constraints applied to a categorical graph product makes it unique. These constraints shape the graph product, giving interesting properties to the resulting graph, making it well suited to represent the design space as discussed in Section 3.1. However, additional constraints should be specified, since the structure of the categorical graph product still allows unfeasible design solutions. Even if a design is feasible, it can be invalidated when checked against non-functional requirements, which must be satisfied by the system. Moreover, designs usually start with pre-defined design decisions, such as specific processors, components, and communication structure, and these decisions should constrain the design space.

The DSE tool provides some structural constraints, in order to avoid unfeasible solutions. It also generates constraints automatically from non-functional requirements specified by means of MARTE stereotypes on UML models. Furthermore, the designer can also specify additional constraints, in order to specify pre-defined design decisions. The existing pre-defined constraints are grouped and detailed next.

Structural Constraints: These constraints are applied in order to avoid unfeasible designs, which could appear as a sub-graph of the design space. They can be applied on the whole design space, when it is small, before starting the DSE process. Other possibility is to apply these constraints during DSE, only for the restricted context of the current vertex and its neighbors, in the case of a large design space. Some available constraints are:

- **Duplicated Processor Assignment**: Avoids different processors to be assigned to the same communication slot.
- **Multi Processor Assignment**: Avoids the same processor to be assigned to different communication slots.
- **Multi Task Assignment**: Avoids the same task to be assigned to different processors.

Non-Functional Constraints: These constraints are applied to avoid an unfeasible design when it is checked against non-functional requirements. These constraints avoid the violation of requirements such as task deadline, maximum delay, and maximum energy consumption. Although designers could specify them, some of these constraints are generated automatically, when the MARTE stereotypes and tags are applied. These constraints are applied after the evaluation of a candidate solution by the estimation tool. In the case of requirements violation, the arcs of the candidate solution sub-graph are removed from the design space and the sub-graph is removed from the population.

Pre-defined Design Decisions: Previously developed components or the selected platform could impose restrictions a designer must respect. Moreover, the design experience may influence how the automated DSE process proceeds. Therefore, these constraints are specified in cases where the designer needs to interfere directly in the DSE process through specific design decisions. For this purpose, the transformation rules must be manually written and loaded by the DSE tool. Our tool already provides three design constraints, namely Specific Task Mapping, Specific Processor Allocation, and Specific Task Execution Frequency, which were specified as transformation rules in the xTend language provided by OAW and are automatically applied when this information is supplied in the UML Deployment diagram.

5. DESIGN FLOW OVERVIEW

5.1 System Modeling

This work relies on UML and the MARTE profile to specify functional and non-functional requirements. UML provides an
adequate abstraction for DSE, allowing a designer to omit details of the model, which can be later specified by an automated DSE tool. A non-intrusive set of modeling guidelines must be followed to allow design automation tools to capture and handle the UML models. In our approach, at least Use Case, Class, and Sequence diagrams should be used to specify the application. Figure 2-a shows the partial class diagram for the wheelchair movement control used in the case study (see Section 6), which shows the \textit{MovementController} class annotated with a stereotype (\textit{ScheduleableResource}) from the MARTE profile representing a class of active objects.

The deployment diagram may be used for a manual definition of system architectural issues, such as the number and types of processors, the task mapping, and the hardware communication structure. By using the proposed DSE method, the DSE tool can automate some of these activities, and the designer is not required to specify the complete deployment diagram. In the case where a deployment diagram is specified, its information is interpreted as predefined design architectural decisions, imposing constraints to the DSE process. Figure 2-b illustrates a deployment diagram, in which task T15 is mapped to DSP processor P0.

Figure 2-c shows the sequence diagram for the behavior of an instance of the class \textit{MovementController}, which must periodically calculate movement information and send it to the actuator interface.

Figure 2. Sample Diagrams: a) class; b) deployment; c) sequence

5.2 Automated Design Space Exploration
After the system specification using UML and its MARTE profile, the tool chain performs the automated DSE process, as illustrated in Figure 3. In this tool chain, the H-SPEX tool [10] was extended to implement the proposed design space abstraction method and to exploit the MDE approach. The automatic process starts by (1) loading the UML models from a UML editor into the MDE framework. From the application, platform, and mapping models (2), H-SPEX builds the graphs used in the graph product (design space graph) from the design information contained in the MDE framework. For the selected alternative designs (sub-graphs from the design space), H-SPEX (3) writes the design choices to set up the implementation model in the MDE framework, according to the selected alternative designs. After that, the estimation tool (4) is executed in order to perform the model-based estimation. The estimation tool (5) reads the resulting implementation model from the MDE framework, (6) performs the estimations, and writes the result back into the MDE framework. By using the estimated values in the Implementation Model, H-SPEX (7) evaluates the design alternatives by analyzing the costs that are estimated for each sub-graph. H-SPEX iterates again from step (2) to generate another set of alternative designs, if the selected set does not meet the design constraints or the stop condition was not reached.

Figure 3. Tool workflow to perform DSE

6. CASE STUDY
A DSE scenario for the design of a real application, concerning the automated control of a wheelchair with functions such as movement control, collision avoidance (ultrasound and stereo vision), and navigation, has been developed. These functions are implemented by 17 tasks, having communication dependencies between them. The platform provides up to four processors and a hierarchical bus with two segments to support the system implementation. Firstly, the system was specified as described in Section 5.1, using UML and MARTE. Figure 2 shows the partial UML models for the case study. Following the process described in Section 3.2, three graphs were extracted from the UML model, to serve as input to H-SPEX: (i) the task graph containing the referred 17 tasks (Figure 4-a); (ii) the processor graph (Figure 4-b) defining that four processors can be used without communication restriction; and (iii) the communication graph (Figure 4-c), which defines the hierarchical bus, where selected processors must be allocated into. These three graphs are used to calculate the graph product, which represents the design space (Figure 4-d).
Task 15 implements a stereovision function, presenting heavy image processing algorithms. Figure 5-a shows a deployment diagram specifying that the DSE tool must map Task 15 into the DSP processor P0, in order to take advantage from the DSP processor architecture. Figure 5-b illustrates the design space at the vertex <T13, P1, C1>. Figure 5-c shows the reduced design space that is available after the evaluation of the Specific Mapping constraint, showed in the Figure 5-d, which is automatically applied when the deployment diagram is specified.

One of the future directions to be considered is the extension of the design exploration activities, by means of a generalization of the rules used to extract the graph from the design repository, in order to compose the categorical graph product. Also more experiments should be performed, in order to better evaluate the benefits of the proposed approach.

### References


* This work is supported by CNPq - Brazil