Multilevel Planning for Self-Optimizing Mechatronic Systems

Christoph Rasche
C-LAB
University of Paderborn
crasche@c-lab.de

Steffen Ziegert
Department of Computer Science
University of Paderborn
steffen.ziegert@uni-paderborn.de

***Abstract*—This paper presents a multilevel planning approach for the use in complex self-optimizing mechatronic systems. The approach has been designed for the RailCab system, which is an autonomous railbound transportation system. The term multilevel planning denotes planning on different levels of abstraction where each level involves different aspects and thus raises different planning tasks to solve. These planning tasks are not independent of each other. Plans for the higher level planning tasks involve reconfigurations of the system's software architecture and influence parameters used by the lower level planner to compute Pareto optimal behavior. Additionally, the higher level planner relies on plans computed by the lower level planner in order to meet the execution times (of system reconfigurations) that it assumed. To actually assure that the lower level planner computes Pareto optimal plans and takes multiple objectives (which are conflictive) into account, it is based on multi-objective optimization (with Pareto fronts as output).**

Keywords*—*hybrid planning; graph transformation; temporal planning; Pareto front; optimal planning

I. INTRODUCTION

The ever increasing complexity of mechatronic systems and the integration of more and more sophisticated functionality leads to new challenges for their design and development. The Collaborative Research Centre 614 "Self-optimizing Concepts and Structures in Mechanical Engineering" (CRC 614) at the University of Paderborn treats problems that occur during the design of complex mechatronic systems. The goal is to design self-optimizing systems, that are able to react autonomously to environmental changes by changing their parameters, as well as their objectives, if necessary. The developed concepts go far beyond simple control strategies.

To be able to test the approaches under real world conditions the RailCab system [1], an innovative autonomous railbound transportation systems has been built at the University of Paderborn. It consists of single RailCabs for the driverless transportation of passengers and goods while each vehicle drives on demand. The RailCabs are not coupled mechanically but convoys can be created in order to decrease energy consumption.

A highly complex system like the RailCab system involves various tasks that need to be achieved during runtime. These tasks involve behavior at different levels of abstraction. Each RailCab has an individual goal, e.g., transporting passengers or goods to a specified target station within a specified time. On a high level of abstraction RailCabs plan their route to the target station. Each route in the railway network is assumed to consist of a number of track segments, leading to a discrete planning task at this level. The choice of route and driving speed depends on the routes of other RailCabs in the system. Thus, this abstraction level includes planning of system reconfigurations, like the establishment or breakup of a convoy of RailCabs. These reconfigurations require precise timing information and affect the software architecture of the system, e.g., a convoy operation requires a communication link between the participating RailCabs.

The planning task on the lower level considers continuous behavior and parameters of a single RailCab that has to be planned according to external requirements. For example, a RailCab has to provide a certain level of driving comfort and must not run out of energy before reaching its destination. The fulfillment of these conflictive requirements depends on a variety of control parameters of the RailCab, for which the planner has to determine Pareto optimal settings. The approach is called hybrid planning because it uses an initial discrete plan and forecasts continuous system behavior by simulation during runtime.

Decisions on the higher level, like joining or leaving a convoy, also affect the lower level planning task: due to changing operating conditions, the parameter settings of the lower level planner have to be adapted online to still move with Pareto optimal settings. Since, the planning technique on the lower level refines the higher level actions into continuous behavior, it can guarantee that the next track segment is reached in exactly the time that the higher level temporal planner assumed. Thus, replanning on the lower level will not cause the higher level plan to be invalid afterwards.

This paper presents a hierarchical planning system for self-optimizing mechatronic systems. The temporal planning technique deployed on the higher level of abstraction has been published before in [2], but was treated in isolation only. Here, we specifically address the interrelation of this planning technique with a lower level planner and its ability to adapt its parameter settings online.

The paper is structured as follows. The temporal planning technique, which is used on the higher level of abstraction for planning routes and cooperative behavior, is introduced in Section II. Afterwards, the hybrid planning approach, which is based on multi-objective optimization and used as the lower level planning technique, is described briefly in Section III. The ability to adapt the environmental changes, like emerging drag or temperature changes during movement, is outlined in

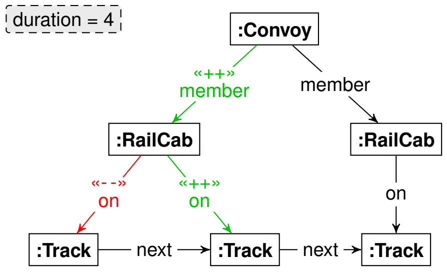

Fig. 1: Story pattern `joinConvoy`

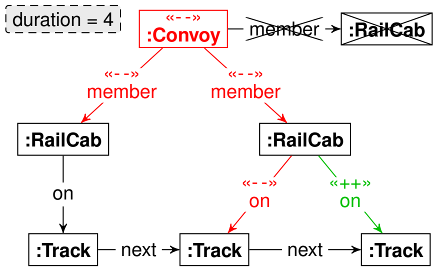

Fig. 2: Story pattern `breakConvoy`

Section IV. The remainder of the paper shows related work in Section V before giving a conclusion in Section VI.

II. Temporal Planning of Software Architecture Reconfiguration

We employ a model-based approach to the design of self-optimizing mechatronic systems. System models are given in MechatronicUML [3], a UML profile for the development of such systems. In the model for the RailCab scenario, the railway system consists of track segments that are connected to each other via `next` links. A RailCab that operates in the system can occupy such a track segment. Furthermore, RailCabs can coordinate with other RailCabs to form a convoy. To safely operate in a convoy, acceleration and braking has to be coordinated and managed between convoy members. Such an active convoy operation is represented by an instance of the `Convoy` type. A `Convoy` instance has a `member` link to each participating RailCab.

This approach to temporal planning deals with software architecture reconfiguration of self-optimizing systems. The communication behavior of components is not considered by this technique. Reconfigurations concerning the software architecture, e.g., the instantiation of a convoy, are modeled with story patterns [4], an extension of UML object diagrams.

Story patterns have a formal semantics based on (typed) graph transformation systems [5]. A graph transformation system consists of a graph representing the initial configuration of the system and a set of rules. Each rule consists of a pair of graphs, called left-hand side (LHS) and right-hand side (RHS), that schematically define how the graph representing the system's configuration can be transformed into new configurations. Elements that are specified in both graphs are preserved, other elements are deleted (if specified in the LHS only) or created (if specified in the RHS only). Syntactically, a story pattern represents such a rule by integrating the LHS and RHS into one graph and using stereotypes to indicate elements that are only present in the LHS or in the RHS.

Fig. 1 provides an example of a reconfiguration, which takes 4 time units: a RailCab joining a convoy of RailCabs. Objects and links that are being created or deleted by the application of the story pattern are labelled with the stereotypes «++» and «--», respectively. The story pattern specifies the creation of a `member` link representing the RailCab's participation in the convoy operation simultaneously with its movement to the next track segment. The story pattern can be executed to transform the state graph into a new configuration if it contains a subgraph that *matches* the LHS of the story pattern.

Our modeling formalism also allows to express that certain objects or links are not permitted to appear in the current state graph. See for example the story pattern given in Fig. 2. The crossed out `RailCab` object and the link connecting it to the `Convoy` object are not allowed to appear in the state graph. Such a restriction to the applicability of a story pattern is called a negative application condition (NAC).

In addition to the story patterns that define possible transformations, we need an initial configuration and a goal specification to feed the planning system with. A goal specification is a partly specified configuration that can be modeled as an ordinary object graph. Goal specifications are either generated from user input or predefined by the system designer. Initial configurations for the planning system are generated from actual runtime states of the system.

Consequently, we are interested not only in which graph transformations to execute but also in the points in time when a graph transformation is supposed to start. In this temporal planning approach, a plan is therefore a set of tuples of points in time and graph transformations. The graph transformations itself have annotated durations.

We solve these planning tasks by translating the models into the Planning Domain Definition Language (PDDL) [6] and feeding them into an off-the-shelf planning system, like SGPlan$_6$ [7]. In PDDL, a domain is defined by action schemata, as well as types and predicates that can be used within action schemata. An action schema consists of a list of parameters, a precondition, and an effect. In the precondition, a list of literals that are required for applying the action can be specified. Similarly, the effect of an action specifies a list of literals that are obtained when the action is applied. An action is instantiated – in the context of PDDL this is called *grounding* – by substituting the parameters with existing objects. Our translation scheme builds the declarations (of types and predicates) from the class diagram and generates an action schema for each story patterns.

In PDDL, action schemata for time-consuming actions split the literals used in their precondition and effect into different sets according to their time of evaluation. Literals can be required `at_start`, `over_all`, and `at_end` when used in the precondition and be effective `at_start` and `at_end` when used in the effect. The obvious approach to assume that the applicability check happens (in zero time) at the beginning of the reconfiguration and the actual change at its end is not suitable for many situations. Unintended interferences, e.g., the deinstantiation and use of a software component at the same time, could occur. The planning domain has to be generated in a way such that conflicts due to a concurrent execution of

```
60.041: (MOVE rc0 t16 t17) [4.0000]
60.042: (MOVECONVOY convoy0 rc2 rc3 t18 t19 t20) [4.0000]
64.043: (BREAKCONVOY convoy0 rc2 rc3 t19 t20 t21) [4.0000]
64.044: (MOVE rc0 t17 t18) [4.0000]
68.045: (MOVE rc3 t21 t22) [4.0000]
68.046: (CREATECONVOY convoy0 rc2 rc1 t19 t25 t26) [4.0000]
72.047: (MOVE rc3 t22 t23) [4.0000]
72.048: (MOVECONVOY convoy0 rc2 rc1 t25 t26 t27) [4.0000]
72.049: (MOVE rc0 t18 t19) [4.0000]
```

Fig. 3: Excerpt of a concurrent reconfiguration plan

reconfigurations are not possible. However, we do not want to burden the designer of the planning domain with its complicated and error-prone definition. Therefore, the questions that our translation scheme needs to address are: does the concurrent execution of two graph transformations result in any conflicts, and how can such a concurrent execution be avoided? To safely control whether a concurrent execution is allowed, our solution generates additional literals that *lock* access to graph nodes and edges when they are in use by a reconfiguration.

Consider for instance the application of the story patterns `joinConvoy` and `breakConvoy` given in Fig. 1 and 2. Let us assume that one of the reconfigurations, e.g., `breakConvoy`, is currently being applied. This means, its condition has already been checked but the alteration of the configuration has not yet been executed. The execution of a reconfiguration of RailCab `r1` joining the convoy makes no sense in this situation and should not be allowed because the convoy will be deinstantiated by `breakConvoy`. The problem is that the configuration is in the process of being changed, but this is not reflected in the intermediate state graph. Checking the applicability at the beginning of a reconfiguration and executing the alteration at its end is ineligible as a general solution. Our solution to this problem encodes information about the deinstantiation of the convoy into the configuration by acquiring a write lock of the `Convoy` object when the `breakConvoy` reconfiguration starts and releasing the lock when the reconfiguration ends. In the opposite case, i.e., if `joinConvoy` starts first, it encodes into the configuration that it requires the `Convoy` object by acquiring a read lock and releasing it when the reconfiguration ends. This approach is very suitable for a translation into PDDL since locking functionality can simply be realized by defining new predicates and functions for the locks. Since acquiring and releasing all locking literals of a reconfiguration is done as an atomic step (at the beginning and the end of the reconfiguration, respectively), there can be no deadlocks when acquiring the locks. For more details on the translation scheme we refer the reader to [2].

Our model includes story patterns to move RailCabs or convoys of RailCabs and story patterns related to convoy de-/instantiation and membership change. All these story patterns are available in the generated planning domain as action schemata.

Listing 3 shows an excerpt of a plan that was generated by SGPlan$_6$ for a planning task involving 4 RailCabs. During the interval [60–64], RailCabs `rc2` and `rc3` operate in convoy mode. From 64 to 68, they break up the convoy operation because the underlying domain specifies a Y junction between tracks `t19`, `t20`, and `t25`, and they need to move along different routes to arrive at their target locations. To do so, `rc2` has to fall back, i.e., it still occupies `t19` at 68. Concurrently, i.e., during the interval [60–68], `rc0` moves from `t16` to `t18` but waits from 68 to 72 to not crash into `rc2`. To sum up, the reconfiguration plans that SGPlan$_6$ produces take advantage of parallel execution of actions when possible, while guaranteeing that concurrently executed actions do not interfere with each other. With regard to the application scenario, this means that RailCabs operate in parallel if they are sufficiently apart from each other, but wait for the execution of other RailCabs' reconfigurations if necessary, e.g., to clear a common track segment.

III. Hybrid Planning

Besides the temporal planning approach for the coordination of several RailCabs, e.g., to create a convoy each RailCab computes single behavioral plans in order to move from its initial position to its destination with Pareto optimal settings on the lower level. Such plans are necessary as the RailCab has only limited energy resources. The passengers can, e.g., change the velocity of their RailCab, which influences the energy consumption.

A hierarchical hybrid planner is used for the purpose of creating behavioral plans [8]. This hybrid planning approach first computes an initial plan and forecasts continuous system behavior. The purpose is the creation and adaptation of a plan in order to always move with Pareto optimal settings from the start position to the destination. The problem of planning is modeled as a linear optimization problem and solved using the Simplex algorithm.

Such a plan has to consider conflictive objectives. These objectives have to be optimized which leads to a multi-objective optimization problem. A Pareto set is the solution set of such a problem and forms a $(k-1)$-dimensional object. In this context, k denotes the number of objectives involved in the problem. The computation of a single Pareto front for each track section is neglected in this paper. The interested reader is referred to a detailed description in [10].

A multi-objective optimization, using the program GAIO [9], [10] is performed, which results in Pareto fronts. Based on the Pareto fronts several Pareto optimal configurations of system parameters with different degrees of performance regarding each objective are provided to the planner. The planner then selects one of these sets of Pareto points for each track section. A plan then consists of a set of Pareto points leading to Pareto optimal settings at each track section.

The initially computed plan, based on the Pareto points computed before starting the journey, does not take into account the actual operating conditions during the journey, which might deviate from the a priory assumed conditions, e.g., due to environmental changes. Hence, only preliminary parameters can be taken into account for the computation of feasible plans.

Creating and dissolving convoys, for example, leads to states where the RailCabs do not move any longer with Pareto optimal settings due to a considerable change of drag. In a convoy each RailCab behind the leader moves in the slipstream of the RailCab in front of it. This changes the energy consumption at consistent movement speed dramatically for most of the RailCabs and a new plan considering the new energy consumption should be computed for every RailCab

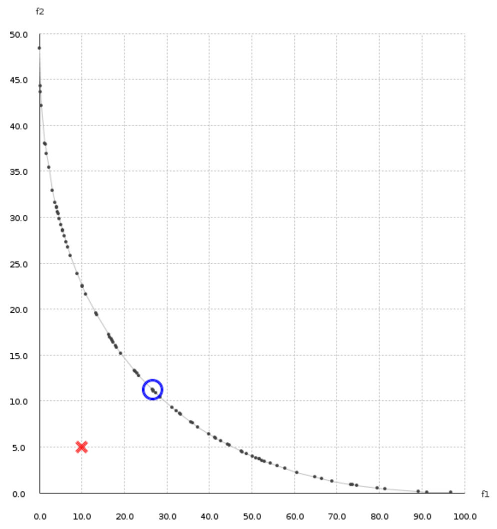

Fig. 4: An example Pareto front. The blue circle represents the Pareto point selected by the planner while the red cross shows the working point.

that is affected by these changes. Additionally, the energy consumption has to be decreased when moving in a slipstream as the current energy consumption would increase the speed of the RailCab and the timing assumptions, on which the temporal planner relies, would not be met.

The same is true, if a RailCab leaves a convoy. It then faces a higher drag and thus needs more energy to keep its speed constant. Therefore, an adaptation of the Pareto front is performed, as described in the following section. The planner then computes a new plan that is based on new parameters, which are calculated by the adaptation approach, in order to always work on valid data.

IV. Behavioral Adaptation of Differing Model Parameters

As mentioned, it is hardly possible to avoid deviations between the settings achieved by the use of previously computed Pareto fronts and the real settings, reached during operation. From this it follows that the Pareto points, on which the original plan is based, become invalid and thus the entire plan becomes invalid. To still be able to use a plan, which is as close to be optimal as possible, a change of the Pareto front has to be conducted.

Fig. 4 shows an example for a parameter change. The line denotes the Pareto front computed by the use of the model values and the blue circle shows the Pareto point selected by the planner. Each Pareto point considered in this paper has a comfort value, given by f_1 and an energy consumption value, given by f_2. The currently measured comfort and energy consumption leads to the working point denoted by a red cross. This example includes considerable changes in the energy consumption as well as between the measured comfort value and the comfort value given by the selected Pareto point. Such differences make the entire plan invalid and a recalculation, based on the newly determined working point must be conducted.

To be able to detect such deviations and to change the plan, several values like energy consumption and passenger comfort have to be measured continuously during RailCab operation. Based on the measured values the current working point has to be calculated.

It is not possible to simply compute new Pareto fronts, leading to the measured working point as the changes of the environmental parameters are not computable and it is possible that the current working point is not Pareto optimal. In that case no Pareto front, containing the working point exists. Thus, an iterative approximation of the model Pareto front towards the measurements is conducted.

A. *Taylor Series Approximation*

An approximation using Taylor Series expansion can be performed, to successively approximate a Pareto front given by model parameters closer to a working point, obtained by measured values [11].

The functions, used to compute Pareto fronts for an entire RailCab are rather complex and mainly a combination of the multi-objective functions for single systems, presented in [8], [12], [13] and [14]. The resulting Pareto points selected by the planner are n-dimensional. To adapt a Pareto point to a working point each dimension is considered individually.

Let x_0 be the Pareto point and let x_m be the working point. First, a Taylor series for each parameter p_i in dimension i of the multi-objective function is computed, using the partial derivatives as follows:

$$\begin{aligned} T_i(x) = f(x_0) &+ \frac{\frac{\partial f(x_0)}{\partial x_i}}{1!}(x - x_0) + \frac{\frac{\partial^2 f(x_0)}{\partial x_i}}{2!}(x - x_0)^2 \\ &+ \frac{\frac{\partial^3 f(x_0)}{\partial x_i}}{3!}(x - x_0)^3 + \cdots \end{aligned} \tag{1}$$

Additionally, the differences between the Pareto point and the working point for each parameter $\Delta p_i = \|x_0 - x_m\|_i$ are computed. These differences are used to compute new parameter values p_{i_n}, as shown in the following equation.

$$p_{i_n} = p_i + \frac{\Delta p_i}{T_i(x_m)} \tag{2}$$

The Pareto front is then newly computed, based on this new parameters and the procedure is started over again until no further reduction of the differences is achieved. The resulting Pareto front is close to the working point and based on the new environmental parameters. The planner then uses Pareto fronts, based on the newly determined parameters to conduct a replanning, leading to a new and feasible path.

B. *Result*

Fig. 5 shows an example for a recalculated Pareto front. The values used in this example are abstract values without units of measurement. Two simple functions were used to represent passenger comfort and an energy consumption at a specific movement speed. The functions lead to a two-dimensional Pareto front. The axis f_1 depicts the current energy consumption and the axis f_2 the current passenger comfort.

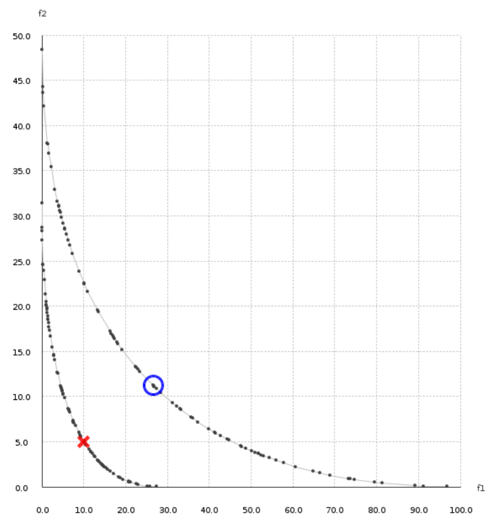

Fig. 5: Recalculated Pareto front based on measurement point. The blue circle shows the Pareto point selected by the planner. The red cross denotes the working point. The original Pareto front has been adapted towards the working point.

A low comfort value results in a high passenger comfort. In the depicted example scenario the planner selected the Pareto point at position (26.5,11.2) from the model Pareto front for the current track section, framed by a blue circle in order to reach the required comfort without consuming too much energy. The measured data revealed a current working point at position (10,5), which is framed by a red cross.

Such a difference between the Pareto point and the working point makes the overall plan, used to move to the destination invalid. In order to be able to compute a new plan an adaptation of the Pareto front towards the working point is conducted, as shown in Fig. 5. Based on these resulting values new Pareto fronts can be computed, used by the planner to create a feasible plan.

V. Related Work

The multilevel planning approach presented in this paper is based on the assumption that the planning tasks are inherently hierarchical and can be separated into higher level and lower level tasks. No interleaving between the planning techniques is necessary; thus, they can be applied sequentially on their respective abstraction levels (beginning at the highest level). For systems where a strict separation of the planning tasks is not possible, Marthi et al. [15] proposed an approach called *angelic* hierarchical planning. Their approach provides the higher level planner with models that allow to make guarantees on the satisfiability of the lower level planning tasks. This prevents the system from having to backtrack to a higher abstraction level, which results in a significant speedup.

When researchers try to tie AI planning techniques with the software engineering domain, their techniques often rely on graph transformation systems. Estler and Wehrheim [16] developed a heuristic search planner along with a technique to learn domain-specific heuristics from modeling artifacts (among other things from a meta-model). These techniques are promising because of their intuitive representation and close association to model-based software engineering. However, up to today, they usually do not support time-consuming reconfigurations.

Similar to our approach for the generation of temporal plans, Tichy and Klöpper [17] presented an automatic translation of graph transformation rules into PDDL actions to plan self-adaptive behavior. The support for time-consuming reconfigurations was addressed only in terms of stereotypes; concurrency issues were not treated. Our temporal planning technique can be seen as an extension of their approach.

The use of Pareto optimal solutions is a common approach for planning in multi-objective applications. Hongfu et al. [18], e.g., solve the multi-objective problem of intelligent mission planning in dynamic environments by a combination of Pareto fronts, receding horizon control, fuzzy inference systems and expert knowledge. In contrast to the approach presented in this paper they select a Pareto point from the computed front, based on expert knowledge.

Klöpper et al. [19] use Pareto based planning in multi-agent mechatronic systems. In their system operation strategies exist, that correspond to Pareto optimal configurations. These operation strategies represent trade-offs between the system objectives and expected system state changes regarding limited resources, as e.g., energy and execution time. The resulting Pareto optimal configurations are used as input to the planning model, which is based on a state-action formalism. They also use a hybrid planning approach, combining local planning and reactive behavior for decision making in real-time.

VI. Conclusion

We presented a multilevel planning approach for self-optimizing mechatronic systems that adapt itself to envoronmental changes. Many other planning approaches do not consider environmental influences. A plan is often computed a priori and followed by the autonomous system. Changes of the environment, like newly detected obstacles, can then force a replanning. Nevertheless, most times it is assumed that parts of the environment, like weather, temperature, etc., have no influence to the system. There are several applications for which such an assumption does not hold. The presented RailCab system is such a system that is directly influenced, e.g., by drag changes which influences the energy consumption.

In our approach, Pareto optimal plans cannot be created a priory due to unpredictable environmental influences, like emerging headwind or temperature changes during movement. Such influences lead to changing parameter values that have to be taken into account. Therefore, the Pareto front that the initial plan is based on is adapted online towards a measured working point. After such an adaptation, the planner replans using the updated parameter values.

In addition, there are planning tasks on a higher abstraction level that are solved by an independent planning system. Plans for the higher level planning tasks involve reconfigurations of the system's software architecture and influence parameters used by the lower level planner to compute Pareto optimal behavior. This higher level temporal planner allows to execute system reconfigurations in parallel. Its planning tasks are solved by translating them into a PDDL representation that can be handled by off-the-shelf planning systems.

Our combined approach is a first step towards a self-optimizing system that computes temporal reconfiguration plans, which change the software architecture of the system and solve conflictive objectives. Furthermore, it is able to adapt its control parameters to environmental changes like changing drag. In theory, it can be applied to several applications that need to take temporal properties and environmental changes into account.

Acknowledgment

This work was developed in the course of the Collaborative Research Centre 614 "Self-optimizing Concepts and Structures in Mechanical Engineering" and funded by the German Research Foundation (DFG).

References

[1] C. Henke, M. Tichy, T. Schneider, J. Bocker, and W. Schafer, "System architecture and risk management for autonomous railway convoys," in Systems Conference, 2008 2nd Annual IEEE, April 2008, pp. 1–8.

[2] S. Ziegert and H. Wehrheim, "Temporal reconfiguration plans for self-adaptive systems," in Software Engineering (SE 2013), ser. Lecture Notes in Informatics (LNI). Gesellschaft für Informatik e.V. (GI), February 2013.

[3] S. Becker et al., "The MechatronicUML design method – process, syntax, and semantics," Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, Tech. Rep., 2012.

[4] T. Fischer, J. Niere, L. Torunski, and A. Zündorf, "Story diagrams: A new graph rewrite language based on the unified modeling language," in 6th Int. Workshop on Theory and Application of Graph Transformations (TAGT 1998), 1998.

[5] H. Ehrig et al., "Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach," in Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations, G. Rozenberg, Ed. World Scientific, 1997, ch. 4, pp. 247–312.

[6] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," Journal of Artificial Intelligence Research (JAIR), vol. 20, 2003, pp. 61–124.

[7] Y. Chen, B. W. Wah, and C.-W. Hsu, "Temporal planning using subgoal partitioning and resolution in SGPlan," Journal of Artificial Intelligence Research (JAIR), vol. 26, 2006, pp. 323–369.

[8] N. Esau et al., "Hierarchical hybrid planning for a self-optimizing active suspension system," in 7th IEEE Conference in Industrial Electronics and Applications, IEEE. Singapore: IEEE, 18 - 20 Jul. 2012.

[9] M. Dellnitz, O. Schütze, and T. Hestermeyer, "Covering Pareto sets by multilevel subdivision techniques," Journal of Optimization Theory and Application, vol. 124 (1), 2005, pp. 113–136.

[10] O. Schütze, K. Witting, S. Ober-Blöbaum, and M. Dellnitz, "Set oriented methods for the numerical treatment of multi-objective optimization problems," in EVOLVE – A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation, ser. Studies in Computational Intelligence, E. T. et al., Ed. Springer Berlin Heidelberg, 2013, vol. 447, pp. 187–219.

[11] J. Li, H.-C. Zhang, and Z. Lin, "Asymmetric negotiation based collaborative product design for component reuse in disparate products," Computers & Industrial Engineering, vol. 57, no. 1, 2009, pp. 80–90.

[12] C. Romaus, J. Bocker, K. Witting, A. Seifried, and O. Znamenshchykov, "Optimal energy management for a hybrid energy storage system combining batteries and double layer capacitors," in Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE, September 2009, pp. 1640–1647.

[13] A. Trachtler, E. Munch, and H. Vocking, "Iterative learning and self-optimization techniques for the innovative railcab-system," in IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on, November 2006, pp. 4683–4688.

[14] A. Pottharst et al., "Operating point assignment of a linear motor driven vehicle using multiobjective optimization methods," in Proc. of the 11th International Power Electronics and Motion Control Conference (EPE-PEMC 2004), 2004.

[15] B. Marthi, S. J. Russell, and J. Wolfe, "Angelic hierarchical planning: Optimal and online algorithms," in Int. Conf. on Automated Planning and Scheduling (ICAPS 2008), 2008.

[16] H.-C. Estler and H. Wehrheim, "Heuristic search-based planning for graph transformation systems," in Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2011), 2011, pp. 54–61.

[17] M. Tichy and B. Klöpper, "Planning self-adaptation with graph transformations," in Int. Symp. on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2011), 2011.

[18] H. Liu, X. Gu, J. Chen, and H. Liu, "Intelligent multi-objective receding horizon control for ucav mission planning," in Computer Science and Information Processing (CSIP), 2012 International Conference on, August 2012, pp. 1154–1158.

[19] B. Klöpper, S. Honiden, and W. Dangelmaier, "Divide & conquer in planning for self-optimizing mechatronic systems - a first application example," in Computational Intelligence in Control and Automation (CICA), 2011 IEEE Symposium on, April 2011, pp. 108–115.