Design-Space Exploration in Model Driven Engineering
–An Initial Pattern Catalogue–

Ken Vanherpen[1], Joachim Denil[1,2],
Paul De Meulenaere[1], and Hans Vangheluwe[2,3]

[1] CoSys-Lab, FTI, University of Antwerp, Belgium
[2] AnSyMo, FWET, University of Antwerp, Belgium
[3] MSDL, McGill University, Canada
{ken.vanherpen, joachim.denil, paul.demeulenaere}@uantwerpen.be
hv@cs.mcgill.ca

Keywords: Constraint/Rule-Based, Evolutionary Algorithm, DSE, MDE, MILP

Abstract. A designer often has to evaluate alternative designs during the development of a system. A multitude of Design-Space Exploration (DSE) techniques exist in the literature. Integration of these techniques into the modelling paradigm is needed when a model-driven engineering approach is used for designing systems. To a greater or lesser extent, the integration of those different DSE techniques share characteristics with each other. Inspired by software design patterns, we introduce an initial pattern catalogue to categorise the embedding of different DSE techniques in an MDE context. We demonstrate their use by a literature survey and discuss the consequences of each pattern. Finally, we demonstrate the application of our initial pattern catalogue on two examples.

1 Introduction

Model-Driven Engineering (MDE) uses abstraction to bridge the cognitive gap between the problem space and the solution space in complex software system problems. To bridge this gap, MDE uses models to describe complex systems at multiple levels of abstraction, using appropriate modelling formalisms. Model transformations play a key role in MDE to manipulate models. Transformations are used for code synthesis, mapping between models at the same or multiple levels of abstraction, etc. Model transformation is even regarded as the "heart and soul of model-driven software and system development" [1].

While designing a system, the need often arises to explore different design alternatives for a specific problem. Design Space Exploration (DSE) is an automatic process where possible alternatives of a particular design problem are explored. The exploration is guided with imposed constraints and optimality criteria on the different candidate solutions. In the literature a multitude of design-space exploration techniques are available, for example (Mixed Integer) Linear Programming, evolutionary algorithms and constraint satisfaction.

In our experience with embedding DSE in a model-driven engineering context and a survey of the literature, we observed the use of different models, expressed using different formalisms, for both design, exploration and the modelling of goal functions. Combining the different models, using transformations, with the multitude of techniques available for searching design-spaces revealed similarities between the models and transformations of the different exploration techniques. To consolidate this knowledge, we organise these methods into an *initial pattern catalogue*, inspired by software design patterns. The goal of this effort is to create a more complete pattern catalogue for model-driven engineering approaches for design-space exploration with the support of the community.

The remainder of this paper is structured as follows. Related work is elaborated in Section 2. Section 3 introduces the Initial Pattern Catalogue. In Section 4, we discuss other useful techniques for DSE in an MDE context. Finally, Section 6 concludes our contributions and elaborates on future work. This technical report discusses some case studies, as an elaboration of our work presented in [2].

2 Related Work

The concept of patterns is widely used in Software Engineering. They provide generalized solutions to common software problems in the form of templates. The templates can be used by software developers to tackle the complexity in a larger software problem. One of the most highly cited contributions to pattern catalogues in the field of software is the work of the "Gang of Four" [3], which presents various design patterns with respect to object-oriented programming. Inspired by the Gang of Four, Amrani et al. [4] presents a model transformation intent catalogue which identifies and describes the intents and properties that they may or must possess. Their catalogue can be used for several purposes such as requirements analysis for transformations, identification of transformation properties and model transformation language design. Their presented catalogue is a first attempt to introduce the concept of patterns in MDE.

A more in-depth literature study is integrated in Section 3 such that each pattern is illustrated by known uses. This motivates one to the application of the introduced patterns.

3 Initial Pattern Catalogue for DSE

In this section we first discuss the need for a pattern catalogue specific to the Design Space Exploration domain. Next, our proposed pattern structure is described by analogy with the seminal work of the "Gang of Four" [3]. Finally, Subsections 3.2 and 3.3 will elaborate our *initial pattern catalogue*.

3.1 The need for patterns

By definition design patterns are used to formalise a problem which recur repeatedly. They help a designer to evaluate alternatives for a given design problem

in order to choose the most appropriate design. The usefulness of such patterns has already been proven in the Software Engineering domain where the "Gang of Four" [3] gave impetus to the creation of a widely accepted software design patterns catalogue. The successful impact of its widespread use is undoubtedly the well defined structure of each pattern. More specifically, each pattern is typed by: (1) Pattern Name and Classification, (2) Intent, (3) Also Known as, (4) Motivation, (5) Applicability, (6) Structure, (7) Participants, (8) Collaborations, (9) Consequences, (10) Implementation, (11) Sample Code, (12) Known Uses and (13) Related Patterns. Each of these sections is textually described and where necessary graphically supported using Class Diagrams, describing structure, and/or Activity Diagrams, describing the workflow of the pattern. At least one case study demonstrates how the patterns can be applied in practice.

In accordance to software design patterns, we define the format of each proposed pattern as follows. **Intent:** Gives a short explanation of the intention of the pattern. **Structure:** Describes the general structure of the pattern. **Consequences:** Describes the trade-offs in using the pattern. **Known Uses:** Lists the applications of the pattern in the literature. While this is not meant to be an exhaustive literature review of all the applications of the pattern, one can draw inspiration from these examples to apply the pattern. **Application:** Gives a short description in which cases this pattern can be useful and how it can be implemented.

The ***Structure*** is graphically supported by the Formalism Transformation Graph and Process Model (FTG+PM). The left side of the FTG+PM clearly shows all involved formalisms (boxes) and their relations using model transformations (circles). The right side shows the process with the involved models (boxes), transformed by a model transformation (roundtangle). Complex dataflow (dashed line) and control-flow (full line) relations can exist in the process part of the FTG+PM. This can be summarized as a legend, which is shown in Figure 1. The reason behind this latter supported formalism is threefold: (1) It clearly represents the structure of the approach by connecting the different formalisms with transformations on the left side of the FTG+PM. The FTG+PM also shows the workflow of combining the different models and transformations in a process on the right side. (2) The FTG+PM can be used to (semi-)automatically execute the defined transformation chains (yellow coloured). Manual operations are also possible that allow for experience based optimisation and design (grey coloured). (3) Different patterns described in this formalisms are easily connected to each other. This enables the embedding of DSE within the MDE design of systems.

As mentioned in section 1, we would like to refer to our technical report [2] where we apply our *initial pattern catalogue* to some case studies.

3.2 Exploring Design Spaces

Performing design-space exploration in a model-driven engineering context can be abstracted in some steps: (1) A meta-model defines the structural constraints of a valid solution. (2) A DSE-tool generates valid candidate solutions conforming

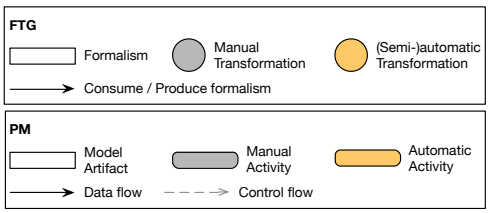

Fig. 1: FTG+PM Legend

to the meta-model. An initial model adds other structural constraints to the set of candidate solutions. (3) A transformation transforms the set of candidate solutions to an analysis formalism to check the feasibility of the solution with respect to a set of constraints. (4) If necessary, a second transformation generates a model in a performance formalism to check the optimality of the solution with respect to certain optimisation goals. (5) Depending on the optimisation technique, the process is iterated multiple times. Information from feasibility and performance models is used to guide the exploration.

Depending on the exploration technique, we classify different model-driven engineering approaches to solve this generic design-space exploration strategy.

Model Generation Pattern

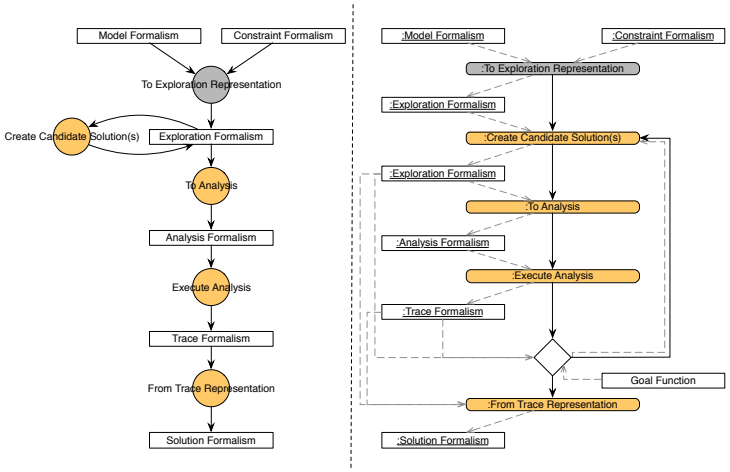

Fig. 2: Model Generation Pattern

Intent: This pattern transforms the meta-model of a problem space together with constraints to a constraint-satisfaction problem. The exploration of the design consists of the generation of a set of models that satisfy the structural

constraints imposed by the meta-model and the other constraints provided using a constraint formalism.

Structure: The pattern, shown in Figure 2, starts with a meta-model and some constraints. A transformation transforms these models into a constraint satisfaction problem. By invoking a solver, an exploration of the design space generates candidate solutions. Each candidate solution is transformed into an analysis representation. The analysis produces traces of each candidate solution. Based on the goal function model, the optimal trace is transformed to a solution model. This solution model can either be expressed in the exploration formalism, the original model formalism or a specific solution formalism.

Consequences: Depending on the used solver, this method might be computationally and memory intensive because an exhaustive search of the design space is executed. A transformation is needed to translate the meta-model with constraints to a model that is usable by the DSE-tool. Domain knowledge can be introduced by adding constraints to the meta-model. Note that adding extra constraints helps the search for a solution. An initial model, where some choices are predetermined, adds extra constraints. A less generic alternative is to add the initial model when evaluating candidate solutions.

Known Uses: Neema et al. [5] present the DESERT framework used for Model-Driven constraint-based DSE. It implements an automated tool which abstracts the Simulink design space to generate candidate solutions. In [6] the FORMULA tool is presented, where candidate solutions are generated from a meta-model. A similar tool called Alloy is used by Sen et al. [**?**] to automatically generate test models. Saxena and Karsai [7] present an MDE framework for generalized design-space exploration. A DSE problem is constructed out of a generalized constraint meta-model combined with a domain specific meta-model.

Application: The pattern is not recommended when one searches for an optimal solution out of a large search space without a lot of constraints. On the other hand, this pattern is very useful to rapidly obtain candidate solutions conforming to the meta-model.

Model Adaptation Pattern

Intent: This pattern transforms the model or a population of models to a generic search model used in (meta-) heuristic searches. Depending on the problem and search algorithm, different search representations can be used.

Structure: A model or population of models expressed in a certain formalism is transformed to a specific exploration formalism. Based on the guidance of a goal function, an algorithm creates new candidate solutions. A (set of) candidate solutions are transformed to an analysis model in order to evaluate. Finally, the result is transformed to a solution model. This solution model can either be expressed in the exploration formalism, the original model formalism or a specific solution formalism.

Consequences: A dedicated search representation has to be created as well as manipulation functions to create alternative designs. This requires an adequate understanding of the problem and domain knowledge. A translation from the

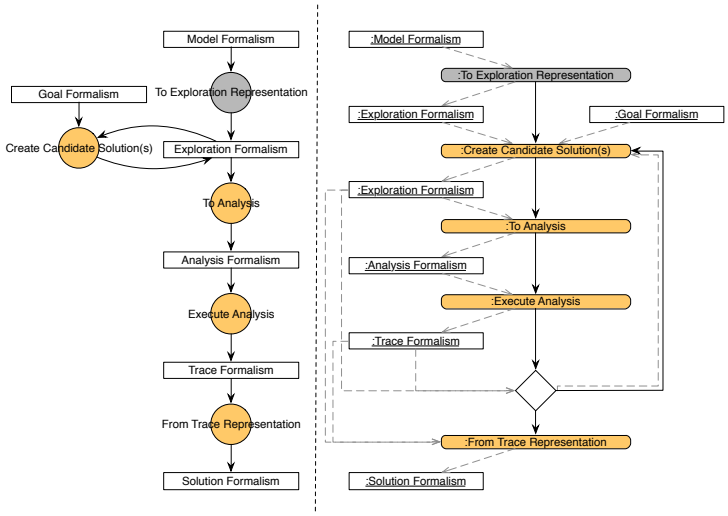

Fig. 3: Model Adaptation Pattern

problem domain to the search representation and vice-versa is required. An initial model, as a constraint, can be added by fixing the generated solution or by rewriting the functions to create new solutions (cross-over, mutation, etc.).

Known Uses: Williams et al. searched for game character behaviour using a mapping to a genetic algorithm [8]. Burton et al. solve acquisition problems using MDE [9]. Genetic algorithms are used to create a Pareto front of solutions. A stochastic model transformation creates an initial population. Finally, Kessentini and Wimmer propose a generic approach to searching models using Genetic Algorithms [10]. The proposed method is very similar to the described pattern.

Application: This pattern is recommended when a design problem can easily be transformed to an optimal search representation, e.g. a list or tree representation. Different operations on this new representation are implemented in the solution space (usually a generic programming language). Well-known algorithms, like genetic algorithms and hill-climbing, implement the search.

Model Transformation Pattern

Intent: This pattern uses the original model to explore a design-space. Model transformations encode the knowledge to create alternative models. Guidance to the search can be given by selecting the most appropriate next transformation or by adding meta-heuristics to the model transformation scheduling language.

Structure: A model combined with a goal function is used to create a set of candidate solutions that are expressed in the original model formalism. These are transformed to an analysis representation to gather some metrics that are expressed by a trace. Using (meta-)heuristics, a new set of candidate solutions

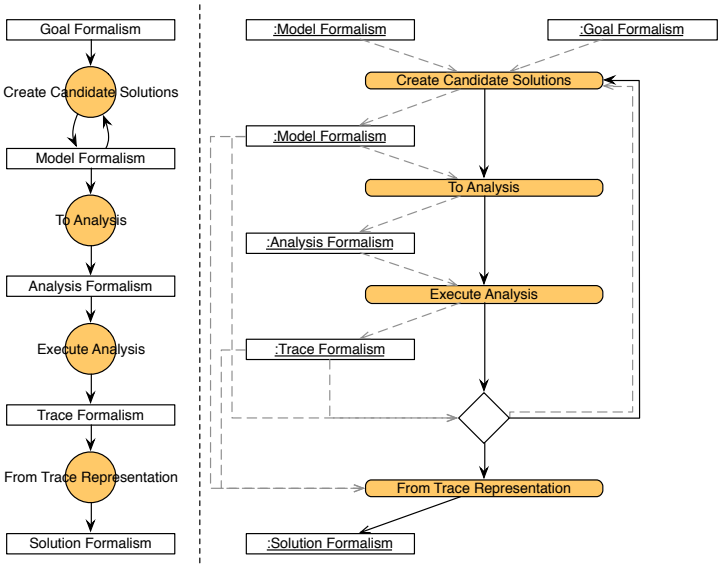

Fig. 4: Model Transformation Pattern

can be generated according to a goal function. Finally, if required, the most optimal solution or set of solutions can be transformed into a solution model.

Consequences: A high degree of domain knowledge about the problem is required to design the transformation rules. On the other hand, the rules encode domain knowledge to guide the exploration. Model-to-model or model-to-text transformations are required to evaluate a candidate solution. An initial model as a constraint can be added by adjusting the meta-model with variation tags. Similarly to the *Model Adaptation Pattern*, the initial conditions can also be implemented as fix operations using model transformations. Model transformations to create new candidate solutions are computationally expensive because of the subgraph isomorphism problem.

Known Uses: In [**?**] a model-driven framework is presented for guided design space exploration using graph transformations. The exploration is characterised by a so called exploration strategy which uses hints to identify dead-end states and to order exploration rules. This way the number of invalid alternatives is reduced. Denil et al. [11] demonstrates how search-based optimization (SBO) techniques can be included in rule-based model transformations.

Application: The pattern is used when it is hard to obtain a generic search representation. Model transformation rules, expressed in the natural language of the engineer, are implemented using current model transformation tools. Guidance is implemented through the scheduling of the model transformation rules.

3.3 Exploration Chaining Pattern

In order to prune the design space more efficiently, multiple of the proposed patterns can be chained. This technique is called "Divide and Conquer" and may as well be described by a pattern. To represent the chaining of multiple FTG+PMs, this pattern is graphically supported by means of a principle representation.

Intent: This pattern adds multiple abstraction layers in the exploration problem where candidate solutions can be pruned. High-level estimators are used to evaluate the candidate solutions and prune out non-feasible solutions and solutions that can never become optimal with respect to the evaluated properties. Figure 5 shows the overall approach of this pattern.

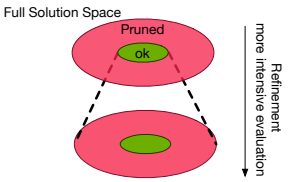

Fig. 5: Exploration Chaining Pattern

Structure: At each of the abstraction layers an exploration pattern is used to create and evaluate candidate solutions. Non-pruned solutions are explored further in the next exploration step.

Consequences: Domain knowledge about the problem is required to add levels of abstraction. High-level estimators are needed at each of the abstraction layers to evaluate a candidate solution. Because more information is introduced at each of the abstraction layers, the evaluation of a single candidate solution becomes more complex and usually more computationally intensive. Finally, a pruning strategy is required to decide what solutions have to be pruned at each of the abstraction layers.

Known Uses: Sen and Vangheluwe add different levels of abstraction in the design of a multi-domain physical model [12]. This numerically constraints the modeller to create only valid models. Kerzhener and Paredis introduce multiple levels of fidelity in [13]. Finally, multiple levels of abstractions for an automotive allocation and scheduling problem are introduced in [14].

Application: This pattern provides a solution when memory and time complexity are an issue during the exploration of the design space. It tackles the complexity by its layered pruning approach. Therefore, this pattern is preferred when searching for (an) optimal solution(s) in a large search space. Different exploration patterns are chained to create solutions.

4 Discussion

In this section we describe some other techniques that are useful for design-space exploration in a model-driven engineering context. Some techniques could potentially become a pattern in a new version of the catalogue.

Dealing with Multiple Objectives : Multi-objective optimisation deals with the decision making process in the presence of trade-offs between multiple goal functions. Certain DSE and search algorithms can deal with multi-objective functions

by construction. However, some techniques do not have this features. Here we give two ways of dealing with the problem.

Scalarize the Objective-Function: When scalarizing a multi-objective optimisation problem, the problem is reformulated as a single-objective function. The goal function model becomes a combination of individual objective functions. A model defines how the combination of the different individual goal function models is done, for example in a linear fashion, or other more complex functions.

Create Variants: In certain cases the designer would like to compare the different trade-offs using a Pareto curve. We use the scalarizing pattern to create multiple variants of the combined objective function. Intermediate results of the exploration are used to select an appropriate recombination that could potentially add a new Pareto solution.

Meta-model reduction: By using sensitivity analysis of the involved modelling elements and parameters, the meta-model can be reduced with the elements and parameters that have a small influence on the result of the goal function. An example of this technique can be found in [15].

5 Examples

In this section we illustrate the implementation of the previously described patterns by means of two examples. The first example searches for an electrical filtering circuit. The second example is a resource allocation problem in the automotive domain based on [16].

5.1 Electrical Network

We regard a filter design as a black box with an input, output and mass node with some passive electrical components in between which are connected to each other. When focussing on the exploration of passive analogue filters, those electrical components can be *Resistors*, *Capacitors* and *Inductors*. The corresponding meta-model is shown in Figure 6. Various configurations of those components lead to the construction of different types of filters. An example is a Low-Pass Filter (LPF) which passes low-frequency signals and attenuates signals with frequencies higher than the cut-off frequency (ω_c). A filter its behaviour is specified using a gain-magnitude frequency response, also called Bode plot, whereof an example is shown in Figure 7.

The goal of our DSE is to find a filter where its Bode plot has a minimal deviation compared to the theoretical one shown in Figure 7. Therefore, we see the deviation as a difference value between the theoretical and measured gain for each frequency point. In that case, the goal-function or fitness-function can be formulated by Equation 1. A larger deviation will result in a higher score, while a solution containing fewer components will result in a lower score.

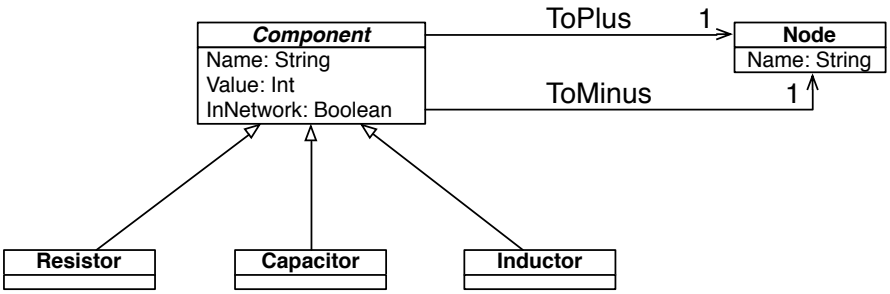

Fig. 6: Meta-model of a passive electrical design filter

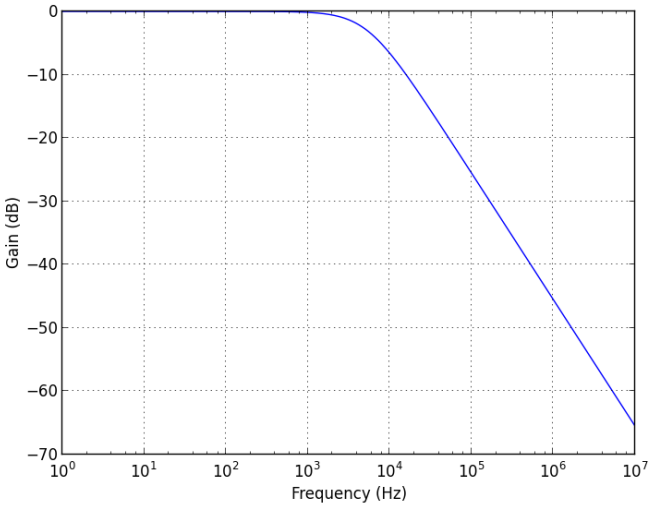

Fig. 7: LPF Bode plot

$$\text{Score} = (\text{Deviation} * 20) - \text{Number of components} \quad (1)$$

Which pattern of our proposed *initial pattern catalogue* can one apply to solve this design problem? Since the search space is quite large without a lot of constraints, we exclude the *model generation* pattern. This implementation has a very high memory and time complexity, resulting in a uncompleted exploration of the design space and thus a non-optimal solution. When choosing the *dedicated search representation* pattern, one will notice we are dealing with a design problem which is hard to transform to a generic search model such as a list or tree. The most appropriate pattern to implement this design problem is the *search using model transformation* pattern. The designer has to create a set of model transformation rules that apply mutation operations on the initial model. Example of such search rules are: adding or removing a serial or parallel connections, etc.

5.2 Allocation Problem

In the second example, we have a set of communicating software functions which need to be assigned to a set of Electronic Control Units (ECUs) connected by a communication bus. An example of such an allocation problem is shown in Figure 9. The corresponding meta-model is shown in Figure 8. It contains a *SWC* (Software Component) with a Name, a Period and a WCET (Worst Case Execution Time) describing the maximum time a task could be executed on a specific ECU. A *SWC* can be mapped to a single *ECU*. When a *SWC* is mapped to an *ECU* the Load attribute of the *ECU* is increased by (WCET/Period) of the mapped software function. A similar calculation is used for the Load on the Bus, based on the Size of the communication messages (*Comm*) and the Period of the sending software function. The Load on the *Bus* only increases when the sending and receiving *SWC* are mapped to a different *ECU*.

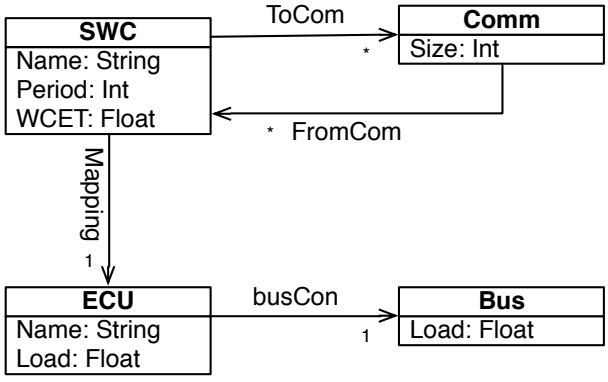

Fig. 8: Meta-model of the allocation problem

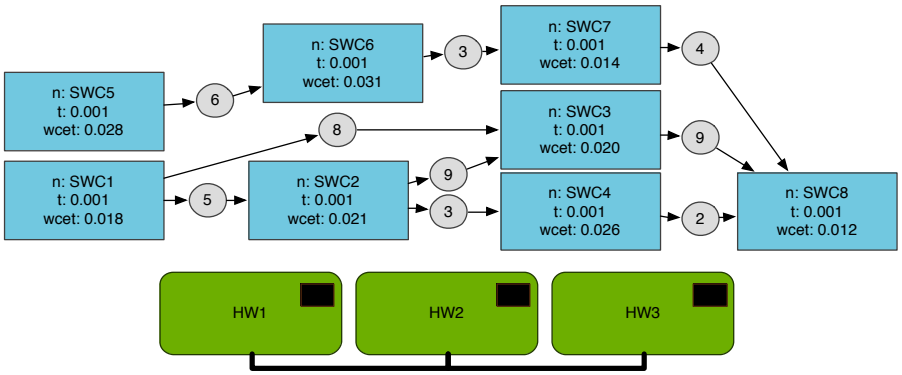

Fig. 9: Example of an allocation problem

Zheng et al. [16] approach the problem by searching for a mapping where the total load of the different ECUs are below a threshold of 69 % (the schedulability test for rate-monotonic systems [17]). The goal-function or fitness-function for finding an optimal solution is to minimise communication between the different ECUs because the communication on the bus introduces delays that impact the timing behaviour of the final solution. Therefore, the goal-function can be formulated by Equation 2. The penalty for unfeasibility is one order magnitude higher than the communication cost.

$$\text{Score} = \text{Eff. Communication Cost} + \text{Penalty} \tag{2}$$

This design problem lends itself for chaining two design patterns: the *model generation* pattern followed by the *dedicated search representation* pattern. Since the design problem can easily be transformed to a list representation, the choice of the *dedicated search representation* pattern is obvious. The list index defines the software component, the value defines the ECU on which the component is mapped. We use a genetic algorithm with a single cross-over point to generate new allocations. The preliminary *model generation* pattern is used for generating an initial population of models. This population of models is a precondition when searching for candidate solutions using Genetic Algorithms.

6 Conclusions and Future Work

Resulting from our own experiences with DSE and a literature survey, we presented an initial pattern catalogue which categorizes different approaches of Model-Driven Design Space Exploration. We described the patterns by the use of the FTG+PM to visualise the involved formalisms and their relations using model transformations. We demonstrated the use of those patterns by references to the literature and by means of two examples.

With the support of the community, it is our ambition to extend this towards a more complete this initial pattern catalogue, similar to the widely available software design patterns used in software engineering. Finally, we would like to investigate the parts of patterns that can be fully or partially automated.

Acknowledgements

This work has been carried out within the framework of the MBSE4Mechatronics project (grant nr. 130013) of the agency for Innovation by Science and Technology in Flanders (IWT-Vlaanderen).

References

1. S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *IEEE Software*, vol. 20, pp. 42–45, Sept. 2003.
2. K. Vanherpen, J. Denil, P. De Meulenaere, and H. Vangheluwe, "Design-Space Exploration in Model Driven Engineering: An Initial Pattern Catalogue," in *Proceedings of the First International Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA)*, co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), pp. 42–51, CEUR Workshop Proceedings (Vol-1340), September 2014.
3. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
4. M. Amrani, J. Dingel, L. Lambers, L. Lúcio, R. Salay, G. Selim, E. Syriani, and M. Wimmer, "Towards a Model Transformation Intent Catalog," in *Proceedings of the First Workshop on the Analysis of Model Transformations*, AMT '12, (New York, NY, USA), pp. 3–8, ACM, 2012.
5. S. Neema, J. Sztipanovits, and G. Karsai, "Constraint-Based Design-Space Exploration and Model Synthesis," pp. 290–305, 2003.
6. E. K. Jackson, M. Dahlweid, D. Seifert, and E. Kang, "Components, Platforms and Possibilities : Towards Generic Automation for MDA," 2010.
7. T. Saxena and G. Karsai, "MDE-Based Approach for Generalizing Design," in *Model Driven Engineering Languages and Systems*, pp. 46–60, Springer, Saxena2010.
8. J. R. Williams, S. Poulding, L. M. Rose, R. F. Paige, and F. A. C. Polack, "Identifying Desirable Game Character Behaviours through the Application of Evolutionary Algorithms to Model-Driven Engineering Metamodels," in *Proceedings of the Third International Symposium on Search Based Software Engineering*, pp. 112–126, 2011.
9. F. R. Burton and S. Poulding, "Complementing Metaheuristic Search with Higher Abstraction Techniques," *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pp. 45–48, May 2013.
10. M. Kessentini, P. Langer, and M. Wimmer, "Searching Models, Modeling Search," in *Proceedings of the 1st Workshop on Combining Modelling with Search-Based Software Engineering*, pp. 51–54, 2013.

11. J. Denil, M. Jukss, C. Verbrugge, and H. Vangheluwe, "Search-Based Model Optimization using Model Transformations," SOCS-TR-2014.2, School of Computer Science, McGill University, 2014.
12. S. Sen and H. Vangheluwe, "Multi-Domain Physical System Modeling and Control Based on Meta-Modeling and Graph Rewriting," in *IEEE Conference on Computer-Aided Control Systems Design*, pp. 69–75, Ieee, Oct. 2006.
13. A. A. Kerzhener and C. J. Paredis, "Combining SysML and Model Transformations to Support Systems Engineering Analysis," *Electronic Communications of the EASST 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)*, vol. 42, 2011.
14. 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.
15. B. Eisenhower, Z. O'Neill, S. Narayanan, V. a. Fonoberov, and I. Mezić, "A methodology for meta-model based optimization in building energy models," *Energy and Buildings*, vol. 47, pp. 292–301, Apr. 2012.
16. 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.
17. C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.