AN APPROACH FOR THE AUTOMATED SYNTHESIS OF TECHNICAL PROCESSES

Tino Stanković[1], Kristina Shea[2], Mario Štorga[1] and Dorian Marjanović[1]
(1) University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture, Croatia
(2) Technical University of Munich, Institute of Product Development, Germany

ABSTRACT
This paper considers the implications of introducing the computational method for technical process synthesis founded on the Theory of Technical Systems as an addition to the current Computational Design Synthesis (CDS) methods and tools. A computational method containing formal model of technical process based on labelled multidigraph and formal model of technical process synthesis that is based on labelled multidigraph graph-grammar transformations will be presented. The result of applying transformation to the multigraph is generation of variants showing how technical process could be accomplished.

Keywords: computational design synthesis, formal modelling, technical process synthesis

1 INTRODUCTION
Efforts in the research and development of computational support for conceptual design are motivated by a notion that such support could provide designers with novel or even creative concept alternatives as the result of computationally augmented solution space search. Generation, evaluation and selection of a concept variant are processes based on designers knowledge, experience in the field and information retrieved from the external sources [1]. If a computational system as an aid to design process can provide concept alternatives then designer being aware of these can make well established decisions in order to design new product more efficient. Computationally generated concept alternatives can thus serve as valuable starting points to design solution development. The field of Computational Design Synthesis (CDS) [2] brings together advanced computational algorithms with existing design theory and methodology in order to provide means for computational creation of design solution alternatives. Going beyond just dealing with multi-objective optimisation on well established solution concepts, the CDS considers design synthesis as its focal point thus being more or less aimed at early design phases where the synthesis is the most intensive. Most common [2], CDS is called for with a solution creation requiring generating and testing alternatives in numbers too large for a designer to seek out.
Work that is presented within this paper aims at the introduction of computational support intended for the very beginning of the computational design phase where according to the Theory of Technical Systems (TTS) [3] a technical process needs to be established. This work is thus coherent with the efforts made within the CDS research community as to provide better search and solution generation possibilities to designers by developing computational support for the early design stages [3]. However, what differs in respect to the current CDS efforts is the selection of the TTS as theoretic fundament thus recognizing the technical process synthesis as a requirement if the function of technical system that has to be designed is considered as the consequence of its participation within technical process. Moreover, according to the TTS, a technical processes synthesis is understood as a key element if an innovative design of new product is to be accomplished. As Section 3 of this paper will show, due to different methodology adopted the CDS methods in general do not consider technical process level, thus specifying the function of the product as an input to the solution generation what imposes limitations to the search space. Although none of the current CDS methods presents a complete framework adopting all of the conceptual design stages, the introduction of the technical process synthesis will prove beneficial when such frameworks will appear regardless to the methodology applied.
This work will present a computational method containing formal model of technical process (TP) based on labelled multidigraph and formal model of technical process synthesis that is based on graph

grammar transformations. The assumption is that if engineering knowledge about technical processes, technological principles and the necessary effects is formalized and embedded within set of graph grammar productions than the variants of operand transformation can be created. The result of applying productions is decomposition of technical process from the initial black-box form to the establishment of chain of operations with operands in their initial, intermediate and desired states supported by the necessary effects. The paper will include an example showing method's application and the resulting implications to the definition of technical system's function. Conclusion will close this paper.

2 THEORETICAL BACKGROUND

According to the Theory of Technical Systems [3] technical evolution, design and product development are explained as a response to those needs and requirements within human society for which, to be satisfied, an assistance of technical means was necessary. Such teleological view implies as a starting point to a development of a new product a definition of technical process as a process in which necessary effects must be delivered by technical product and human beings in order to enable purposeful transformation of operands. Following the systemic reasoning [3], technical process is modelled as transformation system composed of series of operations interrelated with operand flows and supported by the necessary effects. Changing of state of operands in the desired manner can be accomplished in different ways in respect to the various technological principles which prescribe and establish a sequence of necessary operations supported with effects that must be performed within transformation processes.

Based on designer's experience, the knowledge of existing technologies (technological principles) and on the understanding of the task, a decomposition of technical process is performed in order to gain insights and to reveal details about the transformation process. The consequence of decomposition performed is a conception of information necessary for design of technical system [1], [3]. In the decomposition process designers must consider different duties that human operator and technical system have to fulfil in order to enable transformation of operands. By reasoning how and with what to perform a transformation in order to achieve a desired state of operands designer establishes technical process. In the end, it is up to designer's knowledge to choose the most suitable technological principle by which the transformation will be accomplished.

Design theory states [1], [3], [4] that technical processes are established as sequences of operations based on different technological principles. Depending on the complexity of the given tasks, the technical process might be supported by several technical systems of different levels of complexity. The interplay between human operators and technical systems, i.e. the product being designed, results with the provision of the necessary effects. Possibilities of product realisation are therefore understood in respect to the extents of technical system's participation in the transformation. Determination of these duties as an ability to deliver necessary effects thus defines a technical system's function as an entry point by which the organ structure of a technical system will be established [2]. Implications and importance of the consideration of technical processes in respect to whole of the conceptual design was thus one of the prime motivators why to research computational means that could aid designers at that particular stage of design process. Taking into account how a technical system would participate within technical process is clearly at least equally important to the other design phases and since it is the first stage it may contribute the overall success of design process by the most.

3 RELATED WORK

To consolidate various approaches, methods and tools that emerged over the years, efforts were made to establish a generic model of a computationally supported design synthesis process. Two correlated models appeared in the literature: a generic framework [2] that proposed representation, generation, evaluation and guidance as four basic steps which must be addressed inside a computationally driven design synthesis process; and a performance-based framework [5] emerged for topological synthesis proposing investigation, generation, evaluation and mediation as steps of a parametric based computational synthesis. Although the two approaches differ slightly by the nomenclature, the content of the proposed is almost the same. To reflect on the CDS methods and tools development, the nomenclature according to the generic model of CDS [2] is adopted.

A compilation of the more recent and some of the older but very well known approaches to the CDS is shown in Table 1. Table columns are arranged according to: theoretical fundaments undertaken,

design phase for which support is intended and according to four steps comprising the CDS [2] (brackets denote fixed inputs). Rows 1-5 denote methods which differ significantly in the respect to solution representation and generation: A-design applies agent based approach in order to create meaningful solution concepts using a catalogue retrieved components [6], Hutcheson et. al. apply heuristic based search using genetic algorithm (GA) for morphological matrix search [7], the Concept generator [8] seeks out product function realisation possibilities using matrix algebra defined over function-to-component (FCM) and dependency structure matrices (DSM), CAM [9] uses state based search to create product architecture alternatives and SOPHY [10] aims at supporting designers by generating concept sketch, thus depicting the working principle on which that concept is based.

Rows 6-10 (Table 1) include CDS knowledge-driven approaches that apply production rules to formalise engineering knowledge of a particular domain. Solution alternatives are generated by automating derivation process computationally. All of these approaches are aimed at provision of computational support for the conceptual stages of product development. As seen from the Table 1, the most common is the application of graph grammars. Graph grammars are defined as production rule systems consisting of vocabulary and alphabet, and a set of rules for implementing graph transformations. In most cases computational support is provided for product function structuring and component configuration. Schmidt and Cagan developed GGREADA [11] which is an approach to support of mechanism synthesis developed as a mixture of configuration and catalogue selection design. With its predecessor the FFREADA, a function-to-form recursive annealing was applied to string of symbols to generate hand drill designs, where as GGREADA extended to graph grammars to generate concepts using components based on a *Meccano*® parts set. Siddiqque and Rosen apply graph grammars to develop a Product Family Reasoning System (PFRS) which would help designers in development of product platforms [12]. In their work two questions were addressed: how to establish common platforms for a set of different products, and the opposite, how to specify the product portfolio supported by the platform. First, the production rules were applied to generate a variety of product function structures which were then mapped to components containing relationships among functions and components. Answering how to specify the product portfolio supported by the platform required grammar definition as an acceptance grammar thus parsing the product architectures to see whether they fit in the language of the specific product family. Whereas HiCED [13] involves hierarchical coevolutionary approach in order to simultaneously evolve functions and components by combining genetic algorithms with graph grammar, BOOGGIE [14] developed according to function-behaviour-structure (FBS) product model tries to make use of the available graph grammar transformation tools and other open-source software packages in order to integrate them into a framework for synthesis of mechatronic products. Currently the framework only aims at variants generation without the optimisation support.

Rows 11-14 (Table 1.) contain the list of CDS methods intended for the embodiment design phase support. In general these approaches are shape or spatial grammar based, what is again a type of graph grammar transformation applied for topological synthesis. Schmidt et. al. developed a method for the automated synthesis of mechanisms, for epicyclical gear trains in specific [15]. Graph grammars were used to add vertices and loops to the initial start graph. With the interpretation of the resulting structure by processing vertices and edge labels the desired gear transmission ratio was obtained. Computational support of simulation driven microelectromechanical system (MEMS) synthesis was presented by Bolognini et. al. [16] based on the CNS-Burst method. The method was developed as a combination of Connected-Node System which is in fact a hypergraph based representation of MEMS systems, and a multi-objective generate-and-test search algorithm denoted as BURST. As the continuation of the previous work done by Starling and Shea for simulation driven synthesis [5] founded on the FBS modeling approach, a simulation-driven method for gearboxes synthesis was developed by Lin et. al. [17]. The component structure was represented using a virtual graph consisting of gear pairs and shafts thus depicting a power flow inside a gear-box. The system topology and geometry modification were derived by following a set of spatial grammar rules inside a simulated annealing search process. Grammar rules where ranked according to the performance of designes they created. Wu et. al. developed a systematic approach for automated support for design of mechatronic dynamic systems based on bond graph formalisms [18]. It is a simulation driven approach which requires as an input a conceptual definition of dynamic system to define a state space. For that purpose, a conceptual dynamics CD graph is introduced representing the information about the connections between components of the system.

Table 1. Overview of CDS methods and tools

Y	Medi Guid	Evaluation	Generation	Invest Rep	Scope: Plat A	Scope: Str C	Scope: Bet str Orga	Scope: S I	Scope: p	T D	Metho	Au	
		Method											
2008	hy Bui Evo	simulation Perfo Simulation-dr	graph sy A	Bond graph		x	x				D E	Wu a	1
2009	P	multiple weighting factors Multi-objective,	-gramm Spatial	Graph,		x					Synth A	Lin et	1
2007	princi nc Cl	Simulation Perfo Multi-obje Simulation dri	transf Ru	(Hypergraph) CNS		x					fo sy Cor	et. B	1
2000	Enumerative	on b Powertrain ra	iso based C	Labeled graph		x					of S	et. Sc	1
2009	Enumerative	P Simulation Dr Compone	Graph-grammar	Graph F-B-S		x	x	x		Be P	synth M BOOC	et. He	1
2007	Bui Evo	multiple weighting factors Multi-objective,	Graph-grammar	Binary GP Tree Fu		x		x		Be P	HiCED	Jin, Li	9
2005	S search Hy	Simulation Perf Multi-obje Simulation dr	grammar Struc Graph	Graph F-S		x	x	(x)		Be P	D driv for Pa	She St	8
1999	Enumerative	Acceptance Sub-graph	Graph-grammar	Graph Fu	x	x		(x)		Be P	PFRS	Ro S	7
1997	SA S M	Perfomace metrics embedding rules	Graph-grammar	Graph C Fu		x		x		Be P	GGREADA	Cag Sc	6
2010	Enumerative	Causality	autor expre Va	elem (PH Schema		x	x			TTS	SOPHY	et. Ri	5
2009	L C	Constraint based	applic Eler	(Graph	x						CAM	Wyatt	4
2006	Enumerative	Com C	Matrix algebra	Tree FCM		x		(x)		Be P	Concept Generat	et. Bry	3
2006	Hy Bui Evo	multiple weighting factors Multi-objective,	Heuristics	chart Morphological		x				Be P	Select Co	et. H	2
2003	learni TAB	Multi-objective, Pareto	Heuristics Agent based/	Catalogue based design							A-Design	et. C	1

4 FORMAL MODEL OF TECHNICAL PROCESS

Following the systemic reasoning, technical processes and technical products are most often modelled as transformation systems [1] which can be both formally and visually represented as graphs. The latter clarifies why the most of the early design computational support tend to utilize graph transformations for solution generation. Thus, depending on the abstraction level and context of respective early design stage, transformation system's elements can differ, but the basic graph representation is retained.

To model technical process formally it is necessary to introduce technical process related entities (*TP entities*); operands (*Od*), effects (*Eff*) and operations (*Op*) namely, into a graph's structure. Hence, operations are mapped to graph's nodes, where operands and effects are mapped to arcs thus creating multidigraph labelled over vocabulary of *TP entities* set Σ_G. Multigraphs are considered as non-simple graphs in which multiple edges between vertices, i.e. nodes, are allowed but no loops are permitted. In general, a multigraph G can be defined as an ordered pair (V, E), where V is a set of nodes and E a bag of edges. If a direction is required to represent binary relation between the vertices, than edges are replaced by directed edges or put succinctly by arcs.

Formal model of technical processes is defined as a labelled multidigraph $G = (V, E, s, t, l_E, l_V)$ with operands, operations and effects as ordered sextuple where V is a set of nodes and E is a bag of arcs, s and t are mappings returning source and target for each edge, and l_E are l_V mappings which assign every directed edge from E to operand *Od* or effect *Eff* and every node from V to operation *Op* respectively.

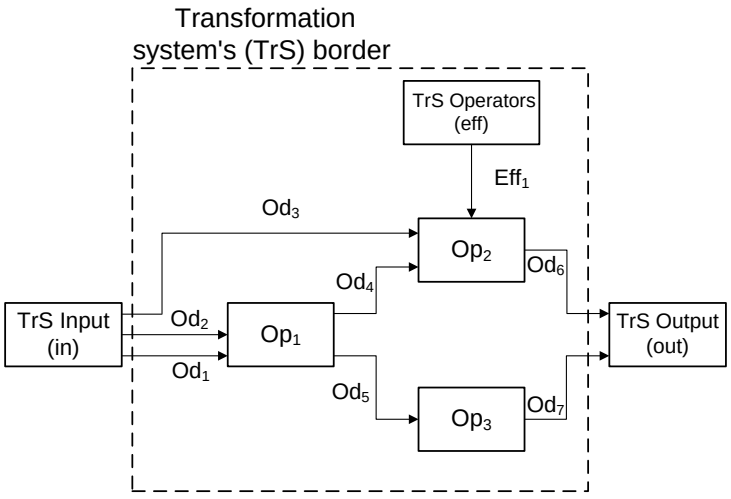


Figure 1. Example of technical process modelled with labelled multidigraph G

An arbitrary structure of technical process modelled as labelled multidigraph is shown in Figure 1. Operands $Od_1,...,Od_7$ can be understood as operands of different types (classes), or as operands of the same type but in varying states or both. These may be either the operands which transformation directly satisfies the existent users' needs, or the operands that emerge secondary as required or generated by the transformation system. Operations are represented as graph nodes labelled as $Op_1,...,Op_3$. Source and target nodes of transformation system's inputs and outputs are represented with *TrS Input* and *TrS Output* or simply as *in* and *out* labelled nodes. Likewise the source node of effects is denoted as *TrS Effects* or simply as *eff* labelled node. Necessary effect is represented as Eff_1.

In order to suffice the purposes of technical process modelling the following must be met:

1. the existence of minimally one *Op* labelled node, an operation that is, alongside *in*, *out* and *eff* labelled nodes, $|V| \geq 4$,
2. graph G has to be well connected only in respect to the transformation of operands, meaning that there must exist operation chain transforming *Od's* including *in* and *out* labelled nodes,
3. the isolation of the *eff* labelled node is permitted, denoting that the necessary effects do not have to be present at each and every decomposition step (concurs with 2.).

Labelled multidigraph based model of technical processes accepts *TP entities* as any type of process related objects which may possess their own set of attributes. At the current method's development stage *TP entities* are reduced to being labels only.

5 FORMAL MODEL OF TECHNICAL PROCESS SYNTHESIS

Graph grammars are means to perform a rule-based transformation of graphs. The application of rule first identifies a target structure, a sub-graph that is, inside a host graph, which has to be replaced by a new sub-graph. As the result of deletion of the old and integration of the new sub-structure with the remainder of original graph a transformed graph structure emerges.

Thus, in the respect of developing computational support for technical process synthesis applying a sequence of rules implies carrying through a series of transformations necessary for the creation of design solution obtaining the following [3]:

- structure of operand transformation process in technical processes corresponding to the operands' initial and desired states,
- identification of necessary effects required by the technology applied, and
- recognition of secondary flows which can appear as the result of operands' transformation.

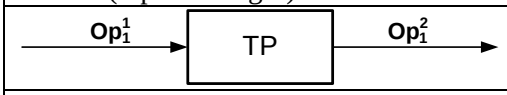
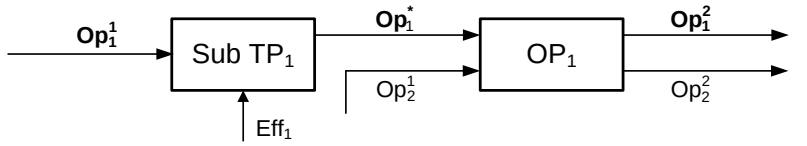
Thus, a successive derivation of all possible combinations of rules creates a design search space regarded as a formal language of technical processes for which the rules were defined for. The resulting design solution can be understood in a linguistical sense as a syntactically correct expression or a sentence composed of alphabet of formal language of technical processes.

Performing a local change to graph's structure is performed under the instructions given by the production rule $p : L \rightarrow R$. In fact, the engineering knowledge about how to decompose sub-process into set of interrelated operations is formalised within each of the productions. Interpretation and application of production p is addressed with the following:

- the left hand side of the rule L - a sub-graph modelled as labelled multidigraph that will be inserted in the host graph G (G is current decomposition step), L has only one Op node $|L| = 4$,
- the right hand side of the rule R - a sub-graph that will be inserted at desired place inside the graph G,
- exactly which part of host graph's structure will be replaced is defined by matching procedure $m(L)$, $m : L \rightarrow G$ which finds L in G,
- specification of how to reconnect R into the structure of G is defined by the transformation algorithm and the connecting procedure ρ.

Instead of performing graph based search, the matching procedure $m(L)$ is resolved by attaching a token to each operation within rule p or host graph G. Same token assumes literally the same mappings to *TP entities*. An example of the rule definition both using token based Backus-Naur notation [19] and graph-grammar representations is given in Table 2 (*in*, *out* and *eff* labelled nodes omitted for the sake of simplicity, principle operand transformation shown in bold):

Table 2. Rule definition example in token and graph-grammar representations

< *TP* > ::= < $SubTP_1$ > < OP_1 > (representing $p : L \rightarrow R$)
< *TP* > (representing *L*)

< $SubTP_1$ > < OP_1 > (representing *R*)


Transformation algorithm and connection procedure ρ are defined as following:

1. identify L in G,
2. subtract L from G,
3. identify interfaces as all of the dangling arcs that have lost source or target node as the result of applying 2. and collect these within interface set $intf_1$,
4. delete all the edges which are mapped to the effects both from G and $intf_1$,
5. delete *in* and *out* labelled nodes from R,
6. collect interfaces from R and put them in the interface set $intf_2$,
7. collect effect labelled edges form R and put them $intf_3$,

8. delete *eff* labelled node from *R*,
9. add *R* to *G*,
10. reconnect interfaces using matching of $intf_1$ and $intf_2$ with deletion of all the duplicates from $intf_2$, matching implies recognition as true if an edge from $intf_1$ and edge from $intf_2$ have the same operand labelling, and that if the edge from $intf_1$ has no source than the corresponding edge from $intf_2$ must be deprived of target node (vice versa is also acceptable),
11. copy effects from $intf_3$ to *G*,
12. reconnect the reminder of the interfaces from $intf_2$ as completely new secondary flows emerging from the input *in* or going out from the system towards *out*.

The definition of rule *p* is completely performed by designer. It involves definition of *TP entities* and their mappings to labelled multidigraph both for *L* and *R*. Designer defines a black-box technical process formulation with operands in their initial and desired states thus reflecting existing market and societal needs. As well, the constraints as an input to the search can be created. Technical process synthesis is then performed according to matching procedure *m*, set of productions of type $p : L \rightarrow R$ and transformation algorithm with connection procedure ρ. An example of technical process synthesis depicted as derivation sequences both in token based notation and graph-grammar is shown in Figure 2. (*in*, *out* and *eff* labelled nodes omitted for the sake of simplicity):

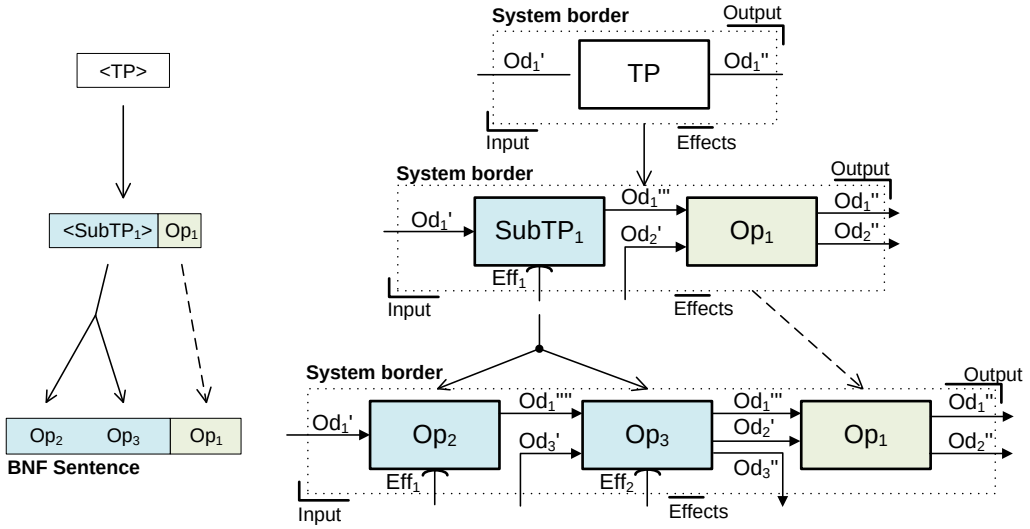


Figure 2. Derivations example of TP synthesis in BNF and in graph grammars

Reusing the secondary flow Od_2' as an output of Op_3 in order to feed Op_1 rather than importing the Od_2' from the outside of transformation system is the result of applying the connecting procedure ρ.

In order to provide goal based search a grammatical evolution (GE) [20] was added to the technical process synthesis method. Grammatical evolution is genetic algorithm based stochastic optimizer that seeks out the rule application sequence that is able to produce optimal solution under the given criteria. Since GE operate with token based BNF rules, it proved ideal as an addition to the method for graph grammar based technical process synthesis.

6 EXAMPLE

The purpose of the example is dual: to show the potential of the method for technical process synthesis and to elaborate how the variation on technical process level can yield in different function structure of technical system. The formulated task is a design of an automated assembly line that is able to deliver stiffened steel panels. What designer needs to gain are insights about working principles on which the transformation of operands is performed, as well as the necessary effects that need to be provided to sustain the transformation. Within this example's grammar (Figure 4. and Figure 5.), the process of stiffened panel assembly is divided within three logical steps: step one is the positioning of steel plates and their assembly into a steel panel, step two comprises of cutting of panel to desired dimensions and then, possible surface cleaning and setting of the markings for placement of stiffeners. The final step comprises of stiffener transport and its positioning. Step three is concluded with further distribution of the welded panel. In order to exemplify differences on a technical system level emerging as the result

of technical process search, welding and riveting are considering as two alternatives for the creation of stiffened panel. It is assumed that steel plates and stiffeners enter transformation in the state appropriate for appliance of those two technologies including welding joints or holes required for riveting. A black-box formulation of such process as it might be specified by designer, with operands in their initial and a desired state is given in Figure 3. Goal based search is specified as a completely automated welding process involving the least possible number of operations. Derivation stopping rule is set to length of 100 steps.

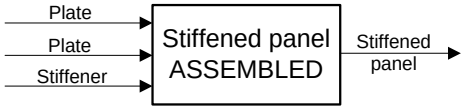


Figure 3. Stiffened panel assembly black-box process formulation

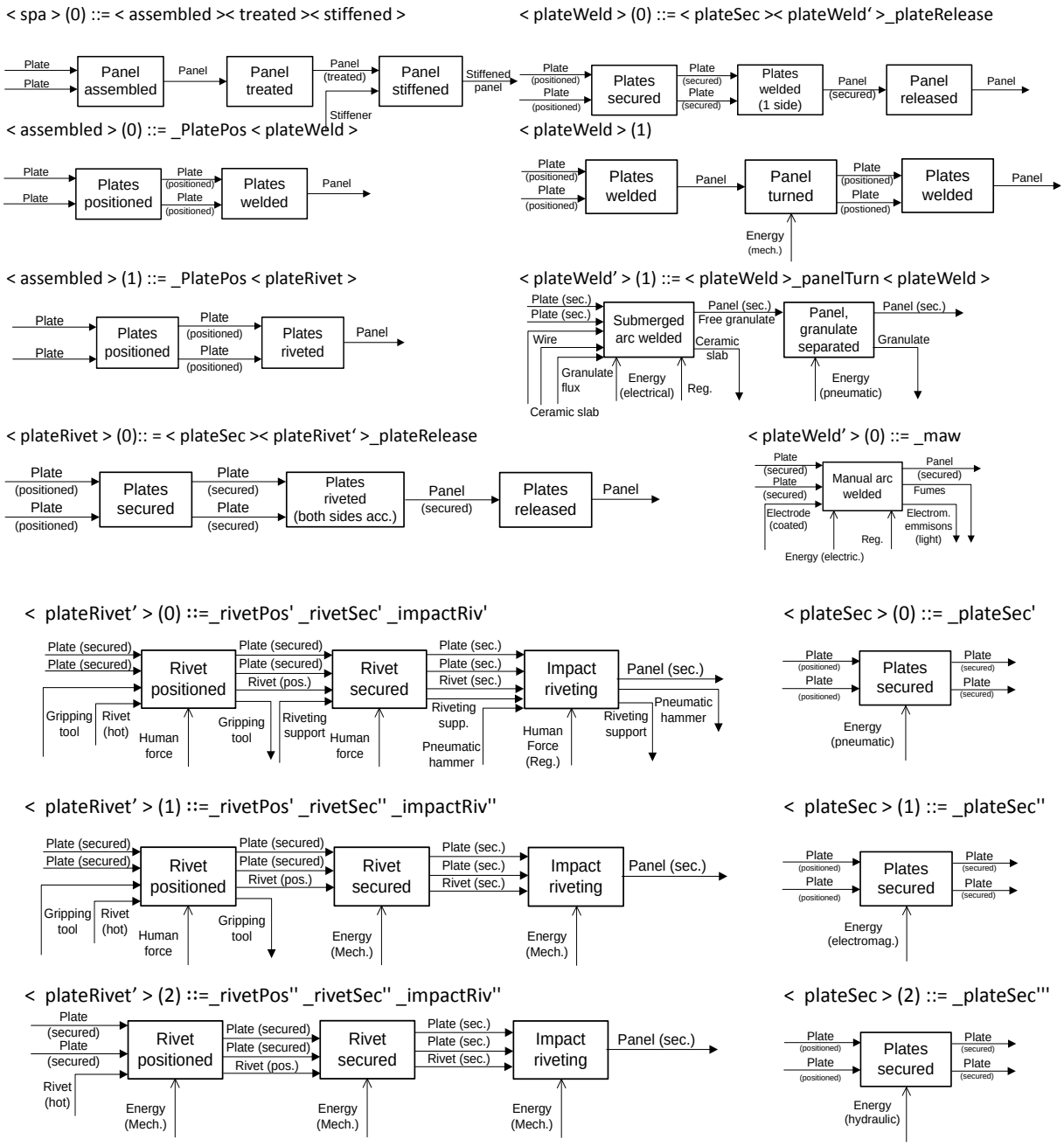


Figure 4. Graph grammar of stiffened panel assembly (Part I)

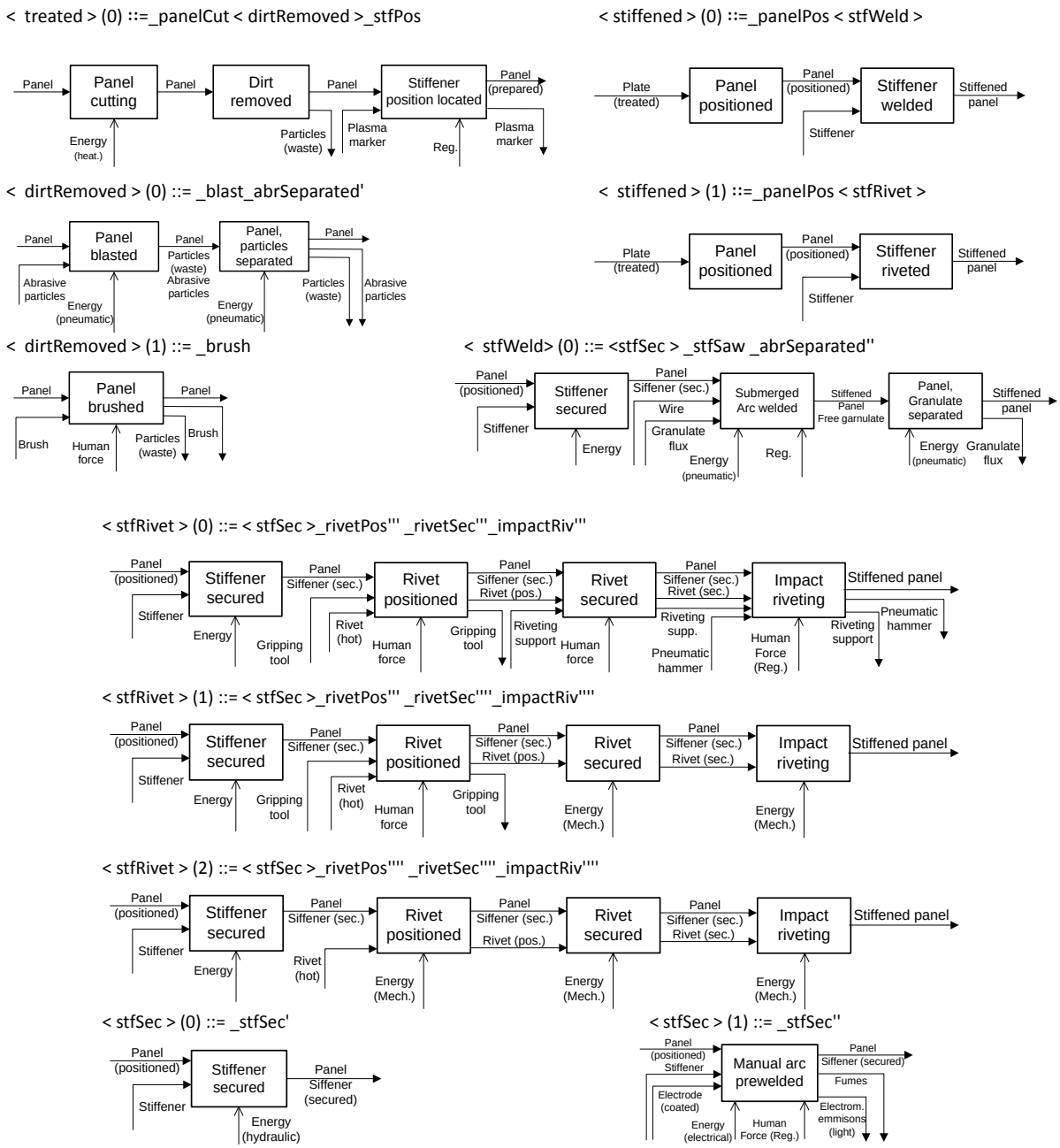


Figure 5. Graph grammar of stiffened panel assembly (Part II)

6 DISCUSSION

Number of technical process variants that can be created using grammar as defined in Figure 4 and Figure 5 equals 600 not taking into an account 4 branches involving < *plateWeld* > (1) (see Figure 5) for which the introduction of stopping rule was necessary. If only welding alone is considered, than 168 variants exists (Figure 6), only riveting yields in 108 variants. Combination of technological principles as welding of plates with riveting and riveting of plates with welding of stiffeners are the rest of transformation variants. It is important to stress out that the mechanisms on which GE is based, a combination of genetic algorithm and formal grammar, enable the creation only of the meaningful alternatives. Figure 6 depicts welding branch productions sequences only (left hand sides of productions shown, search goal solution is gray-shaded).

An example of how a variation on technical processes (TP) level may yield in different technical systems is shown in Figure 7. Because of the lengthy results involving multitude of operations, only excerpts from two technical variants are being depicted; one with fully automated panel riveting and the other with technical process variant involving fully automated panel welding (as gray-shaded

derivation sequence in Figure 6). Based on the required effects one or more technical systems could be designed in order to sustain technical process.

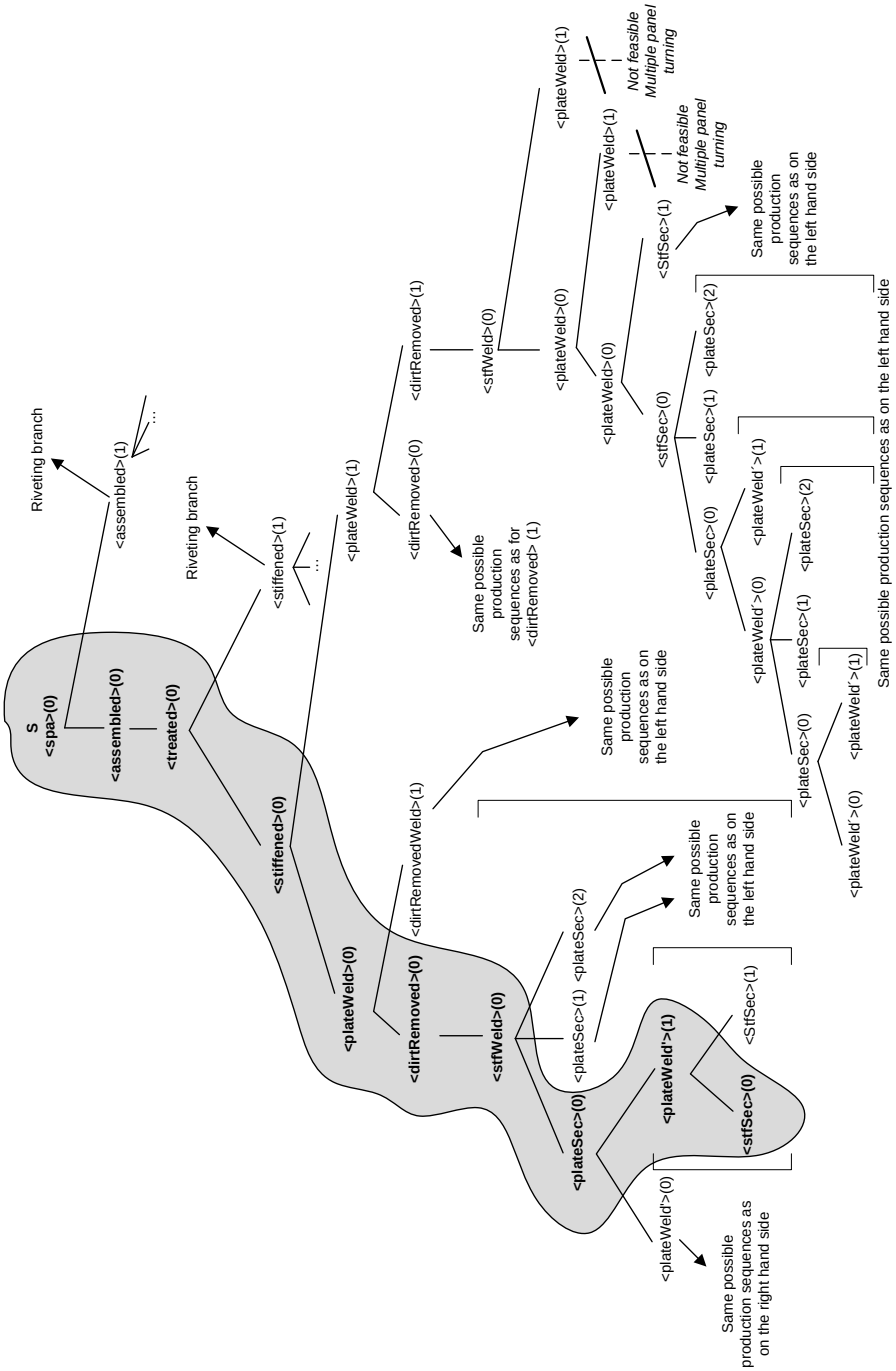


Figure 6. Welding branch productions sequences (search goal solution is gray-shaded)

This is the reason why the riveting based process is depicted with two technical systems instead of one. Technical system for riveting must be capable of provisioning the impact force, thus specifying one of the system's functions. Consequently, the technical system for welding must be capable of providing an electrical arc to be able to perform join of two plates into a panel. These two functions are direct consequences of different technological principles on which the operand transformation variants were founded. There is no other way in which these two technical system's functions could have emerged. The same reasoning holds for securing of rivets and removal of granulate. Moreover, the necessary output flows of technical systems, like rivets or welding wire for instance, are also the result of different technical process that needs to be supported (assuming that inputs and secondary outputs of technical systems are not the same).

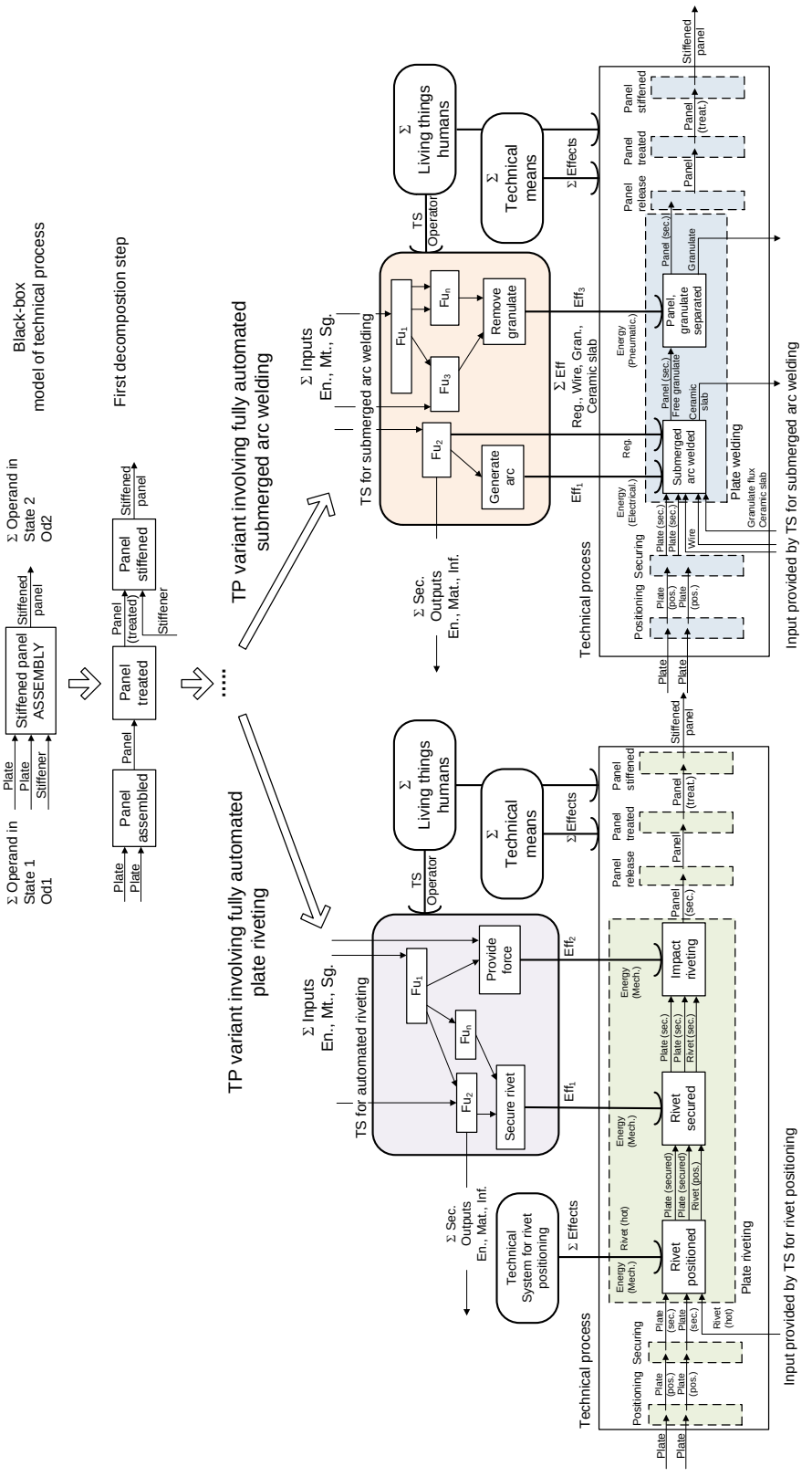


Figure 7. An example of how a variation on TP level yields in different technical systems

7 CONCLUSION

For the creation of computational support for the establishment of technical process synthesis formal models of technical process and technical process synthesis were presented. As shown in Table 1 the highest point of abstraction from which current approaches start is the functional level not recognizing technical processes at all. Further consideration of technical system at lower levels of abstraction cannot add new effects since they would redefine what a technical system should do within a

transformation system, and the only other way to accomplish that change is to affect the technical process inside where the main operand transformation is realized. As a proof an example involving stiffened panel assembly was shown.

REFERENCES

[1] Hubka V., Andreasen M.M., Eder E.W. *Practical Studies in Systematic Design*, 1998 (Butterworth & Co., London).

[2] Cagan, J., Campbell M.I., Finger S., Tomiyama T. A Framework for Computational design synthesis: Model and Applications, *Journal of Computing and Information Science*, 2005.

[3] Hubka V. and Eder W.E. *Engineering Design: General Procedural Model of engineering Design*, 1992. (Springer-Verlag, Berlin, Heidelberg).

[4] Hubka V. and Eder W.E. Theory of Technical Systems and engineering design synthesis, Chakrabarti A. (Ed.), 2002, (Springer-Verlag, London).

[5] Shea K. and Starling A.C. From Discrete Structures to Mechanical Systems: A Framework for Creating Performance-Based Parametric Synthesis Tools, *American Association of Artificial Intelligence-Technical Report*, 2003.

[6] 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*, 1998, Kluwer Academic Publishers, Netherlands.

[7] Hutcheson R.S., Jordan R.L., Stone R.B. *Application of a Genetic Algorithm To Concept Variant Selection*, 2006, In Proceedings of the ASME-DETC.

[8] Bryant C.R., Stone R.B., McAdams D.A., Kurtoglu T., Campbell M.I. *Concept Generation from the Functional Basis of Design*, 2005, In the Proceedings of the ICED05.

[9] Wyatt D.F., Wynn D.C., Clarkson P.J. *A Computational Method To Support Product Architecture Design*, 2009, In the Proceedings of the ASME-IMECE.

[10] Rihtaršić J., Žavbi R., Duhovnik J. *Sophy – Tool for Structural Synthesis of Conceptual Technical Systems*, 2010, In the Proceedings of DESIGN 2010, Dubrovnik, Croatia.

[11] Schmidt L.C. and Cagan J. GGREADA: A Graph Grammar-Based Machine Design Algorithm, *Research in Engineering Design*, 1997, Vol. 9, pp. 195-213.

[12] Siddique, Z. and Rosen D.W. *Product Platform Design: A Graph Grammar Approach*, 1999, DETC99/DTM-8762, ASME.

[13] Jin Y. and Li W. Design Concept Generation: A Hierarchical Coevolutionary Approach, *Journal of Mechanical Design*, 2007, Vol. 129, pp. 1012-1022.

[14] Helms B. and Shea K. *Object-Oriented Concepts for Computational design synthesis*, 2010, In the Proceedings of DESIGN 2010, Dubrovnik, Croatia.

[15] Schmidt L.C., Shetty H., Chase S.C. A Graph Grammar Approach for Structure Synthesis of Mechanisms, *ASME - Journal of Mechanical Design*, 2000, Vol. 122, pp. 371-376.

[16] Bolognini F., Seshia A.A., Shea K. *Exploring the Application of a Multi-domain Simulation-based Computational Synthesis method in MEMS Designs*, 2007, ICED'07, Paris, France.

[17] Lin Y., Shea K, Johnson A., Coultate J., Pears J. *A Method and Software Tool for Automated Gearbox Synthesis*, 2009, ASME, IDETC/CIE.

[18] Wu Z., Campbell M.I., Fernandez B.R. Bond Graph Based Automated Modelling for Computer-Aided Design of Dynamic Systems, *Journal of Mechanical Design*, 2008, Vol. 130, ASME.

[19] Naur P. *Revised report on the algorithmic language ALGOL 60*, 1963, ACM, 6(1), pp. 1-17.

[20] O'Neill M. and Ryan C. Grammatical evolution, *IEEE Transactions on Evolutionary Computation*, 2001, Vol. 5 (4), pp. 349-358.

Contact: Dr. Tino Stanković, Dipl.-Ing., Ph.D. ME
University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture
Ivana Lucica 5, Zagreb, Croatia
Tel: +385 1 6168 432, Fax +385 1 6168 286
Email: tino.stankovic@fsb.hr, URL: www.cadlab.fsb.hr

Tino is postdoctoral researcher at the Department of Design at the University of Zagreb Faculty of Mechanical Engineering and Naval Architecture. He is interested in computational design synthesis, formal modelling, multi-objective optimization and evolutionary computation.