Model-Driven Engineering Approaches to Design Space Exploration

Joachim Denil*†, Gang Han‡, Magnus Persson§,
Paul De Meulenaere†, Haibo Zeng‡, Xue Liu¶ and Hans Vangheluwe*‖
*Ansymo-Lab, University Of Antwerp, Belgium
†Tera-Labs, Karel de Grote University College, Belgium
Email: Joachim.Denil@kdg.be
‡Department of Electrical and Computer Engineering, McGill University, Canada
Email: Haibo.Zeng@mcgill.ca
§ Department of Machine Design, Royal Institute of Technology (KTH), Sweden
Email: magnper@kth.se
¶Cyber-Physical Systems Lab McGill University, Canada
Email: xue@cs.mcgill.ca
‖MSDL-Lab, McGill University, Canada
Email: hv@cs.mcgill.ca

***Abstract*—During the design and deployment of increasingly complex distributed embedded systems, engineers are challenged by a plethora of design choices. This often results in infeasible or sub-optimal solutions. In industry and academia, general and domain-specific optimization techniques are developed to explore the tradeoffs within these design spaces, though these techniques are usually not adapted for use within a Model-Driven Engineering (MDE) process. In this paper we propose to encode metaheuristics in transformation models as a general design exploration method. This is complemented by an MDE framework for combining different heterogeneous techniques at different abstraction layers using model transformations. Our approach results in the seamless integration of design space exploration in the MDE process. The proposed methods are applied on the deployment of an automotive embedded system, yielding a set of Pareto-optimal solutions.**

I. Introduction

The design of distributed embedded systems has changed radically over the last decades. To deal with the increasing complexity of these complex systems, companies are moving to model-driven engineering (MDE) techniques for the design and testing of these systems. This has the advantage that domain knowledge can be explicitly modelled using languages closer to the domain experts. The model artefacts can be reused for later design variants, documentation, etc. To manipulate the created models, MDE uses model transformations. Examples of the introduction of MDE can be found in the automotive industry with the use of Architecture Description Languages (ADLs) to formalize the design information and AUTOSAR [1] for the component-based design of automotive systems.

During the design of distributed embedded systems many design choices need to be made. For example, an optimal mapping of a set of software components to a set of hardware components. The process of analysing the feasibility and optimality of several solutions by searching through the design space is called Design-Space Exploration (DSE). The solution space of DSE problems is constrained by the functional and extra-functional requirements of the system. These constraints can be either structural, behavioural or derived. On the other hand, designers want to optimize the solution with respect to one or a set of goal functions.

In industry and academia, a lot of heterogeneous techniques have been developed to search a solution space. These techniques include but are not limited to Mixed Integer Linear Programming (MILP), Constraint Satisfaction (CSP) and Meta-heuristics, like hill climbing and simulated annealing. Another set of techniques have been specifically developed for some domain specific problems, for example list scheduling techniques for the optimization of scheduling problems. The multi-criteria and trade-off nature of design space exploration makes it a challenging task, where manual work needs to be combined with efficient partial automation. Most traditional optimization techniques are not designed for use with multi-criteria optimization. They commonly solve this by reducing the optimization problem to a single optimization goal through a weighted goal function based on the optimization parameters. This is not sufficient for the task at hand in design space exploration, as it is a broader task than design optimization, as it also includes e.g. uncovering relations between different design parameters.

In the research community there has not been a lot of focus on integrating these heterogeneous techniques within the MDE development cycle. Integrating these techniques, has the advantage that the whole process is uniform. Several major tool integration research projects have explicitly chosen the model transformation approach as their integration mechanism [2], [3]. The approach also has the advantage that the design variability in the model under design is made explicit, in the *same* formalism as the model itself, making it readable not only to optimization experts but also to domain engineers. This has the effect that the domain knowledge that is used within

the optimization techniques is made explicit. The models can also be reused, both for later design variants with different requirements, documentation, etc.

In this paper we propose a solution for integrating these different techniques in an MDE development cycle. The proposed solution uses model transformation techniques to (a) integrate meta-heuristics in the transformation models so they can be used for general optimization problems on models and (b) a technique to combine multiple transformations on different levels of abstraction/approximation with different heterogeneous techniques.

The rest of the paper is structured as follows: section II gives a general introduction to model transformation and the principles of implementing meta-heuristics using a model transformation language, section III explains the principles behind combining several subsequent transformations, section IV validates the approach using an automotive case study, section V discusses the results, section VI gives an overview of other similar approaches in the research literature, and section VII concludes the paper and discusses future work.

II. A Model Transformation Language with Search Capabilities

In this section we introduce model transformation techniques and discuss how general search techniques can be used within model transformations. While it is impossible to give a complete overview of these techniques, we show that by using a very expressive transformation language, meta-heuristics can be implemented in the transformation models.

A. Model Transformation Languages and T-Core

Model transformation languages work on typed, attributed, labelled graphs that represent the model. A rule represents manipulation operations on the represented model. A rule consists of a left-hand side (LHS) pattern representing the precondition for the applicability of the rule. The right-hand side (RHS) pattern defines the outcome of the operation. A set of negative application condition (NAC) patterns can be defined to block the application of the rule.

The developed transformation language is based on the T-core transformation framework that allows the construction of custom transformation languages [4]. Figure 1 shows some of the components of a transformation language. We briefly discuss the components used in this work. More information can be found in [4].

- Operators:
  - Matcher: The matcher finds the matches of the LHS condition in the model and stores them in a match-set.
  - Iterator: The iterator is used to select one match to rewrite.
  - Rewriter: The rewriter rewrites the model using the RHS pattern.
  - Rollbacker: The rollbacker enables backtracking in the transformation language.
- Scheduling Language: The scheduling language is used to schedule the different rules after each other. Different kind of scheduling languages can be used. In this work we use the Python programming language as our scheduling language.

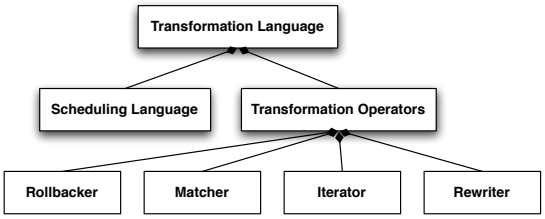

Fig. 1. Composition of a Transformation Language

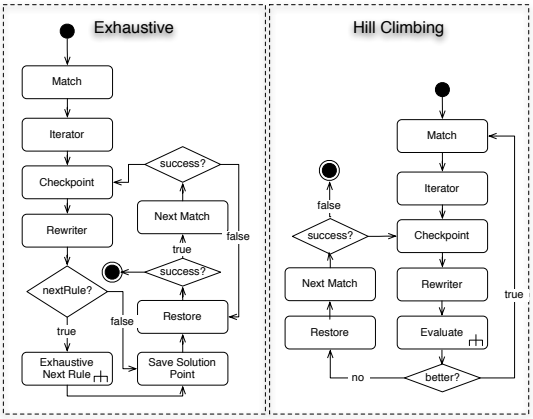

Fig. 2. Exhaustive Search and Hill climbing

In the following subsections we show how to implement, using the T-Core Transformation Framework, three well known search techniques that are used in optimization. The first two: *exhaustive and random* search create a number of solution points in the search space. The latter: *Hill Climbing* starts optimizing a single solution.

B. Exhaustive Search

While the exhaustive search is infeasible in most problems, it can be used for the optimisation of small problems. Exhaustive search will generate all solutions of the design space. Figure 2 shows an activity diagram of the implementation of the exhaustive search method. The transformation starts by matching all the occurrences in the start model. The iterator chooses the first match in the match-set. At this point a checkpoint is made. This checkpoint contains (a) the model, (b) the match-set, without the chosen match and (c) the selected match. The selected match is rewritten in the model. If more rules are available or the same rule that has to be executed multiple times, this is done by using the same method (match, select match, checkpoint and rewrite). The complete solution is archived for further analysis. Then backtracking can start. Since the backtracker can contain multiple checkpoints (from multiple rule applications), the last one is selected. This

restores (a) the model, (b) the match-set and (c) the match that was selected at the time instant the checkpoint was made. The iterator is used to replace the previously chosen match with another one from the match-set, again this is check-pointed and rewritten. The process continues as described above until all matches in the match-sets of all checkpoints have been applied.

C. Random Search

In random search a set of solutions is created in a random way. Random search uses only the matcher, iterator and rewriter. After matching all occurrences of the pattern in the model, a random match is selected for rewrite. This requires a different iterator than in exhaustive case. The rewriter applies the randomly chosen match on the model. Another rule, or the same rule can be executed after that until a solution point is obtained. A loop is used to create a set of solutions.

D. Hill Climbing

Hill climbing is a local search technique that uses an incremental method to optimize a single solution. It examines neighbouring states and accepts the change if it is a better solution. Figure 2 shows the building blocks of the Hill Climbing transformation. After matching a (set of) rule(s), the iterator picks one match at random and rewrites this in the model. The solution is evaluated and compared with the original solution. In case the solution is not better, the original solution (with the matches) is restored and another match is randomly selected and evaluated. If the solution is a better one, it is accepted. The evaluator contains a set of transformation rules to calculate the metrics of the solution or to generate an analysis or simulation model that can be executed. The metrics obtained are given back to the hill climbing solution so a decision can be made. When a better solution has been found, the process is restarted until no more improvements can be found.

III. Combining Multiple Transformations

The second part of our contribution consists of combining different transformations, search-based transformations and Model-to-Model transformations, in sequence or in parallel to optimize a system. We leverage a number of techniques to alleviate the state space explosion problem during the optimization of a system:

- *Different levels of abstraction or approximation:* Based on the idea of platform based design [5], clear approximation or abstraction levels[1] can be added where intermediate solutions can be evaluated and pruned early.
- *Other optimization techniques:* When a general solution method is already available, for example through the capabilities of standard tool, we transform the model to this representation. The results are transformed back to the original representation for further exploration or synthesis activities.
- *Manual activities:* When the designer has a solution (manually or using external tools) without an available automatic transformation, the designer can manually change the model. The exploration activity resumes from this point.

In this work the Formalism Transformation Graph and Process Model (FTG+PM), defined in [8], is used as the base for chaining different transformations. The framework can be used to incorporate all the different steps of the MDE lifecycle and thus allows the embedding of the exploration activity in the design and verification of complex systems [9].

The FTG+PM framework is comprised of the Formalism Transformation Graph (FTG) and its complement, the Process Model (PM). The FTG is a hypergraph with *languages* as nodes and *transformations* as edges. It charts the relationships among the multitude of languages and transformations used to develop systems within a domain. The PM precisely models the control and data flow between the transformation activities taking place throughout the software development lifecycle starting from requirements analysis and design, to verification, simulation, and deployment. The elements used in the PM-side are typed by the nodes and edges in the FTG. We use a subset of the UML 2.0 activity diagram formalism for the PM. The FTG+PM can be enacted to allow the automatic and semi-automatic execution of the MDE design process.

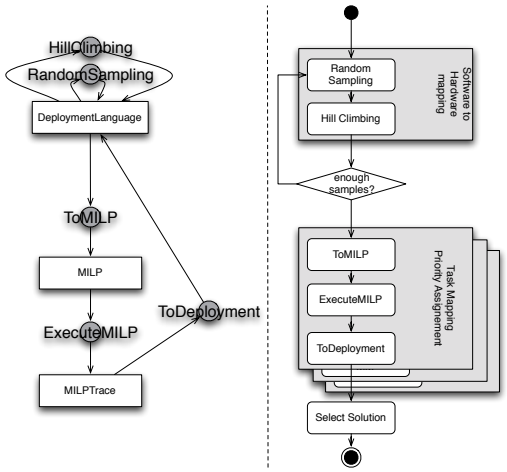

Fig. 3. An example FTG+PM for design space exploration

Figure 3 shows an example FTG+PM for the purpose of design space exploration. On the left side of the figure, all languages involved in the optimization chain are displayed. We only present three languages: (a) The deployment language where the design space should be explored, (b) The MILP language, representing the model in a Mixed Integer Linear

[1] Abstraction levels are used to describe the same system in different levels of detail in order to hide lower-level implementation details when they are not relevant. Approximation levels are similar, however they describe the same model. For an example of approximation levels, a scheduling problem may be analysed through the classical Liu-Layland utilization bound, using exact response time analysis [6], or even more intricate analysis, e.g. using timed automata [7] if the task models do not fit the classical assumptions, such as absence of task communication.

Program, and (c) The MILPTrace language, representing the results produced by the MILP solver. At the FTG level, five transformations are defined: (a) Random Sampling, (b) Hill Climbing, (c) ToMILP, (d) Execute MILP and (e) ToDeployment. For each of these transformations the input and output languages are defined. On the right side of the figure these transformations are scheduled after each other in the process. The optimization chain starts with the Random sampling of a number of solutions. These solutions are further explored using an instance of the hill climbing transformation. Later, the intermediate results are stored and this process repeats until a fixed number of solutions are found. At another approximation level, a set of these solutions are transformed to a MILP representation. This MILP representation is solved using standard solvers such as CPLEX. Finally, the traces are translated to the original representation.

IV. Case Study

We show our contributions using an automotive case study based on [10] where a set of software functions need to be assigned to a set of electronic control units (ECU), and the designer can select the types of ECUs. The software functions are further executed on a real-time operating system with a fixed priority preemptive scheduler, where the priorities of the tasks need to be selected, and the communication signals need to be packed into communication frames on a communication bus with fixed-priority non-preemptive scheduler (like in Controller Area Network). The solutions are constrained by typical end-to-end deadlines. Due to space constraints of this paper, we cannot show all transformations and (meta-) models involved in this case study.

We use two case studies in our experiments. An industrial size design consists of 40 software functions, 81 signals, and 9 ECUs (each has two types to choose from). A small subset of the design containing six software functions and two ECUs was created, to allow the validation using exhaustive search.

A. Validation

Figure 4 shows a part of the start model. The blue boxes represent the software functions, with periods, that need to be deployed on the ECUs (green boxes). Signals between the software functions are represented by purple circles. ECU-types are shown as white boxes with a *T* inside. The worst-case execution times of a software function on a specific hardware type is represented as a scope icon.

To validate the different search methods of section II, an exhaustive search was implemented. The monetary cost of ECUs, communication cost and processor utilisation are used as metrics. In parallel, 100 random solutions are created and hill-climbed.

B. Industrial size model

The industrial size model uses the same deployment language as shown in Figure 4. We defined two approximation levels where bad solutions can be pruned:

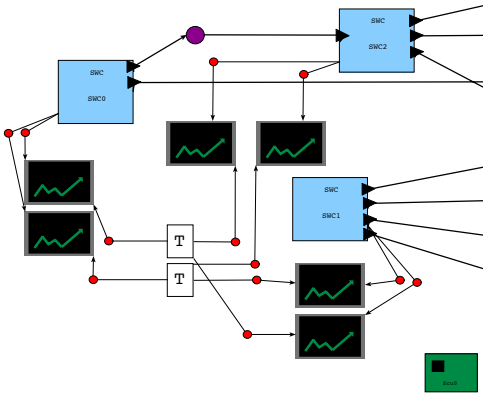

Fig. 4. A small fraction of the case study start model

- **Architecture Level:** At the architecture level, we assign the different software functions to an ECU and assign a type to the ECUs. We can calculate the classical schedulability test, defined by Liu and Layland [11], to prune infeasible solutions. Optimality is defined in terms of cost (based on the hardware cost, the extensibility of the solution (based on computation power left with a type penalty for the slow processor), and communication cost.
- **Scheduling Level:** At this approximation level the priorities are assigned to the tasks, signals are packed to messages and messages are assigned a priority. Optimality is defined by minimizing the end-to-end latencies of the sum of all the paths within the application.

The FTG+PM of the case study is shown in Figure 3. On the first approximation level, 1000 random solutions are created. These solution points are hill-climbed. The hill climber only accepts feasible solutions that are equal on all goal functions and at least better on one goal function. From the solutions found after the first approximation level, the Pareto optimal solutions are selected and transformed to a MILP representation. This MILP representation is executed and the results are transformed back to the deployment model representation.

C. Transformations

We give a brief overview of the transformation rules involved in this case study. The transformation rules for randomly searching the design space consist of:

- Ecu2Type: This rule maps an ECU to a type. The first part of the rule selects an unmapped ECU and assigns it a pivot. The second part of the rule assigns the selected ECU (using the pivot) to a type. The rule is executed until all ECUs have type information.
- Task2Ecu: This rule, shown in Figure 5, maps an unmapped software function to an ECU. As with the previous rule it is executed until all software functions are mapped to an ECU.
- CheckFeasible: This transformation rule checks whether all software functions are mapped to an ECU and that no ECU has a load bigger then 69%.

Hill climbing is done with two rules that can move a software function from one ECU to another ECU (Figure

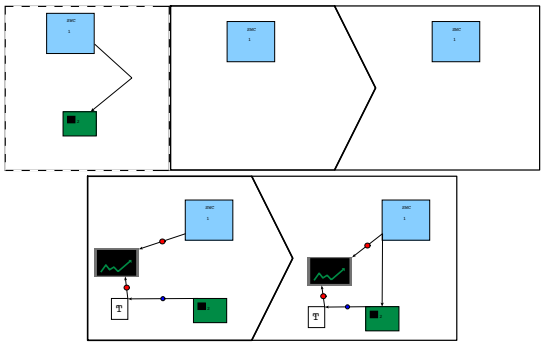

Fig. 5. An example rule to match a software function to an ECU

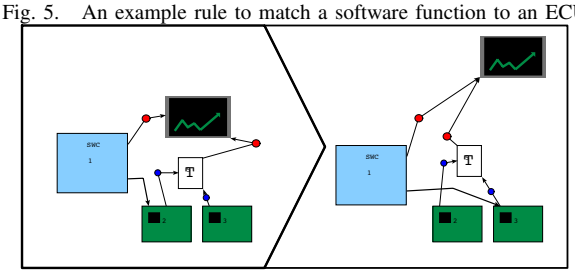

Fig. 6. An example rule to match a software function to an ECU

6). For the evaluation of a single solution point, we use transformations as well. One rule calculates the total cost and the total load of the system. The second rule calculates the total communication cost. Note that this could also be done by transforming the model to another language where analysis or simulation can be done.

Finally, three rules create a Mixed Integer Linear Program from a model after the architecture deployment step. We regard the execution of the MILP as a transformation as well since it manipulates the MILP-model and returns a set of traces containing the solution of the problem. Two rules transform the traces back to the original representation.

D. Results

For the small validation case study, the Pareto-curve of the exhaustive search and the Pareto-curve of the random and hill climbed solutions gave the same solutions as Pareto optimal. This is due to small size of the model where only 256 different solutions exist. Though the proposed techniques presented in Section II can be used to search through the design space.

The industrial size model has $2^9 \times 9^{40} = 7.57 \times 10^{40}$ possible solutions at the first approximation level. The created solutions before hill climbing are shown in Figure 7. After the hill-climbing, 254 unique local optima remained. These are shown in Figure 8. As can be seen, the search is maximizing the extensibility while reducing the communication cost. This process used 7500 minutes of computation time on an 8 core Intel Xeon processor running at 2.66 GHz. This is due to the size of the model, though our focus was not the performance of the approach.

At the second level of approximation, the number of possible solutions depends on the configuration after the first step. Nine Pareto front solutions got selected for the second part.

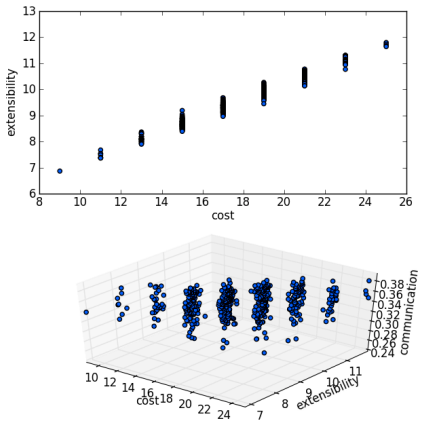

Fig. 7. The solutions before hill climbing

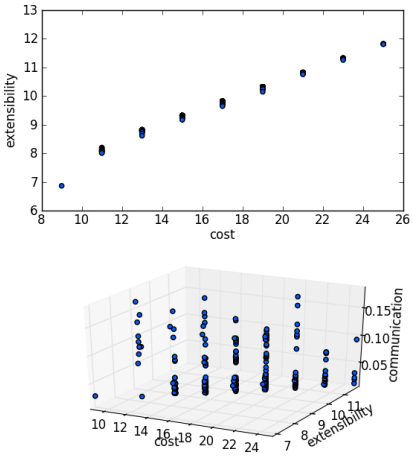

Fig. 8. The local optima after hill climbing

The transformations and execution of the second approximation level used 10 minutes, on the same machine, to produce the optimal solution. The nine solutions are scheduled all within the deadlines of the case-study.

V. Discussion

In this section we discuss some of the issues and opportunities of having a transformation based approach to design space exploration. The most limiting factor in our prototype search transformation model is the execution time. The most expensive operation in the transformation language is the matching algorithm. The complexity of this operation is exponential with respect to the size of the model. When using algorithms like hill climbing, a lot of matching restarts from the beginning while only a small portion of the model has been changed. To increase the speed of the matching operation, an incremental approach to model transformation can be used as proposed in [12]. Parallelism can also be used to speed up the process, since a lot of the branches in the approach can in fact be executed in parallel. Other techniques like pivots and scoping can be used to define regions where to start the search for matches or to define a specific region where to optimize, reducing the load on the matching process.

On the other hand different opportunities exist besides those mentioned in the introduction. The transformations make domain knowledge explicit but they can also encode domain knowledge already known by the domain and/or integration experts. For example, when it is known that certain software functions have to be mapped together, a rule can be written that encodes this knowledge. Other domain knowledge can be discovered by mining the traces of the transformations. This can uncover the sensitivity of parameters, where the change of certain parameters has more effect then others. These are the choices that should be focussed on during design space exploration. The mining of the traces can also be used to uncover domain knowledge, for example when certain choices always lead to good or bad solutions.

VI. Related Work

Transformation based approaches to Design Space Exploration are relatively new topics in the field.

The DESERT tool-suite [13] provides a framework for design space exploration. It allows an automated search for designs that meet structural requirements. Possible solutions are represented in a binary encoding that can generate all possibilities. A pruning tool is used to allow the user to select the designs that meet the requirements. These can then be reconstructed by decoding the selected design. In [14], Saxena and Karsai present an MDE framework for general design space exploration. It comprises of an abstract design space exploration language and constraint specification language. Model transformation is used to transform the models and constraints to an intermediate language. This intermediate language can then be transformed to a representation that is used by a solver. As in our approach, a set of solvers can be supported by using model transformations. Though our approach combines different optimization steps where the process of exploration is defined explicitly. The constraints in our approach are made explicit in the transformation rules and not in a separate language. Schätz et al. developed a declarative, rule-based transformation technique [15] to generate the constrained solutions of an embedded system. The rules are modified interactively to guide the exploration activity. In [16] a transformation-based approach is proposed to generate the full design-space of a cyber-physical system. The transformation language is based on Answer-Set Programming. Different approximation levels are introduced where non-feasible solutions can be pruned. In [17], a framework for guided design space exploration using graph transformations is proposed. The approach uses hints, provided by analysis, to reduce the traversal of states. The OCTOPUS toolchain [18] is a domain specific tool for the design space exploration of embedded systems. The tool is organised around an intermediate language used for connecting different tools together. The FTG+PM is used for the same purpose but uses an alternative approach without the need for an intermediate language.

VII. Conclusions

In this paper, we have shown that it is feasible to implement design space exploration through the usage of model transformations. An expressive transformation language can be used to implement meta-heuristics in the transformation models. This is complimented with the FTG+PM to combine different heterogeneous techniques at different levels of abstraction or approximation. In the approach, the design space exploration step is integrated in the MDE development cycle with the benefit of having a uniform process. This results in documented, explicit design space exploration models that can be reused for later design variants, documentation, etc. The approach was applied to an automotive case study, yielding a set of Pareto-optimal solutions.

In future work, we plan to integrate incremental matching techniques in the T-Core transformation language to speed up the matching process. Other work will focus on integrating other techniques like Simulated Annealing, Heuristic Search, etc.

References

[1] AUTOSAR Consortium, "The AUTOSAR standard," www.autosar.org.

[2] CESAR, "Cost-efficient methods and processes for safety relevant embedded systems," http://www.cesarproject.eu/.

[3] iFEST, "industrial framework for embedded systems tools," http://www.artemis-ifest.eu/.

[4] E. Syriani and H. Vangheluwe, "De- / Re-constructing Model Transformation Languages," *Electronic Communications of the EASST Ninth International Workshop on Graph Transformation and Visual Modeling Techniques*, vol. 29, 2010.

[5] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Design & Test of Computers*, vol. 18, no. 6, pp. 23–33, 2001. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=970421

[6] K. Tindell and A. Burns, "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems*, 1992. [Online]. Available: http://www.springerlink.com/index/J426140471414475.pdf

[7] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability analysis of fixed-priority systems using timed automata," *Theor. Comput. Sci.*, vol. 354, no. 2, pp. 301–317, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2005.11.019

[8] Anonymous, "The Formalism Transformation Graph as a Guide to Model Driven Engineering," A University, Tech. Rep., 2012.

[9] Anonymous, "An Overview of Model Transformations for a Simple Automotive Power Window," A University, Tech. Rep., 2012.

[10] W. Zheng, Q. Zhu, M. D. Natale, and A. S. Vincentelli, "Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems," *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 161–170, Dec. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4408301

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, Jan. 1973. [Online]. Available: http://www.cs.ru.nl/~hooman/DES/liu-layland.pdf

[12] G. Bergmann, A. Ökrös, I. Ráth, D. Varró, and G. Varró, "Incremental pattern matching in the viatra model transformation system," *Proceedings of the third international workshop on Graph and model transformations - GRaMoT '08*, p. 25, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1402947.1402953

[13] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts, "Constraint-based design-space exploration and model synthesis," in *Embedded Software*. Springer, 2003, pp. 290–305. [Online]. Available: http://www.springerlink.com/index/h5c60mea3vx9g2q0.pdf

[14] T. Saxena and G. Karsai, "MDE-based approach for generalizing design space exploration," in *Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 46–60. [Online]. Available: http://www.springerlink.com/index/C620L7531771M18U.pdf

[15] B. Schätz, F. Hölzl, and T. Lundkvist, "Design-Space Exploration through Constraint-Based Model-Transformation," in *2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. IEEE, 2010, pp. 173–182. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/ECBS.2010.25

[16] J. Denil, A. Cicchetti, M. Biehl, P. D. Meulenaere, R. Eramo, S. Demeyer, and H. Vangheluwe, "Automatic Deployment Space Exploration Using Refinement Transformations," *Electronic Communications of the EASST Recent Advances in Multi-paradigm Modeling*, vol. 50, 2011.

[17] A. Hegedus and A. Horváth, "A model-driven framework for guided design space exploration," in *Automated Software Engineering (ASE), 2011*, no. i, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6100051

[18] T. Basten and E. V. Benthum, "Model-driven design-space exploration for embedded systems: the octopus toolset," in *LEVERAGING APPLICATIONS OF FORMAL METHODS, VERIFICATION, AND VALIDATION, LNCS*. Springer, 2010, pp. 90–105. [Online]. Available: http://www.springerlink.com/index/P4760221932026M3.pdf