Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic

Jan Winkels[1(✉)], Julian Graefenstein[2], Tristan Schäfer[1], David Scholz[2], Jakob Rehof[1], and Michael Henke[2]

[1] TU Dortmund, Otto-Hahn-Str. 12, 44227 Dortmund, Germany
{Jan.Winkels, Tristan.Schaefer, Jakob.Rehof}@tu-dortmund.de
[2] TU Dortmund, Leonhard-Euler-Str. 5, 44227 Dortmund, Germany
{Graefenstein, Scholz, Henke}@lfo.tu-dortmund.de

Abstract. Increasing competition, stronger customer focus, shorter product lifecycles and accelerated technological developments imply that companies are faced with the challenge of adapting their own production to the circumstances at ever shorter intervals. The factory planning project is becoming increasingly complex, but there is less and less time available for adaptation. Particularly in the initial planning phase, targets are defined without reliable planning information for the further course, which have far-reaching consequences for the outcome of a successful planning. This paper shows a possibility to generate meaningful solution alternatives at an early stage of the target planning in order to enable an efficient planning process in terms of time and costs. With the help of a constraint-based variant compilation on the basis of previously defined target and frame parameters as well as existing information on the current factory system, various possible solution variants for target planning are to be created. A specific use case scenario was used to develop and test the presented methodology. By comparing combinations of the most diverse possible solutions, the use of a combinatory logic approach enables the first rough and plausible solution variants to be generated automatically, on the basis of which the detailed planning process for achieving the determined solution variant can be created. This way, planning bottlenecks due to the wrong choice of variants as well as large time expenditure for the creation of solution variants can be avoided.

Keywords: Automatic composition · Combinatory logic · Factory planning
Target planning

1 Introduction

When initiating a new factory planning project, it is assumed that the status quo of the factory system is not suitable for future requirements. Due to changing framework conditions and influencing factors which directly affect companies, the factory system

T. Margaria and B. Steffen (Eds.): ISoLA 2018, LNCS 11247, pp. 487–503, 2018.
https://doi.org/10.1007/978-3-030-03427-6_36

has to be adapted in order to remain competitive. In a first step, it is necessary to determine which target state has to be achieved. Correspondent with the company's goals, project objectives are developed to provide a specific framework under certain conditions for the future planning project [1].

However, the strategically oriented goals from the management level cannot be transferred directly to an operative project planning. Figure 1 shows the different levels of influences and responsibilities which are involved in a factory planning project. Each level has different topics to deal with which cumulates in specific information relevant for the following level. For example, a legal guideline might be the restriction to use different materials because of ecological concerns. These materials were used before by the company and now have to be replaced with other materials that follow these new legal guidelines and laws. An example for an objective of the corporate management might be to address a wider range of wealthier range of customers which accepts a much higher price for a product that fulfills certain ecological requirements. In order to follow these objectives of the strategic management in combination with the legal guidelines that have to be followed, the operational management, the level of "Solution variants", uses these cumulated information formulated by every preceding level to consider different solution variants. The last level is responsible for generating a planning process workflow based on the solution variants worked out before which transfers the chosen variant in specific tasks.

An intermediate stage is required in which the management compares the corresponding goals with the strategic corporate goals and defines specific objectives for the planning project in cooperation with the project management [2].

The target planning phase is intended for this purpose in the context of several factory planning procedures [3–5]. Within this first planning phase, individual planning scenarios are worked out which can fulfill the respective objectives and other framework conditions. However, the task of target planning is not responsible for developing implementation measures for the solution variants. These implementation measures are considered by the factory planning procedures to be developed in the following, rather operationally shaped planning phases [2, 5].

It is a requirement of the target planning to consider possible developments and future scenarios in order to develop suitable solution variants [6]. However, such solution variants are initially developed on a very rough basis. They are elaborated with the help of creative methods such as brainstorming or scenario techniques and are based on fuzzy information and knowledge [5, 6]. On the one hand, this is necessary in order to start the development of implementation measures and the general planning process as soon as possible in order to be able to begin with the implementation of the adaptation as quickly as possible. On the other hand, at this early stage a detailed elaboration of the problem definition is not possible due to missing or insufficient and, as already mentioned, fuzzy information [7]. This fast and less detailed and therefore very vague procedure brings along the danger of ending up in a planning dead-end and thus inefficiently working on the whole planning project [8].

If the percentages of cost responsibility of the respective planning phases are considered in a classical factory planning procedure (see Fig. 2), it becomes apparent that the target planning phase occupies half of the total financial volume. This means that decisions made in the target planning phase have a strong influence on the financial

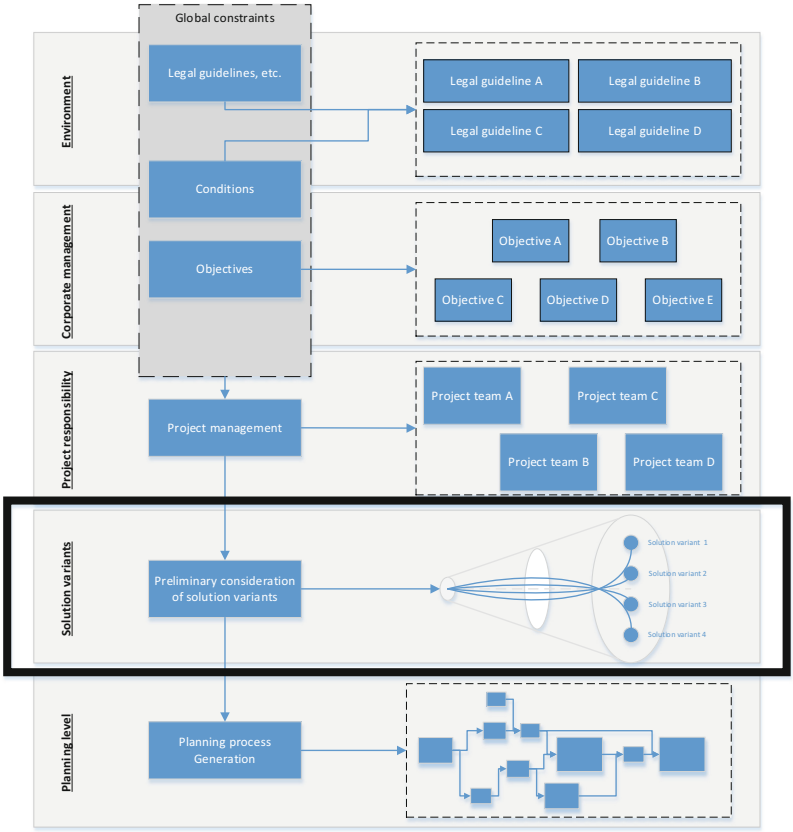

Fig. 1. Levels of responsibility in factory planning

dimension of the whole planning project. As mentioned before there are several factory planning procedures which consider the target or goal planning as a very first step of the whole planning procedure. In this early phase, however, decisions can be adjusted to avoid possible financial damage, which could occur later in the planning process. The decisions of the target planning thus have a significant influence on the successful completion of the project, as it lays the foundation for further procedures.

The objective of the target planning is to develop suitable rough solution variants based on different framework conditions and performance targets for the factory system. These variants are to serve as a guideline and orientation for successful planning. They specify a planning direction with corresponding tasks, which are further detailed in the following planning steps and phases. This first planning phase lays the foundation for the ongoing planning project and is a necessary basis for further decisions made in the following planning phases. Classical planning procedures in factory planning follow the principle of rough approximations to fixed specifications and also make it clear that the results of target planning are on a low level of detail [3, 5, 11].

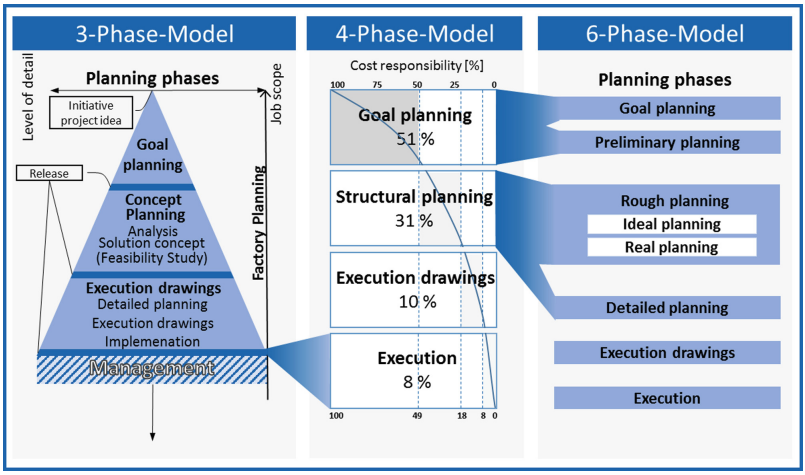

Fig. 2. Cost responsibility in factory planning projects [3, 9–11]

In order to achieve the objectives of the target planning, specific topics are addressed and worked out. First, a definition of the problem for the present planning project must be elaborated, on which the subsequent planning tasks can refer to. In addition, the framework conditions for the project must be clarified. These include not only legal requirements, but also requirements that are in line with corporate management requirements, such as the planning period, the budget for planning or available human resources. The persons involved in the planning team are decisive, as they provide a certain solution space for the development of alternative solution variants based on their individual knowledge and background. This can result in a restricted number of variants because of their individual creativity and knowledge.

Based on the objectives set by the company's management and the identified framework conditions, options of action for formulating solution variants can now be developed. Options for action represent individual aspects of a solution variant, which result as a combination in a possible scenario for achieving the defined targets. The solution variant created by the project team and then agreed on by the management is the aforementioned intermediate stage between the strategic and operative level and the basis for the more detailed planning in the following planning phases (see Fig. 3). An example of a solution variant could be to increase the output of the production system by changing the existing layout. Therefore the layout has to be changed in a way that several segments have to be rearranged because of additional machines and the shift schedule might be necessary to reconsider etc. The details concerning in which way the goals and tasks will be achieved and processed and how to develop the necessary information will be part of the following phases.

Figure 3 shows on the left side the classic approach of formulating and considering several solution variants which then results in evaluating and choosing a specific variant which seems to be the best possible option for the planning project. As mentioned before several information and framework conditions are considered to derive

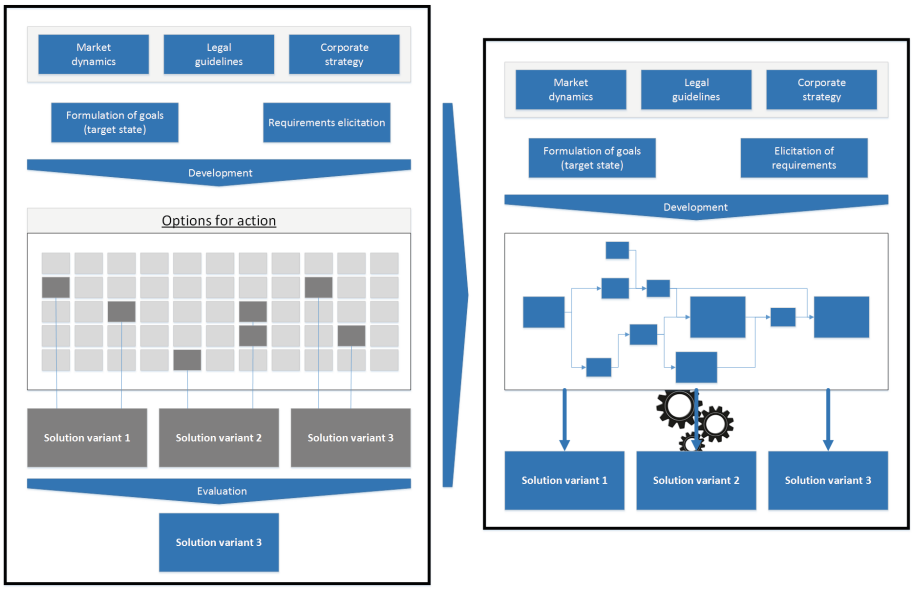

Fig. 3. Classic and new approach for generating solution variants [10]

options of actions. These options of action will then be combined to form individual solution variants based on the creativity and knowledge of the participating project members. There is a vast number of possible variants that can be formed out of the options for action, which cannot be handled in its entirety by some participants on their own. Because of the huge variety of possible combination of options, it might be too complex to formulate every existing solution for the specific project by hand. The process of combining several options for action to a specific solution variant is very hands-on and therefore the variants have a high risk of creating mistakes and future problems in the ongoing planning process.

To optimize this process of formulating solution variants and in combination with that to reduce the risk of failure and possible, high costly problems for the whole planning project the planning phase of target planning needs to be reconsidered in the way this phase is handled. Therefore, the method of combinatory logic will be used to accelerate the generation of high quality and reliably working solution variants, which is illustrated on the right side of Fig. 3. This paper will show the benefits of a fast, reliable and automated generation of solution variants with the help of combinatory logic based on a specific use case. With the help of combinatory logic, it will be possible to reduce influences of individual creativity and knowledge of all participants in order to reduce the possible problems that come with these factors. This will also improve the following planning process as it points out specific intermediate planning tasks, which can be used to generate an optimized planning process workflow for the whole project. Section 2 summarizes related work for the research fields factory planning and software synthesis. The synthesis is performed by making use of the Combinatory Logic Synthesizer (CL)S that is outlined in Sect. 3. These preliminaries

are followed by a description of the use case, selected implementation details and a discussion of the results.

2 Related Work

Little attention has been paid so far to the research field of target planning as part of the wider field of factory planning [6, 12]. There are only few approaches to optimize the phase of target planning and therefore the development of rough solution variants. Approaches that are more specific can be found in the field of general corporate and strategic planning. Some recent works describe the problems of target planning within the context of factory planning and present approaches for a suitable solution space management along the factory planning process, as well as the scenario analysis in the target planning and the maturity level management of factory planning itself [6, 8, 13]. However, only a few other approaches were developed which specifically address the target planning as part of the factory planning and therefore try to improve and optimize this planning phase.

The phase of goal definition and variant formation represents a small part of the whole factory planning process and does not receive the necessary attention as the percentage of cost responsibility suggests (see Fig. 2). Within the scope of corporate and strategic planning there are more specific approaches which clarify the classification of the target planning of a factory planning project as a link between strategic corporate planning and the operative implementation of the planning project [4, 14, 15]. In addition, it becomes clear that creativity techniques such as the already mentioned scenario technique, SWOT analysis, brainstorming and cognitive methods are being used to an increasing extent for the development of solution variants [16–19]. It also emerges from the strategic planning that the focus of academic research is on the interaction of the responsible planners with adjacent disciplines and not on the optimization of developing solution variants as a main aspect [19, 20]. New approaches for the automatically supported development of solution variants are not in focus. Approaches for the automated compilation of planning tasks within the scope of factory planning have already been shown, but not yet specifically transferred to the field of target planning [21]. In the area of software engineering, however, there are quite a number of approaches that also use the combinatory logic approach like it is shown in this paper, but not with a specific application reference to the field of factory planning.

There are various approaches to the problem of software synthesis that can be distinguished by Search Space, User Intent and Search Technique. These aspects form the *dimensions in program synthesis* [22, 23].

The program to be synthesized contains domain relevant code. Thus, the dimension of search space is strongly connected to the research field of domain modeling. Accordingly, the specification of the search space can be achieved by using a Domain-Specific Model [24]. The development of these models is usually supported by domain specific modeling tools. The additional design effort for these tools can be compensated by the CINCO meta modeling framework [25–27]. Its applicability has been shown by the development of DIME [28]. The user intent is a formulation of the desired program. Common formalisms to express its properties are temporal logic [29–31], first-order

logic [32] or higher-order logic [33]. Moreover, synthesis can be performed according to a set of input/output examples [34, 35]. The search technique determines how the synthesis algorithm is looking for the target program within the search space. There are several approaches that incorporate constraint solving [36], the use of semantic reasoners (i.e. a graph-based search technique) [37] or neural networks [38].

Synthesis can be a powerful technique to support planning tasks, yet only a few studies deal with automated code generation in this domain. The work in [39] demonstrates the synthesis of domain-specific hierarchical task network (HTN) planning software. Planware [40, 41] is a system to synthesize algorithms that produce optimal domain-specific schedules. The synthesis of workflows can also be considered as a part of planning synthesis. The concept proposed in [29] enables for the iterative refinement of the domain specification until a suitable process can be generated. In [37], automated runtime flexibility for software engineering processes is provided by incorporating a semantic reasoner and ad-hoc changes of process instances featured by AristaFlow [42]. As opposed to these studies, the approach for target planning presented in this paper is formally sound and complete. It is particularly well suited to handle synthesis tasks with an emphasis on features and it produces a list of possible solutions that can be further analyzed by experts.

The component-based software synthesis is a powerful technique which can formalize variability in product lines by making use of feature models. Feature modeling and Software Product Lines (SPL) [43–45] are closely related topics that represent individualization and standardization of software. They both contribute to shortening development life cycles of software products by facilitating the reuse of software while comprehending a systematic management of software product families. Research resulted in the programming paradigm feature oriented software development which is supported by comprehensive development environment [46].

Variability modeling is a well developed research topic in the context of business process modeling and still receives continued attention. In 2017, an overview to existing approaches in this field was given in a comprehensive survey by La Rosa, van der Aalst et al. [47].

3 Combinatory Logic Synthesizer

The Combinatory Logic Synthesizer ((CL)S, [48]) is a framework to compose software components or data structures according to their type signature. In this paper, (CL)S is used to form rough solution possibilities for target planning. Thus, its formal foundation will be outlined in this section to provide the reader with an elementary understanding of the theoretical background.

The framework determines a result term in the form of an applicative composition in compliance with combinatory logic. Applicative terms are defined as $e ::= x|(e\,e')$, where x ranges over a denumerable set of variables. The type system of (CL)S is based on intersection types [49].

Type expressions are denoted σ, τ and defined as follows:

$$\sigma, \tau ::= a|\omega|\alpha|\sigma \to \tau|\sigma \cap \tau$$

Type constants are ranged over by a, b, c, … and they can be programming language types (native types) or textual descriptions (semantic types). The special type constant ω is the root element of the subtyping relation. Type variables are ranged over by α, β, γ, … and they are substituted with type constants according to a substitution map. This map is part of the domain specification (i.e. not part of type expressions) and will be used to resolve type variables prior to the computation of an inhabitant. Additionally, type expressions can contain function types (→) and intersections (∩).

(CL)S performs the software synthesis by solving the type theoretic problem of inhabitation. It is often abbreviated as $\Gamma \vdash ? : \tau$ and asks if a term with type τ exists under the type assumptions stated in Γ. The propositions-as-types correspondence [50, 51] connects the type theory with programs. According to the propositions-as-types correspondence, a proof term M for the type judgement $\Gamma \vdash M : \tau$ must be a program or data structure that conforms to the target type τ. Thus, solving the inhabitation problem can be interpreted as type-directed software synthesis.

The synthesis is not building code from scratch but making use of software components held in a repository Γ. The components are also referred to as combinators as they will be composed in a combinatory way. The repository is a finite set that contains a type assumption for each combinator: $\Gamma = \{(x_1 : \tau_1), \ldots, (x_n : \tau_n)\}$ with $x_i \neq x_j$ for $i \neq j$ where $x_1, \ldots, x_n$ represent combinator names and $\tau_1, \ldots, \tau_n$ are the corresponding type assumptions. During inhabitation, the combinator's type is used to determine which elements can be applied to each other in order to satisfy the target type. The current inhabitation algorithm is proven to be complete and enumerates all inhabitants.

In addition to the type signature, implementation details can be provided for combinators. They can contain programs, data, data fragments or functions. Moreover, variability points can be inserted and described with the type expression. The use of type taxonomies and type variables further supports the comprehensive specification of complex combinators. An algorithm to decide the inhabitation problem for intersection types is described in detail in [52].

The component-based synthesis with intersection types can be classified according to the *dimensions in program synthesis* [22]. Domain knowledge can be expressed by the semantic layer and corresponding combinator implementation. The search space is defined by well-formed applicative compositions of available combinators. The inhabitation algorithm represents the search strategy and the user intent must be supplied as a target type expression.

(CL)S has been used to synthesize BPMN 2.0 business process descriptions in interaction with the Combinatory Process Synthesis framework (CPS, [53]), showing that the synthesis approach is suitable to generate structured data. Recent research puts the emphasis on language independent code generation using meta code generation [54]. For this study, the Scala based version of the (CL)S was used[1]. The underlying

[1] The source code of the (CL)S framework is available at https://github.com/combinators/cls-scala.

formalism extends the Bounded Combinatory Logic (BCL, [52]) by introducing n-ary type constructors and demanding the use of substitution maps for type variables. Compared to BCL, this results in higher expressiveness while type variable mappings require a more precise specification.

4 Specific Industrial Use Case for Testing the Approach

To build a test model and to review the developed code, a real planning scenario of a company in the manufacturing industry was used as a basis.

The planning impulse was triggered by the company management. Due to a changing customer structure with correspondingly different products and sales figures, the production system had to be changed. First, framework conditions and targets were formulated at the level of corporate management in order to set a direction for the final design of the planning. The main points were:

- No changes to existing production lines.
- Employees will not be laid off or hired.
- The property or the existing built-up area will not be changed.
- The factory hall is not extended.
- A certain sales figure must be produced with the existing manufacturing equipment.

Further information on these framework conditions and targets were provided in order to be able to develop rough solution variants according to these specifications. First, a sales volume could be determined that differed from the previous one. The demands by the customer of each individual product are to be more volatile than before, which requires increased flexibility for the new system. Likewise, the changed product variants result in increased storage requirements in order to be able to cover the correspondingly greater variety of products.

The goals and general conditions set by the corporate management led to the decision that the biggest variable that would fulfill the requirements and goals was the planning of a new logistics concept. The number of employees as well as the existing structure of the building and production lines should remain unchanged. It was checked whether the production lines and the respective equipment were technically capable of manufacturing the new products. As this could be guaranteed, the focus was on logistics as a variable to be changed. Storage types as well as the conveying means including the corresponding connections to the respective operating means of the production lines had to be redesigned. After it was decided that a new rough logistics concept should be developed, all relevant parameters were identified in order to be able to meet the targets with the new concept (see Fig. 4). The main parameters were the capacity of the storage types, the needed space in the layout and the costs of each logistic element. Based on these parameters three rough solution variants could be generated, which differed in the financial expenditure along other specifics.

For the algorithm, individual parameters, such as type of storage like Kanban racks or high rack store, as well as conveying equipment such as classic forklifts or automated guided vehicles were identified and analyzed. These parameters were subdivided into small groups with their individual attributes such as cost rates, area requirements or

capacities, so that these values could be transferred into a form that can be processed by the algorithm (see Fig. 4). With this breakdown of individual solution elements, individual options for action could be broken down into many smaller elements. This should allow a more diverse generation of solution variants without being limited by the planner and the solution space of his participating personal. In addition, the respective attributes of each element should allow a more detailed compilation according to specific criteria such as costs or used space in the factory.

5 Implementation and Experimental Results

In this section, we will use the previously presented use case as a practical framework for evaluating our approach to automated composition of factory configurations. The scenario is well suited to illustrate process synthesis because it is well-structured and contains inherent variability. The inputs for the CL(S) are available as an accompanying download[2]. It is the aim to use the synthesis algorithm described in Sect. 3 to automatically show which configuration options can be used for the use case and under which conditions they can be implemented. In order to be able to map the variability and the numerous different configuration possibilities in the present scenario, the scenario was converted into a feature model. Feature models are originally a representation of all occurrences of a software product line (SPL). An SPL is a collection (or family) of related programs that are based on a common software kernel but differ in features. A "feature" is defined as a "salient or distinguishable user-visible aspect, quality or characteristic of a software system" [55]. Feature models are visualized through feature diagrams and used throughout the product line development process. The model defines the features, their characteristics, as well as their dependencies, which are reflected in the diagram.

In addition, the models can have other constraints, which can be represented in additional documentation (tables, etc.). A concrete incarnation of a member of an SPL is called a feature configuration. A configuration is only allowed if it does not violate any constraints described in the model. The concept of the software product lines has been adapted to the present work and its underlying scenario. It will be introduced and used as a production product line (PPL) at this point. In our scenario, different variants of individual components of the material flow are considered as features. Different transport systems for example represent different factory features. Configuration is allowed if at least one feature is selected from all the required components and global constraints such as the budget limit are not violated. The scenario model is shown in Fig. 4.

The model shows the existing variation possibilities. For example, it is possible to combine different transport systems. Each system has individual characteristics, such as specific costs, throughput or floor space requirements.

In the next step, the feature model was transformed into combinators for synthesis. The individual combinators are shown in Fig. 5. The name of the combinator follows

[2] https://james.cs.tu-dortmund.de/smjawink/CLS-FactoryConfig.

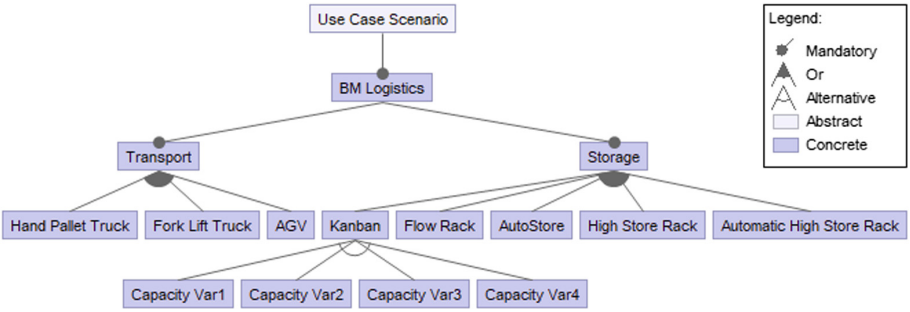

Fig. 4. Feature model

the "prefix" + "name of the feature to be selected" structure. Possible prefixes are "storage-Selector" and "transportSelector" to indicate the corresponding branch of the Feature Model Tree. The combiner for selecting an AutoStorage system is therefore called "storage SelectorAutoStore". In principle, there is a combiner for each entry in the feature tree whose signature can be filled in by various forms of its child nodes. As a result the existing combinator "configuration" needs an expression of the transport and storage selector to be able to be executed. This is equivalent to the entry BA-Configuration in the Feature Model.

Note the two combinators addTransport and AddStorage, which are intended to increase the number of transport or storage systems used. For instance, if you want to use 2 Forklift trucks, the combinator selectTransport (ForkLift) would be executed twice and then combined with the addStorage combinator to form a transport system. With this construction, however, there is a problem: Since the constraints are not checked until after the synthesis algorithm has been completed, it can theoretically happen that the addStorage combinator could be back-interchanged infinitely, resulting in an infinite number of solutions. To prevent this, the maximum depth of the resulting tree grammar in the implementation has been limited.

The inhabitation is executed with a call of the form:

```
lazy val resultsFromRequests: Results =
Results.add(Gamma.inhabit[Form](‘FactoryConfig
(‘AutoStore)).
```

This means that the algorithm is asked if it is possible to generate a solution that meets the required specifications from the given repository Γ. In the example above, the use of an AutoStore system is explicitly specified. For the experiment in the present scenario, no restrictions of this kind were specified in order to enable as many possible solutions as possible. All other constraints were taken directly from the given scenario in order to make the solutions applicable to it.

The solution set is then displayed in a web interface where the user can view and evaluate the solutions. The web interface displaying the solutions of the experimental run can be seen below.

```
Γ = {                          configuration:  (String → String → String → Form) ∩
                                               (Title → NameTransport(a) → NameStorage(b) → OrderMenu)
                       storageSelectorKanban1:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(Kanban1))
                       storageSelectorKanban2:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(Kanban2))
                       storageSelectorKanban3:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(Kanban3))
                       storageSelectorKanban4:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(Kanban4))
                     storageSelectorFlowRack:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(FlowRack))
                     storageSelectorAutostore:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(Autostore))
                  storageSelectorHighRackStore:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(HighRackStore))
              storageSelectorAutoHighRackStore:  (String → String) ∩
                                               (NameTransport(a) → NameStorage(AutoHighRackStore))
                   transportSelectorHandPallet:  (String → String) ∩
                                               (Title → NameTransport(HandPallet))
                  transportselectorForkLift:  (String → String) ∩
                                               (Title → NameTransport(ForkLift))
                    transportselectorAGV:  (String → String) ∩
                                               (Title → NameTransport(AGV))
                          combineStorage:  (String → String → String) ∩
                                               (NameStorage(a) → NameStorage(a) → NameStorage(a))
                          combineTransport:  (String → String → String) ∩
                                               (NameTransport(b) → NameTransport(b) → NameTransport(b))
                                 ConfigTitle:  String ∩ Title
    }

WF = {  {a → AGV}, {a → HandPallet}, {a → ForkLift}, {b → AutoHighRackStore}, {b → HighRackStore}, {b → Autostore}, {b →
        FlowRack}, {b → Kanban1}, {b → Kanban2}, {b → Kanban3}, {b → Kanban4}
    }
```

Fig. 5. Combinator repository overview

In the use case scenario, the following configuration was chosen as the solution: As a transportation system, an AGV System was selected, the storage system was an AutoStore rack. As Fig. 6 shows, this solution also appears in the solution set of the algorithm (solution number 5 of 24 in total). In addition, further alternative configurations can be seen.

These generated solutions differ in terms of various measures, such as the capacity or the number of employees required per shift. For instance, the usage of a forklift truck as a transportation appears to be much cheaper, but also offers a smaller amount of transportation capacity per hour. Which configuration is selected at the end also depends on how the responsible persons weight the individual parameters and which personal preferences they have and fits best to given case.

This shows that our approach does not only provide all possible solutions, but also evaluates them directly with regard to important key figures. This way, the responsible persons not only directly see which options are available, but can also directly see their pros and cons and thus make a quick and well-founded decision.

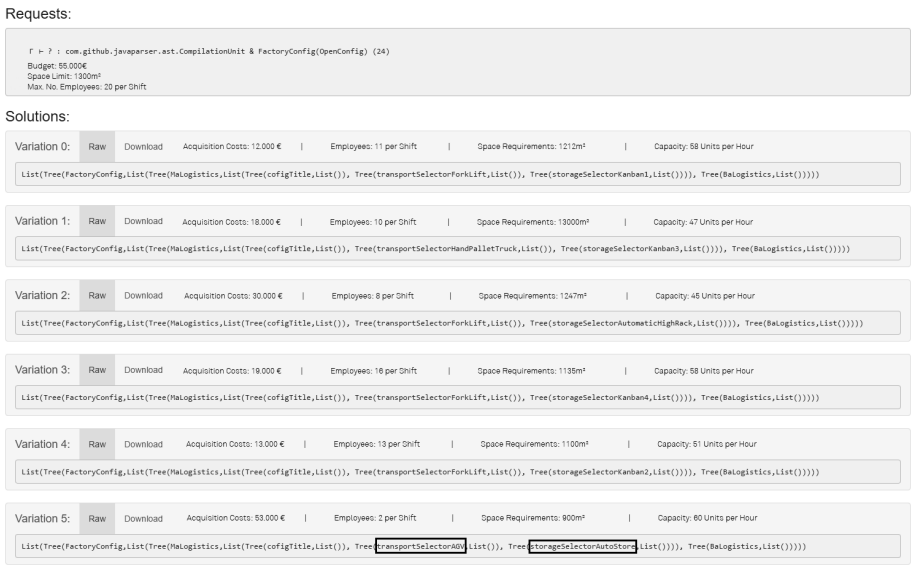

Fig. 6. Inhabitants of the experimental run

6 Conclusion and Outlook

It has been shown that plausible and useful solutions for target planning can be obtained with our approach. By offering a large number of different solutions and a large variability of different aspects of the solution, planners can make quick and reliable decisions based on the generated proposals and thus advance the planning process. Further, it is possible to support the following more detailed planning process by giving specific directions based on the chosen solution variant. Therefore the possible solution space for the whole detailed planning process can also be reduced so that a more precise planning process can be generated afterwards.

In order to further improve the technology and its benefits, various points of dependences are conceivable. Of course, the first starting point to consider is to support more complex decisions and other aspects of the planning process such as the ressource planning or the layout planning through the procedure. In addition to that, other areas where planning workflows are needed to manage complex planning projects could be addressed with the approach shown in this paper. An example would be the management of the construction of a new hospital building where a great amount of different information and dependences are interconnected with each other. It is necessary to be able to handle this complex kind of project and our approach would be beneficial in automatically generating sufficient planning workflows based on our produced solution variants.

The currently still hard-coded combinator repositories are also to be dynamized in the future. The development of a parser that automatically generates the combinators from a domain model is conceivable. In combination with modern domain modeling

tools such as DyWA [25], it would be possible for users without programming knowledge to design repositories and execute inhabitation requests.

Another starting point is to improve the quality of the solutions offered. By adding feedback loops and using machine learning methods, the tool aims to gain knowledge of generated solutions which were considered good and practicable. In the best case scenario, it should be able to learn which possible configurations are the best for certain requests.

Acknowledgement. The study presented in this paper was partly funded by the GRK 2193 (www.grk2193.tu-dortmund.de/de/) and the Center of Excellence for Logistics and IT (www.leistungszentrum-logistik-it.de/) located in Dortmund.

References

1. Brankamp, K.: Zielplanung. In: Eversheim, W., Schuh, G. (eds.) Produktion und Management 3. Gestaltung von Produktionssystemen. Hütte, pp. 9.31–9.39. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-58399-5
2. Both, P.V., Rexroth, K.: SIAS – Konzeption eines planungsunterstützendenWerkzeuges für die Zielplanung. In: Knoll, M., Oertel, B. (eds.) Dienstleistungen für die energieeffiziente Stadt, pp. 109–130. Springer Spektrum, Heidelberg (2012)
3. Aggteleky, B.: Fabrikplanung. Werksentwicklung und Betriebsrationalisierung, 2nd edn. Hanser, München (1987)
4. Pawellek, G.: Ganzheitliche Fabrikplanung. Grundlagen, Vorgehensweise, EDV-Unterstützung, 2nd edn. VDI-Buch. Springer, Berlin (2014)
5. Grundig, C.-G.: Fabrikplanung. Planungssystematik - Methoden - Anwendungen, 5th edn. Hanser, München (2015)
6. Baumeister, M.: Fabrikplanung im turbulenten Umfeld. Methodik zur Zielplanung einer Fabrik unter Berücksichtigung eines turbulenten Unternehmensumfeldes und der übergeordneten Unternehmensziele. Zugl.: Karlsruhe, Univ., Diss., 2003. Forschungsberichte aus dem Institut für Werkzeugmaschinen und Betriebstechnik der Universität Karlsruhe, vol. 115. Inst. für Werkzeugmaschinen und Betriebstechnik, Karlsruhe (2002)
7. Hawer, S., Ilmer, P., Reinhart, G.: Klassifizierung unscharfer Planungsdaten in der Fabrikplanung. ZWF (2015). https://doi.org/10.3139/104.111339
8. Hilchner, R.: Typenorientiertes Lösungsraum-Management in der Fabrikplanung. Zugl.: Aachen, Techn. Hochsch., Diss., 2012, 1st edn. Edition Wissenschaft Apprimus, vol. 2012,13. Apprimus-Verl., Aachen (2012)
9. Schulte, C.: Logistik. Wege zur Optimierung der Supply Chain, 7th edn. Vahlens Handbücher der Wirtschafts- und Sozialwissenschaften (2016)
10. Eversheim, W., Schuh, G. (eds.): Produktion und Management 3. Gestaltung von Produktionssystemen. Hütte. Springer, Heidelberg (1999)
11. Kettner, H., Schmidt, J., Greim, H.-R.: Leitfaden der systematischen Fabrikplanung. Hanser, München (1984)
12. Rexroth, K., Brüggemann, T., Both, P.V.: Methodology of target and requirements management for complex systems concerning the application field of an energy-efficient city. In: Schrenk, M. (ed.) REAL CORP 2009: cities 3.0 - smart, sustainable, integrative. Proceedings of 14th International Conference on Urban Planning, Regional Development and Information Society; Beiträge zur 14. Internatinalen Konferenz zu Stadtplanung, Regionalentwicklung und Informationsgesellschaft; [strategies, concepts and technologies for planning the urban future; 22–25 April 2009, Centre de Disseny de Sitges, Catalonia, Spain; Tagungsband], pp. 353–359

13. Krunke, M.: Reifegradmanagement in der Fabrikplanung. Dissertation, RWTH Aachen (2017)
14. Girmscheid, G.: Projektabwicklung in der Bauwirtschaft - prozessorientiert. Wege zur Win-Win-Situation für Auftraggeber und Auftragnehmer, 5th edn. VDI-Buch (2016)
15. Welge, M.K., Al-Laham, A., Eulerich, M.: Strategisches Management. Grundlagen - Prozess - Implementierung, 7th edn (2017)
16. Wilson, I.: Strategic planning isn't dead—it changed. Long Range Plan. (1994). https://doi.org/10.1016/0024-6301(94)90052-3
17. Glaister, K.W., Falshaw, J.R.: Strategic Planning. Still Going Strong? Long Range Plan. (1999). https://doi.org/10.1016/s0024-6301(98)00131-9
18. Frentzel, W.Y., Bryson, J.M., Crosby, B.C.: Strategic Planning in the Military. Long Range Planning (2000). https://doi.org/10.1016/s0024-6301(00)00040-6
19. Wolf, C., Floyd, S.W.: Strategic planning research: toward a theory-driven agenda. J. Manag. (2016). https://doi.org/10.1177/0149206313478185
20. Liedtka, R.M.D.O.J., Jacobs, D.C., Heracleous, L.: Strategizing through playful design. J. Bus. Strat. (2007). https://doi.org/10.1108/02756660710760971
21. Graefenstein, J., Scholz, D., Henke, M., Winkels, J., Rehof, J.: Intelligente Orchestrierung von Planungsprozessen. ZWF (2017). https://doi.org/10.3139/104.111696
22. Gulwani, S.: Dimensions in program synthesis. In: Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, pp. 13–24 (2010)
23. Gulwani, S., Polozov, O., Singh, R.: Program synthesis. Found. Trends® Program. Lang. **4** (1–2), 1–119 (2017)
24. Roser, S., Lautenbacher, F., Bauer, B.: Generation of workflow code from DSMs. In: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (2007)
25. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. Int. J. Softw. Tools Technol. Transf. (2017). https://doi.org/10.1007/s10009-017-0453-6
26. Naujokat, S., Neubauer, J., Margaria, T., Steffen, B.: Meta-level reuse for mastering domain specialization. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, pp. 218–237. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_16
27. Steffen, B., Naujokat, S.: Archimedean points: the essence for mastering change. In: Steffen, B. (ed.) Transactions on Foundations for Mastering Change I. LNCS, vol. 9960, pp. 22–46. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46508-1_3
28. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
29. Awad, A., Goré, R., Thomson, J., Weidlich, M.: An iterative approach for business process template synthesis from compliance rules. In: Advanced Information Systems Engineering, pp. 406–421 (2011)
30. Yu, J., Han, Y.-B., Han, J., Jin, Y., Falcarin, P., Morisio, M.: Synthesizing service composition models on the basis of temporal business rules. J. Comput. Sci. Technol. **23**(6), 885–894 (2008)
31. Lamprecht, A.-L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-based loose programming. In: 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 262–267 (2010)

32. Alur, R., et al.: Syntax-guided synthesis. Form. Methods Comput.-Aided Des. (FMCAD) **2013**, 1–8 (2013)
33. Srinivas, Y.V., Jüllig, R.: Specware: formal support for composing software. In: International Conference on Mathematics of Program Construction, pp. 399–422 (1995)
34. Le, V., Gulwani, S.: FlashExtract: a framework for data extraction by examples. In: ACM SIGPLAN Notices, pp. 542–553 (2014)
35. Feser, J.K., Chaudhuri, S., Dillig, I.: Synthesizing data structure transformations from input-output examples. In: ACM SIGPLAN Notices, pp. 229–239 (2015)
36. Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. ACM Sigplan Not. **41**(11), 404–415 (2006)
37. Grambow, G., Oberhauser, R., Reichert, M.: Semantically-driven workflow generation using declarative modeling for processes in software engineering. In: Proceedings of EDOCW 2011, pp. 164–173. IEEE Computer Society (2011)
38. Parisotto, E., Mohamed, A.-R., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855 (2016)
39. Ilghami, O., Nau, D.S.: A general approach to synthesize problem-specific planners. University of Maryland, College Park, Department of Computer Science (2003)
40. Becker, M., Gilham, L., Smith, D.R., et al.: Planware II. Synthesis of schedulers for complex resource systems (2003)
41. Blaine, L., Gilham, L., Liu, J., Smith, D.R., Westfold, S.: Planware-domain-specific synthesis of high-performance schedulers. In: Proceedings of 13th IEEE International Conference on Automated Software Engineering, pp. 270–279 (1998)
42. Reichert, M., et al.: Enabling Poka-Yoke workflows with the AristaFlow BPM Suite (2009)
43. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28901-1
44. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37521-7
45. van Gurp, J., Bosch, J., Svahnberg, M.: On the notion of variability in software product lines. In: Proceedings of Working IEEE/IFIP Conference on Software Architecture, pp. 45–54 (2001)
46. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014)
47. Rosa, M.L., van der Aalst, W.M.P., Dumas, M., Milani, F.P.: Business process variability modeling: a survey. ACM Comput. Surv. (CSUR) **50**(1), 2 (2017)
48. Bessai, J., Dudenhefner, A., Düdder, B., Martens, M., Rehof, J.: Combinatory logic synthesizer. In: 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2014, Corfu, Greece, 8–11 October 2014, pp. 26–40 (2014). https://doi.org/10.1007/978-3-662-45234-9_3
49. Coppo, M., Dezani-Ciancaglini, M.: An extension of basic functionality theory for lambda-calculus. Notre Dame J. Form. Log. **21**, 685–693 (1980)
50. Curry, H.B.: Functionality in combinatory logic. Proc. Natl. Acad. Sci. **20**(11), 584–590 (1934)
51. Howard, W.A.: The formulae-as-types notion of construction. To HB Curry: Essays Comb. Log. Lambda Calc. Formalism **44**, 479–490 (1980)
52. Düdder, B., Martens, M., Rehof, J., Urzyczyn, P.: Bounded combinatory logic. In: Proceedings of Computer Science Logic, CSL 2012, pp. 243–258. Schloss Dagstuhl (2012)

53. Bessai, J., Dudenhefner, A., Düdder, B., Martens, M., Rehof, J.: Combinatory process synthesis. In: Proceedings of 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques, ISoLA 2016, Imperial, Corfu, Greece, Part I, 10–14 October 2016, pp. 266–281 (2016). https://doi.org/10.1007/978-3-319-47166-219
54. Bessai, J., Düdder, B., Heinemann, G., Rehof, J.: Towards Language-Independent Code Synthesis (2018)
55. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990