A NEW METAMODEL TO REPRESENT TOPOLOGIC KNOWLEDGE IN ARTIFACTUAL DESIGN

J. M. Jauregui-Becker[1], K.J.W. Gebauer [1], F. J. A. M. van Houten[1]
(1) University of Twente

ABSTRACT
This paper presents the Topology Abstraction Representation Diagram (TARD) as a new metamodel to represent knowledge of components and their connectedness in artifactual design. TARD has been developed with the aim of supporting the Computational Design Synthesis (CDS) process of configuration design tasks. TARD consists of three base building blocks: *elements*, *c-relations* and *h-relations*. Elements represent components of an artifact by grouping parameters. C-relations represent the connectedness of the elements in the topology. H-relations model how a group of c-relations describes the composition of one level of abstraction and its relation with other levels of abstraction. The main improvement of TARD in relation to other representation schemes for CDS (i.e. grammars, multi-level networks, evolutionary approaches, and pure parametric methods like PaRC) is its capability of handling top-down and bottom-up synthesis processes while keeping the topologic consistency of the design solutions as they are being generated. This is achieved by: (1) decoupling levels of abstractions by using the explicit hierarchical h-relations, (2) decoupling the connectedness among elements by using the explicit c-relations and (3) defining the cardinality of both the elements and the c-relations as a parameter that keeps track and allows steering the number of instantiated building blocks.

Keywords: Computational Synthesis, topology, configuration design

1 INTRODUCTION
Since the emergence of computers, design researchers and practitioners have developed software to support engineering design tasks. These tools have gradually transformed the design process, leading to the progressive substitution of paper based techniques by Digital Product Development (DPD) approaches. Nowadays, DPD is mainly supported Computer Aided Design (CAD), Computer Aided Engineering (CAE), Computer Aided Manufacturing (CAM) and Product Data Management (PDM) software, aiding engineers in the design of complex products. However, advances in technology and competitive markets are driving modern products towards further miniaturization, better quality, more functionality and yet lower prices [1]; imposing a great challenge on product development. This has motivated the development of computer systems that support the design synthesis process as well [2] (synthesis defined as the process of transforming design requirements into candidate design solutions). The result is the emergence of Computational Design Synthesis (CDS) as a field in engineering science that researches methods to automate the generation of designs. A well accepted model of the general processes a CDS method should include is shown in Figure 1 [3]. Furthermore, a complete overview on design automation methods and techniques can be found in Chakrabarti [4] and in Antonsson [5].

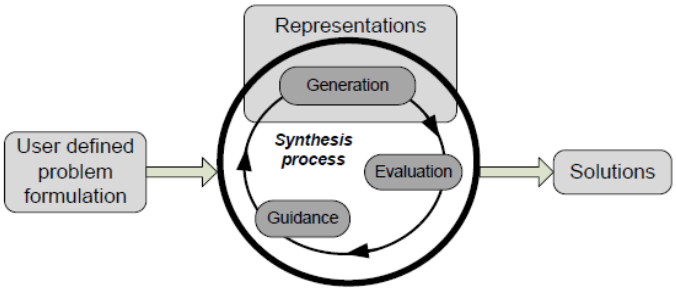


Figure 1. Computational design synthesis diagram [3]

As it is indicated in Figure 1, the first step in setting a CDS method up is formulating the design problem. For technical problems this is done by declaring variables, relations, constraints and objective functions, as it is described in [6]. This information is then translated into a representation, or building blocks, that can be used by algorithms to generate candidate solutions. A candidate solution satisfies all constraints of the problem independent of how well the goal is achieved. An evaluation step analyses the solutions by calculating its performance and decides whether it is accepted, adjusted, or rejected. Guidance drives the generation process in a given direction to improve the generation of candidate solutions.

CDS methods range from low level building blocks manipulation (as for example in [7-10]) up to high level conceptual reasoning (as for example in [11-13]). From the point of view of low level building blocks manipulations, the synthesis process involves mainly two types of activities: generating networks of components aiming at meeting functional requirements and instantiating constrained parameters to assure the design requirements and performances are met. However, when dealing with complex and largely annealed topologies, the generation process can become unmanageable unless a proper understanding of the problem structure is determined first. The question here is how to efficiently compose building blocks into large complexes, as pointed out by Antonsson et al in [5]. It is here stated that "regular structures, hierarchical structures, symmetry, duplication, self-similarity, and minimization of information content are important aspects of making large complexes constructible, manageable and the construction process scalable". Therefore, the challenge is not only one of defining the right generation algorithms, but also one of determining appropriate representation frameworks capable of supporting the different mechanisms involved in the automatic generation of design solutions. This paper works on this challenge by presenting a metamodel to represent topologic knowledge for configuration design. The metamodel is named Topology Abstraction Representation Diagram (TARD), as the focus is set on the topology characteristics of design problems. As this paper concerns CDS, software implementation aspects have been taken into account in the development of this new model. This paper is organized in 5 sections. Section 2 describes how four well known representational schemes have been used in the development of TARD. Section 3 presents TARD and describes the characteristics of its building blocks. Section 4 presents an example that demonstrates how to use TARD. Section 5 makes a discussion about the advantages of using TARD as representation model and elaborates on future research issues.

2 CONTRIBUTIONS AND COMPARISON WITH OTHER FRAMEWORKS

TARD has been developed by combining concepts coming from four different representational schemes, namely, grammars, multi-level networks, evolutionary approaches, and PaRC. The following subsections present a small overview of these techniques as well as a description about the fashion in which they have contributed to the development of TARD.

2.1 Grammars

Using grammars for CDS consists in translating knowledge about a design problem (e.g. from experienced designers) into a set of transformation rules describing how an initial incomplete design can be transformed into a complete one. Here, algorithms generate solutions by applying the rules to an initial design. Grammars are used both as representations (e.g. [14],[10]) and as generation systems (e.g. [15], [16], [17], [13]). As generation systems, grammars have been typically used in architecture (e.g. [18]) and in mechanical design (e.g. [17]). A grammar is defined by a 4-tuple $G = (V, X, R, S)$, where V is the set of objects that are manipulated by the grammar, X is a set of terminal and non-terminal symbols, S is the initial symbol and R is a set of rules of the form outlined above. The language of the grammar G is the set of all results produced from the start symbol that consists of only terminal symbols.

There are three possible tasks for programs that implement grammars [19]. The most common task, and perhaps the first that comes to mind, is to aid in the generation of designs at the hand of a known grammar. Here, grammar rules are combined using special algorithms to generate a new design. A second type of program is a parsing program. A parsing program is given a grammar and a design, and the program determines if the design is in the language generated by the grammar and, if so, gives the sequence of rules that produces the design. A third type of program is an inference program. The grammatical inference problem is: given a set of designs, construct a grammar that generates the design solutions.

In TARD, grammar rules (connectedness rules) are incorporated by means of the c-relations (explained in subsection 3.3). There are two main differences between grammar rules and c-relations. The first is that c-relations are independent from the objects (or nodes) they are connecting. By doing so, TARD can support the tree types of grammar operations aforementioned. The second is that c-rules make use of a parameter termed as cardinality to either keep track or leave open the number of elements it may connect.

2.2 Multi-level Networks

Nested topologies require an approach for separating different levels of design detail. Johnson et al. [20] does so by presenting a multilevel network for relating elements in a topology. This network distinguishes parts and its components at different levels of abstraction. Components at level N are related by an assembly relation R to one component at level $N+1$. By doing so, the component on the higher level $N+1$ can be represented as function of the components on the lower level N and the assembly relation *R*. As a consequence, the components in one level are not directly related with each other.

TARD borrows from multi-level networks the idea of having explicit relations decoupling levels of abstraction. This is done by using the so called h-relations (further explained in subsection 3.4). This property allows algorithms to generate partial design solutions focused on specific levels of abstractions. This also enables the application of bottom-up design strategies, as lower abstraction levels can be first created, and their results used later to decide upon the configuration of higher levels. The main difference between TARD and this approach is that the assembly relation in multi-level networks group components without characterizing their internal composition, while the h-relation groups c-relations.

2.2 Evolutionary Approaches

Evolutionary approaches are based on the principles of biological evolution, and encompass a broad range of search techniques, among which the most prevalent are genetic algorithms (e.g. [21]), evolutionary programming (e.g. [22]) and evolution strategies (e.g. [23]). A characteristic of these methods is that they employ sets of solutions (populations) represented by vectors. Any particular solution is written as a string of objects called chromosomes, and each component is regarded as a gene. Some of the main differences between these techniques lie in their representation schemes. In Genetic Algorithms (GA), the representation is done in the form of string of numbers, which are often binary. Genetic programming, on the other hand, uses lisp-like trees as representation of the solutions, resulting in computer program like solutions. In Evolutionary Programming (EP) the structure of the program is fixed, being the parameters the ones allowed to change. Evolutionary programming uses vectors of real numbers for the representation of solutions.

As it will be explained in subsection 3.5, TARD stores sequences of c-relations within the h-relation. By doing so, the generation of a c-relations chain can be defined prior to the instantiation of this building blocks. The so called sequence resembles the chromosome representation used in evolutionary approaches, suiting TARD for applying evolutionary algorithms for the generation of candidate solutions.

2.5 PaRC

PaRC (acronym for Parameters, Resolved rules and Constrain rules) is a knowledge engineering methodology for design automation [24]. PaRC prescribes the automation of parametric routine design problem by proposing a framework for representing knowledge and methods for automating the design problem. PaRC distinguishes two types of design knowledge: (1) knowledge that defines what the design problem is, and (2) knowledge that defines how the design problem can be solved. Design problem knowledge is represented by parameters and topological elements. Knowledge for solving the problem is represented by resolve rules (R-rules) and constraint rules (C-rules). A generation algorithm, generic to design problem formulated with the PaRC representation, enables the synthesis activity.

From the topologic point of view, PaRC combines knowledge about how to proceed with configuration knowledge in R-rules. As it will be discussed in section 5, the cardinalities parameters (explained in subsection 3.2) can be gathered to make relations specifying how the instantiation of one building block is constrained by the instantiation of others. In turn, these relations can be used to

transform the configuration problem into a parametric design problem. This has two advantages: firstly, design problem formulations can be easily modified and, secondly, parametric optimization can be used to fine-tune previously obtained configurations.

3 TOPOLOGY ABSTRACTION REPRESENTATION DIAGRAM (TARD)

TARD integrates explicit relations to model both the connectedness among elements in one abstraction level and the inter-dependency of elements at different abstraction levels. By doing so, characteristics of grammar, parametric and multi-level networks representations are resembled by a small set of generic building blocks: elements, c-relations and h-relations. Elements represent components of the design, and group the set of parameters used in its description. For example, in Figure 2, the components *E1*, *E2* and *E3* represent an engine, a gear box, and a wheel respectively. C-relations represent the connectedness of the elements in the topology, which is described in Figure 2 by the relations *C*. H-relations model how a group of c-relations are related to describe the composition of a higher level element. A group of c-relations related by one h-relation is termed *abstraction-group*. Both c-relations and h-relations are explicit, meaning they are building blocks by themselves. This characteristic enables the generation of solutions following both top-down and bottom-up approaches, as levels of abstraction are hierarchically independent from each others.

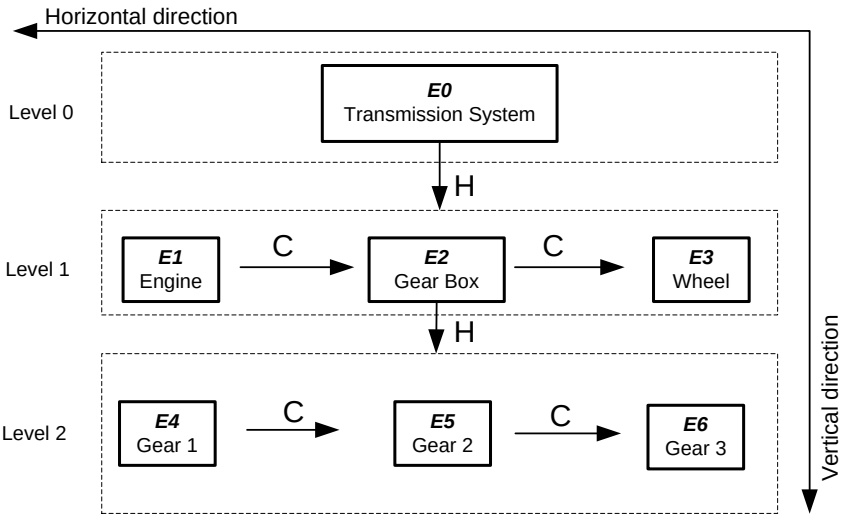


Figure 2. Example for the general usage of elements, c-relation and h-relations on two levels of detail

3.1 Design Problem Structure

Elements, c-relations and h-relations are used at three levels of problem structure: problem class, problem instance and problem solution, as indicated in Figure 3. At the problem class level, these building blocks describe the knowledge involved in the artifact. At a problem instance level, a partial instantiation of the knowledge represents the design requirements. A problem solution is represented by a full instantiation of the building blocks in TARD. Problem instances are transformed into problem solutions by algorithms that generate instances to the unknown descriptions, elements, and relations. Like in Object Oriented Programming (OOP), the instantiation of building blocks is done by objects. As result of this structure, one problem class can represent many problem instances, and one problem instance can have many problem solutions.

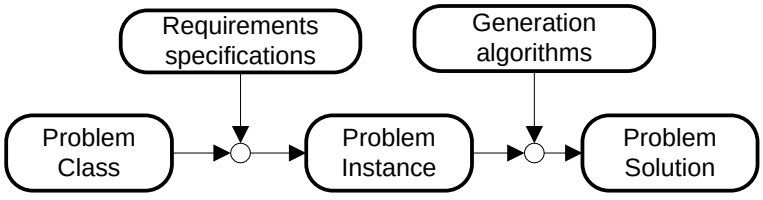


Figure 3. Framework to structure design problems

3.2 Base variables

TARD building blocks make use of the following common variables:

- **Cardinality:** Is a variable that describes and controls the number of instances a building block class can have. TARD distinguished two types of cardinalities: *local* and *global*. Local cardinalities describe the number of instances in a relative fashion (e.g. c-relation in relation to an element). Global cardinalities model the total number of instances of a certain element in the complete design problem.
- **References:** A reference is a pointer to another class or object instance. It is used to define how one building block is related to another. Relations also use references to element parameters to define its form. Depending on the direction of the association, the reference is of the type *owned* or the type *owner*. Owner references follow the hierarchical structure from top to bottom, while owned references do it from bottom to top. References are required to enable communication among the building blocks classes and instances.

3.2 Elements

Elements are used to represent different types of components in an artifact. For example, the transmission system in Figure 2 is composed out of four elements: *input shaft*, *gear1*, *gear2* and *output shaft*. Elements are described in TARD by the following information:

- **Parameters**: Describe the component by modeling its characteristics. At the problem class, parameters are only declared. Values are attributed by setting the requirements at the problem instance level, or by a generation algorithm to create problem solutions.
- **Cardinalities**: The total cardinality e describes the number of elements of a type (one component) in the design solution. The local cardinality e_l describes the total amount of elements within one abstraction-group. Cardinalities can be either known or unknown.
- **References**: Owner-reference to an h-relations and owned-reference to a c-relation.

Elements can be classified according to their position in a TARD as follows:

- **Zero-element:** Is the top element that represents the artifact as a whole. In Figure 4, the zero-element is denoted as Element Z. The cardinality of zero-elements equals one (1) and no c-relations are connected to them.
- **Abstract-elements:** Are elements composed of an abstraction-group. In Figure 4, Z is both a zero-element and an abstract element.
- **Base-elements:** Are elements that are not composed of an abstraction-group. In Figure 4, elements A, B and C are all base elements.

In addition to the former classification, elements can also be classified into parent and child elements. For example, in Figure 4 the element Z is the parent element of A, B and C; while element B is the child of element Z.

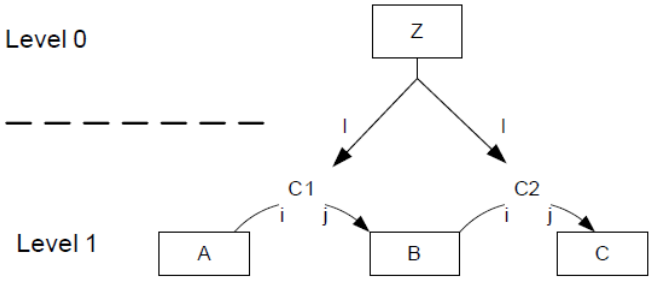


Figure 4. Example of bi-level TARD Diagram

3.3 C-relations

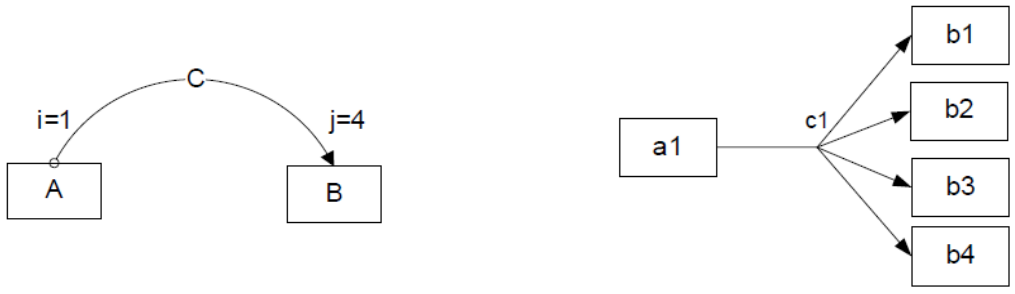
C-relations are used to model the connection between two element classes in an abstraction-group, as indicated in Figure 5. C-relations are modeled using the following variables:

- **Direction**: Indicates how one element is related to other elements of another class.
- **Cardinalities**: C-relations have two local cardinalities, namely, cardinality *i* and cardinality *j*. These cardinalities define the number *i* of instances of an element type that can be connected with

a number *j* of instances to another element. As the framework is developed to support 1-simplex relations, *i* has a constant value of one (1). The example shown in Figure 5 illustrates a c-relation class *C* that relates the element classes *A* and *B* with $i = 1$ and $j = 4$. The instantiated c-relations in Figure 5 shows that one instance of *A* is connected to four instances of *B*.

- **References**: Owner-reference to an element, owned-reference to a h-relation

It is important to notice in Figure 5(b) that although *C* connects one element *A* with four elements *B*, it does not connect the instances of *B (b1, b2, b3, b4)* among each others. So, even if c-relations relate more than two element instances, as in the example in Figure 5(a), they do not model multidimensional simplexes. C-relations support the generation of topologies by generating sequences, enabling the application of grammar approaches for design synthesis (this point will be treated in a future publication). C-relations also enable the assemble sequences of elements independent of the previous existence of instantiated parent elements. Furthermore, elements are related to c-relations by references, which allow its instantiation in two ways: by instantiating the relations and later their referenced elements (a top-down strategy), or by matching previously instantiated elements and then instantiate the c-relation accordingly (a bottom-up strategy).



a. C-relation class connected to two element classes

b. C-relation instance connected to five element instances

Figure 5. C-relations: class and instance

3.4 H-relations

H-relations relate the c-relations of an abstraction-group at level *N* with its parental element on level $N-1$. The c-relations and h-relations create an explicitly formulated skeleton in which, by the means of generation or recognition algorithms, element instances can be created. H-relations are modelled using the following variables:

- **Cardinalities**: Each reference from an h-relation class to a c-relation class is associated with a total cardinality l_c. This cardinality models the amount of instances of c-relations in an abstraction-group. In the example shown in Figure 6(a), a c-relation *C* connects one element *A* with one element *B*. In order to connect 3 element instances *A* with 3 element instances of *B*, three c-relations *C* are required, which results in $l = 3$.
- **Sequence**: The second type of information stored within the h-relation is called sequence. A sequence is a list of c-relations, and can be either known or unknown. Solving h-relations with unknown sequences are equivalent to using grammar approaches.
- **Constraining rules:** Constrained combination of c-relations by specifying required combinations or not allowed combinations. Constraining rules are formulated in the form of sequence fragments. These types of rules are formulated using references to both c-relations and elements classes and instances and logic operators.
- **Reference**: Owner-reference to c-relations, owned-reference to an element.

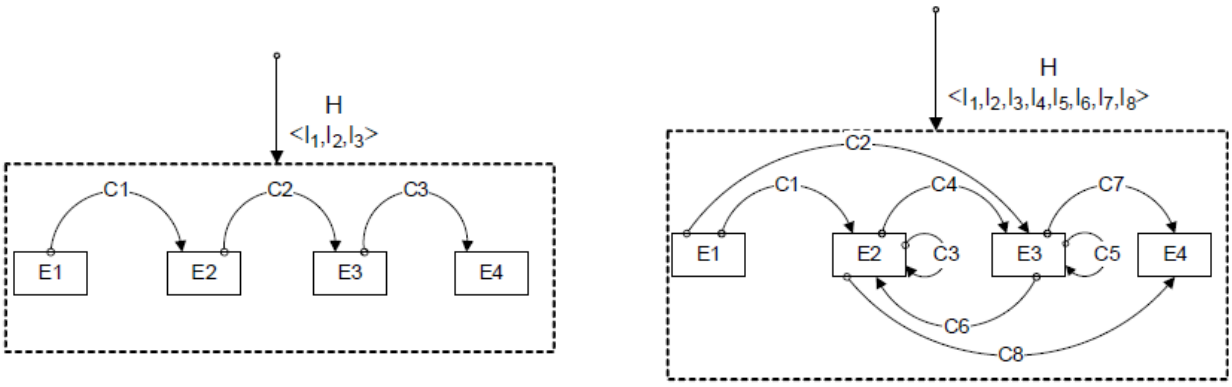
H-relations enable the representation of multiple domains. This can be done by attributing one specific h-relation to each of the domains that are to be modeled. However, as h-relations are always related to parent elements on a one to one basis, an element instance can only have one h-relation of each type.

a. Class representation with $l_C = 3$ *b. Instance representation of c-relations*

Figure 6. H-relations: class and instance

3.5 Abstraction-groups

Creating solutions to problem instances consists in instantiating c-relations and elements in abstraction-groups, which results in a collection of m elements connected by $m-1$ c-relations. This collection is regarded as sequence. Abstraction-groups with one possible sequence, as for example in Figure 7(a), are regarded as simple. On the other hand, abstraction-groups allowing multiple distinctive sequences are regarded as complex, as for example the abstraction-group shown in Figure 7(b). The degree of freedom of a simple abstraction-group only depends on the number of unknown cardinalities, while in a complex one it depends on both the cardinalities and number of possible sequences. An abstraction-group is complex if the number of c-relations n is equal or larger than the number of elements m within the abstraction-group, thus $n \geq m$. The example illustrated in Figure 7(b) shows a complex abstraction-group with 4 elements and 8 c-relations. The example has an infinite set of solutions. On the other hand, the abstraction-group in Figure 7(a) has one possible sequence.



a. Simple abstraction-group *b. Complex abstraction-group*

Figure 7. Simple and complex abstraction-groups

3.6 Bottom-up vs. Top-down

Within one abstraction-group, the concepts of top-down and bottom-up are related to mechanisms used in the instantiation of Elements and C-relations. In top-down approaches, instantiated upper layers determine the **creation** of the lower level. For example, an instantiated C-relation determines the instantiation of an element. In a bottom-up approach occurs the opposite: instantiated lower layers are used to **recognize** upper non instantiated levels. For example, recognize if two instantiated elements can be related by one of the C-relations in the abstraction-group. Determining which of these processes -creation or recognition- is to be applied is a function of which elements and relation have been previously instantiated. Its instantiation is the consequence of how the requirements are distributed in the problem instance.

To explain this concept, let's consider Figure 8. The left side of the figure shows an example of an abstraction-group described by the element classes *A*, *B* and *C*. The right side of the figure shows a pool of element instances *a*, *b* and *c*. The number of instances of each type in the pool is represented by X_a, X_b and X_c respectively. Furthermore, the amount of instances required in a solution is denoted by the element cardinalities. The number of element instances that result from a recognition

process or a creation process is denoted as f_x and f_g, respectively. Element cardinalities that exclusively result from a generation process are given by $e = f_g$ and those resulting from recognition are given by $e = f_x$. Having this, the following types of abstraction-groups and their solving mechanism can be identified according to the combination of instantiated building blocks:

- **System type 1:** It has the following characteristics:

  $$x = 0, e = f(g)$$

  A creation mechanism is used, as no elements have been instantiated yet.

- **System type 2:** It has the following characteristics:

  $$x > 0, x < e, e = f(g)$$

  Instances are only created for element classes that have no existing instances. Recognition is used to determine how to connected existing instances.

  System type 3: Instances are only created for element classes that have no existing instances.

  $$x > 0, x < e, e = f(g) + f(x)$$

  C-relations are recognized for instantiated elements, and created for non instantiated elements. It has the following characteristics:

- **System type 4:** It has the following characteristics:

  $$x > 0, X = e, e = f(x)$$

  C-relations are only recognized.

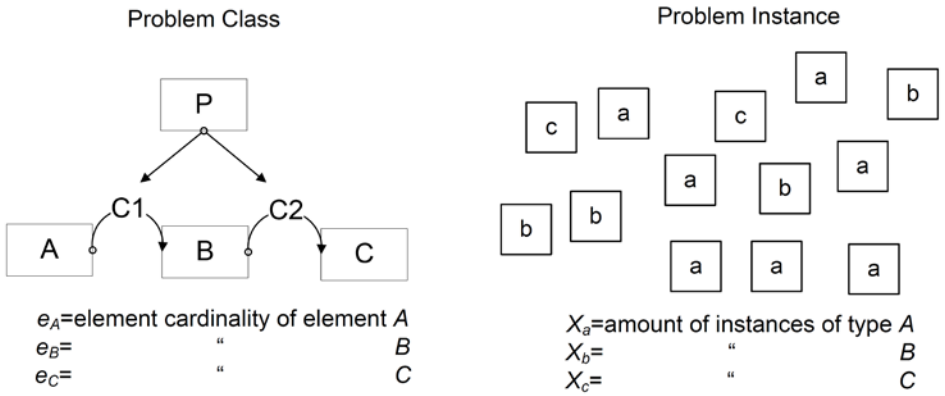


Figure 8. Class and instance representations of an abstraction-group.

3.7 Guiding Search Processes in TARD

A sequence has been described as one possible combination of the elements in an abstraction-group. This combination is stored in the form of a list of successive C-relations (a string), which starts and finishes with an C-relations. In order to manage the generation of sequences, a factor k_C associated to each C-relation *C* instance in the string has been defined. This factor denotes the amount of relation instances at a given position in the sequence. Thus, the total amount of instances l_C of the C-relation in a sequence *C* is given by the sum of its factors k_C. By relating both terms, an expression termed as sequence equation is derived to aid the management of complex abstraction-groups:

$$l_c = \sum K_c \tag{1}$$

In a complex abstraction-group, algorithms iteratively connect elements using C-relations. While this is done, a sequence is generated and expanded, which in turn is used as guidance by the complementary rules. By adding each of the factors k_C, a comparison can be made to the sequence

equations. This enables making predictions over which relation to choose in the next iteration step or, if no alternative can be found, to stop. Thus, the sequence equation provides a guidance to the assembly processes in order to end up with a consistent sequence.

4 EXAMPLE: BELT SYSTEM DESIGN

The following example illustrates the concepts discussed above. Figure 9 shows a sketch of a conceptual belt drive system. The embodiment of the belt system includes an input shaft (*IS*), an input pulley (*IP*), a belt (*B*), an output pulley (*OP*) and an output shaft (*OS*). The components within the embodiment are arranged in order to serve the main functionality of the system, which is transferring force and momentum. This type of problem can be regarded as a topologic design problem, as it concerns the configuration of different types of components. In other words, the topology has to make sense from a functional point of view. This implies that the c-relations are established within the functional domain of transferring energy. Figure 10 presents a TARD model of this belt drive system. It is composed of just one level of abstraction where all components in Figure 8 have been translated into elements. These elements are related to each other by the means of four c-relations (*C1*, *C2*, *C3, and C4*) in such a manner that it resembles the path of the energy flow from input to output. Due to the fact that each component only appears once the cardinalities *i* and *j* equal one (1). Furthermore, the cardinalities *l* contained in the h-relation, which is connected to the zero level element *BS* (Belt system), are equal to one as well.

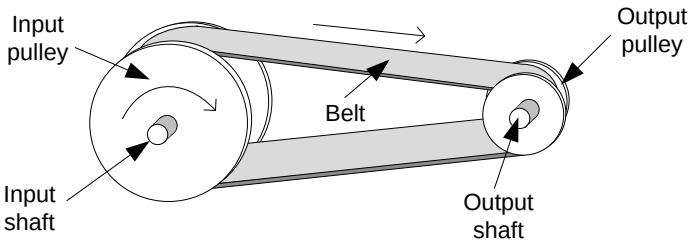


Figure 9. Schematic of a belt drive transmission system

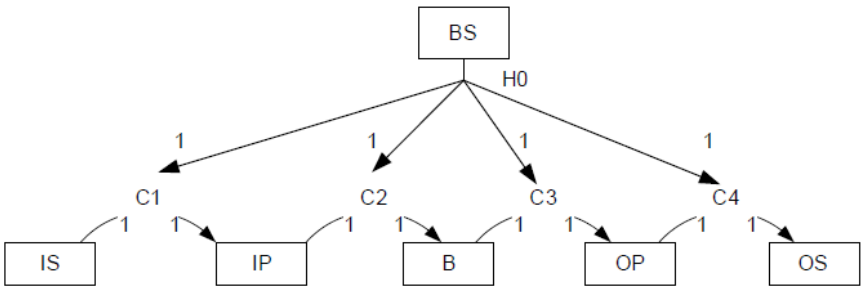
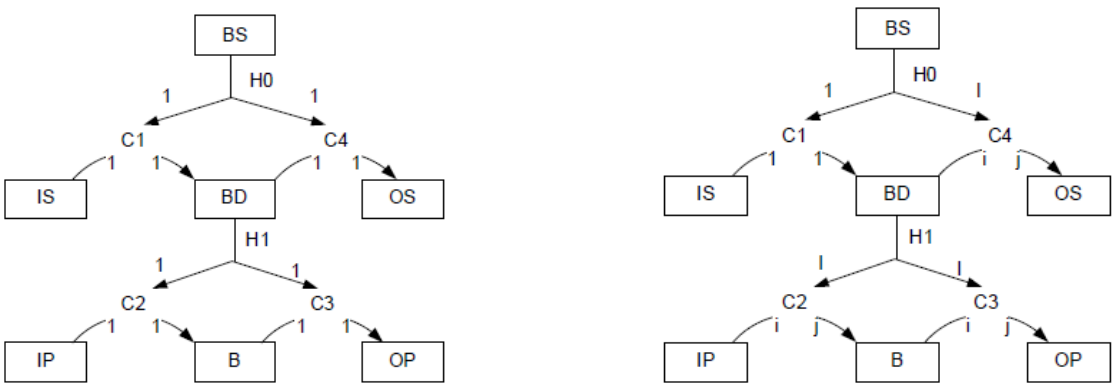


Figure 10. TARD of the belt drive transmission system represented by one abstraction-group

The value of the cardinalities is fixed due to the fact that the artifacts' topology is known. However, the structure of the diagram can be changed by reorganizing the base elements into different abstraction-groups. The topology diagram shown in Figure 11 is constituted by the same physical elements as the structure shown in Figure 10. However, their representation is done differently, given that the structure in Figure 11 contains a new element *BD* (belt drive), which is composed by the elements *IP*, *B* and *OP* in a lower level abstraction-group. Although this model looks more complex than the previous one, it allows separating the problem into two independent chunks. By doing so, level 1 can be concerned with the search of adequate transmission ratios, while level 2 is concerned with the search of parameter values that satisfy the ratios specified in level 1.



a. All cardinalities fixed: structure fully defined

b. Most cardinality unknown: structure undefined

Figure 11. Topological network of the belt system with two levels of detail

The TARD model shown in Figure 11(a) is topologically fully defined, as its cardinalities are known, and it is limited to one possible sequence of c-relations. On the other hand, the TARD model shown in Figure 11(b) is topologically under defined, given that some of its cardinalities are unknown. Depending on the number of unknown cardinalities and its distribution in the TARD network, there are different possible configurations of the solutions. For example, the belt system in Figure 9 is one alternative. Another possible solution for the under defined situation is the design shown in Figure 12, which has two belts and two output pulleys. This solution is achieved by setting the cardinalities *i* and *j* of *C2* to *1* and *2* respectively, introducing a parallel structure, while the cardinalities at *C3* are kept equal to one. It should be pointed out that due to the parallelism all successive objects in the abstraction-group have parallel instances as well. This is kept consistent by setting the cardinality *l* for *C3* in *H1* equal to *2*.

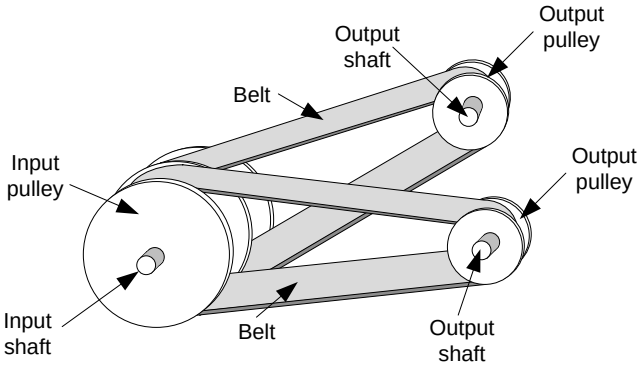


Figure 12. A conceptual double belt transmission: one input, two outputs

P-relations

P-relations are used when it is required to model physical connections of elements in different abstraction-groups. Usually, this relationship exists between the first and last elements of an abstraction-group. Consider the example in Figure 11. Here, the transmission system consists of physically connected parts. Figure 10 shows that the elements of the belt system can be divided into two abstraction-groups. Although the output pulley *OP* and the output shaft *OS* are physically connected, they are allocated into different abstraction-group. If there are two instances of element *OP*, there have to be two instances of *OS* as well and vice versa. The same holds for the elements *IS* and *IP*, however the cardinalities remain 1 in this case. The correlation between those elements is called proximity relation, or p-relation. In the example, the local cardinalities of *IP* and *OP* are set equal to the local cardinalities of *IS* and *OS* respectively, resulting in the proximity relations: $e < IP >= e < IS >$ and $e < OP >= e < OS >$, with $e_1/e_2 = p$, where p is called proximity constraint. The choice of which element to relate by a p-relation depends on the problem itself and its boundary conditions.

5 DISCUSSION

This paper presented a model for representing topologic knowledge of routine design for computational design synthesis purposes. TARD (acronym for Topology Abstraction Representation Diagram), as this model is termed, models topologies at the hand of three base building blocks: elements, c-relations and h-relations. In TARD, elements group parameters, c-relations describe how different elements types can be connected and h-relations describe the relations between different levels of abstraction. This configuration allows developing solving strategy methods as function of the organization of building blocks and solving abstraction-groups independent from each others. TARD also enables the integration of several domains by adding extra h-relations.

Further research will focus on the development of methods to automate the generation of candidate solutions based on TARD. One method should enable the representation of topologies in the form of algebraic equations by relating the cardinalities in TARD. These equations will have three specific functions: keep consistency of the topology, identify distributions of requirements in the form of instantiated building blocks, and control the instantiation of elements. The idea is to enable the usage of constraint solving methods when dealing with simple abstraction-groups. A second method generation method based on grammar approaches should be developed for dealing with complex abstraction-groups. The grammar based approach can make use of cardinalities to control the generation of solutions as well as integrate functions that determine if the generation algorithms should exhibit creation or recognition characteristics. Creation corresponds to a top-down approach, for example, assessing first the ***types*** of elements that can be connected, and then instantiating and connecting them. Recognition, on the other hand, corresponds to a bottom-up approach. For example, a pool of ***instantiated*** elements exists, and its characteristics determine how they can be connected. The end goal is to open new ways of generating design. For example, by assessing the differences of candidate solutions of problems that have been specified in terms of *what we want* (requirements at the top abstraction levels) vs. those specified in terms of *what we have* (requirements set at the bottom levels of abstraction). This moves the task of designers from *finding solutions* towards *designing the right problems*.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program ‘Integrated Product Creation and Realization (IOP-IPCR)’ of the Dutch Ministry of Economic Affairs.

REFERENCES

[1] Minderhoud, S. and Fraser, P., Shifting paradigms of product development in fast and dynamic markets. *Reliability Engineering & System Safety*, 2005, 88(2), pp127-135.

[2] Cagan, J., Grossmann, I.E. and Hooker. A Conceptual Framework for Combining Artificial Intelligence and Optimization in Engineering Design. *Research in Engineering Design*, 1997, 9, pp20-34.

[3] Campbell, M.I. and Rai, R., A Generalization of Computational Synthesis Methods in Engineering Design. *AAAI Spring Symposium Series*, 2003.

[4] Chakrabarti, A., *Engineering Design Synthesis: Understanding, Approaches and Tools.* (Springer Verlag, London, 2002).

[5] Antonsson, E.K. and Cagan, J., *Formal Engineering Design Synthesis*. (Cambridge University Press, 2001).

[6] Jauregui-Becker J.M, Tragter H and van Houten, F.J.A.M., Structure and models of artifactual routine design problems for computational synthesis. *CIRP Journal of Manufacturing Science and Technology*, 2009, 1(3):120-125.

[7] Shea, K. and Cagan, J., The design of novel roof trusses with shape annealing: assessing the ability of a computational method in aiding structural designers with varying design intent. *Design Studies*, 1999, 20, pp3-23.

[8] Campbell, M.I., Cagan, J. and Kotovsky, K., Agent-based synthesis of electro-mechanical design configurations. *Journal of Mechanical Design*, 2000, 122(61).

[9] Schotborgh, W.O., Kokkeler, F.G.M., Tragter, H. and van Houten, F., A Bottom-Up Approach for Automated Synthesis Tools in the Engineering Design Process. *Proceedings of International Design Conference 2006*, 2006, pp. 349-356.

[10] Orsborn, S., Cagan, J., Pawlicki, R. and Smith, R., Creating cross-over vehicles: defining and combining vehicle classes using shape grammars. *Artificial Intelligence in Engineering Design and Manufacturing*, 2006, 20(3), pp217-246.
[11] Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y. and Tomiyama, T., Supporting conceptual design based on the function-behavior-state modeler. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1996, 10, pp275-288.
[12] Kurtoglu, T., Swantner, A. and Campbell, M.I., Automating the Conceptual Design Process: From Black-box to Component Selection. *Design Computing and Cognition '08*, 2008, pp553-572.
[13] Startling, A.C. and Shea, K., A Parallel Grammar for Simulation-Driven Mechanical Design Synthesis. *International Design Engineering Technical Conferences 2005*, 2005, 2(A), pp427-426.
[14] Parunak, V., Case Grammar: A Linguistic Tool for Engineering Agent-Based Systems. *Unpublished white paper at http://www.iti.org/~van/~asegram.ps*, 1995.
[15] Kurtoglu, T. and Campbell, M.I., Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping. *Journal of Engineering Design*, 2009, 20(1), pp83-104.
[16] Shea, K. and Cagan, J., Languages and semantics of grammatical discrete structures. *AI EDAM*, 1999, 13(4), pp241-251.
[17] Soman, A., Padhye, S. and Campbell, M.I., Toward an automated approach to the design of sheet metal components. *AI EDAM*, 2003, 17(03), pp187-204.
[18] Duarte, J., *Customizing mass housing: a discursive grammar for Siza's Malagueira houses.* (Massachusetts Institute of Technology, 2000).
[19] Gips, J., Computer implementation of shape grammars. *NSF/MIT Workshop on Shape Computation*, 1999.
[20] Johnson, J.H., Multidimensional networks in the science of the design of complex systems. In *ECCS 2005 Satellite Workshop: Embracing Complexity in Design.* pp33-48
[21] Koza, J.R., Streeter, M.J. and Keane, M.A., Automated Synthesis by Means of Genetic Programming of Human-Competitive Designs Employing Reuse, Hierarchies, Modularities, Development, and Parameterized Topologies. In *The 2003 AAAI Spring Symposioum: Computational Synthesis: From Basic Building Blocks to High Level Functionality.*
[22] Fan, Z., Wang, J., Seo, K., Hu, J., Rosenberg, R., Terpenny, J. and Goodman, E., Automating the Hierarchical Synthesis of MEMS Using Evolutionary Approaches. *Evolvable Machines*, pp129-1492005).
[23] Zhang, Y., Antonsson, E.K. and Martinoli, A., Evolutionary engineering design synthesis of on-board traffic monitoring sensors. *Research in Engineering Design*, 2008, 19(2), pp113-125.
[24] Schotborgh, W.O., *Knowledge engineering for design automation.* (University of Twente, 2009).

Contact: Juan M. Jauregui-Becker
University of Twente
Faculty of Engineering Technology
Laboratory of Design, Production and Management
P.O. Box 217, 7500 AE Enschede
The Netherlands
T: +31 53 489 4025
F: +31 53 489 3631

Juan Jauregui-Becker currently works as assistant professor at the University of Twente, in The Netherlands. His PhD thesis investigated the development of a computational design synthesis method. He received his BSc degree in mechanical engineering at the University of Los Andes in Venezuela and his master degree in mechanical automation at the University of Twente in The Netherlands.