INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

REQUIREMENTS-DRIVEN DESIGN COMPUTATIONS IN NEXT-GENERATION CAD

Vikram Bapat[1], Bernhard Bettig[2] and Joshua Summers[3]
[1]Michigan Technological University, USA
[2]West Virginia University, USA
[3]Clemson University, USA

ABSTRACT
This paper discusses a new computational framework for requirements-driven automated design synthesis that allows design changes to be accepted seamlessly from multiple stakeholders and the computer software itself. The framework, based on declarative-variational mathematical underpinnings, supports the systematic design process by using requirements as a basis for coordinating designer and computer efforts. We define a design knowledge representation based on the design exemplar that enables mapping a large set of real-world design requirements to a computer-interpretable form. A new generic high-level algorithm that can apply requirements (exemplars) simultaneously is proposed. As opposed to conventional rule-based production systems, which enforce design requirements one at a time, this new algorithm is able to detect incompatibilities between requirements by applying the requirements simultaneously.

Keywords: computer-aided design, automated design synthesis, requirements satisfaction, constraint satisfaction, design exemplar technology

1 INTRODUCTION
Computer-Aided Design (CAD) software is used pervasively in the manufacturing industry to speed up the creation, analysis and realization of products and it is difficult to imagine not using it. However, current CAD software does little more than accept the input of designs that have already been conceived by a designer; CAD software does not provide any help in actually conceiving designs. We have argued that this limitation is inherently due to the mathematical framework on which CAD software is implemented [1]. In this paper we propose a next-generation requirements-driven CAD framework with declarative-variational [2] mathematical underpinnings that would enable a design exploration process as shown in Figure 1. Here designers and all interested stakeholders generate a set of requirements that are mapped to a computational representation and applied to the computational system. At the same time engineering knowledge from domain experts is also applied to the system, also expressed in the form of requirements. The requirements are combined seamlessly within the computation system which can then synthesize alternative satisficing solutions. Requirements can be posed in terms of dynamically applied algebraic, geometric, physical, and semantic relationships, including requirements on attribute values and the existence of the design objects and relationships themselves. The computation system allows designs to be under-constrained, thus allowing for tweaking of the design by designers and the addition of new requirements. The computation system also allows designs to be over-constrained and thus allows for the detection of incompatible requirements. Examining the alternative solutions synthesized (and evaluated and selected) by the computer, the designer can remove unanticipated deficiencies by further refining the requirements, by either changing, adding or removing requirements. Once the requirements have been sufficiently refined, attractive design solutions can be explored in greater detail by recursively repeating this process.
This research tries to answer the question as to how can design requirements and knowledge be used to generate and evaluate designs automatically? Rather than continuing to use the computer as an analysis calculator or document recorder, we propose to use the deductive, algorithmic reasoning of computational support to create a framework in which humans and computers can interactively reason

about the design problem, collaboratively generating, evaluating, and selecting among design alternatives and iteratively refining the design requirements. In this paper we will address the three fundamental aspects of the system as follows:

- The design exemplar data structure and how it can be used to represent design requirements.
- The generic exemplar algorithm and how it can be used to enforce design requirements one at a time.
- A new generic high-level algorithm that can apply design requirements (exemplars) simultaneously and detect incompatibilities in the requirements.

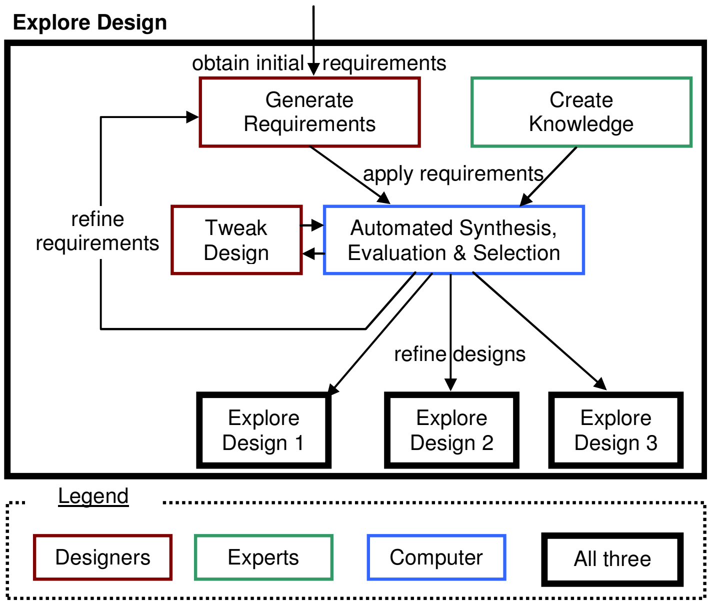


Figure 1. Requirements-Driven Design Exploration Process

2 REPRESENTING REQUIREMENTS USING THE DESIGN EXEMPLAR

2.1 Types of Requirements

Design requirements are defined as conditions or properties which are essential or requisite to the design [3]. Design requirements are also defined as something that constrains the possible solutions of a design problem to a subset of all possible designs [4]. Often, as in this paper, design constraints are considered the same and used interchangeably with design requirements [3, 5, 6]. As well, for us design specifications are the lists or collections of requirements of the product being designed. The specification list defines the design problem, and as we solve for the design we modify the problem itself be changing the specification in order to get a solution [3, 5]. At every stage of the systematic design process, design requirements play an important role in defining the design task.

In order to be able to use requirements in a computational system, the requirements, which are originally provided in human interpretable "natural language," need to be represented in a computer interpretable form**.** Until now, research has mainly been done in the field of requirements engineering. This research has been concerned with four aspects of requirements management and is largely domain independent. The research can be classified into the following broad areas as follows [7-9]:

1. How to elicit requirements [10-13] and validate them [14-16] by analyzing the specification to check whether these are the ones the designers really want.
2. How to achieve Requirements Tracing [17-19]; i.e., trace requirements back to the originating source as well as its subsequent use in the design.
3. Managing evolving requirements in a collaborative environment [20, 21].
4. Applications of managed requirements in evaluating and checking the design for satisfaction of the requirements [22-24].

From a system engineering viewpoint, STEP AP233 standards [25] have been proposed and encapsulate standards used for requirements management. Many commercial software's like Telelogic DOORS [26] and Teamcenter for systems engineering [27], which can work off these standards, are available to assist the users in the above mentioned areas. Requirements management forms a very crucial part of systems engineering. Efforts have also been made to study requirements management covering the four broad areas discussed above and applied to automotive engineering [28]. Information modelling efforts to represent requirements [29, 30] do not provide support for linking the requirements to detailed geometry and driving the creation of geometry. In the research presented in this paper we do not concentrate on requirements management but on how the requirements managed by the user can be effectively used to assist in mechanical design by computationally affecting the creation of geometry and being linked to it. In this research, we are interested in the integration of requirements with the "generate," "evaluate" and "select" phases of every design stage throughout the design process so that they can be driven automatically using the requirements as an input.

At a fundamental computational level we have found that the detailed requirements seem to fall into three different types:

1. Requirements on Design Attributes

    1.(a) Design attribute *X* must be equal to *V*

Here *X* is an attribute or property of the design that can be evaluated or can be directly observed and *V* is a qualitative or quantitative value (loosely – an adjective) which can be absolute or can be an attribute of other design(s) or design element(s). An example of a Type 1(a) requirement is "The length of the truck cargo box must be 3700 mm", or "the length of the cargo box must be the same as the length of the cargo box from a competitor." A key impact of Type 1(a) requirements is that they take away degrees-of-freedom in the design. That is, one or more variables whose value could earlier be chosen freely from a continuum of alternatives are now constrained to take on a controlled value.

    1.(b) Design attribute *X* must be greater than *V*

This is an inequality type of comparative expression. Here again *V* is (i) an absolute value, (ii) an attribute of one other design or design element, or (iii) an attribute of a set of other designs or design elements. For example, (i) "the length of the cargo box must be greater than 3700 mm," (ii) "the length of the cargo box must be greater than the length of the cargo box from a competitor," or (iii) "the length of cargo box must be greater than the lengths of cargo boxes from all competitors." (Think of "big," "bigger," "biggest.") Although Type 1(b) requirements are also comparative requirements as Type 1(a), they do not take away any degrees-of-freedom in the design, but instead only serve to bound the design space.

    1.(c) Design attribute *X* must be as great as possible

This requirement describes an objective. It can be stated in a number of different ways (e.g., "as low as possible," "as light as possible"). Type 1(c) requirements refer to design attributes as in Types 1(a) and 1(b) and in general are also specified with respect to a target value, which implicitly can be positive or negative infinity. However Type 1(c) requirements have the interesting and unique ability to take away degrees-of-freedom "en mass." For example, in the requirement "the length of the cargo box must be as great as possible" every free variable to which the length is sensitive will have all of its degrees of freedom removed.

2. Requirements on the existence of objects and relationships

    2.(a) Design must have *Y*

Here *Y* is a feature, a component or some element of the design. Requirements expressed as "Design must have *Y*" deal with the existence of objects. For example the requirement that "a truck must have wheels, motor, cargo box and cab" deals with the existence of these objects. These requirements also impact the existence of objects later in the design process, for example the existence of more detailed objects such as the faces, edges and vertices that are necessary to describe the shapes of these components.

    2.(b) Design element *Y1* must be *R* with respect to design element *Y2*

Here *R* is a relationship. This includes requirements such as "the cargo box must be PART OF the truck" and "the line represents the GEOMETRY OF the road."

3. Requirements on Function

    Product must do *Z*

Here *Z* is a function or action verb. These are the function or behavioural requirements (e.g., cargo box must store goods). These are typically the most important requirements for mechanical devices but not the only ones. For example aesthetic requirements on form and shape cannot be described using action verbs, but necessitate using Type 1 and 2 requirements.

2.2 Mapping Design Attribute Requirements to Exemplars

2.1.1 Objects and Constraint Types

Type 1 requirements can be represented by constraints and objectives that control object attributes. Current CAD systems are based only on equality type of constraints between geometric objects and parameters, as shown in column two of Table 1 and can map only the Type 1(a) kinds of requirements.[1] However, to truly represent all kinds of Type 1 design requirements, constraints such as those shown in all columns of Table 1 must be allowed [31-33]. Column 3, inequality constraints, must be allowed in order to represent Type 1(b) requirements and column 4, objectives, must be allowed in order to represent Type 1(c) requirements.

Table 1. Object and Constraint Types

Object Types	Equality Constraints	Inequality Constraints	Objectives
(a) Algebraic			
Variable	Equation, Table	Inequality	Minimize, Maximize
(b) Geometric			
Point, direction, line, plane, circle, ellipse, parabola, hyperbola, cylindrical surface, spherical surface, conical surface, toroidal surface, B-spline curve, B-spline surface	DISTANCE, ANGLE, RADIUS, COINCIDENT, INCIDENT, TANGENT, PARALLEL_OFFSET, PARALLEL_DIRECTION, RIGHT_ANGLE	ORDER_ON, IN_FRONT_OF, LEFT_OF, ON_INDICATED_SIDE, SAME_ORIENTATION, ORIENTED_LEFT_OF, (plus opposite versions of these)	FARTHEST_FROM, FARTHEST_IN_INDICATED_DIRECTION, CLOSEST_ORIENTATION, ORIENTED_MOST_LEFT_OF (plus opposite versions)
(c) Topological/Geometric			
Vertex_point, edge_curve, face_surface, solid	LENGTH, AREA, VOLUME, MOMENT	CONCAVE/CONVEX, SMOOTH/SHARP, INSIDE/OUTSIDE	

2.1.2 The Design Exemplar

Although many Type 1 requirements apply directly to specific instances of design objects, more often Type 1 design requirements apply to types of objects or certain situations (e.g., all wheels must be tangent to the road) and must be applied dynamically as the design changes (e.g., as new wheels are created in the design). Design exemplar technology provides a logical reasoning mechanism that can be used in a declarative-variational environment to apply constraints and objectives automatically where they are applicable. The design exemplar is a powerful, generic data structure developed to represent geometric, algebraic, physical, and semantic design characteristics in design problems [31, 34, 35]. Designers, using exemplars, can find geometric properties (e.g. walls with a specific thickness) in CAD models and then change these properties as needed (e.g. change the wall thickness from 2.5 mm to 12.7 mm). In the exemplar graphical representation used below, round nodes are used to represent objects and square nodes are used to represent relationships or constraints, which can be unary, binary, or n-ary. Exemplars represent design characteristics by defining the objects and relationships that must exist explicitly in the design model when the design has the given characteristic. For example, the Designed_Offset_Faces exemplar shown in Figure 2(a) defines a characteristic in which there exist in the design model two faces (*f1* and *f2*), that have been explicitly constrained by the designer to be offset from one another. Once an exemplar has been defined, a pattern matching algorithm (e.g. [36]) can be executed to identify faces in the design that the designer has offset. One difference of exemplars from conventional face adjacency graphs [66] and graph grammars [37, 38] is that exemplars allow defining sub-graphs that repeat (repetition block), sub-graphs that must not exist in the design (NOT block) and sub-graphs that can occur in different ways (OR, XOR blocks).

An exemplar can also describe objects and relationships that are implicit in the design. For example, Figure 2(b) shows an exemplar, "Offset_Faces," in which two faces exist in the design that are

[1] I-DEAS is an exception because it allows variables to also be controlled by inequality expressions.

implicitly offset. That is, they happen to be offset in the design, but are not directly being controlled to be so. To find instances of patterns with implicit relationships, pattern matching must be combined with solving and evaluating constraints. Specialized domain-specific solvers are called by a generic solving algorithm (see [31, 39] for complete details on the generic solving algorithm and supported solvers). Implicit objects and relationships are shown with dashed lines in the exemplar graph.

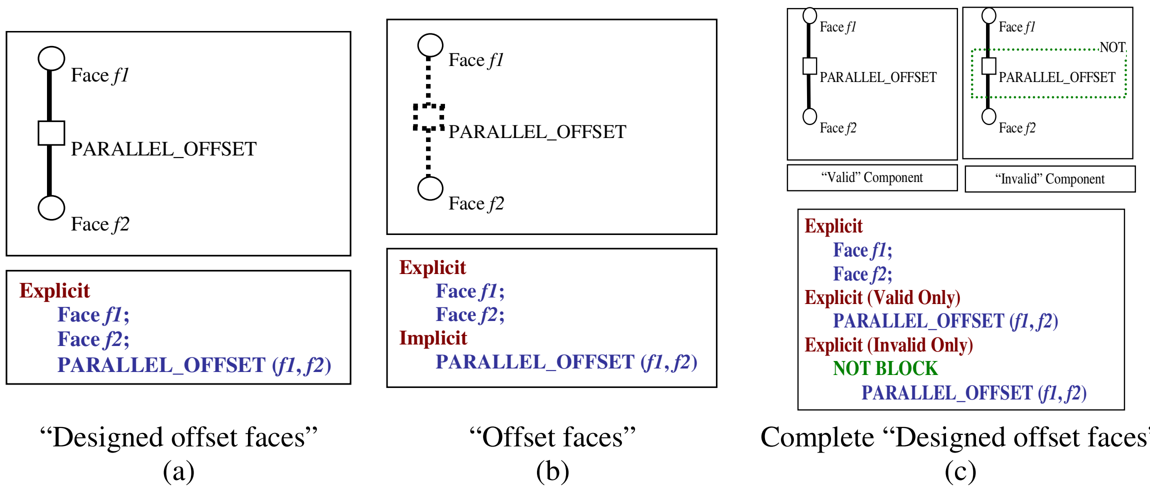


Figure 2. (a) Exemplar for "Designed offset faces" (b) Exemplar for "Offset_Faces" (c) Exemplar for Complete "Designed offset faces"

Besides distinguishing explicit and implicit characteristics, an exemplar also distinguishes what a design is like when it has the described characteristic and when it does not. For example, Figure 2(c) shows an exemplar distinguishing between faces that have been designed to be offset and those that have not. The "valid" and "invalid" parts of an exemplar describe the two situations: *valid*, the identified faces are explicitly designed to be offset, and *invalid*, the identified faces are not explicitly designed to be offset. The "valid" and "invalid" components enable modifying or enforcing a design characteristic in terms of both the existence of objects and the values of attributes. The exemplar algorithm works somewhat like a graph grammar production system [37, 40] in that the graph from one component is recognized in the design graph using a sub-graph isomorphism algorithm (with evaluation of any implicit relationships). Matches that are found in the design of the *invalid* component are converted to the *valid* component (or vice versa) by adding and removing explicit objects and relationships and enforcing any implied relationships. Modifications can include changing values (e.g., modify hole diameters) or changing existence (e.g., add or delete hole features).
So far, exemplars have been used for single task problems [31, 41-45] such as (i) identifying design objects with a given characteristic (e.g. undercut faces), (ii) evaluating design properties (e.g. length of a duct), (iii) validating designs (e.g. not too big), (iv) comparing two designs with respect to a characteristic, and (v) modifying a design (e.g. to have or not have a fillet). This research is the first time that the design exemplar has been considered for use in automated design synthesis.

2.1.3 Composite Constraints and Dynamically Applied Constraints
Composite constraints are required in order to express, in a comprehensible way, requirements that can only be expressed as system of constraints. For example, a requirement that distance between two objects be greater than a specified value has to be expressed using a set of two primitive constraints: a DISTANCE constraint and an algebraic inequality expression.
In some instances a design requirement can be easily described by a single geometric constraint, but it is not known ahead of time which objects are the ones on which the constraint should be applied. For example, a constraint on the overall length of a part can be imposed by a constraint DISTANCE (*object_1*, *object_2*, *distance_measure*) where *distance_measure* is the distance between *object_1* and *object_2*, which are at opposite ends of the part. The objects referenced by this constraint could change as the design progresses, as occasionally different objects become the end objects. Figure 3(a) shows an example of a situation where this can happen. If we require the overall length to be maintained to be less than or equal to 10 units then as the shape changes, the DISTANCE constraint

must be applied to different objects. A dynamic composite constraint must provide rules to identify which objects are the ones that should be constrained. This makes the constraint much more complicated but can be implemented using exemplars as shown (compressed) in Figure 3(b).

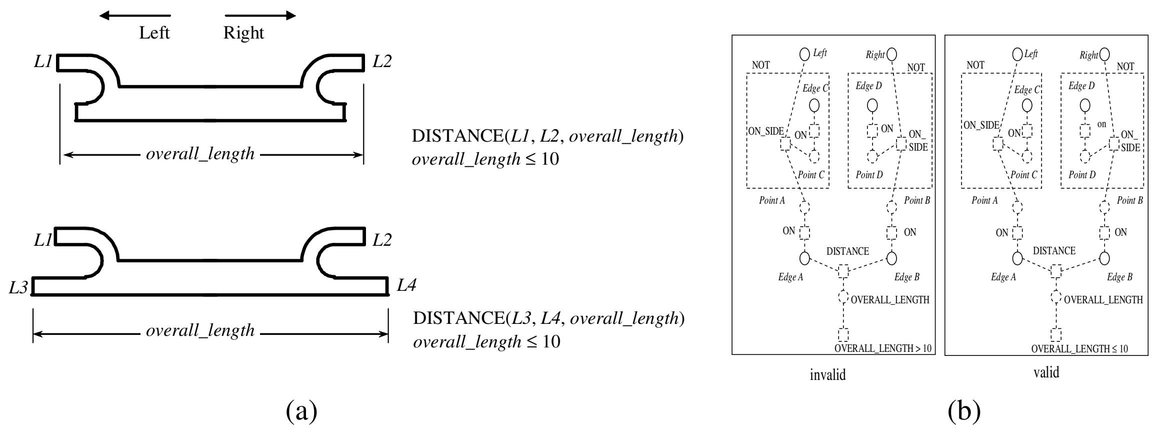


Figure 3.(a) Overall Length constraint acts on different objects as shape changes (b) The dynamically applied "overall length" constraint implemented using an exemplar

2.3 Mapping Existence Requirements to Exemplars

In order to represent Type 2 requirements in a computational representation we must also be able to somehow express requirements on the existence of objects and relationships. Figure 4 shows the exemplar used to enforce the requirement that the geometry of a wheel be added to the design if an object of type "Wheel" exists in the design. In general the exemplar algorithm will enforce that the design stays with the "valid" and not the "invalid" for whatever exemplars are given. Hence when the algorithm detects that an object called wheel exists it will try to make the design valid by adding the geometry for wheel into the design if necessary.

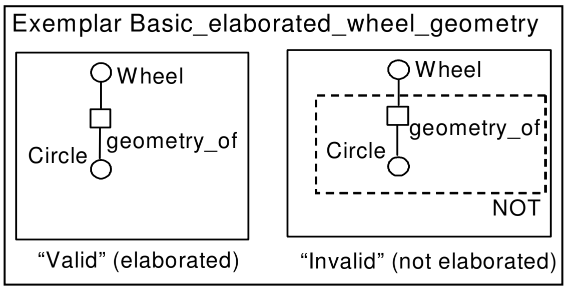


Figure 4. Exemplar for "Wheel Elaboration"

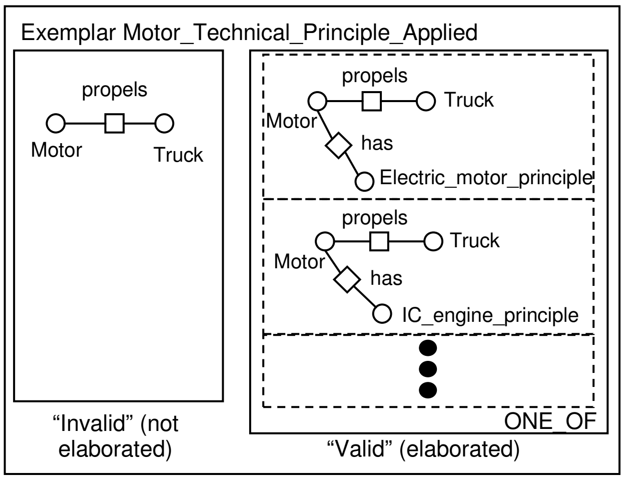


Figure 5. Application of Technical Principle for the "Propels" Function

2.4 Mapping Function Requirements to Exemplars

Requirements like "<u>Assemble</u> and <u>Glue</u> cartons" and "<u>Count</u> number of cartons" are requirements on the function of a design. In systematic design, requirements on function of the Type 3 must map to alternative function structures, which must map to alternative technical principles, which must map to alternative embodiments in order to realize a design [3]. The embodiments ultimately seem to map to specific requirements on what the design must have and must be (i.e., Type 3 requirements ultimately seem to map to Type 1 and 2 requirements).

Let us consider the requirement on function that, "motor must propel truck" for the design of a truck. As shown on the left side of Figure 5, the functional requirement "motor must propel truck" can be expressed as a relationship that must be enforced between the motor object and the truck object. Without elaborating on the meaning of the "propel" concept it is possible to define allowable solution principles, one of which must be chosen if the product is to be realized.[1] These are shown on the right hand side of Figure 5. In turn, each of the solution principles can have another exemplar elaborating it with possible embodiments.

3 ENFORCING DESIGN REQUIREMENTS ONE AT A TIME

The existing generic exemplar algorithm can be applied manually to enforce requirements one at a time [31]. The user first selects the requirement (exemplar) to be applied and then presses the "Find" button to highlight the next occurrence of the *invalid* component of this exemplar in the design. If an occurrence is found, pressing the "Change" button causes the algorithm to remove from the design objects and relationships that are only in the *invalid* component, and add objects and relationships that are only in the *valid* component. Objects and relationships that are *explicit* are added permanently whereas objects and relationships that are *implicit* are added temporarily until the whole system has been solved using a generic propagation-based constraint satisfaction solver that calls domain-specific subsystem solvers [31].[2] The next invalid occurrence is then highlighted automatically and "Change" can be pressed again right away. This process can be repeated until all occurrences have been handled or, alternatively, the "Change All" button can be pressed to change all of the occurrences, one at a time. This procedure can then be repeated with a different exemplar. Though this process is easier than creating objects and relationships manually, and allows standardizing how "requirements" are applied, it is still not automated design synthesis and it is still possible to apply a design change that invalidates a previously applied requirement.

4 APPLYING REQUIREMENTS SIMULTANEOUSLY

Many automated design synthesis systems exist. Automated design synthesis systems produce candidate solutions that are later analyzed, refined and tested automatically[46-48]. One approach to automated design synthesis is known as function-based synthesis, where an abstract functional description for a device is transformed into a structural description that satisfies the functional requirements [49]. In function-based synthesis the design decisions are driven by the function of the final design through the theory of functional reasoning where the design structure consists of a set of geometric objects (features and parts) and a set of functions [50]. Each geometric object facilitates its own set of functions and the geometric objects may be combined to provide more complex functions [51]. To generate the design, one must find or construct a structure that has the necessary functions [52]. Systems based on functional reasoning are limited in that they have difficulty performing reasoning with combinations of objects. Solutions in which the intended function is facilitated by a combination of objects are not identified [50]. Other approaches to design synthesis do not consider the function of the design explicitly but include it implicitly in rules or procedures (e.g. [53]).

Rule-based methods for design synthesis create candidate designs by applying pre-defined operations one after the other to build up designs. The sequence of operations may or may not be pre-defined, but an operation can only be applied if its pre-conditions are met. Directed search algorithms drive designs towards characteristics that best meet the given set of design objectives, by varying the operation sequence. One sub-class of rule-based design methods is grammatical design in which a grammar, consisting of a vocabulary of elements, a set of rules, and an initial structure, transforms structured arrangements of elements into new structures [54]. Design using grammars involves recursively selecting transformation rules and applying them to a candidate structure, until a final structure that satisfies design requirements emerges [55]. In "shape grammars," the vocabulary elements are symbols, that is, shapes made up of points and lines. In "graph grammars," the vocabulary elements are different types of nodes and edges. Grammars have been utilized to create spatial and architectural designs [56, 57], mechanisms [58, 59], and machinable feature models [38] . Other rule-based methods for design synthesis have been presented by [60] and are represented by powerful commercial knowledge-based systems such as ICAD [61] and Knowledge Fusion [62]. These systems provide a general programming language for defining rules; specifying when the rule is

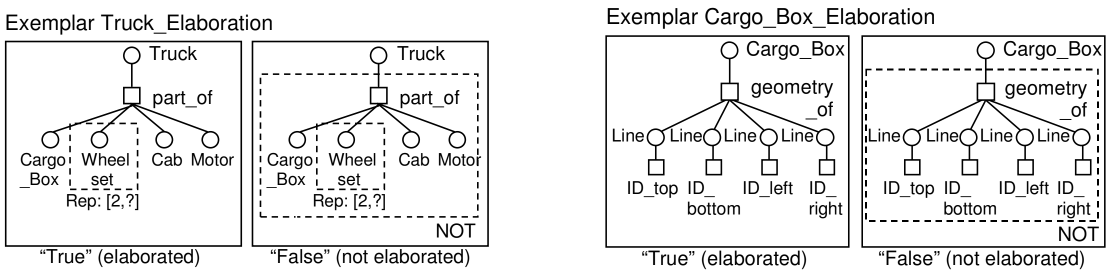
[1] We make an assumption that there are only a finite number of solution principles for any given function and that these can be identified ahead of time in design catalogs. This assumption is justified for practical purposes since if a solution principle is not known it cannot be used.

[2] This is the intended approach. The currently implemented constraint solver only satisfies the *valid* component of the exemplar without considering the rest of the design.

applicable and what action is performed. These systems can create geometry or flag design problems. Rule-based systems have the ability to deal with the physical rather than the abstract form but still they are inherently parametric by nature. However, rule based systems still apply requirements one at a time.

4.1 Proposed Methodology

To generate valid designs, two types of knowledge are required: design refinement knowledge and design validation knowledge. Design refinement knowledge defines the details that correspond to a more abstract object. These details must be created and added to the design by the synthesis methods. This allows the design to be elaborated and to progress from one state to another. Figure 6 shows design exemplars that characterize an elaborated truck and an elaborated cargo box. To enforce these characteristics, the synthesis engine must create the missing objects: Cargo_box, Wheel_set, Cab, and Motor. Note that the Wheel_set is in a repetition block, with an unspecified number of repetitions (except that the minimum is two). Because the number of wheel sets is under-defined, a variety of solutions can result, as shown in Figure 7.



(a) Truck Elaborating Exemplar (b) Cargo Box Elaborating Exemplar

Figure 6. Truck Refinement Exemplars (Simplified)

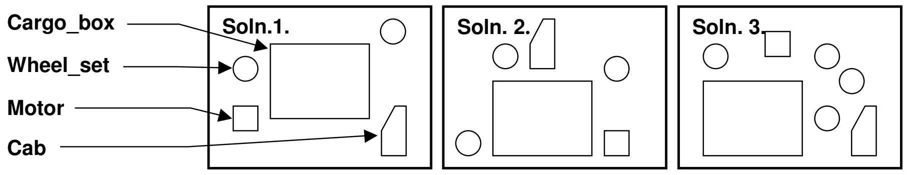
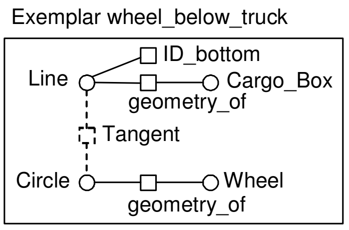


Figure 7. Three Possible Solutions from Truck Refinement



Other validation exemplars
bottom_of_cargo_box_and_motor_lined_up
all_truck_parts_connected
wheels_not_touching
clear_view_in_front_of_cab

Figure 8. Validation Exemplars (Simplified)

Design validation knowledge must be used to control the validity of a design with respect to good design practice, available sizes of material, natural laws governing design behavior, etc. For example, to control the positions of the truck elements from Figure 7, validity requirements (Figure 8) are applied. To solve the implicit "tangent" geometric constraint between design objects, the geometric solving algorithms are used. Applying the validity requirements to Solution 1 from Figure 7, results in several solutions as shown in Figure 9, some of which may be novel or may lead the designer toward a novel design.

4.2 Proposed Algorithm

Two approaches are typically employed in design synthesis systems: a sequenced production system of rules (rule chaining) and unordered application of rules (graph grammars). Both approaches apply sequential design changes as shown in Figure 10(a) which illustrates the application of a series of mapped requirements (R0, R1, R2) one at a time to generate a design solution based on an initial design model (M0).This is typical of a graph grammar approach of applying a sequence of rules, modifying an initial model into a final solution.

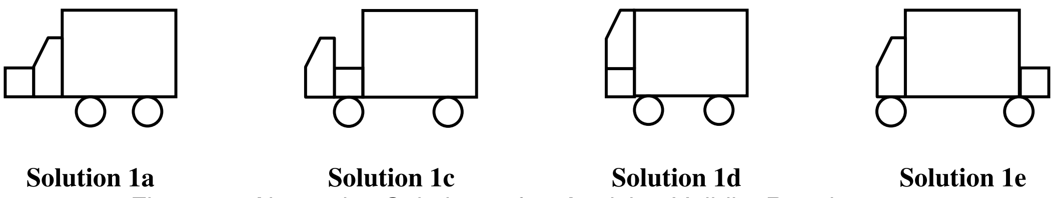


Figure 9. Alternative Solutions after Applying Validity Requirements

Here, the rules may be repeated or reordered. The manual application of the generic exemplar algorithm also employs this approach. Figure 10(b) shows a new generic high-level algorithm we hope to implement which applies design requirements simultaneously and detect incompatibilities in the requirements. Rather than applying discovered changes incrementally as with graph grammars and current knowledge-based systems, the algorithm records a set of proposed changes, compares them with each other for consistency, and then applies them, effectively executing the exemplar algorithm simultaneously for many different requirements. This declarative-variational synthesis approach is given by the following steps:

1. For each requirement exemplar, R_m, test the applicability to the same current model, M_i, and store applicable changes in M_{i-j}.
2. Compare all in M_{i-j} with each other to detect incompatible object additions or deletions. Stop if there is an incompatibility.
3. Apply stored changes to M_i to get M_{i+j}. Perform constraint satisfaction to obtain attribute values. Re-evaluate applicable R_m to see if they are still valid. Stop if there is an incompatibility.
4. Repeat until no more R_m are applicable.

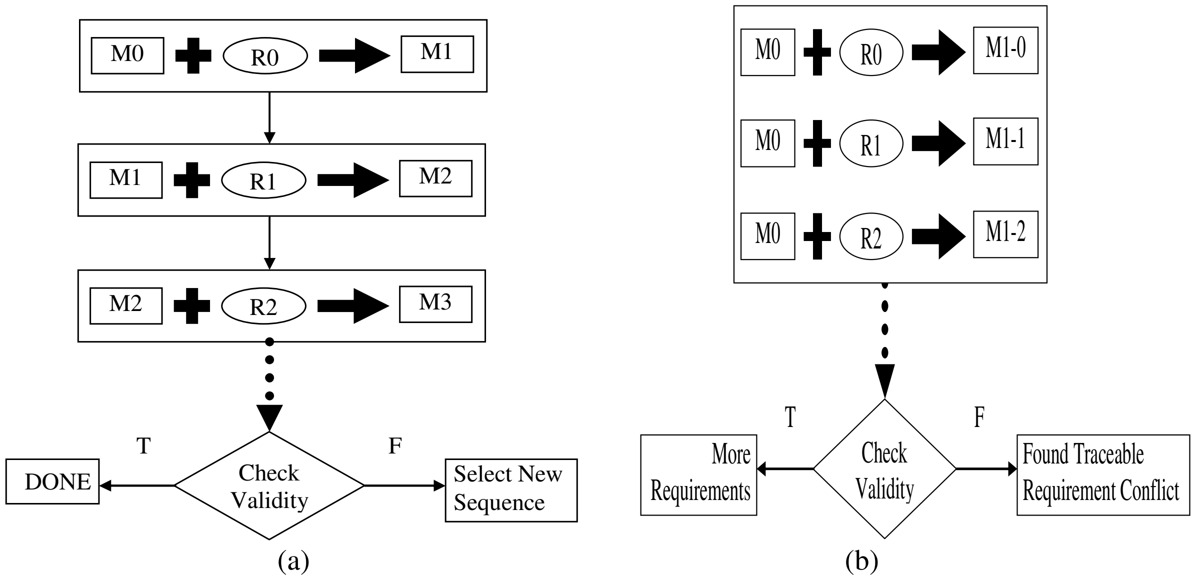


Figure 10. (a) Graph Grammar Synthesis (b) Requirements-Driven Variational Synthesis

5. DISCUSSION

As a result of this research the activities of synthesis, evaluation, and selection in the systematic design process will be supported in a common framework. Once prototype software has been created we expect that novel solutions will arise both automatically and through designer initiative:

1. Automatically during object synthesis, as different configurations are created by the computer,
2. Automatically in constraint solving, when choosing from multiple solutions, or choosing solution values from a range of possible values,
3. Through the designer, as design requirements are added, removed or modified, and
4. Through the designer changing assumptions or restrictions in the engineering knowledge,

5. Through ideas that might be sparked in the user's mind when seeing an odd configuration generated by the computer.

The paradigm of requirements driven computations is expected to be a vital part of next-generation CAD systems. Besides promoting innovative designs, the use of knowledge-bases will promote safer, less costly products with fewer design errors. The use of requirements as a basis for collaboration will also reduce instances in which design modifications detrimentally impact the intent of earlier design decisions.

ACKNOWLEDGEMENT

The authors gratefully acknowledge support for this research from the National Science Foundation (grant CMMI 0638393).

REFERENCES

[1] Bettig, B., Bapat, V. and Bharadwaj, B. Limitations of parametric operators for supporting systematic design. *CDROM Proceedings DECT-2005, ASME International Design Engineering Technical Conferences, September 24--28, Long Beach, California, USA*, 2005.
[2] Chung, J.C.H. and Schussel, M.D. Technical Evaluation of Variational and Parametric Design. *Computers in Engineering*, 1990, 1, 289-298.
[3] Pahl, G. and Beitz, W. *Engineering Design : A systematic Approach*. (Springer, 1995).
[4] Ullman, D.G. *The Mechanical Design Process*. (McGraw-Hill, 2002).
[5] Cross, N. *Engineering Design Methods*. (John Wiley &Sons New York, 1989).
[6] Dixon, J.R. and Poli, C. *Engineering Design and Design for Manufacturing*. (Field Stone Publishers, Conway, Massachusetts, 2000).
[7] Maiden, N. What has requirements research ever done for us? *Ieee Software*, 2005, 22(4), 104-105.
[8] Wieringa, R., Maiden, N., Mead, N. and Rolland, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 2006, 11(1), 102-107.
[9] Bashar, N. and Steve, E. Requirements engineering: a roadmap. *Proceedings of the Conference on The Future of Software Engineering* (ACM Press, Limerick, Ireland, 2000).
[10] Jacobs, G. and Ip, B. Establishing user requirements: incorporating gamer preferences into interactive games design. *Design Studies*, 2005, 26(3), 243-255.
[11] Hammori, M., Herbst, J. and Kleiner, N. Interactive workflow mining - requirements, concepts and implementation. *Data & Knowledge Engineering*, 2006, 56(1), 41-63.
[12] Dag, J.N.O., Thelin, T. and Regnell, B. An experiment on linguistic tool support for consolidation of requirements from multiple sources in market-driven product development. *Empirical Software Engineering*, 2006, 11(2), 303-329.
[13] Arthur, J.D. and Groner, M.K. An operational model for structuring the requirements generation process. *Requirements Engineering*, 2005, 10(1), 45-62.
[14] Gorschek, T. and Wohlin, C. Requirements abstraction model. *Requirements Engineering*, 2006, 11(1), 79-101.
[15] Gnesi, S., Lami, G., Trentanni, G., Fabbrini, F. and Fusani, M. An automatic tool for the analysis of natural language requirements. *Computer Systems Science and Engineering*, 2005, 20(1), 53-62.
[16] Katasonov, A. and Sakkinen, M. Requirements quality control: a unifying framework. *Requirements Engineering*, 2006, 11(1), 42-57.
[17] Hayes, J.H., Dekhtyar, A. and Sundaram, S.K. Advancing candidate link generation for requirements tracing: The study of methods. *Ieee Transactions on Software Engineering*, 2006, 32(1), 4-19.
[18] Jane, C.-H. Toward improved traceability of non-functional requirements. *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering* (ACM Press, Long Beach, California, 2005).
[19] Balasubramaniam, R., Curtis, S., Timothy, P. and Michael, E. Requirements traceability: Theory and practice. *Ann. Softw. Eng.*, 1997, 3, 397-415.
[20] Yang, H.S., Kim, M., Park, S.Y. and Sugumaran, V. A process and tool support for managing activity and resource conflicts based on requirements classification. *Natural Language*

Processing and Information Systems, Proceedings, 2005, 3513, 114-125.
[21] William, N.R., Suzanne, D.P. and Vecheslav, V. Requirements interaction management. *ACM Comput. Surv.*, 2003, 35(2), 132-190.
[22] Damian, D., Chisan, J., Vaidyanathasamy, L. and Pal, Y. Requirements engineering and downstream software development: Findings from a case study. *Empirical Software Engineering*, 2005, 10(3), 255-283.
[23] Jiau, H.C. and Yu, D.F. Comments on "Automatic analysis of consistency between requirements and designs". *Ieee Transactions on Software Engineering*, 2006, 32(4), 279-280.
[24] Feather, M.S., Cornford, S.L., Hicks, K.A. and Johnson, K.R. Applications of tool support for risk-informed requirements reasoning. *Computer Systems Science and Engineering*, 2005, 20(1), 5-17.
[25] TC184/SC4, I. ISO 10303-233, STEP Part 233, Application protocol: Systems engineering data representation (International Organization for Standardization, 2005).
[26] Telelogic. Telelogic DOORS - Requirements managements for co-located teams. (Telelogic AB, 2006).
[27] UGS. UGS: Teamcenter solutions by product : Systems Engineering. (UGS Corp., 2006).
[28] Almefelt, L., Berglund, F., Nilsson, P. and Malmqvist, J. Requirements management in practice: findings from an empirical study in the automotive industry. *Research in Engineering Design*, 2006, 17(3), 113-134.
[29] Schachinger, P. and Johannesson, H.L. Computer modelling of design specifications. *Journal of Engineering Design*, 2000, 11(4), 317-329.
[30] Balmelli, L. and Moore, A. Requirements Modeling for System Engineering Using SYSML, The Systems Modeling Language. *Proceedings of DETC, Computers & Information in Engineering Conference*, 2004.
[31] Bettig, B. A Graph-Based Geometric Problem Solving System for Mechanical Design and Manufacturing. *Mechanical and Aerospace Engineering* (Arizona State University, Tempe, AZ, 1999).
[32] Bettig, B. and Shah, J. Derivation of a standard set of geometric constraints for parametric modeling and data exchange. *Computer-Aided Design*, 2001, 33(1), 17-33.
[33] Bettig, B. and Shah, J. Solution selectors: A user-oriented answer to the multiple solution problem in constraint solving. *Journal of Mechanical Design*, 2003, 125(3), 443-451.
[34] Summers, J.D., Shah, J.J. and Bettig, B. The Design Exemplar: A New Data Structure for Embodiment Design Automation. *Journal of Mechanical Design*, 2004, 126(5), 775-787.
[35] Anandan, S., Bettig, B., Summers, J., Maier, J. and Bapat, V. Semantics in Engineering Design. *International Conference on Engineering Design*Paris, France, 2007).
[36] Joshi, S. Feature Recognition and Geometric Reasoning for some Process Planning Activities. In Wozny, M.J., Turner, J.U. and Preiss, K., eds. *Geometric Modeling for Product Engineering*, pp. 363-384 (Elsevier Science Publishers, North-Holland, 1990).
[37] Schmidt, L.C. and Cagan, J. GGREADA: A graph grammar-based machine design algorithm. *Research in Engineering Design-Theory Applications and Concurrent Engineering*, 1997, 9(4), 195-213.
[38] Fu, Z., Depennington, A. and Saia, A. A Graph Grammar Approach to Feature Representation and Transformation. *International Journal of Computer Integrated Manufacturing*, 1993, 6(1-2), 137-151.
[39] Summers, J.D. Development of a Domain and Solver Independent Method for Mechanical Engineering Embodiment Design. *Mechanical and Aerospace Engineering*, p. 286 (Arizona State University, Tempe, 2004).
[40] Fu, Z. and Depennington, A. Geometric Reasoning Based on Graph Grammar Parsing. *Journal of Mechanical Design*, 1994, 116(3), 763-769.
[41] Bettig, B., Shah, J.J. and Summers, J.D. Geometric Exemplars: A Bridge Between AI and CAD. In Wozny, ed. *From Knowledge Intensive CAD to Knowledge Intensive Engineering*, pp. 13-25 (Kluwer Academic Press, the Netherlands, 2001).
[42] Summers, J.D., Bettig, B. and Shah, J.J. The design exemplar: A new data structure for embodiment design automation. *Journal of Mechanical Design*, 2004, 126(5), 775-787.
[43] Summers, J.D., Lacroix, Z. and Shah, J.J. Case-Based Design Facilitated by the Design Exemplar. In Gero, J., ed. *International Conference on Artificial Intelligence in Design*, pp.

pp. 453-476 (Kluwer Academic Press, Cambridge, UK, 2002).

[44] Summers, J.D. and Shah, J.J. The Design Exemplar: A New Data Structure for Design Automation. (Design Automation Lab, Arizona State University, Tempe, AZ, 2001).

[45] Summers, J.D. and Shah, J.J. Exemplar Networks: Extensions of the Design Exemplar. In ASME, ed. *Design Engineering Technical Conferences*, pp. CIE-57786 (Computers in Engineering, Salt Lake City, UT, 2004).

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

[47] Bryant, C., McAdams, D., Stone, R., Tolga, K. and Campbell, M. A Computational Technique for Concept Generation. *Design Engineering Technical Conferences*, pp. DTM-85323 (ASME, Long Beach, CA, 2005).

[48] Rajagopalan, V., Bryant, C., Johnson, J., McAdams, D., Stone, R., Kurtoglu, T. and Campbell, M. Creation of Assembly Models to Support Automated Concept Generation. *Design Engineering Technical Conferences*, pp. DTM-85302 (ASME, Long Beach, CA, 2005).

[49] Chandrasekaran, B. Design Problem Solving: A Task Analysis. *AI Magazine*, 1990, 11(4), 59-71.

[50] Ligman, D. DEIMOS: A Functional Paradigm for Mechanical Design. *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 773-780Charleston, SC, 1990).

[51] Freeman, P. and Newell, A. A Model for Functional Reasoning in Design. *Second International Joint Conference on Artificial Intelligence*1971).

[52] Kurtoglu, T., Campbell, M., Joah, G., Bryant, C., Stone, R. and McAdams, D. Capturing Empirically Derived Design Knowledge for Creating Conceptual Design Configurations. *Design Engineering Technical Conferences*, pp. DTM-84405 (ASME, Long Beach, CA, 2005).

[53] Fitzhorn, P. Engineering Design as a Computable Function. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 1994, 8, 35-44.

[54] Gips, J. and Stiny, G. Production Systems and Grammars: A Uniform Characterization. *Environment and Planning*, 1980, 7, 399-408.

[55] Brown, K. Grammatical Design. *IEEE Intelligent Systems*, 1997, 12(2), 27-33.

[56] Gips, J. and Stiny, G. Shape Grammar and Generative Specification of Painting and Sculpture. *Information Processing*, 1972, 71, 1460-1465.

[57] Agarwal, M. and Cagan, J. On the Use of Shape Grammars as Expert Systems for Geometry-Based Engineering Design. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing*, 2000, 14, 431-439.

[58] Schmidt, L. and Cagan, J. Optimal Configuration Design: An Integrated Approach Using Grammars. *Journal of Mechanical Design*, 1998, 120(1), 2-9.

[59] Schmidt, L., Shetty, H. and Chase, S. A Graph Grammar Approach to Mechanism Synthesis. *Journal of Mechanical Design*, 2000, 122, 371-376.

[60] Dixon, J. Knowledge-Based Systems for Design. *Transactions of ASME, Special 50th Anniversary Design Issue*, 1995, 117(11).

[61] KTI. Knowledge Technologies International: KTI Home Page. (Knowledge Technologies International, 2004).

[62] UGS. UGS - Product Life Cycle Management Solutions. (UGS, 2005).

Contact: Bernhard Bettig
West Virginia University Institute of Technology
Department of Mechanical Engineering
405 Fayette Pike,
Montgomery, WV 25136
USA
Tel: Int +1 304 442 3289
E-mail: Bernhard.Bettig@mail.wvu.edu