Journal of Engineering Design
iFirst, 2012, 1–28

Formal modelling of technical processes and technical process synthesis

Tino Stanković[a]*, Mario Štorga[a], Kristina Shea[b] and Dorian Marjanović[a]

[a]*Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Zagreb, Croatia;* [b]*Engineering Design and Computing Laboratory, Department of Mechanical and Process Engineering, ETH Zurich, Zurich, Switzerland*

(*Received 4 October 2011; final version received 14 August 2012*)

The computational design synthesis approach considered in this paper, proposes directed multigraph and graph grammar-based models of technical processes and technical process synthesis. The theory of technical systems, which is adopted as a theoretical foundation for this work, assumes a teleological viewpoint bringing together the purpose of technical systems and fulfilment of customer demands and societal needs. These demands and needs are met by means of a technical process inside which the operands are transformed with the assistance of a technical system to achieve a desired state. Formal models of technical processes and technical process synthesis establish the foundation for further application of search algorithms to support early engineering design. The engineering knowledge about technical processes is provided within a set of graph grammar rules. As the result of the proposed approach, the designer is enabled to consider different operand transformations in an expedient fashion with the possibility of the generation of novel alternatives. The proposed approach is illustrated through an example of the design of a stiffened panel assembly line involving welding and riveting as two basic principles.

Keywords: theory of technical systems; computational design synthesis; graph grammars; formal modelling; technical processes

1. Introduction

Technical evolution, with design as its principal activity (Simon 1996), can be understood as a response to needs and requirements of human society, for which to be satisfied, assistance by technical means is necessary (Asimow 1962). In a teleological sense, the justification for a particular technical system's existence is enclosed within its purpose. Thus, engineering design implies at least an acknowledgement of the socio-technical context that amongst others examines the interactions between human operators, technical systems, and the environment (Asimow 1962). These interactions define a technical system as being far from just a physical embodiment realised with a disregard for its surrounding environment but that it contains the principles according to which technical systems participate, or how and in what way they might be used, in the processes of fulfilling various needs and requirements. For that matter, according to the systems

*Corresponding author. Email: tino.stankovic@fsb.hr

ISSN 0954-4828 print/ISSN 1466-1837 online

http://dx.doi.org/10.1080/09544828.2012.722193
http://www.tandfonline.com

theory (Gharajedagi 2011), the viewpoint of a single cause–effect relationship is extended by the notion of plurality, which considers the possibility that systems in general may attain multiple structures and exert multiple behaviour modes. Additionally, they can be run by, or participate in, different processes. Rather like problem solving, the technical process in which a technical system participates (Hubka and Eder 1992) is a transformation process, the execution of which exerts purposeful change on the environment by transforming inputs to outputs. The plurality of processes states that there might be numerous working principles by which that transformation may be accomplished (Bertalanffy 1969). Thus, as its structure is not solely dependent on the inputs and outputs, small changes in inputs may result in completely different outputs. Therefore, the structure and behaviour of a technical system are interconnected with the technical process in which it participates and supposedly propels. Both the modalities of a technical system's usage as governed by its operator and the various principles, whose combination accounts for the transformations results in a satisfactory fulfilment of need, should be considered or at least, should be acknowledged before the establishment of a technical system's structure.

This work discusses and proposes formal modelling of technical process synthesis, which according to the theory of technical systems (TTS) (Hubka and Eder 1992, 2002) initiates the conceptual design phase, especially for novel engineering design. In abstracting the design process as a spanning tree to show the transformation of the problem specification into a detailed technical description through a series of design decisions (Andreasen and Hein 1987), the causality within the design process is clearly indicated, emphasising the importance of the initial stages as they effect significant alterations in the steps that follow. The formal modelling that is presented in this work is intended to establish a foundation for the creation of a computational-based support for designers in the technical process synthesis design phase. As such, this work also belongs to the research area of computational design synthesis (CDS), in which there is a deficiency in the support of technical process synthesis (as elaborated in Section 3). With respect to the CDS generic framework (Cagan *et al.* 2005), this work addresses the representation and generation steps of the formal synthesis procedure outlined. A formal system realised by means of a graph grammar, encapsulating the engineering knowledge regarding working principles (technology), is used to perform the synthesis of technical processes. It is expected that the formalisation of technical process synthesis through the developed logical framework presented in this paper, will allow currently available algorithms and search techniques to speed up and to make the solution search process more thorough and efficient.

The motivation for proposing the formal model of technical process synthesis, as well as the theoretical foundations, is discussed in Section 2 of this paper. Section 3 presents the state-of-the-art of CDS. The most recent approaches tackling design and product development are compared and analysed to show that almost none of these methods address transformation processes in general or technical process synthesis in particular. The conclusions are drawn to justify the selection of graph grammars for technical synthesis formalisation. Section 4 proposes a graph-based formal model of technical processes, as well as a graph grammar-based decomposition procedure, in order to conduct the synthesis of technical processes. An illustrative example is presented that will show how through a series of graph grammar transformations sufficient information is gained to understand and specify the function of the technical system. Discussion of the results and recommendations for future work close the paper.

2. Design theory background

The TTS, the viewpoints of which are adopted as the design theory foundations of this research, is concerned with studying technical systems as artefacts that are of a technical or engineering nature (Hubka and Eder 1992). It also brings together how technical systems came to be with the methods and processes that conceived and created them. Acknowledging the interactions between the technical system, human operator, and the environment, the TTS models the socio-technical context through the introduction of a technical process synthesis, inside which designers are urged to reason about the possible ways and principles by which societal needs can be purposefully met with the assistance of a designed technical system. With respect to the latter, a technical process is defined as an artificial process in which the state of an operand is intentionally transformed under the influence of effects delivered from a technical system, a human operator, and the environment (Hubka and Eder 1992). Backed by the available state-of-the-art reviews on approaches to functional modelling of technical systems (Erden *et al.* 2008) and design process activity structuring (Sim and Duffy 2003), it can be concluded that the vast majority of design theoretical approaches blur the distinction between technical processes and the functions of technical systems. Unlike the transformation within a technical process, the transformation occurring within technical systems is performed solely by the technical system itself with the goal of delivering the necessary effects to support the technical process (Hubka and Eder 1992, 2002). With respect to the latter, the transformation within the technical system is conceived with the goal to realise the effects necessary to make possible the transformation within the technical process. These effects are the result of technology (working principles) that is based on the principles of the physical laws used to structure the technical system. Thus, the task of a designer is to consider a number of variants of technical processes based on different technological principles and to select the most suitable variant that produces the required transformation of operands under some given set of criteria. Here, the technology can be understood as a linking element that clearly puts into the relation a purpose and a function of the technical system, with the latter being defined as the capability to produce the necessary effects. It can be concluded that by omitting technical process synthesis from the design process, valuable information, such as modes of user-product interaction or technological principles, by which the technical system is driven and correspondingly the function of the technical system is conceived, can be overlooked and realisation possibilities left unexplored.

2.1. *Synthesis of technical processes*

Synthesis of technical processes is carried out by the decomposition of higher-level sub-processes, inside fixed systems boundaries, resulting in a growing number of primitive elements, i.e. operations, interconnected with flows (Figure 1).

The object that is undergoing the transformation in the technical process is regarded as an operand (see Figure 1); a passive member in the technical process that is the subject of both structural and behavioural changes. Depending on the selected available technologies, which differ in the principles by which the change of the operand's state is performed, the designer decomposes the high-level processes into sub-processes and operations, in order to obtain more detailed transformation sequences. In terms of TTS, applying a technology is always considered together with the assistance of the supporting technical systems, which is realised by providing various effects to sustain the transformation (Figure 1). As required by the applied technology, the additional secondary inputs (disturbances) and resulting outputs may appear as well. Decomposition lasts until the designer gains sufficient insight to grasp all of the relevant aspects, clearly identifying the functional requirements of the technical system that will have to be designed.

Although the decomposition of the technical processes at first glance may appear as purely analytical, to obtain the technical system function specification also requires synthesis. Namely, the establishment of the technical processes involves decomposition, but to perform this, a structuring of the technical process is necessary. The definition of a system's structure is a synthetic

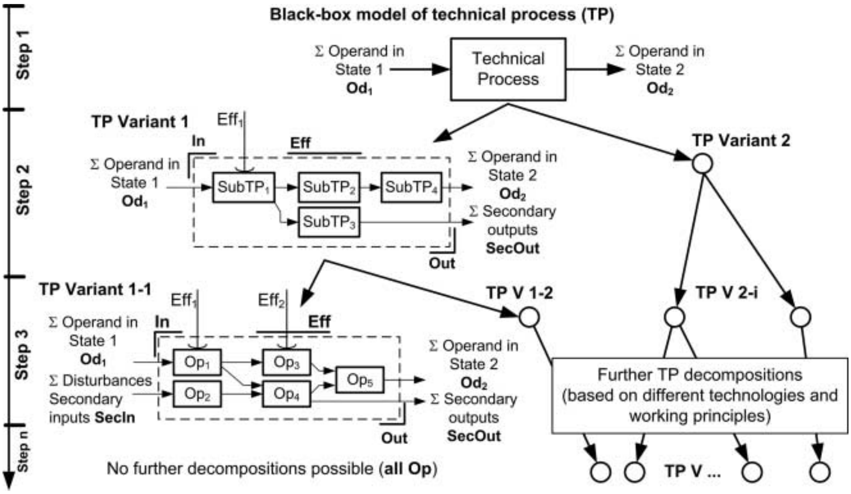

Figure 1. Decomposition of technical process.

activity (Simon 1996), which in the case of the technical processes, occurs at every level of the decomposition process: the determination of operations with necessary effects based on selected technological principles, the setting up of the system boundaries and the relating of the operations with operand flows in different states are all required to specify the transformation process. It can be said that the analytic part of the technical process decomposition is concerned with the insights by which the function of the technical system is determined, but to attain these, a transformation process synthesis is required at each and every step of decomposition.

3. State-of-the-art review on CDS

CDS is a complex multidisciplinary research area that involves the algorithmic creation of designs using computers and often includes a formal approach to design and design space modelling (Cagan *et al.* 2005). Advanced computational techniques and search algorithms are used to conduct efficient search across the design space, in order to provide a foundation on which designers can make well-informed decisions. A summary of the state-of-the-art of recent CDS approaches for the early product development phase is given in Table 1. The columns in the table are organised to show the scope and theoretical foundations of each of the approaches analysed, as well as how it managed to realise the four generic steps involved in the formal synthesis process, i.e. representation, generation, evaluation, and guidance (Cagan *et al.* 2005). The bracketed entries in scope columns of Table 1 denote the required inputs for the synthesis approach to operate.

Another view by which CDS methods and tools may be differentiated is according to their general approach to computational problem solving (Goldstein and Papert 1977): method and tool development may be founded on the design of better mathematical algorithms, search techniques, and computational paradigms (e.g. parallel processing), or through an epistemological approach that looks for better ways to express, recognise, and use diverse and particular forms of knowledge. Therefore, a non-strict division could be applied to Table 1 with the result that the analysed approaches in rows 1–10 are in fact knowledge based, whereas the approaches in rows 11–14 rely on mathematical algorithms and search techniques. The non-strictness of the division is to emphasise that approaches that are, for instance, agent based (row 13 in Table 1), depend on emergent complex behaviour that is not explicitly algorithmic nor knowledge based.

Table 1. Overview of CDS methods and tools.

				Scope					Realisation of CDS steps			
	Authors	Method	Design theory	Technical process	Function structure	Organ structure/ behaviour	Components/ structure	Architecture/ platform	Representation	Generation	Evaluation	Guidance
1	Bolognini *et al.* (2007)	Computational synthesis for MEMS design					X		CNS Graph	Rule-based spatial graph transformations	Pareto multi-objective Simulation driven performance metrics	CNS-BURST with non-domination principle
2	Lin *et al.* (2009)	Automated gearbox synthesis					X		Graph, Power flow paths	Spatial grammar	Multi-objective, multiple weighting factors	Past performance rule selection Simulated annealing SA
3	Schmidt and Cagan (1997)	GGREADA	Pahl and Beitz		X		X		Function structure Component structure Graph	Graph grammar	Performance metrics embedded inside rules	Simulated annealing SA
4	Siddique and Rosen (1999)	PFRS PC/PSPV	Pahl and Beitz		X		X	X	Function structure Component structure Graph	Graph grammar	Sub-graph isomorphism Acceptance grammar	Enumerative

(*Continued*)

Table 1. Continued.

			Scope					Realisation of CDS steps			
Authors	Method	Design theory	Technical process	Function structure	Organ structure/ behaviour	Components/ structure	Architecture/ platform	Representation	Generation	Evaluation	Guidance
5 Starling and Shea (2005)	Parallel grammar for simulation-driven mech. design	Pahl and Beitz		X	X	X		FBS Graph	Graph grammar Structure grammar	Pareto multi-objective Simulation driven performance metrics	Hybrid pattern search Simulated annealing SA
6 Schmidt *et al.* (2000)	Structure synthesis of mechanisms					X		Labelled graph	Graph grammar Grammar based rules for isomorphism detection	Powertrain ratios on basis of labelled nodes and edges	Enumerative
7 Jin and Li (2007)	HiCED	Pahl and Beitz		X		X		Function structure GP tree Binary string	Graph grammar	Multi-objective, multiple weighting factors	Evolutionary Building-block hypothesis GP, GA

8	Wu *et al.* (2008)	Bond graph, CAD of dynamic systems			X	X	Bond graph	Automated mapping from system concept to bond graph	Simulation driven performance metrics	Evolutionary Building-block hypothesis GA
9	Helms and Shea (2012)	BOOGGIE mechatronic design synthesis	Pahl and Beitz	X	X	X	FBS Graph	Graph grammar	Component compatibility Simulation driven Performance metrics embedded in rules	Enumerative
10	Wyatt *et al.* (2012)	CAM toolbox EPA			X	X	Schema Configuration graph	Elementary operations applied to schema	Constraint based	Constraint-based Depth-first search
11	Hutcheson *et al.* (2006)	Concept Variant Selection	Pahl and Beitz			X	Morphological chart	Heuristics	Multi-objective, multiple weighting factors	Evolutionary Building-block hypothesis GA

(*Continued*)

Table 1. Continued.

				Scope					Realisation of CDS steps			
	Authors	Method	Design theory	Technical process	Function structure	Organ structure/ behaviour	Components/ structure	Architecture/ platform	Representation	Generation	Evaluation	Guidance
12	Bryant *et al.* (2005)	Concept Generator	Pahl and Beitz		X		X		FCM Tree	Matrix algebra	Component occ. frequency based ranking Component compatibility from DSM	Enumerative
13	Campbell *et al.* (1998)	A-Design					X		Catalogue-based design	Agent-based Heuristics	Pareto multi-objective	Tabu-based learning
14	Rihtaršić *et al.* (2010)	SOPHY	TTS			X	X		Schema Physical law, wirk elements-ports	Variation of chains of linear expressions Semi-automated sketch generation	Causality	Enumerative

3.1. *Formal grammars-based CDS approaches*

A production system is a formal system designed to perform transformation of a certain input to a particular output, using a set of condition-action rules that can be applied whenever conditions to do so have been met (Levet 2008). Initially, Post's (1943) production systems transformed strings as sequences of symbols belonging to a specified fixed vocabulary. These are found extensively in linguistic theory for establishing formal grammars, because they are a type of production system capable of describing linguistic structures, thus formalising the language under consideration (Chomsky 1957). Formal grammars became widespread in other domains, such as: theory of computation, artificial intelligence, automated problem solving, image processing, resulting in the development of formal systems that are able to accept and transform more varied types of structures, including terms, trees, and graphs.

3.1.1. *Shape and spatial grammars*

The use of formal grammars for architecture and visual arts applications was first presented by Gips and Stiny (1980), who developed shape grammars as a production system that specifies a set of design solutions called a language, by the transformations required to generate that set. Shape grammars assume the transformation of shapes, which involves recognition of sub-shapes and their replacement with new shapes. To help specify the context of the productions, a marker is commonly used to denote the replacement origin. If required for the purpose, the labelling of shapes may be applied as well. With respect to engineering design and product development, shape grammars are most commonly applied to support topological synthesis in the design phase, e.g. shape annealing arising by unifying shape grammars with a simulated annealing algorithm (Shea 1997, Shea and Cagan 1998). These were successfully applied for solving the topological optimisation problems of truss structures involving both in-plane and in-space problems. Also, a series of industrial design papers attempted to identify brand style features and then generate solutions using these specific styles embedded within grammar rules; this research being predominantly motivated for vehicle applications (Pugliese and Cagan 2002, McCormack and Cagan 2004).

An interesting approach is the one developed for computational support of simulation-driven microelectromechanical system (MEMS) synthesis (Bolognini *et al.* 2007). A connected-node system (CNS)-Burst method was developed as a combination of a CNS, which is in fact a hypergraph-based representation of the MEMS system and a multi-objective generate and test search algorithm, called Burst. The search principle is based on the procedure where the CNS modification operators are applied in short bursts to the current system's layout. Frequencies of modifications are user-defined. A special evaluation module was designed to obtain performance metrics of the created system, by which a non-dominated solution population is identified.

A spatial grammar-based method for gearbox synthesis was developed by Lin *et al.* (2009). The component structure is represented using a virtual graph consisting of gear pairs and shafts, which depicts a power flow inside the gear-box. The system topology and geometry transformations are derived by following a set of spatial grammar rules inside a simulated annealing search process. Grammar rules are ranked according to the performance of the design they created.

3.1.2. *Graph grammars*

Graph grammars are defined as production systems consisting of vocabulary and a set of rules for implementing graph transformations. As either vertex or edge replacing, the productions must contain some additional embedding procedures through which the inserting structure is integrated with the remainder of the graph's structure. Most commonly, these procedures include edge reconnection or node labelling to provide guidance for the matching. Schmidt and Cagan (1997) developed a graph grammar-based machine design algorithm (GGREADA). Built on the foundations of the function to form recursive annealing design algorithm (FFREADA) that uses a string of symbols to generate hand drill designs, GGREADA uses graph grammars to generate concepts using components based on Meccano® parts. GGREADA is a mixture of configuration and catalogue selection design. Function-to-form transformation is realised top-down with components realising product functions. Also, function-sharing was implemented to allow for one component to realise several functions. GGREADA uses simulated annealing to recursively evolve a product on both function and component levels. The objective function uses multiple weighting factors.

Siddiqque and Rosen (1999) applied graph grammars to develop a product family reasoning system to assist in the design of product platforms. Two questions were addressed: how to establish a common platform for a set of different products and conversely, how to specify the product portfolio supported by the platform. By using sub-graph isomorphism to recognise similarities in the products' structures, common functions are identified. Based on this, production rules are defined to generate a variety of product function structures, which are then mapped to components containing function-to-component relationships. To address the second question, an acceptance type grammar was applied to parse the product architectures, in order to see whether they fitted the language of the specific product family. A different domain approach, which also involved detection of similarities among graphs, was developed for the automated synthesis of epicyclical gear trains (Schmidt *et al.* 2000). Graph grammars are used to add vertices and loops to the initial start graph. By processing the vertices and edge labels to interpret the resulting structure, the desired gear transmission ratio is obtained. Additional graph grammar rules are added for identifying isomorphic graphs to produce unique solution variants only.

A parallel grammar for mechanical design synthesis was developed by Starling and Shea (2005), to investigate the feasibility of the simulation-driven environment, in order to produce better quality designs. To achieve this, a cross-domain modelling language Modelica® inside Dymola® is used to obtain the simulation results. Precompiled simulation executables are used that only required a parameter value update within the input files. A parallel grammar is established based on the function-behaviour-structure (FBS) product representation (Umeda *et al.* 1996). Two types of rules are applied: function-grammars, which generate function structure using predefined building-blocks and structure-grammars, which then create parametric component structure as a simulation starting point. The Pareto optimal set is identified using a hybrid pattern search algorithm.

A good example of how to tackle the problem of computational concept generation using grammars was presented by Jin and Li (2007). A hierarchical co-evolutionary design approach assumes iterative co-evolution of products on different abstraction levels. First, based on the knowledge stored inside a rule library, an initial population of functional decompositions is created. Then, genetic programming and a genetic algorithm are triggered to co-evolve the products' functions and components as functional means. Functional and component structures are represented as simple flow graphs. The fitness function is formulated using weighting factors.

Wu *et al.* (2008) developed a systematic approach for an automated design of mechatronic dynamic systems based on bond graph formalisms. It is a simulation-driven approach, which requires a conceptual definition of a dynamic system as an input to define a state space. For this purpose, a conceptual dynamics graph is introduced, which represents information about the relationships between the components of a system. Generic models of components having various types of connection possibilities are stored within a repository. A dynamic model of a system represented with state-space equations is automatically generated out of the defined concept using bond graph transformations with user-defined goals. Optimisation is performed using a real-valued genetic algorithm with an individual solution genotype derived based on a hierarchical representation of the component design rules, constraints, and physical laws.

A recent framework that brings object-oriented graph grammars into engineering is developed according to the FBS product model (Helms and Shea 2010, 2012). The framework considers mainly the top-down approach of automatically decomposing the product on all three levels of FBS to generate, for example, alternative hybrid powertrain concepts. An object-oriented meta-model is used incorporating the different levels of abstraction, i.e. FBS, and defines interconnections between vocabulary, both within one level and between levels, based on the definition of ports. The definition of ports enables the use of generic rules at all levels, which are independent of the vocabulary definition, through port matching, in addition to allowing the graphical description of application specific graph rules.

3.2. *Other approaches in CDS*

An approach to computationally explore possible architectures (EPA) was developed by Wyatt *et al.* (2009, 2012) and implemented as a part of Cambridge advanced modeller toolbox. The basic principle of EPA claims that for any given initial architecture, any other architecture of that product is reachable through a state-space search process, by carrying out a sequence of transformations. Therefore, a designer using a graphical modelling language defines a schema as a graph composed of a finite set of different relations, components, and logical constraints. The schema logically frames an architecture state space. Using a depth first search, elementary transformations on the initial product architecture are executed and then tested against the proposed schema. Two evaluation metrics are changeability to represent immunity to change propagation and designability to show a required design effort.

The problem of generating optimal concepts out of a morphological chart was reduced to a combinatorial genetic algorithm-driven approach by Hutcheson *et al.* (2006). The approach only dealt with problems having a number of functions for which it was meaningful to represent their relationship inside a simple chain. When considering more complex structures, the user had to compose the function structure in such a manner that the overall structure is reducible into a series of function chains. For the creation of functional models, it was proposed to use a standard taxonomy as defined by the National Institute of Standards and Technology of USA (Stone and Wood 2000). The validation of the solution principles is performed by the energy flow compatibility check with the search objective formulated as a weighted fitness function.

A similar approach was undertaken with the Concept Generator, which is a computational tool developed by Bryant *et al.* (2005), intended to create design solutions by establishing a mapping from a predefined function structure to components using matrix algebra. Solutions are generated on the basis of a web-based repository of function-to-component matrices (FCM). The FCM shows those technical solutions that can realise a given function and design structure matrices (DSM) in which the component-to-component compatibility with respect to energy flows is defined. Ranking is achieved by comparing the frequency of occurrence of the components inside the generated solutions, with the data gathered from over 70 consumer products.

A somewhat different approach was presented with A-Design (Campbell *et al.* 1998), which included a collection of software agents to create meaningful solution concepts out of a component catalogue. Different agent types are developed: configuration agents that perform an interface-based connection of components managed by an input–output type compatibility check, instantiation agents the duty of which is to retrieve new components from the catalogue and fragmentation agents that segmented solutions and preserved them to be improved in later iteration steps. Based on their merit of performance, a manager agent determined those agents that would cooperate with a bit of randomness included, to avoid local optima. Learning was achieved in a process similar to the Tabu search algorithm; designs were identified as Pareto optimal and as good or bad and were stored as such. The user had a chance to affect the evolution course adapting it to their own preference so that the created designs successfully met the criteria.

SOPHY, which stands for a synthesis of physical laws (Rihtaršić *et al.* 2010), is a tool aimed at supporting designers by generating a sketch of a concept, clearly depicting the working principle on which that concept is based. A similar approach based on a chaining of equations was utilised some years ago for the development of mechatronic systems (Weber 2005). The process of sketch generation is performed with the designer's assistance, using predefined physical law-based schemas, which are coupled into a complete concept.

3.3. *Discussion on state-of-the-art review*

What is clearly visible from the presented analysis (Table 1) is that most of the CDS approaches for the support of the early design phase use graph-based representations and graph-grammar-based solution generation as an established principle. Almost all of the design theories model high-level product representations as systems that are often formally and visually represented as graphs, i.e. function structures, process flow diagrams, and so forth. Taking the latter with the possibilities of formal grammars to efficiently capture the knowledge on graph transformations (Kreowski *et al.* 2006), it becomes clear that the graph grammars are suitable formalisms for the modelling of technical process synthesis.

As shown in Table 1, the highest level of abstraction, according to TTS, from which the recent CDS approaches are able to provide support, is the functional level of product abstraction making no reference to technical processes or transformation systems at all. The methods analysed most often address only the technical system's functions and their realisation by a catalogue-based selection of components, or the methods deal only with the establishment of a component structure, i.e. a system's architecture design. The way in which a technical system participates in the technical process in order to fulfil its purpose by delivering effects necessary for operands transformation is not the focus of the analysed methods.

4. Formal modelling

This section offers a formal model of technical processes and technical process synthesis. To be able to describe the transformation process, a labelled directed multigraph is selected. This particular graph type is selected to provide sufficient semantic richness necessary for technical process modelling. Operands, effects, and operations are associated as labels to the graph's elements, i.e. operations to vertices, operands, and effects to directed edges. Multiple edges between nodes account for operand flows in order to express the operand transformation process.

The engineering knowledge about technical processes synthesis, including technological principles and the effects necessary for technical process realisation, are formalised within a graph grammar that enables a rule-based transformation mechanism to achieve synthesis of a technical process. Formalisation of part of the TTS, to show that engineering design synthesis is in the early phase of conceptualisation can be performed computationally, is the key contribution of this paper, both in design theory as well as in the field of CDS.

Rule-based transformation of graphs used in this work is understood as performing a local change to the graph's structure under the conditions given by a production rule $p : L \rightarrow R$ (König 2004, Ehrig *et al.* 2006, Kreowski *et al.* 2006). The basic procedure of the transformation algorithm, to decompose a sub-process into a chain of operations using a production rule p, is shown in Figure 2:

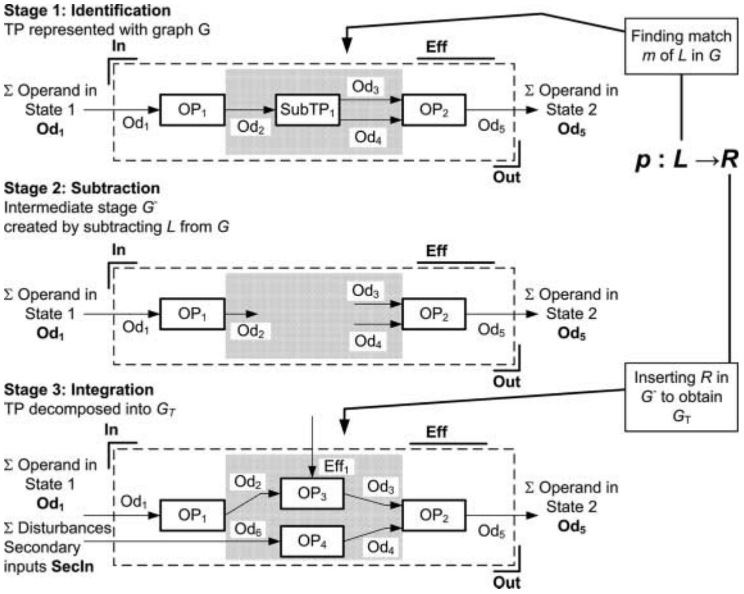

Figure 2. Example of transformation algorithm to perform a TP decomposition step under the production rule p.

The production p is defined by the terms of its left-hand side L, specifying the element of a technical process structure that will be replaced, i.e. process or sub-process. The right-hand side R, specifies the decomposed structure, i.e. a sub-graph consisting of sub-processes and operations that will be inserted and reconnected into the technical process (TP) structure (Figure 2). In addition to the production rule definitions that account for the formalised knowledge on technical processes, the transformation algorithm must involve the definition of matching procedures, because L from $p : L \rightarrow R$ must be identified in G and the specification of connecting procedures must be defined as well, to integrate R into the structure of G after L has been subtracted.

4.1. *Modelling of technical process*

Multigraphs are considered as non-simple graphs in which multiple edges between vertices, i.e. nodes, are allowed but no loops are permitted (Rosen *et al.* 2000, Weisstein 2009). In general, a multigraph G, can be defined as an ordered pair (V, E), where V is a set of nodes and E is a multiset of edges. To formally model technical processes, it is necessary to introduce related technical process entities, namely operands, effects, and operations into a graph's structure. Hence, operations are mapped to the graph's nodes and operands and effects are mapped to the arcs. The definition of a set of TP entities Σ_G is given as follows:

Definition 1 *A finite non-empty set of TP entities is defined as* $\Sigma_G = \Sigma_{Od} \cup \Sigma_{Eff} \cup \Sigma_{Op}$, *where* Σ_{Od} *denotes a set of operands* $Od \in \Sigma_{Od}$, Σ_{Eff} *is a set of effects* $Eff \in \Sigma_{Eff}$ *and* Σ_{Op} *is set of operations* $Op \in \Sigma_{Op}$.

Definition of TP entities provides meaning to the multigraph structure. Thus, the operands are the subject of transformations within the technical process. The TTS recognises three classes of operands: materials and object of biological origin, energy, and information. The transformation within the operations affects the properties of the operands, such as structure, form, or position in space and time. These operations are supported by the effects, i.e. any action or means that are

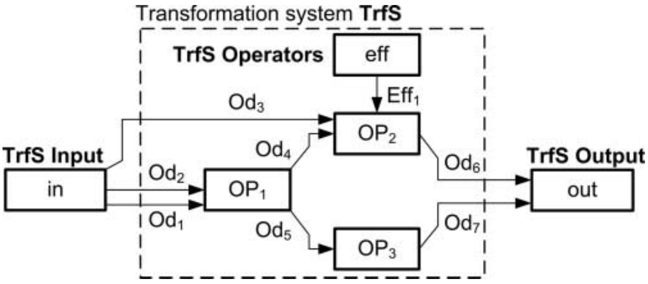

Figure 3. An example of TP modelled by G with TP entities.

required to sustain the operation, delivered from the technical system and human operator by that describing a human–machine interaction and clearly specifying the function of technical system. If Σ_G is understood as an alphabet, then a multigraph G can be defined over Σ_G as follows:

DEFINITION 2 *A multigraph G defined over alphabet* Σ_G *is represented as an ordered tuple* $G = (V, E, s, t, l_E, l_V)$ *where:*

- V *is a finite non-empty set of vertices,*
- $E \subseteq \{(u, v)|u, v \in V \wedge u \neq v\}$ *is a finite non-empty bag of directed edges e,*
- $s : E \to V$ *is a mapping, which for each edge e assigns a source vertex u,*
- $t : E \to V$ *is a mapping, which for each edge e assigns a target vertex v,*
- $l_E : E \to \Sigma_{Od} \cup \Sigma_{Eff}$ *maps for each edge e an operand or an effect,*
- $l_V : V \to \Sigma_{Op}$ *is a mapping, which for each vertex v assigns an operation.*

Figure 3 shows an arbitrary structure of technical process. Operands $Od_1, \ldots, Od_7$ can be understood as operands of different types (classes), or as operands of the same type but in varying states. Some of these operands may be the operands of the transformation that directly satisfy the existent users' needs and some may emerge as secondary, as required or generated by the transformation system. Operations OP_2 and OP_3 are performed in parallel, thus creating a sequence when coupled together with OP_1.

The vertices labelled in and out are added to a graph's structure to represent flows crossing the system border. The eff vertex accounts for the source of the effects (i.e. human operator and technical system) within the transformation system. According to the TTS, for all operations to be executed, the effects are a necessary condition. However, a formal model, as considered in our research, allows that the effects are not specified over each and every operation, which assumes that the designer does not know in advance all of the necessary effects required to sustain the transformation within the technical process, e.g. black-box abstraction of the problem. The interpretation is as follows: if an effect already exists (e.g. Eff_1 in Figure 3), then it must be obeyed when applying the production rules, otherwise eff can be left isolated. Thus, the following definition introduces additional constraints that have to account for the formal modelling of the technical processes:

DEFINITION 3 *A labelled directed multigraph* $G = (V, E, s, t, l_E, l_V)$ *is a formal model of technical process iff (if and only if) the following is satisfied:*

- $|V| \geq 4$ *with the following conditions for vertices:*
  - ○ $\exists_1 v \in V | l_V(V) = \text{in}$
  - ○ $\exists_1 v \in V | l_V(V) = \text{out}$
  - ○ $\exists_1 v \in V | l_V(V) = \text{eff}$

- *restrictions to edges*:
  - ○ $\nexists e \in E | l_V(t(e)) = \text{in}$
  - ○ $\nexists e \in E | l_V(s(e)) = \text{out}$
  - ○ $\nexists e \in E | l_V(t(e)) = \text{eff}$
- *G has to be well connected.*

To clarify Definition 3, the number of vertices being greater or equal to four implies the existence of at least one operation within the technical process, which exerts a transformation on the operands. Additionally, there must be exactly one of the in, out, and eff-labelled vertices. Restrictions to the edges define the roles of the in, out, and eff-labelled vertices, which are to supply the input operand flows (in), to accept the output flows (out) and to provide the effects (eff). Finally, the well-connectedness states that there are no isolated graph areas to perform operand transformations independently of the main transformation.

4.2. *Matching procedure*

In order to be able to perform a graph grammar transformation using productions $p : L \to R$, the mechanism, i.e. a match of $m(L)$ in G, has to be defined (Ehrig *et al.* 2006, Kreowski *et al.* 2006). The result of matching is the identification of a sub-process L inside a technical process G. Thus, if found in G, L is formally a sub-graph of G the definition of which is given as follows:

DEFINITION 4 *Let $\mathcal{G}_\Sigma$ be a finite set of all possible graphs that can be constructed over the alphabet of technical processes Σ_G, then a graph $C \in \mathcal{G}_\Sigma$ is called a sub-graph of $H \in \mathcal{G}_\Sigma$, if and only if the following conditions are satisfied*: $V_C \subseteq V_H, E_C \subseteq E_H, s_C(e) = s_H(e), t_C(e) = t_H(e), l_{E_C}(e) = l_{E_H}(e), l_{V_C}(u) = l_{V_H}(u)\ \forall e \in E_C \wedge u \in V_C$.

Definition 4 states that if a graph is also a sub-graph of another graph, then the former must match: its structure, nodes and directed edges, and its labelling, i.e. operations, effects, and operands. Building on the latter, it is possible to define a structure that preserves mappings over the $\mathcal{G}_\Sigma$ by providing means to structures match. Matching assumes respecting the contact conditions – the identity of TP entities, including both $l_E : E \to \Sigma_{Od} \cup \Sigma_{Eff}$ and $l_V : V \to \Sigma_{Op}$ through which operands, effects, and operations have been assigned to the directed edges and vertices. The following gives the definition addresses of the contact conditions of a structure-preserving mapping:

DEFINITION 5 *For graphs $C, H \in \mathcal{G}_\Sigma$, a mapping $m : C \to H$ is a pair of structure preserving mappings $m_V : V_C \to V_H$ and $m_E : E_C \to E_H$ such that the following holds*:

1) $\forall e \in E_C - (e | l_{E_C}(s_C(e)) \in \{\text{in}, \text{eff}\} \wedge l_{E_C}(t_C(e)) = \text{out}) \wedge u \in V_C - u | l_{V_C}(u) \in \{\text{in}, \text{out}, \text{eff}\}$:
   - ◦ $m_V(s_C(e)) = s_H(m_E(e)),\ m_V(t_C(e)) = t_H(m_E(e))$
   - ◦ $l_{E_H}(m_E(e)) = l_{E_C}(\text{e}),\ l_{V_H}(m_V(u)) = l_{V_H}(u)$
2) $\forall e \in E_C | l_{E_C}(s_C(e)) = \text{in}$:
   - ◦ $m_V(t_C(e)) = t_H(m_E(e)), \quad l_{E_H}(m_E(e)) = l_{E_C}(\text{e})$
3) $\forall e \in E_C | l_{E_C}(t_C(e)) = \text{out}$:
   - ◦ $m_V(s_C(e)) = s_H(m_E(e)), l_{E_H}(m_E(e)) = l_{E_C}(\text{e})$
4) *iff* $\exists e \in E_C | l_{E_C}(s_C(e)) = \text{eff}$:
   - ◦ $m_V(s_C(e)) = s_H(m_E(e)), l_{E_H}(m_E(e)) = l_{E_C}(\text{e})$

The definition of connection conditions for all operations and edges, which do not interact with in, out, and eff labelled vertices is given by (1). Directed edges that model the operand input to the transformation are addressed with (2). The source of input flows is not being taken into the consideration by condition (2) as it will be dependent on the connecting procedure as defined in Section 4.5. Similarly, (3) applies for the outgoing operand flows with their target left open to be determined by the connection procedure. Finally, the effects are addressed by condition (4) which applies only if the effect exists. Again, for the effects, the same assumptions hold as in Definition 3. If the effects are specified inside the production rule p, then the source and label are conserved, whilst the target depends only on the contents of the right-hand side of the production. Given Definitions 4 and 5, a match of L in G can be defined with the following:

Definition 6 *A match of L in G is found by the existence of $m : L \to G$ with $m(L) \subseteq G$, thus satisfying connection conditions in Definition 5.*

The match of L in G is defined to support an arbitrary number of operations. However, it is important to stress that the match applied for the production rule will always correspond to only one operation vertex that will be identified in G. The right-hand side of the rule, R, can have more than one Op. The graphical interpretation of matching $m(L)$ within the first stage of the transformation algorithm can be seen in Figure 2.

4.3. *Graph grammar of technical processes*

A rule p application results with technical process or sub-process decomposition. Driven by a set of production rules p, matching procedures m, and connecting procedure ρ (see Section 4.5), it is possible to traverse between any two points in the technical process decomposition synthesis. The following definition formally states this point:

Definition 7 *Using alphabet $\Delta \subseteq \Sigma_G$, as the graph is labelled over Σ_G and taking a finite non-empty set of production rules $p : L \to R$, then for every existing match $m : L \to G$ a direct derivation can be established as $G \overset{p,m,\rho}{\Longrightarrow} H$.*

The terminals symbols are determined as the symbols for which no more decomposition can be found in TP grammar. Thus, what can be stated is that the terminal graph structure will contain terminal TP entities $\Delta \subset \Sigma_G$ as the graph is labelled over Σ_G. Hence, terminal entities are operations from Σ_{Op}. All graphs composed of terminals are defined as $\mathcal{G}_\Delta \subset \mathcal{G}_\Sigma$.

Definitions 8 and 9 define the graph grammar of the technical processes and in that respect, a formal language of the technical processes as a set of all possible technical process variants defined in $\mathcal{G}_\Delta$:

Definition 8 *A graph grammar of technical processes $\mathcal{GG}$ is defined as an ordered triplet $\mathcal{GG} = (\mathcal{S}, \mathcal{P}_G, \Delta)$, with $\mathcal{S} \in \mathcal{G}_\Sigma$ as starting symbol, $\mathcal{P}_G$ as a finite non-empty set of productions $p \in \mathcal{P}_G$ of type $p : L \to R$ and alphabet $\Delta \subseteq \Sigma_G$ over which graph is labelled.*

Definition 9 *A language of technical processes $\mathcal{L}_{TP}$ generated by graph grammar $\mathcal{GG}$ is a set of graphs $G \in \mathcal{G}_\Delta$, which can be derived according to $\mathcal{GG} = (\mathcal{S}, p, \Delta)$ as $\mathcal{L}_{TP}(\mathcal{GG}) = \{G | G \in G_\Delta, \mathcal{S} \Rightarrow^*_{\mathcal{GG}} G\}$ (the asterisk denotes derivation sequence).*

4.4. *Context-free language of technical processes*

According to the literature (Kreowski *et al.* 2006), if a string is composed as a sequence of symbols $a_1a_2a_3 \ldots a_n, n \in \mathrm{Z}, |a| = 1$, with symbols being elements of some given alphabet $a_n \in \Sigma$, then it

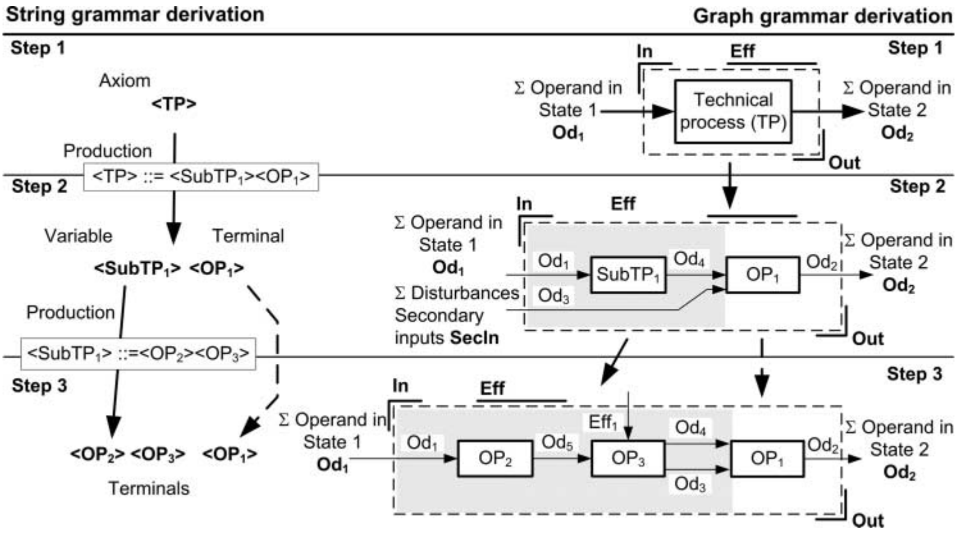

Figure 4. Example of the string grammar derivation process and corresponding graph grammar derivation (in, out, and eff vertices omitted).

is possible to construct a string graph consisting of $n + 1$ nodes and n edges. Similarly, the grammar of technical processes $\mathcal{GG}$ applied for technical process synthesis modelling could be related to string grammars as well. Furthermore, if each of the graph nodes with inbound and outgoing flows is mapped to a symbol, i.e. token, then a graph grammar $\mathcal{GG}$ can be represented as a string context-free grammar, written in the Backus–Naur form (BNF) (Naur 1963) providing the transformation algorithm and the connecting procedures. Mapping a token to a graph operation assumes the reuse of the node labelling in Σ_{Op}, which has already been applied in the multigraph G definition (see Definition 2). Each derivation step in the context-free grammar has to correspond to one decomposition step of the technical processes in the graph grammar. The motivation for introducing a string grammar can be justified as it eases the determination of the structure-preserving mapping, because it provides the layout followed by graph grammars when decomposing the technical process. The latter implies that $m(L)$ is easily resolved by referring to the corresponding $a_n \in \Sigma$. Furthermore, the introduction of string grammars enables a straightforward application of heuristic search algorithms, which operate by string building-block recombination, such as BNF and genetic algorithm-based grammatical evolution (O'Neill and Ryan 2001). An example of string grammar derivation and its corresponding graph grammar derivation is depicted in Figure 4:

The left-hand side of Figure 4 shows a derivation tree of the context-free string grammar (CFG). Decompositions of TP and SubTP$_1$ performed by CFG production rules are accompanied by graph grammar $\mathcal{GG}$ connecting together OP_2 and OP_3 with OP_1. As an example of connecting procedures, on the right-hand side of Figure 4 (derivation Step 2) it can be clearly seen how operand Od_3 enters the transformation from outside the system as a secondary input required to feed OP_1. After decomposing sub-process SubTP$_1$, operation OP_3 manages to supply the OP_1, thus eliminating the unnecessary secondary flow. However, to succeed in the transformation operation OP_3 requires a supporting effect provided by the human operator or technical system. The context-free grammar of technical processes CFG_{TP} and its language of $\mathcal{L}_{TP} = \mathcal{L}_{TP}(CFG_{TP})$ are defined as follows:

Definition 10 *A context-free grammar of technical processes* CFG_{TP} *expressed in BNF is a quadruple* $(\Sigma_s, \mathcal{V}_s, \mathcal{S}_s, \mathcal{P}_s)$ *where:* $\Sigma_s \subset \Sigma_{Op}$ *is a finite non-empty set of terminals belonging to operations,* $\mathcal{V}_s \subset \Sigma_{Op}$ *is a finite non-empty set of non-terminal symbols or variables satisfying*

$\Sigma_s \cap \mathcal{V}_s = \emptyset$, $\mathcal{S}_S$, is a starting symbol or axiom with $\mathcal{S}_s \in \mathcal{V}_s$, and $\mathcal{P}_s$ is a finite non-empty set of production rules of the type $p_s : a \rightarrow b$ where: $a \in \mathcal{V}_s$ and $b \in (\Sigma_s \cup \mathcal{V}_s)^*$ (here the asterisk denotes the set of all possible combinations).

Definition 11 *A formal language $\mathcal{L}_{TP} = \mathcal{L}_{TP}(CFG_{TP})$ generated by grammar $CFG_{TP} = (\Sigma_s, \mathcal{V}_s, \mathcal{S}_s, \mathcal{P}_s)$ is defined as $\mathcal{L}_{TP}(CFG_{TP}) = \{\omega | \omega \in \Sigma_s^*, \mathcal{S}_s \Rightarrow^*_{CFG_{TP}} \omega\}$.*

Finally, it is important to stress that the graph grammar inherits the orderings of operations Op as prescribed by the CFG_{TP} generated $a_1 a_2 a_i \cdots a_n$ string, thus making the connecting procedure ρ not completely invariant.

4.5. *Transformation algorithm and connection procedure ρ*

The transformation algorithm for the synthesis of the technical process represented over graph $G = (V, E, s, t, l_E, l_V)$ and under the set of production rules $p : L \rightarrow R$ and connecting procedure ρ, is given as follows:

- First, according to the $p : L \rightarrow R$, a match of process or sub-process L has to be identified in technical process G. The decomposition is always performed by replacing only a single process or sub-process with the structure in R consisting of an arbitrary number of operations and effects.
- Let $u_{Op} = m(L)$ such that $L \ni u_{Op} \in G$, $u_{Op} | l_V(u_{Op}) \in \Sigma_{Op}$ meaning that this particular single process or sub-process is present both in the left-hand side L of rule p and in the TP as G. An intermediate structure $G^- = G - u_{Op}$ is then created by subtracting TP of u_{Op}. The result is a number of dangling edges, either one of these edges is deprived of only one source or target but not of both at the same time. These flows $e_{G_i^-}$ are collected into interfaces set by satisfying the following: $E_{G_i^-} = \{(u, \text{nil}) \vee (\text{nil}, u) | u \in V_{G^-}\}$.
- Using the effect deleting function $DelEff(G^-, E_{G_i^-})$, all the effects $e_{G^-_{Eff}} \in E_{G_i^-}$ satisfying $l_{V_G}(s(e_{G^-_{Eff}})) = \text{eff}$ are deleted from G^- and $E_{G_i^-}$ as they will all be provided by R.
- As the right-hand side R of the production rule p also complies with Definition 2, R has to be deprived of all in and out vertices thus $R^- = R - V_{TrfS}$, with $V_{TrS} = \{u_R \in R | l_{V_R}(u_R) \in \{\text{in}, \text{out}\}\}$. The interfaces $e_{R_i^-} \in R^-$ resulting from the subtraction are collected as $E_{R_i^-} = \{(u, \text{nil}) \vee (\text{nil}, u) | u \in V_{R^-}\}$.
- The effects $e_{R^-_{Eff}} \in R^-$ are collected as well to be used for further matching as $E_{R^-_{eff}} = \{((u_{R^-} | l_{V_R^-}(u_{R^-}) = \mathit{eff}), v_{R^-}) | u_{R^-}, v_{R^-} \in V_{R^-}\}$.
- Furthermore, R^- is deprived of the eff labelled vertex thus producing $R^- \leftarrow R - V_{Eff}$, $V_{Eff} = \{u_{R^-} \in R^- | l_{V_R^-}(u_{R^-}) = \text{eff}\}$.
- To proceed with the transformation, R^-must be added to the G^- resulting in $G'_T \leftarrow G^- + R^-$.
- To integrate R^- in G'_T the expressions (1)–(3) comprising the connecting procedure ρ (see the following page) have to be performed to complete the transformation. Matching of the interfaces is regulated by the interface function $Inter(e_{G_i^-}, e_{R_i^-})$, which yields truth if both of the following are satisfied:

$$\begin{aligned} l_E(e_{G_i^-}) = l_E(e_{R_i^-}) \wedge s(e_{G_i^-}) = u \in V_{G^-} \wedge s(e_{R_i^-}) = \text{nil} \\ l_E(e_{G_i^-}) = l_E(e_{R_i^-}) \wedge t(e_{G_i^-}) = u \in V_{G^-} \wedge t(e_{R_i^-}) = \text{nil} \end{aligned} \tag{1}$$

If $Inter(e_{G_i^-}, e_{R_i^-}) = \text{true}$, then $e_{G_i^-}$ will take for source/target the operation v from R^- for which it holds the following $v \in R^- | s(e_{R_i^-}) = v \vee t(e_{R_i^-}) = v$. The interface reconnection procedure

Table 2. Transformation algorithm.

```
Input: $G, p_s : \alpha \to \beta, p : L \to R$
Output: $G_T$

1:   $u_{Op} \leftarrow m(L);$                        Identification of $L$ in $G$
2:   $G^- \leftarrow G - u_{Op};$                    Subtracting $L$ from $G$
3:   $E_{G_i^-} \xleftarrow{i} G^-;$                 Interfaces (flows) collected from $G^-$
4:   $DelEff(G^-, E_{G_i^-});$                      Effects deletion from $R$
5:   $R^- \leftarrow R - V_{TrfS}$                   in, out vertices deletion from $R$
6:   $E_{R_i^-} \xleftarrow{i} R^-;$                 Interfaces (flows) collected from $R^-$
7:   $E_{R^-_{eff}} \leftarrow R^-;$                 Effects collected from $R^-$
8:   $R^- \leftarrow R^- - V_{Eff};$                 eff vertex deletion from $R^-$
9:   $G'_T \leftarrow G^- + R^-;$                    Inserting $R^-$ into $G^-$
10 : compare each $e_{G_i^-}$ with each $e_{R_i^-}$ using $Inter\left(e_{G_i^-}, e_{R_i^-}\right)$ do
11 :     if $Inter\left(e_{G_i^-}, e_{R_i^-}\right)$ then
12 :         $Reconn\left(e_{G_i^-}, e_{R_i^-}\right);$
13 :         delete $e_{R_i^-}$ from $E_{R_i^-};$                              $\rho$
14 :     fi
15 : od
16 : $Copy(E_{R^-_{eff}}, G^-);$
17 : foreach $e_{R_i^-}$ reconnect secondary flows using $TrS(e_{R_i^-});$
```

$Reconn(e_{G_i^-}, e_{R_i^-})$ is defined as follows:

$$Reconn(e_{G_i^-}, e_{R_i^-}) = \begin{cases} \text{iff } s(e_{R_i^-}) = \text{nil}, t(e_{G_i^-}) \leftarrow t(e_{R_i^-}) \\ \text{iff } t(e_{R_i^-}) = \text{nil}, s(e_{G_i^-}) \leftarrow s(e_{R_i^-}) \end{cases} \tag{2}$$

The effects collected within $E_{R^-_{eff}}$ are copied to G^- using $Copy(E_{R^-_{eff}}, G^-)$ to reconnect these to eff source vertex in G^- with proper operation v from R^- that satisfies $t(e_{R_i^-}) = v$. Finally, the remainder in $E_{R_i^-}$ are secondary input/output flows and are reconnected to G^- as follows:

$$TrS(e_{R_i^-}) = \begin{cases} \text{iff } s(e_{R_i^-}) = \text{nil}, u \in G^- | l_V(u) = \text{in} \leftarrow s(e_{R_i^-}) \\ \text{iff } t(e_{R_i^-}) = \text{nil}, u \in G^- | l_V(u) = \text{out} \leftarrow s(e_{R_i^-}) \end{cases} \tag{3}$$

Transformation algorithm $G_T \xleftarrow{\rho} G^- + R^-$ is given in pseudo-code in Table 2 (connection procedure ρ is defined in lines 10–17):

5. Example

The purpose of the example is to show how technical process synthesis can be performed using the formalisms proposed in the previous sections. The design task being considered is the design of a stiffened panel assembly line. In order to perform the synthesis, the designer needs to gain knowledge about the different working principles on which the transformation of operands can be performed for this particular task as well as the necessary effects needed that should be provided in order to sustain the different transformations. Depending on the required effects for the different possible transformations (technology, working principles), the conceptual design of an assembly line involves solutions that may result in a multitude of different technical systems included as a part of the assembly line. Plates and stiffeners welded or riveted together are two working principles for joining structural parts assumed in this example. To show the difference in the technical system's function specification, as a consequence of different working principles being applied, the following example's grammar explains in more detail the decompositions of the panel welding and riveting, while the sub-processes of panel stiffening by welding and riveting are only considered as further exploration possibilities. The process of stiffened panel assembly is decomposed within the 10 sub-processes. The production rules that address those sub-process are given in Table 3 in CFG_{TP} grammar (Definition 10) (sub-processes as non-terminals are represented as tokens, while the operations as terminals begin with '_' symbols):

Further explanation of the production rule variants as expressed in Table 3 is as follows:

- *Stiffened panel assembled* $< spa >$ is the initial non-terminal. The process of stiffened panel assembly is decomposed within three logical steps: step one is the positioning of steel plates and their assembly into a steel panel $< assembled >$, step two comprises cutting the panel to the desired dimensions $< treated >$ and then, possible surface cleaning and setting of the markings for placement of stiffeners $< stiffened >$. It is assumed that steel plates and stiffeners enter the transformation in the state appropriate for the application of those two technologies, assuming appropriate welding joints and holes required for riveting.
- *Panel assembled* $< assembled >$ involves the positioning of steel plates $\_platePos$ as a preparation for the joining of the plates to the panel by welding $< plateWeld >$ or riveting $< plateRivet >$.

Table 3. CFG of stiffened panel assembly in BNF.

1	$< spa >$	$::=$	$< assembled >< treated >< stiffened >$	(0)
2	$< assembled >$	$::=$	$\_platePos < plateWeld >$	(0)
		$\mid$	$\_plate\ Pos < plateRivet >$	(1)
3	$< plateWeld >$	$::=$	$< plateSec >< plateWeld' > \ \_panelRelease$	(0)
		$\mid$	$< plateSec >< plateWeld' > \ \_panelTurn$ $< plateWeld' > \ \_panelRelease$	(1)
4	$< plateRivet >$	$::=$	$< plateSec >< plateRivert' > \ \_panelRelease$	(0)
5	$< plateWeld' >$	$::=$	$\_maw$	(0)
		$\mid$	$\_saw\_granRemoved$	(1)
6	$< plateRivet' >$	$::=$	$\_rivetPos' \ \_rivetSec' \ \_impactRiv'$	(0)
		$\mid$	$\_rivetPos' \ \_rivetSec'' \ \_impactRiv''$	(1)
		$\mid$	$\_rivetPos'' \ \_rivetSec'' \ \_impactRiv''$	(2)
7	$< plateSec >$	$::=$	$\_plateSec'$	(0)
		$\mid$	$\_plateSec''$	(1)
		$\mid$	$\_plateSec'''$	(2)
8	$< treated >$	$::=$	$\_panelCut < dirtRemoved > \ \_stf\ Pos$	(0)
9	$< dirtRemoved >$	$::=$	$\_blast\_abrSeparated'$	(0)
		$\mid$	$\_brush$	(1)
10	$< stiffened >$	$::=$	$\_panelPos \ \_stfWeld$	(0)
		$\mid$	$\_panelPos \ \_stfRivet$	(1)

- *Plates welded* $< plateWeld >$ assumes two process variants; the first (0) involves only one side welding and the other (1) assumes both sides welding that requires a panel being turned upside down. Also involved are the sub-processes of plate securing $< plateSec >$ and operations of panel turning *_panelTurn* and releasing *_panelRelease*.
- *Plates riveted* $< plateRivet >$ performs similarly as variant (0) of plate welding with respect to plates being secured and released, with distinction in the actual panel join performed by riveting $< plateRivet' >$.
- *Plates welded (one side)* $< plateWeld' >$ allows two distinct approaches: one is a manual arc welding operation, i.e. variant (0) involving a human operator and the other is a fully automated, submerged arc welding under a protective coating, which needs to be removed after the procedure (1).
- *Plates impact riveted* $< plateRivet' >$ assmues that all three variants comprise operations involving rivets being positioned, secured and then impact riveted. The difference between them is the level of automation being applied. Namely, the operations *rivetPos′_rivetSec′_impactRiv′* in variant (0) are performed manually using appropriate tools, while variant (2) is completely automated *_rivetPos″_rivetSec″_impactRiv″* (see Table 4 for details on operations). Although the transformation results in the same main operand transformations, the secondary flows defer, as well as the necessary effects.
- *Plates secured* $< plateSec >$ involves three different principles for securing plates required for panel joining. The first variant (0) applies a pneumatic principle to create suction to grip plates, (1) utilises electromagnetic force, whilst the last, (2) is hydraulic based to grip plates.
- *Panel treated* $< treated >$ considers operation of panel cutting to fit the exact measurements *_panelCut*, a sub-process of cutting produced dirt removal $< dirtRemoved >$ and operation *_stfPos* of locating and marking of positions to which stiffeners will be attached.
- *Dirt removed* $< dirtRemoved >$ can be executed in two ways, manually using a brush*_brush*, or fully automatically by blasting the panel *_blast* with abrasive particles that are later collected and stored*_abrSeparated′*.
- *Panel stiffened* $< stiffened >$ defines either welding *_stfWeld* or riveting *_stfRivet* to attach stiffeners to the panel, together with a pre-operation of panel positioning *_panelPos* to relocate the steel panel.

Table 4 shows building-blocks that are used to compose the graph grammar $\mathcal{GG}$ of the technical processes based on the knowledge represented in Table 3. A detailed specification of the input and output operand flows, as well as of the required effects for each sub-process/operation, is presented.

When formalising knowledge, the proposed method first requires a definition of TP entities Σ_G (Definition 1). Then, these are used to label directed multigraph G (Definition 2) to create a set of building-blocks. The building-blocks are used to define the graph grammar in compliance with Definition 8. Both sides of the graph grammar productions have to be in compliance with Definition 3 and they must all satisfy the connecting procedures ρ from Section 4.5. For example, a black-box formulation of stiffened panel assembly $< spa >$, thus specifying the left-hand side of the graph grammar production rule p with operands in their initial and desired state, is given in Figure 5. The operands in their required input states are both plates and a stiffener and the desired output is a stiffened panel. Effects are rendered as unknowns at the beginning. The corresponding decomposition of the stiffened panel assembly $< spa >$ into $< assembled >< treated >< stiffened >$ is according to the string grammar in Table 3 and building blocks in Table 4. Figure 6 presents two possible ways of the $< assembled >$ sub-process decomposition according to Tables 3 and 4 and the transformation algorithm in Section 4.5.

Table 4. Graph grammar building-blocks.

Process/Operation	Input *Od* flows	Output *Od* flows	Effects *Eff*
Stiffened panel assembled ⟨*spa*⟩	Plate Plate	Stiffened panel	
Panel assembled ⟨*assembled*⟩	Plate Plate	Panel	
Panel treated ⟨*treated*⟩	Panel	Panel (treated)	
Panel stiffened ⟨*stiffened*⟩	Panel (treated) Stiffener	Stiffened panel	
Plates positioned _*platePos*	Plate Plate	Plate (positioned) Plate (positioned)	
Plates welded ⟨*plateWeld*⟩	Plate (positioned) Plate (positioned)	Panel	
Plates riveted ⟨*plateRivet*⟩	Plate (positioned) Plate (positioned)	Panel	
Plates secured ⟨*plateSec*⟩	Plate (positioned) Plate (positioned)	Plate (secured) Plate (secured)	
Plates welded (1 side) ⟨*plateWeld'*⟩	Plate (secured) Plate (secured)	Panel (secured)	
Panel release _*panelRelease*	Panel (secured)	Panel	
Panel turned _*panelTurn*	Panel (secured)	Plate (secured) Plate (secured)	Energy (mechanical)
Plates riveted (both sides) ⟨*plateRivet'*⟩	Plate (secured) Plate (secured)	Panel (riveted)	
Manual arc welded _*maw*	Plate (secured) Plate (secured) Electrode (coated)	Panel (secured) Fumes Light	Energy (arc) Human force Regulation
Submerged are welded _*saw*	Plate (secured) Plate (secured) Wire Ceramic slab Granulate (flux)	Panel (secured) and granulate Ceramic slab	Energy (arc) Regulation
Panel granulate separated _*gran Removed*	Panel (secured) and granulate	Panel (secured) Granulate (free)	Energy (pneumatic)
Rivet positioned _*rivetPos*'	Plate (secured) Plate (secured) Gripping tool Rivet (hot)	Plate (secured) Plate (secured) Gripping tool Rivet(positioned)	Human force
Rivet secured _*rivetSec*'	Plate (secured) Plate (secured) Riveting support Rivet (positioned)	Plate (secured) Plate (secured) Rivet (secured) Riveting support	Human force
Plate riveted _*impactRiv*'	Plate (secured) Plate (secured) Riveting support Rivet (secured) Pneumatic hammer	Panel (secured) Riveting support Pneumatic hammer	Human force Regulation
Rivet positioned _*rivetPos*	Plate (secured) Plate (secured) Rivet (hot)	Plate (secured) Plate (secured) Rivci (positioned)	Energy (mechanical)
Rivet secured _*rivetSec*"	Plate (secured) Plate(secured) Rivet (positioned)	Plate (secured) Plate (secured) Rivet (secured)	Energy (mechanical)

Plate riveted _*impact Riv*"	Plate (secured)
	Plate(secured)
	Rivet(secured)
		Panel (secured)	Energy (mechanical)
			Regulation

Plate secured _*plateSec*'	Plate (positioned) Plate (positioned)	Plate (secured) Plate (secured)	Energy (pneumatic)
Plate secured _*plateSec*"	Plate (positioned) Plate (positioned)	Plate (secured) Plate (secured)	Energy (electromag.)
Plate secured _*plateSec*"'	Plate (positioned) Plate (positioned)	Plate (secured) Plate (secured)	Energy (hydraulic)
Panel cutting _*panelCut*	Panel	Panel	Energy (heat)

(*Continued*)

Table 4. Continued

Process/Operation	Input *Od* flows	Output *Od* flows	Effects *Eff*
Dirt removed ⟨*dirlRemoved*⟩	Panel	Panel Particles (waste)	
Stiffener positioning _*stfPos*	Panel Marker	Panel (marked) Marker	Regulation
Panel brushed _*brush*	Panel Brush	Panel Brush Particles (waste)	Human force
Panel blasted _*blast*	Panel Abrasive particles	Panel and abrasive particles	
Panel and particles separated _*abrSeparated*’	Panel and abrasive particles	Panel Abrasive particles Abrasive particles (waste)	Energy (pneumatic)
Panel positioning _*panelPos*	Panel (treated)	Panel (positioned)	Energy (mechanical) Regulation
Stiffener welded _*stfWeld*	Panel (positioned) Stiffener	Stiffened panel	Energy (arc) Regulation
Stiffener riveted _*stfRivet*	Panel (positioned) Stiffener	Stiffened panel	Energy (mechanical) Regulation

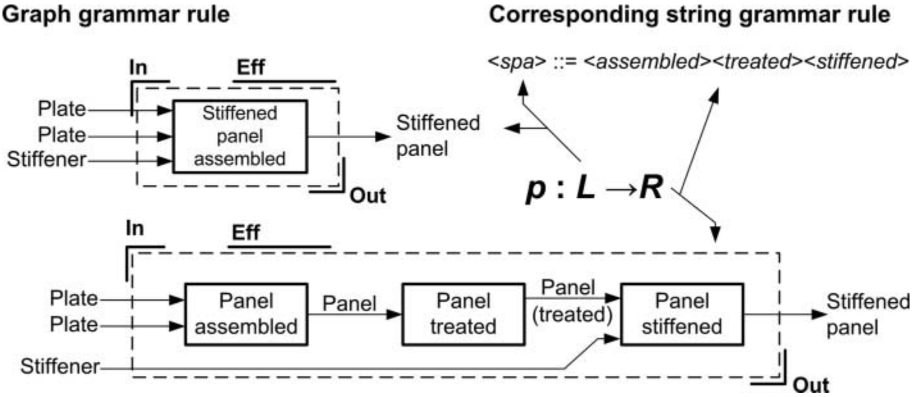

Figure 5. Black-box process formulation of stiffened panel assembly and its corresponding decomposition (in, out, and eff omitted).

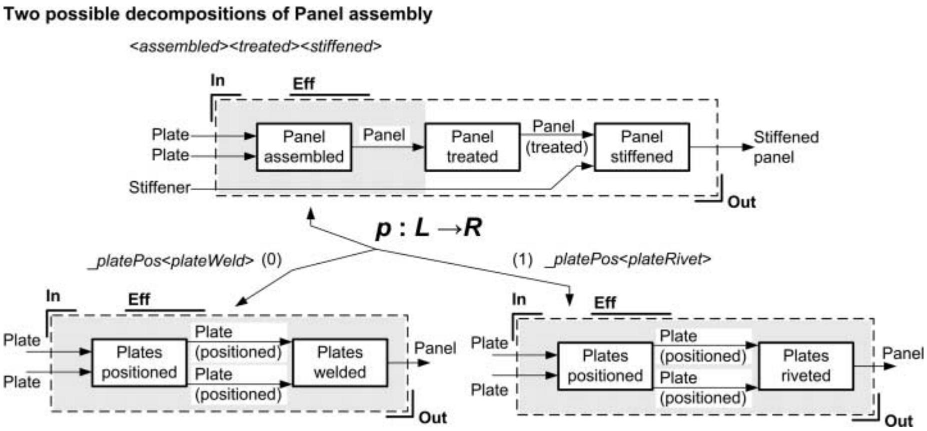

Figure 6. Decomposition of stiffened panel assembly (in, out, and eff omitted).

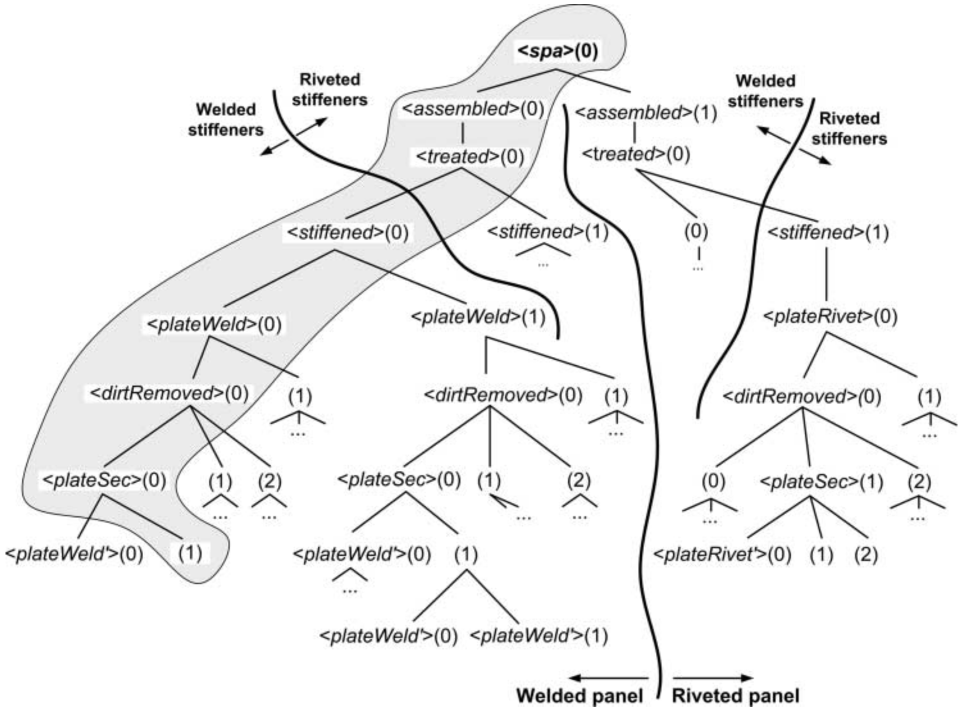

Figure 7. Production derivation tree (left-hand sides of production shown).

6. Results and discussion

A derivation tree showing all theoretically possible production application sequences for the stiffened panel assembly line based on Tables 3 and 4 is depicted in Figure 7. Only the left-hand side of the BNF production is shown, as well as the corresponding rule alternative, which was applied for the decomposition (Table 3). For convenience and in order to make the figure more comprehensive, only the rule-variant ordinals were given in the areas where the space did not permit the full corresponding left-hand side. The number of technical process variants that can be created using this simple grammar is almost 100. As an example, the variant with the minimal number of operations involving a fully automated process is presented in Figure 7 as a shaded area. Four areas divide the derivation tree clearly showing the solution variants, which assume panel welding or panel riveting and the welding or riveting of stiffeners, altogether creating four principle solution groups.

Based on the variant production trees presented in Figure 7 and grammar, as defined in the previous section, an example of how a variation at the technical process level may yield different technical systems is shown in Figure 8. Because of the number of operations involved, only excerpts of two technical process variants are depicted; one with fully automated panel riveting and the other involving fully automated panel welding sub-process. Plate riveting and welding sub-processes are shown in detail, while the rest of the sub-processes are not decomposed to preserve space (Figure 8).

The technical system for riveting must be capable of providing the impact force, thus specifying one of the system's functions. In the other technical process, which corresponds to shaded area in Figure 7, the technical system for welding must be capable of providing an electrical arc able to perform unification of two plates into a panel. These two functions, provision of impact force for riveting and arc for welding, are direct consequences of the different technological principles on

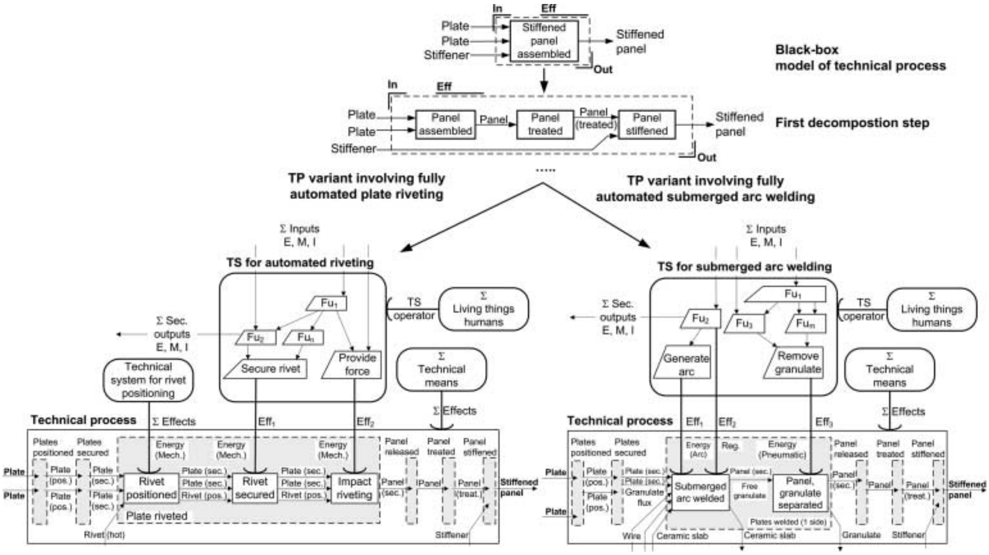

Figure 8. How a variation on the technical process level may yield different technical systems.

which the operand transformation variants were founded. The same reasoning holds for securing of rivets and removal of granulate. Moreover, the necessary input flows of technical systems, such as rivets or welding wire for instance, are also the result of different technical process that need to be supported (it is assumed that inputs and secondary outputs of technical systems are not the same).

The example in Section 5 also exemplified the process of the formation of production rules by which the engineering knowledge is formalised. However, in order to define production rules, the definitions about terms used must be clarified to the users, in order to be able to produce coherent results in the end. Thus, a prerequisite for a successful knowledge-based system and knowledge generalisation is at least having a shared understanding about domains of concern (Gruber 1993), i.e. taxonomy of related terms thus creating possibilities to organise knowledge more efficiently. Such generalisation will enable only what is necessary to describe each of the objects to be put forward, thus eliminating irrelevant details and maintaining the production rule redundancy. The illustrative example shown in Section 5 is label bounded, having no knowledge generalisation possibilities within the bounds of the context-free grammar. Effects and operations attached to the multigraph's edges are technical process labels and not objects of their respective classes. The extension to include attributes, types, and inheritance and to even define operations would require definition of type graph and typed graph, thus introducing semantics and creating a robust system. In that respect, the definitions within Section 4 hold, assuming that additional definitions on flows, effects, and operations are equal to enable comparison. Thus, the definition of type graphs will enable the more efficient utilisation of stochastic search, because the rules would not have to be defined so strictly and label bounded but would instead be tied to types of objects, similar to that presented in Helms and Shea (2012). It can only be suggested to define a type graph of technical processes according to some of the available engineering design formalisms, such as Merged Ontology for Engineering Design (Ahmed and Štorga 2009) or Design Ontology (Štorga *et al.* 2010).

7. Conclusions and future work

The aim of this work was to provide the formal modelling of technical processes and technical process synthesis. As such, this work contributes to the research area of CDS, as it may serve as a foundation for creation of a computational-based support to aid designers in the technical process synthesis design phase. As a result of this work, the following points can be highlighted:

- For the modelling of technical processes, a labelled directed multigraph with operations, operands, and effects is defined (Definitions 2 and 3). Directed multigraphs offer richness in description as demanded for technical process modelling. Operations, operands, and effects are process-related entities defined as TP entities (Definition 1) and they constitute the graph's vocabulary. A graph of technical processes, as a carrier structure, is invariant to the operations, effects, and operands, i.e. objects that have been assigned, thus creating possibilities for further development. As the method for synthesis is defined over the same multigraph type, it is also invariant to the type of objects being assigned to the graph's nodes and directed edges. At the current stage of development only the association of labels to the technical processes structure is performed.
- It was shown that by applying graph grammar transformations using a set of predefined rules and by following a breadth-first node rewriting principle, it is possible to support computational synthesis of technical processes. In order to achieve embedding of each of the decompositions into the technical processes' structure, a transformation algorithm with special connection procedure ρ is defined. The knowledge about technical processes synthesis based on technological principles is formalised within a set of production rules. The synthesis process is modelled within a formal language of technical processes (Definitions 10 and 11) using BNF as a layout for synthesis.
- By writing down the formal language of technical processes in BNF notation, it was shown that it is possible to directly apply heuristic search algorithms, which operate over string encoded solution variants. For instance, the algorithm of grammatical evolution (O'Neill and Ryan 2001) that operates on a genetic algorithm using BNF notation is a reasonable candidate as the search algorithm. As at the current stage of development, TP entities do not permit more attributes as these would require technical process knowledge generalisation and systematisation, it is not possible to construct useful metrics beyond the trivial. Thus, to describe the universal virtues (Hansen and Andreasen 2002) of technical process, it is necessary to be able to perform search and optimisation.
- As the proposed method for the generation of operand transformation variants is knowledge-driven, it was necessary to explore the requirements that need to be met in order to formalise the knowledge about the technical processes within a set of production rules. It is important to stress that the aims of this work did not include research about the content of knowledge about technical processes in respect to its systematisation and generalisation. It was intended to provide a means to formalise that knowledge within a set of production rules and to utilise these productions by the developed method in order to generate operand transformation variants.

Further research efforts might be directed towards extending the technical process model to include type graphs of TP entities. By doing so, a cumbersome process of production rules definition would become easier as the rules become more generalised with respect to TP entities. For such generic rule building-blocks, a more expressive context-sensitive grammar could be applied to formulate productions. The latter could enable a reuse of productions with respect to classes of TP entities and consequently reduce the number of rules.

References

Ahmed, S. and Štorga, M., 2009. Merged ontology for engineering design: contrasting an empirical and a theoretical approach to develop engineering ontology. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 23 (4), 391–407.

Andreasen, M.M. and Hein, L., 1987. *Integrated product development*. Berlin: IFS-Springer.

Asimow, M., 1962. *Introduction to design*. Engelwood Cliffs, NJ: Prentice Hall.

Bertalanffy, L., 1969. *General system theory – foundations development, applications*. New York: George Braziller.

Bolognini, F., Seshia, A.A., and Shea, K., 2007. Exploring the application of a multi-domain simulation-based computational synthesis method in MEMS designs. *In*: *Proceedings of the 16th international conference on engineering design (ICED 07)*. Glasgow: Ecole Centrale Paris & The Design Society, 81–82.

Bryant, C.R., *et al*., 2005. Concept generation from the functional basis of design. *In*: *Proceedings of the 15th international conference on engineering design (ICED 05)*. Glasgow: The Design Society, 280–281.

Cagan, J., *et al*., 2005. A framework for computational design synthesis: model and applications. *Journal of Computing and Information Science*, 5 (3), 171–181.

Campbell, M., *et al*., 1998. A-design: theory and implementation of an adaptive, agent based method of conceptual design. *In:* J. Gero and F. Sudweeks, eds. *Artificial Intelligence in Design '98*. Dordrecht: Kluwer Academic Publishers, 579–598.

Chomsky, A.N., 1957. *Syntactic structures*. The Hague: Mouton.

Ehrig, H., *et al*., 2006. *Fundamentals of algebraic graph transformation*. Berlin: Springer-Verlag.

Erden, M.S., *et al*., 2008. A review of function modelling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22 (2), 147–169.

Gharajedagi, J., 2011. *Systems thinking: managing chaos and complexity*. Burlingoton: Morgan Kaufmann.

Gips, J. and Stiny, G., 1980. Production systems and grammars: a uniform characterization. *Environment and Planning B*, 7 (4), 399–408.

Goldstein, I. and Papert, S., 1977. Artificial intelligence, language, and the study of knowledge. *Cognitive Science*, 1 (1), 84–123.

Gruber, T., 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5 (2), 199–220.

Hansen, C.T. and Andreasen, M.M., 2002. Two approaches to synthesis based on the domain theory. *In*: A. Chakrabardi, ed. *Engineering design synthesis – understanding, approaches and tools*. London: Springer-Verlag, 93–108.

Helms, B. and Shea, K., 2010. Object-oriented concepts for computational design synthesis. *In*: D. Marjanović, *et al*., eds. *Proceedings of the 11th international design conference DESIGN 2010*. Glasgow: Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb and The Design Society, 1333–1342.

Helms, B. and Shea, K., 2012. Computational synthesis of product architectures based on object-oriented graph grammars. *Journal of Mechanical Design*, 134 (2), 021008-1, 021008-14.

Hubka, V. and Eder, W.E., 1992. *Engineering design: general procedural model of engineering design*. Berlin: Springer-Verlag.

Hubka, V. and Eder, W.E., 2002. Theory of technical systems and engineering design synthesis. *In*: A. Chakrabardi, ed. *Engineering design synthesis – understanding, approaches and tools*. London: Springer-Verlag, 49–65.

Hutcheson, R.S., *et al.*, 2006. Application of a genetic algorithm to concept variant selection. *In: ASME conference proceedings (IDETC/CIE2006)*, Philadelphia, PA, 43–52.

Jin, Y. and Li, W., 2007. Design concept generation: a hierarchical coevolutionary approach. *Journal of Mechanical Design*, 129 (10), 1012–1022.

König, B., 2004. *Analysis and verification of systems, with dynamically evolving structure*. Habilitationsschrift zur Erlangung der Venia Legendi in Informatik, Institut für Formale Methoden der Informatik, Universität Stuttgart.

Kreowski, H., Klempien-Hinrichs, R., and Kuske, S., 2006. Some essentials of graph transformations. *In*: Z. Esik *et al*., eds. *Recent advances in formal languages and applications*. Berlin: Springer-Verlag, 229–254.

Levet, W.J.M., 2008. *An introduction to the theory of formal languages and automata*. Amsterdam: John Benjamins.

Lin, Y., *et al*., 2009. A method and software tool for automated gearbox synthesis. *In*: *ASME conference proceedings (IDETC/CIE 2009)*. San Diego, CA.

McCormack, J.P. and Cagan, J., 2004. Speaking the Buick language: capturing, understanding, and exploring brand identity with shape grammars. *Design Studies*, 25 (1), 1–29.

Naur, P., 1963. Revised report on the algorithmic language ALGOL 60. *Communication ACM*, 6 (1), 1–17.

O'Neill, M. and Ryan, C., 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5 (4), 349–358.

Post, E., 1943. Formal reductions of the general combinatorial decision problems. *American Journal of Mathematics*, 65, 197–215.

Pugliese, M.J. and Cagan, J., 2002. Capturing a rebel: modeling the Harley–Davidson brand through a motorcycle shape grammar. *Research in Engineering Design*, 13 (3), 139–156.

Rihtaršić, J., Žavbi, R., and Duhovnik, J., 2010. Sophy – tool for structural synthesis of conceptual technical systems. *In*: D. Marjanović *et al*., eds. *Proceedings of the 11th international design conference DESIGN 2010*. Glasgow: Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb and The Design Society, 1391–1398.

Rosen, K.H., *et al*., 2000. *Handbook of discrete and combinatorial mathematics*. Boca Raton: CRC Press LLC.

Schmidt, L.C. and Cagan, J., 1997. GGREADA: a graph grammar-based machine design algorithm. *Research in Engineering Design*, 9 (4), 195–213.

Schmidt, L.C., Shetty, H., and Chase, S.C., 2000. A graph grammar approach for structure synthesis of mechanisms. *ASME Journal of Mechanical Design*, 122 (4), 371–376.

Shea, K., 1997. *Essays of discrete structures: purposeful design of grammatical structures by directed stochastic search*. Thesis (PhD). Carnegie Mellon University.

Shea, K. and Cagan, J., 1998. Generating structural essays from languages of discrete structures. *In:* J. Gero and F. Sudweeks, eds. *Artificial Intelligence in Design '98*. Dordrecht: Kluwer Academic Publishers, 365–384.

Siddique, Z. and Rosen, D.W., 1999. Product platform design: a graph grammar approach. *In*: *ASME conference proceedings (DETC 1999)*. Las Vegas, NV.

Sim, S.K. and Duffy, A.H.B., 2003. Towards an ontology of generic engineering design activities. *Research in Engineering Design*, 14 (4), 200–223.

Simon, H.A., 1996. *The sciences of the artificial*. Cambridge, MA: The MIT Press.

Starling, A.C. and Shea, K., 2005. A parallel grammar for simulation driven mechanical design synthesis. *In*: *ASME conference proceedings (IDETC/CIE 2005)*, Long Beach CA.

Stone, R.B. and Wood K.L., 2000. Development of a functional basis for design. *Journal of Mechanical Design*, 122 (4), 359–370.

Štorga, M., Andreasen, M.M., and Marjanović, D., 2010. The design ontology – contribution to the design knowledge exchange and management. *Journal of Engineering Design*, 21 (4), 427–454.

Umeda, Y., *et al.*, 1996. Supporting conceptual design based on the function-behaviour-state modeller. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, 10 (4), 275–288.

Weber, C., 2005. CPM/PDD – an extended theoretical approach to modelling products and product development processes. *In*: *Proceedings of the 2nd German–Israeli symposium on advances in methods and systems for development of products and processes*. Stuttgart: Fraunhofer-IRB-Verlag, 159–179.

Weisstein, E.W., 2009. *The CRC encyclopedia of mathematics*. Boca Raton, FL: CRC Press/Taylor & Francis.

Wu, Z., Campbell, M.I., and Fernandez, B.R., 2008. Bond graph based automated modelling for computer-aided design of dynamic systems. *ASME Journal of Mechanical Design*, 130 (4), 041102–041111.

Wyatt, D.F., Wynn, D.C., and Clarkson, P.J., 2009. A computational method to support product architecture design. *In*: *ASME conference proceedings (ASME-IMECE 2009)*. Lake Buena Vista, FL.

Wyatt, D.F., *et al.*, 2012. Supporting product architecture design using computational design synthesis with network structure constraints. *Research in Engineering Design*, 23 (1), 17–52.