INTERNATIONAL DESIGN CONFERENCE - DESIGN 2010
Dubrovnik - Croatia, May 17 - 20, 2010.

DESIGN 2010

DEVELOPING COMPUTATIONAL TOOL FOR GENERATION OF OPERAND TRANSFORMATION VARINATS IN TECHNICAL PROCESSES

T. Stanković, N. Bojčetić and D. Marjanović

Keywords: technical process, grammatical evolution, computational design support

1. Introduction

The Theory of Technical Systems (TTS) models technical processes as systems of transformations through which the needed states of operands can be achieved [Hubka and Eder 1992]. The role of designed technical system inside technical processes is understood in a teleological sense with its function defined as a capability to deliver necessary effects that are required to made those operand transformations possible. Aim of this paper is to present a computational tool for generation of operand transformation variants in technical processes. Such tool, when provided, can be used by designers to consider different product realization possibilities based on existent technological principles in respect to customer needs, requirements and given constraints. Proposed computational support can enhance the design process significantly since transformation alternatives can be obtained in an expedient fashion with the probability of the generation of novel alternatives.
Proposed computational tool is based on a method developed for that purpose [Stanković et al. 2009] that relies on Grammatical evolution (GE) which is a population based heuristic search that creates solution to a given problem by recombination of the rule-based rewriting sequence [O'Neill and Brabazon 2005]. Decomposition performed by designers at the beginning of conceptual design phase in order to establish optimal operand transformations will be performed computationally. The needed knowledge about technological principles on which the transformations are based is formalized within set of production rules in Backus-Naur form (BNF). Computational tool enables user to visually define vocabulary and compose production rules using that vocabulary. That presents an enhancement to the current practice where rules are being defined and stored manually inside a text file. To accomplish computational implementation, a rule based system for decomposition of operands transformation inside technical processes had to be examined from a more formal point of view. It is proposed to base the decomposition system on an established mathematical concept of *double-pushout* approach [Kreowski et. al. 2006] which is a frequent method for conducting grammar based graph transformations. For modeling of operands transformation inside technical processes a directed multi-digraph with labeled vertices and edges is required, i.e. multi-digraph with operations and operands. In the next section our reflections about the translation of existent design methods to computational tools has been elaborated.

2. Background

Systematic design approaches are devised with a purpose of providing support to designers in order to better utilize their knowledge and skills, critical thought is developed and search for solutions is made more comprehensive and rational based [Hubka and Eder 1992]. Formalizing design process into a

step-wise problem solving procedure introduces structure to the process thus enabling activities like standardization and planning. It can be said that systematic approaches offer each in their own way a collection of different methods which can help designers to recognize important aspects of given tasks and encourage them to think creatively when solving these tasks. From studies in cognitive psychology of human achievements in engineering, science and arts it has been noted that inventive and creative solutions yielded as a result of intentional processes conducted under principles and strategies used for everyday problem solving [Weisberg 2006], therefore design methodologies often lean towards applying heuristics for problem solving. Two of many examples may be generation of concept variants through morphological chart [Pahl, Beitz 1988] and substance-field analysis in TRIZ where grammatical based rules are used to create a transformation system with structure analogous to solutions given inside a catalogue [Savransky 2000]. To select one of generated variants designers must evaluate potential solution problems which happens quite often since systematic approaches usually organize design process as being composed of iterative logical cycles resulting with a verified solution at completion at different product abstraction levels. Sometimes a method prescribes an evaluation criteria, like need for establishing diagonal or upper diagonal system design matrix thus creating uncoupled or decoupled design solutions in Axiomatic Design [Suh 2001], but the final decision is always left to designers.

If translation of existent design methods to computational environment can be made possible then design process could be enhanced in respect to speed and time necessary to process and generate solution alternatives which in the end may improve products being designed. Likewise, time to market period will be shortened and whole product development process could become more efficient. Purpose of existing design methods is to stimulate and excel creative reasoning and to help designers at ideation processes. Design methods are built around the same principles which are valid for general. problem solving and the solution strategy is chosen carefully to best-fit intended purpose. However to use computers to simulate higher semantic reasoning needed for performing even simple everyday problem solving is a difficult task and complete translation to computer environment of methods that are intended for and build around the human user may be impossible. For example, to enhance search process in respect to quality and number of solution alternatives generated heuristic search strategies are applied. Creating concepts using morphological chart requires abstract reasoning process since designers must find a way how to patch individual components together in an synergetic and holistic manner. Energy flow compatibility is just a requirement to point out and eliminate impossible solutions. Sometimes establishing similarities and analogies between concepts will devise a solution and expectation and knowledge will help to explain the nature of the task at hand. Good example is substance-field analysis in TRIZ which by designers must recognize patterns in systems structure in order to establish analogies with given solution principles inside catalogues. The Theory of Technical Systems [Hubka, Eder 1992] examines the level of computational support in respect to different search strategies and application domains, concluding that absent or limited computational support using trial and error search is highly inefficient but applicable to various range of problems. The opposites are fully automated computational methods which are highly efficient but very limited to narrow specialized domains of application. If considering design not just as a product development process but as an complex phenomenon involving people, organization and micro-economic and macro-economic context [Blessing, Chakrabarti 2009] then the effort needed to translate design methods to computational tools becomes even greater. Present computational design support is therefore usually oriented towards performing tasks that are hard for humans to do and do not require simulation of semantic and abstract reasoning. Such tasks are these that have iterative nature which involve lots of tedious repetitive data processing that needs to be accomplished in an error free manner like numerical computation or structural and topological optimization, In most cases solution space is a discrete or non-discrete multimodal hyper space often filled with discontinuities as a result of given constraints. Because of the size of the search space these are hard problems for a designer to solve. Translation of these problems to computer environment is usually accomplished by usage of advanced discrete stochastic solvers like genetic algorithms with the problem being formalized using the language of mathematics as set of objective functions and constraints.

Developing computational support for early stages of design process is worth an effort and can be justified. Deciding on an solution alternative at some point in course of designing, will consequently reduce the overall solution search space. So, for the design steps that will follow there would still be room left for minor improvement but the possibility of significant changes is reduced. The decisions made early in design processes appear at the beginning of causal chain and therefore influence the whole design process by the most. This is one of the reasons behind setting the aim of this research to development of computational support that will help designers to establish optimal operand transformations in technical processes [Stanković et. al. 2009]. Task of computational tool is then to generate variants of operand transformations inside technical processes subjected to set of constraints and goals. The goal of performing the decomposition of technical processes by examining operations and flow of operands is to gain insight and information about the transformation process. It is important to stress out that computational support is understood in an advisory form, meaning that the solution alternatives created will help designer to gain new insight about the problems and ideas to work with. In the following section the state of the art is presented.

3. Related work

From the field of evolutionary computation it is known that search done by recombination with exchange and inheritance of good building blocks can yield innovative results [Goldberg 2002]. Reasons for that can be explained by the facts that search mechanism by recombination of solution building blocks simulates heuristic reasoning process performed by humans. That search is performed in an objective non-biased manner, therefore not influenced or guided by established conventional solutions to a given problem. A good example on how to apply an existent search method for conceptual design is a genetic algorithm based search inside a morphological chart. To produce optimal concepts a genetic algorithm is used to select a set of solution principles for the product sub-functions in order to [Hutcheson et al. 2006]. The validation of the solution principles is performed by the energy flow compatibility check and a pre-defined cost function is used. Similar approach is a Concept generator tool [Bryant et. al. 2005] that uses matrix algebra to establish mapping from a predefined function model to lists of components that are capable of resolving each of the given functions.

Different approach was undertaken by A-Design [Campbell et al. 1998] which included a collection of software agents with embedded knowledge enabling them to perform their specific duties in order to create meaningful solution concepts. The validity of the library component interconnections was tested through an interface by an input-output type compatibility check. For the optimization genetic algorithms where considered.

For the support of conceptual design and concept generation, graph grammar representations were used the most [Kurtoglu and Campbell 1996, Schmidt and Cagan 1997, Starling and Shea 2007]. The design knowledge for the specific area of application is formalized within the grammars. The possibilities of utilizing grammars where recognized since they were used extensively for structural and topological optimization. The boundaries of the design search space are defined by a generative grammar thus creating large but finite number of possible solution alternatives. Almost all of the design theories model product as transformation system of some sort which is then in a formal and visual manner represented as graph. Therefore for purpose of creating computational support for the early stages of product development the graph grammars are often used. Survey of the literature has showed that available methods consider graph grammars for decomposition from functions to components where the optimization is usually targeted at the component level.

Merging grammars and genetic algorithms for the computational concept generation resulted with method which iteratively evolved products on different abstraction levels in parallel [Jin, Li 2007]. Then, based on the knowledge stored inside a rule library, an initial population of functional decompositions is created. Genetic programming and genetic algorithm co-evolve functions and functional means. The fitness function is formulated using multiple weighting factors.

Recent methods tend to combine available graph grammar transformation tools like *GrNET* and to achieve standardized approach for synthesis of mechatronic products [Helms et al. 2009]. The underlining modeling is done using Function-Behavior-Structure.

The computational tool presented in this paper is targeted at very beginning of the conceptual phase of product development. Grammatical evolution searches for the rule sequence that can perform the decomposition of the technical process black-box level to a structured system consisting of sub-processes, operations and operand flows according to TTS. State of the art review of computational design support methods and tools shows that technical processes are not considered usually taking as a starting point function level of conceptual phase. Since establishing technical processes also considers examination of the process of using the product that has to be designed, then skipping that imposes serious constraints to the search space. From the literature [Hubka et al. 1988] it is known that variations on the process level yield different function decompositions. According to TTS the ability to deliver desired effects is considered as a function of the product, meaning that the existence of these effects must be recognized before functional decomposition. Methods such as the morphological chart method cannot broaden the search space by addition new effects to the established functional structure. The only way to accomplish that is to return back and affect the technical process from within, which is an iteration that has to be avoided. That is why arguments it would be of value to include technical process level in the overall computational design synthesis framework.

4. Method

Proposed method for generation variants of operand transformation in respect to the given customer needs and constraints relies on the grammatical evolution for controlling and directing the search. Engineering knowledge about technological principles on which operand transformation are based is formalized using set of Backus-Naur form (BNF) type production rules and context-free grammar (CFG). Since grammatical evolution is a stochastic optimizer, then it can search for the sequence of BNF rules through which the optimal decomposition can be generated rather than just generating all of the possible solution variants. Model of proposed approach is depicted according to IDEF0 in Fig. 1 [Stanković et. al. 2009]:

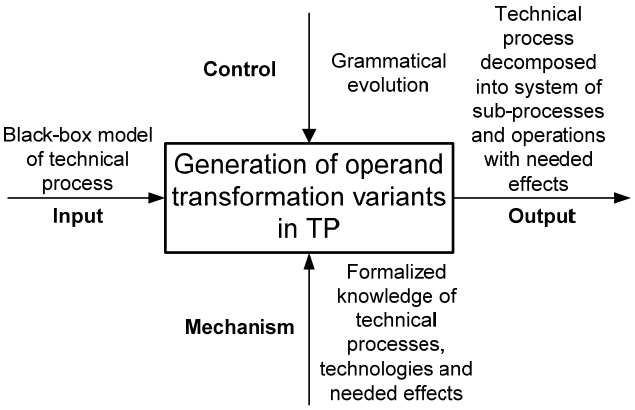
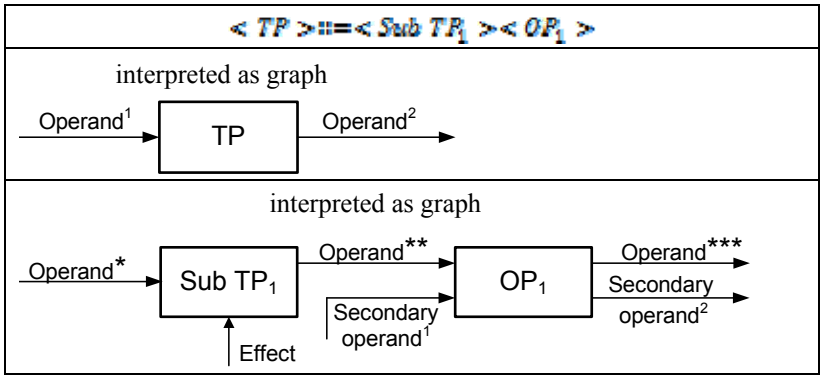


Figure 1. IDEF0 model of generation of operand transformation variants

Grammatical evolution was selected as a search backbone because it is built upon on genetic algorithm therefore inheriting its properties and robustness. Principle on which GE creates valid solutions does not suffer from mixing incompatible operators and operands (the *closure* problem) which occurs very often when using similar stochastic optimizers like genetic programming. [O'Neill, Brabazon 2005]. More so, it has been shown that equivalence exists between Chomsky's grammars and graph grammars [Kreowski et al. 2006] which enables to utilize BNF type rules and context-free grammar to define graph transformations rules. Such approach creates robust symbol like rewriting system which is easier for computer implementation. Technical process as an operand transformation system is then modeled as a token like sentence which has to be interpreted as graph accordingly. The definition of context-free grammar CFG [Jiang 1999] of a formal language $L = L(\mathrm{CFG})$ is expressed by the quadruple $CFG = (\Sigma, V, S, P)$ where Σ is a finite nonempty set of terminal symbols or, V is a finite nonempty set of non-terminal symbols or variables with $\Sigma \cap V = \emptyset$, S is a starting symbol or axiom where $S \in V$ and P is set of production rules of type $\alpha \rightarrow \beta$ satisfies the relation $|\alpha| = 1$

where $\alpha \in V$ and $\beta \in (\Sigma \cup V)^*$. The asterisk denotes the set of all possible combinations of terminal and non-terminal symbols.
Under the BNF formalism each rewriting rule can have multiple alternatives that can successfully replace rule left hand side. Writing sentences in CFG language starts from a predefined single symbol S. Then the initial symbol is rewritten in the first derivation step with the new sentence consisting of string composed of tokens. Afterwards, in the same manner every token from new sentence is again replaced following according the finite set of rules P. Tokens that can be replaced are called variables and are elements of V. The process stops at derivation step on which all of created sentence is composed just of terminal symbols from Σ. Search for optimal variants using GE is accomplished by evolving population of binary encoded chromosomes. For solution generation chromosomes are decoded into sequence of integers and then by using modulo. Using simple expression $r = I \ mod \ N$ it is calculated which rule alternative r to trigger in respect to α, decoded gene value I and the number of alternatives N on the rule right hand side. Going over decoded integers inside chromosome defines whole rule application sequence.

Table 1. Rewriting rule example in BNF and its graph interpretation



Performing decomposition of operand transformation in technical process requires that each side of the rule can be interpreted as a labelled multi-digraph with operations, operands and effects. Table 1 shows one such rule for decomposition of technical process (TP) into a system consisting of sub-process (Sub TP) and operation (OP) with flows and effects; operation OP does not permit any further decompositions:
Since the CFG is used, the left hand side of the rule is composed always of one token denoting the operation that needs to be decomposed. Right hand side of the rule is a small transformation system based that is able to accomplish transformation of operands on some existent technological principle as specified on the rule left hand side. In Table 1 the asterisk denotes transformation of main operand. Supplementary or unwanted secondary operands with needed effects may appear as result of technological principles on which the transformation is based upon. Designer formulates the input information by choosing the initial token representing technical process, main operands in desired states prior and after the transformation thus enabling rewriting procedure to starts. After all processes have been rewritten into operations which are elements of Σ the decomposition stops.In the following section mathematical model of operand transformation is given.

5. Mathematical model of operand transformation

Mathematical modeling of the operands' transformation in technical process is accomplished using multi-digraph with operations, operands and effects. For this purposes a multigraph will be considered as a non-simple graph inside which multiple edges between vertices are allowed but no loops are permitted [Weisstein 2009]. Formal definition states that a multi-digraph G is an ordered pair (V_G, E) where V_G is set of vertices and E is a multi-set of arcs (edges) between vertices however for purposes of this research an extension is necessary. To capture and describe transformation in technical process let one element from the set of all operands, effects and operations $\Sigma_G = \Sigma_{Od} \cup \Sigma_{Eff} \cup \Sigma_{Op}$, be related to each vertex V and edge E. Put succinctly, T Set Σ_G contains all possible graph labels. Introduction of labels to the multigraph structure is necessary in order to assign effects and operands in their

particular states to the graph's edges. Same principle is applied for assignment of operations to the graph's vertices. Let operand Od be an element from finite set of all possible operands Σ_{Od}, $Od \in \Sigma_{Od}$. Operand is an object having *ID*, *name*, *state, type* and *label* for attributes. In the same manner effect Eff is an element of finite set of all possible effects Σ_{Eff}, $Eff \in \Sigma_{Eff}$ having *ID, name, type* and *label* for attributes. Thus, in the presented model edge label is not an alphanumeric string but an object container which can accept both operands and effects. Same reasoning is applied with the graph vertices where each vertex is an object, namely an operation Op, where $Op \in \Sigma_{Op}$ and Σ_{Op} is a finite set off all possible operations. Op is an object with *ID, name, label* and three collections of *input operands*, *output operands* and *effects* as attributes. Each operation consists of a single operation and input operands in their states prior to and after the main transformation. At the present research/implementation state dealing with operations semantics is avoided since the production rules are case specific for each and every operand.

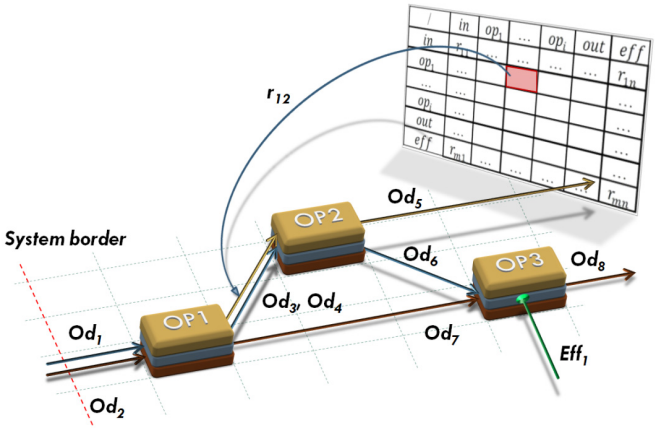


Figure 2. Multi-digraph with operations, operands and effects and its adjacency matrix

Formally, multi-digraph G with operations and operands defined over finite non-empty set of labels Σ_G is an ordered sextuple $G = (V_G, E, s, t, l_E, l_V)$, where V_G is finite non-empty set of vertices, $E \subseteq \{(u, v) | u, v \in V_G \wedge u \neq v\}$ is a bag of directed edges (arcs) e, $s: E \to V_G$ and $t: E \to V_G$ are mappings that assign source and target vertices to each edge e, mapping $l_E: E \to \Sigma_{Od} \cup \Sigma_{Eff}$ assigns operand Od or effect Eff to each edge e and finally $l_V: V_G \to \Sigma_{Op}$ is mapping that assigns operation Op to each edge e. Relation of multi-digraph with operations, operands and effects with to its adjacency matrix is sown on Fig. 2.

From perspective of graph grammars Σ_G contains alphabet from which graph-grammar rules are later constructed by the user. Adjacency matrix r_{ij} of directed multigraph is defined so that each cell can contain multiple relations that may occur inside the transformation process. Taking into account that relation as container excepts effects or operands, then r_{ij} may be written as $r_{ij} \subseteq \{(u_i, v_j) | u_i, v_j \in V_G \wedge u_i \neq v_j\}$. Mapping adjacency matrix cell and multiple operands between operations is depicted on Fig. 2. In next section procedure for applying rewriting rules will be elaborated.

6. Definition of rules and rules rewriting system

To successfully apply rewriting procedure the conditions of replacement have to be prescribed and algorithm of rewriting must be specified. Using BNF rewriting rules and CFG enables that replacement and insertion points are easily identified where single token is always replaced with set of tokens. Within proposed tool a *double-pushout* approach for graph grammar transformation has been applied, however to suit the requirements of technical process modeling additional conditions have been created that have to be met in order to patch up the operand flows correctly. According to literature [Kreowski et al. 2006] the rule $r \in P$ required for graph grammar transformation is composed of three graphs $r = (L \supseteq K \subseteq R)$ where all of the graphs are elements of set of all possible graphs G_Σ defined over Σ_G, $L, K, R \in G_{\Sigma_G}$, such that K is sub-graph of L and R. Left hand side of the

rule is graph L and the right hand side graph is graph R. Graph K is called the gluing graph. In relation to CFG in the Section 4., the gluing graph enables replacement of technical process or sub-process in G defined with α with the with the new process defined in β resulting with new technical process H as shown in Fig 3. Please note that graph grammars and Chomsky's grammars are equivalent, so instead of L, α is used to denote left hand side.

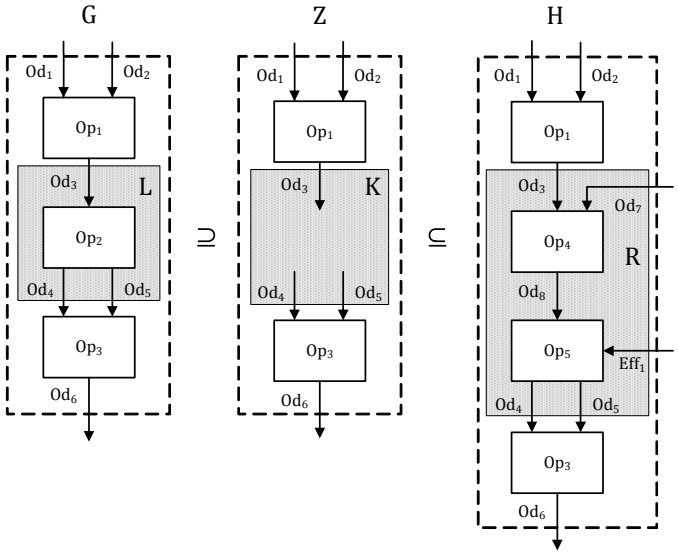


Figure 3. Double push down approach [Kreowski et. al. 2006] tailored to suit purposes of technical process variants generation

To accomplish successful $\alpha \rightarrow \beta$ rewriting which will suit the purposes of technical process (TP) decomposition following conditions must be met:

- each TP consists of minimally 4 operations $Op \in \Sigma_{Op}$ which must be connected by operand flows,
- fictive operations $Op_{In}, Op_{Out}, Op_{Eff} \in \Sigma_{Op}$ are part of each TP in order to describe operand flows and effects going or coming from outside the systems borders,
- fictive operations $Op_{In}, Op_{Out}, Op_{Eff} \in \Sigma_{Op}$ do not transform operands,
- α is an input to rewriting, it consists of a single sub-process or process and input operands in their states prior to and after the main operands' transformation $Op \in \Sigma_{Op}$; as an additional input the secondary flows of operands and desired effects may appear,
- β is a rewritten sub-system; it is a multigraph $G = (V_G, E, s, t, l_E, l_V)$ which consists of at least two sub-processes (operations) where at least one must accomplish the transformation of the input operand; the secondary flows as well as the new effects may appear,
- compatibility of operand flow is checked using gluing graph *K*.

7. Computational tool architecture

To be able compare results methods and to realize draw backs and possible improvements of the proposed grammar based transformation system an computer program application was developed. Current state of tool development suffices testing purposes. Key modules of the developed tool are shown in Fig. 4. Program application architecture is organized in modular fashion since it requires diverse program libraries including grammatical evolution library, user interface components and multi-digraph library. Modular approach enables re-use of developed libraries for further development of computational support aimed at other stages of conceptual design. More so, much of the today advances in cutting edge programming tools are considered and some of them are developed anew to

build this application (MS C#, LINQ, relation database, dynamic creation of graphical representation of multi-digraphs).

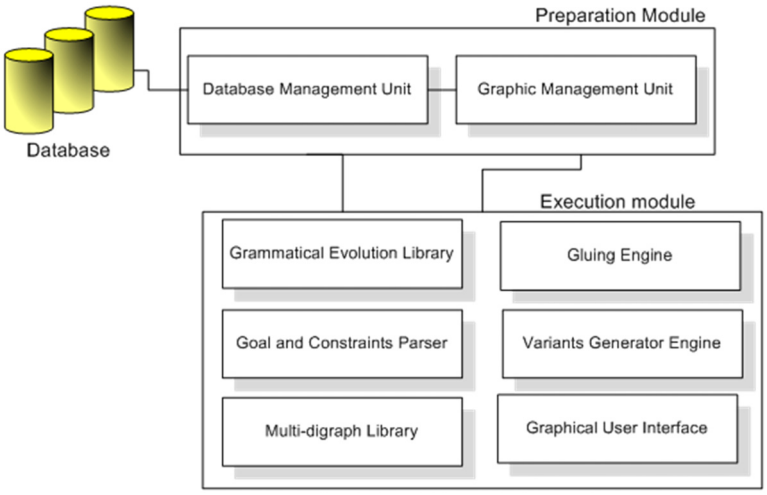


Figure 4. Architecture of proposed tool

Preparation module consists of database management unit which has a task to establish connection to the database and to retrieve and store data to the database. Special attention is put on the rule validation. All data retrieved from the database is run through the parser to get appropriate rule syntax. Then the result of the parser is validated so basic rule semantic can be tested and in that way minimize possible errors in data transmission. Architecture of the preparation module is shown in Fig.

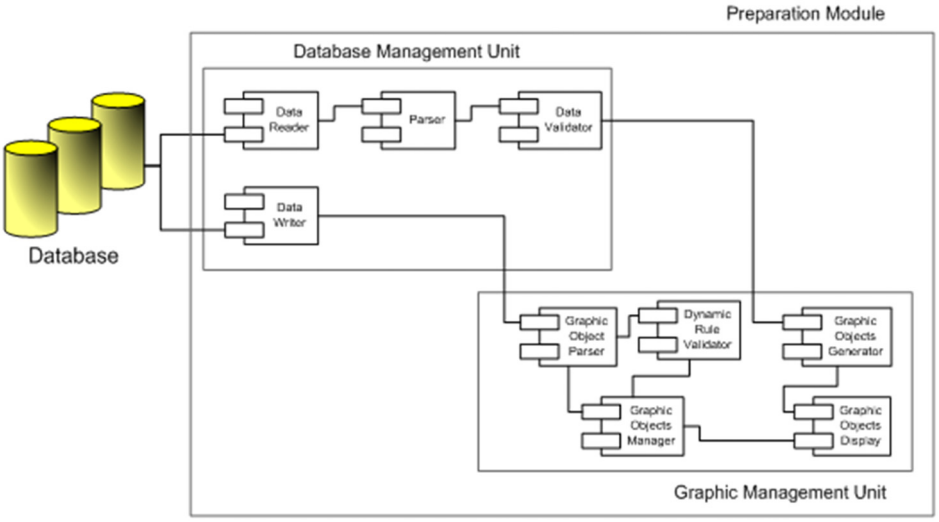


Figure 5. Architecture of Preparation Module

data that might have been stored incomplete due to lost connection during write process or tampering with data using other database management applications is discarded as not valid. Graphic management unit has to manage the graphical representation of the rules, operations, operands and effects. After the validation of the data retrieved from the database , the data is being sent to the graphic objects generator. This module based on the nature of the data creates, in memory, graphical representation of the objects. Objects from the memory are then read by the graphic objects display module and displayed on the main window of the application.

Displayed objects are not just the static pictures but they are interactive thus responding to the user actions. Because that user has large freedom in interaction with the objects constant surveillance of his or hers actions must maintained. Responsibility to assure correct or valid rule creation is assigned to dynamic rule validation module. This module constantly validates any user action and informs user of

correct and incorrect ones. When user decide to save created rule graphical rule representation is parsed and formatted to the form suitable to be written to the database. Manipulation of the graphical representation of the rules is responsibility of the graphical object manager module. This module ensures consistency of the graphical objects during users interaction. Consistency is ensured using graphical links and object overlap monitoring capability.
Execution module consists of several interconnected modules but it is reusing some of the functionalities of the preparation module. Database interaction and graphical representation manipulation are developed robust and therefore could be used in both of the main modules. The variants generator engine and the grammatical evolution library are performing decomposition, transformation and search and optimization under specified goals and constraints which are defined by the user using goal and constraints parser. Same module is also performing conversion of search goal into multigraph so that it can be interpreted by the GE. The gluing engine implements the patching up of operand flows when insertion of sub-graphs into graph when occurs. As mentioned earlier, proposed system is based on the multigraph functionality. This functionality is implemented using multi-digraph library) which consists of several routines and methods for addition, manipulation and deletion of elements that constitute a multi-digraph.

8. Example

In order to use preparation and execution modules user has to understand only the basic idea that is behind the applications. Both applications have intuitive GUI that is easy to learn how to work with. Hopefully, the potential user will see both applications as windows to his own ingenuity. After connection to the database is established (selecting connect command from the file menu) user can select his or hers next action as follows:

- Operands – to manage operands (create, modify, delete),
- Effect – to manage effects (create, modify, delete),
- Operations – to manage operations by combining existing operands and effects,
- Rule Assembler – to mange different operations combinations that can be substitute for an existing one.

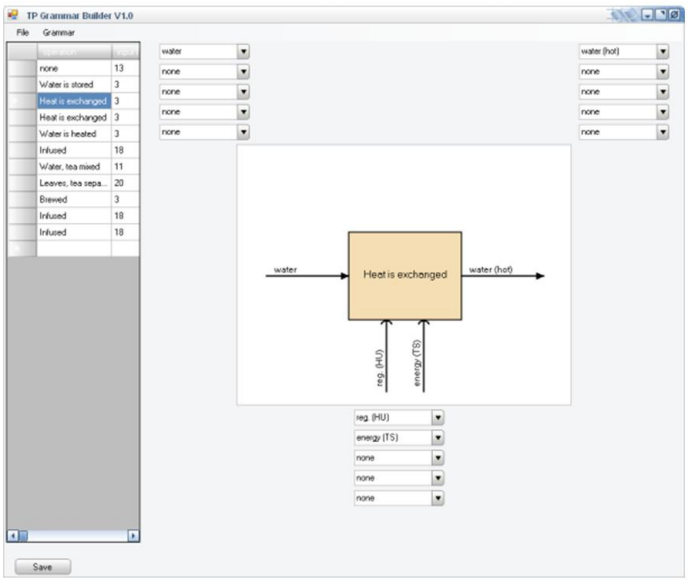


Figure 6. Screenshot of the operation builder

Operations are usually named according to their purpose. There can be more that one operation with the same name but to avoid ambiguity it has to be composed of different combinations of operands and effects. On Figure 6. an example of is presented depicting one operation with one input and one output operand and two effects. New operands on each operation side can be added or existing ones can be changed by selecting from the drop down boxes on the window.

As was explained in more details in previous sections to be able to thoroughly use execution module rules must be constructed from existing user defined set of operations. This task can be done by selecting rule assembler command form the grammar menu. After activation of this command window presented in the Figure 7 will open. First user must select one of the available operation in the table on the left side of the window. Selecting an operation will create graphical representation that will be presented in the right part. On the right hand side of the window slots for the operations are available so the user can create new rule to suit his or hers needs. Rules are created by selecting operations from the available slots. Links represented operand flows between the operations in the active rule can be created by selecting compatible output and input nodes.
After adequate number of rules are created user can run execution module to find all possible solutions that satisfies imposed constraints for a given operation. EM graphical user interface is similar to Preparation Module so additional learning is not necessary. First the connection to the database has to be established. Then all operations stored in the database are displayed in the tabular form on the left side of the window. Constraints can be defined an modified to govern search and optimization process.

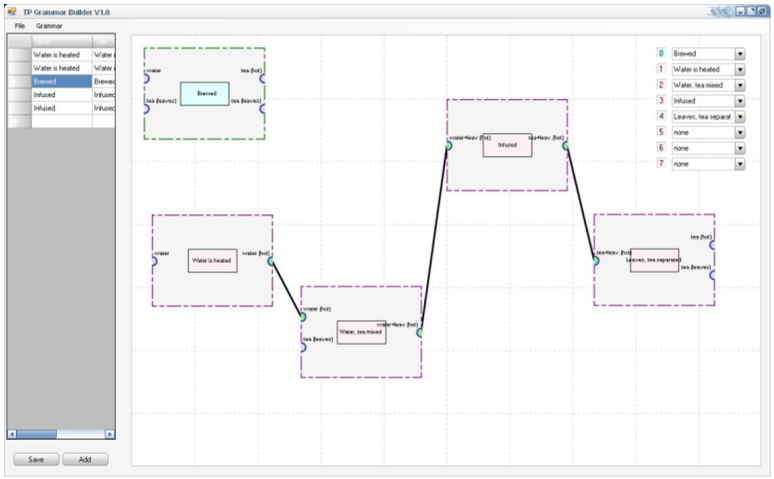


Figure 7. Screenshot of visual interactive rule builder

Finally, when user defines goals and constraints the generation of optimal variants can start. If no stopping criteria has been set the user is can always stop the search process.

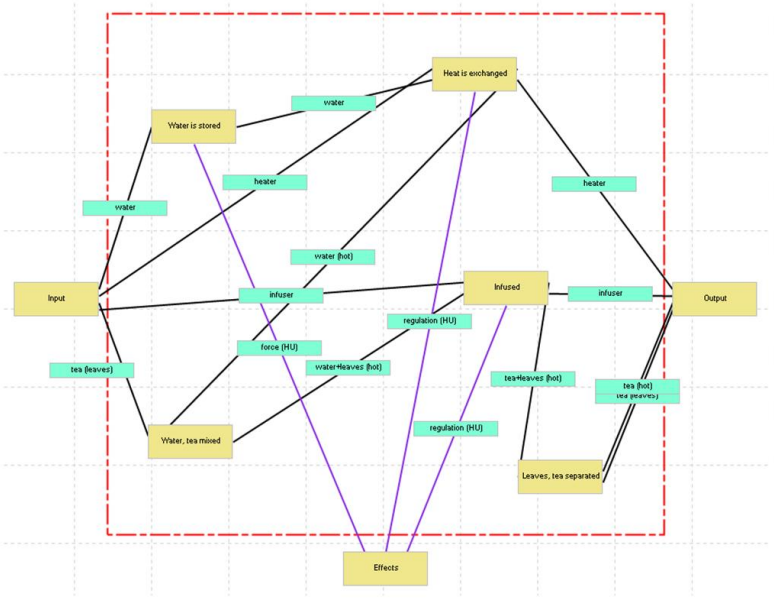


Figure 8. Example of decomposed TP

At the current level of tool development with no usage of advanced algorithms for graph visualizations the result will be displayed in the main window as in example shown on Figure 8.

9. Discussion

Currently, the presented tool is at its prototype stages of development still requiring design of a large enough BNF library for testing purposes. Valuable assessment of the tool performance can be obtained only by real-life testing. Within presented tool triggering of BNF rule alternatives is conducted only by following rule ID. That approach is limited for test and development purposes. If the search would be elevated to more semantic levels and be conducted over the name of operation, then the rule definition would be easier for the user and possibilities of creating more creative solutions would increase. More so, algorithms that are able to untangle and depict graph in a manner comprehensible to user have not been applied yet. Design management methods and tools like DSM will also be considered for that purpose. For instance, motivation for designing interactive visual builder of grammar rules was triggered by negative experiences of designing BNF rules in text files by hand. Usually, the text file approaches result in slow string parsing routines that reduce effectives of the application.
Recent approaches for creating computational methods for conceptual design support tend to reuse existent, already developed computational tools. Decision to design and build whole computational method anew was motivated by the fact that, although time consuming, such approach offers maximum possibilities. Developed libraries shown in architecture of presented computational tool are build in modular and generic fashion. In that way robustness is achieved thus permitting reuse for creation of computational support for other stages of conceptual design phase.
Once created, a large BNF rule library will result with the number of feasible solutions being raised significantly. Performing combinatorial search over all of feasible solutions would not be an effective strategy. GE is an optimization method that converge to a single solution and sometimes require multiple algorithm runs to identify possible variants that have equal fitness. However, since engineering problems are almost always multi-objective, multi-objective genetic algorithms (MOGAs) are appropriate to obtain population diversity while converging to an optimum. Application of MOGAs presents a suitable way to avoid exhaustive search for obtaining a population of feasible Pareto optimal solution variants.

10. Conclusion

The tool presented in this paper is based on grammatical evolution (GE) driven method for the generation of operand transformation variants inside technical processes. The method uses stochastic optimization algorithm to perform decomposition from the black-box model to the transformation process composed from chain of operations in order to generate alternatives. Formalized knowledge about technological principles and needed effects to support technical processes stored inside the BNF rule library.
By using developed tool, designer can obtain valuable insights about existing technologies and technological principles on which technical process can be based on. Solutions obtained provide the input on which the function structure of the product can be determined.
For the modeling of transformation of technical processes *double-pushout* approach was extended to accommodate operations, operands and effects. Additional conditions had to be specified in order to patch operand flows correctly. Multi-digraph is known mathematical concept, and it was selected because of its applicability for modeling of product at later stages of conceptual design. Bringing all of these methods together presented considerable effort in order to in an synergetic manner create a tool that will aid designer at very beginning of conceptual design.

Acknowledgement

This research is founded by the Ministry of Science, Education and Sports of the Republic of Croatia under scientific research project no. 120-1201829-1828 “Model and Methods of Knowledge Management in the Product Development”.

References

Blessing L., T, M, Chakrabarti, A,, „DRM, a Design Research Methodology", Springer-Verlag, London Ltd., 2009.

Bryant, C. R., Stone, R. B., McAdams, D. A., Kurtoglu, T., Campbell, M. I., „Concept Generation from the Functional Basis of Design", In the Proceedings of the ICED'05, 2005.

Campbell, M., Cagan, J., Kotovsky K., „A-Design: Theory and Implementation of an Adaptive, Agent based Method of Conceptual design", Artificial Intelligence in Design'98, Kluwer Academic Publishers, Netherlands, 1998.

Helms, B., Shea K., Hoisl, F., „A Framework for Computational Design Synthesis Based on Graph-grammars and function-behavior-structure", In the Proceedings of the ASME DETC 2009.

Hubka, V., Andreasen, M. M., Eder, E. W., 1988. Practical Studies in Systematic Design, Butterworth & Co., London.

Hubka, V., Eder, W. E., "Engineering Design: General Procedural Model of engineering Design", Springer-Verlag Berlin Heidelberg, 1992.

Hutcheson, R. S., Jordan, R. L., Stone R. B., „Application of a Genetic Algorithm to Concept Variant Selection", In Proceedings of the ASME DETC 2006.

Jiang, T., Li, M., Ravikumar, B., Regan, K. W., „Formal Grammars and Languages, Algorithms and Theory of Computation Handbook / edited by Atallah M. J., CRC Press, Boca Raton, New York, 1999.

Jin, Y., Li, W., „Design Concept Generation: A Hierarchical Coevolutionary Aproach", Journal of Mechanical Design, 2007, pp. 1012-1022.

Kreowski, H. J., Klempien-Hinrichs, R., Kuske, S., „Some Essentials of Graph Transformation", Recent Advances in Formal Languages and Applications, Esik, Z., Martin-Vide, C., Miltrana, V. (Eds.), Springer-Verlag Berlin Heidelberg, 2006, pp. 229-254.

Kurtoglu, T., Campbell, M., I., „A graph Grammar Based Framework for Automated Concept Generation", In the Proceedings of DESIGN 2006, Dubrovnik, Croatia, 2006.

Pahl, G., Beitz, W., "Engineering Design – A Systematic Approach", Springer Verlag, 1988.

O'Neill, M., Brabazon, A., „Recent Adventures in Grammatical Evolution, Computer Methods and Systems", CMS'05, Krakow, Poland, 2005, pp. 245-253.

Savransky, S. D., "Engineering of Creativity - Introduction to TRIZ: Methodology of Inventive Problem Solving, CRC Press, Boca Raton, New York, 2000.

Schmidt, L. C., Cagan, J., „GGREADA: A Graph Grammar-Based Machine Design Algorithm", Research in Engineering Design, Vol. 9, 1997, pp. 195-213.

Starling, A., Shea, K., „A Clock Grammar: the use of parallel grammars in performance-based mechanical synthesis", In Proceedings of the ASME DTM, Montreal, Canada, 2002.

Shea, K., Cagan, J., „Innovative Dome Design: Applying Geodesic Patterns with Shape Annealing", Artificial Intelligence for Engineering Design, Analysis and Manufacturing , 1997, pp. 379-394.

Stanković, T., Shea, K., Štorga, M., Marjanović, D., „Grammatical Evolution of Technical Processes", In the Proceedings of the ASME DETC 2009.

Suh, N. P., "Axiomatic Design", Oxford University Press, New York, 2001.

Weisstein, E. W., „The CRC Encyclopedia of Mathematics", CRC Press, 2009.

Dipl. Ing. Tino Stanković, Naval Architect
University of Zagreb
Faculty of Mechanical Engineering and Naval Architecture
Chair of Design and Product Development
I. Lučića 5, 10000 Zagreb, Croatia
Telephone: (+385 1)6168 369
Telefax: (+385 1)6168 284
Email: tino.stankovic@fsb.hr
URL: http://www.cadlab.fsb.hr