VariMod: A Structured Approach to Variability in 3D Modelling

Jef Jacobs
Jens Nicolay
Wolfgang De Meuter
Software Languages Lab
Vrije Universiteit Brussel
Brussels, Belgium

ABSTRACT

Today's manufacturing industry is confronted with an increasing demand for product variability that stems from product customisation needs and the engineering process. Different customer demands and the mass-customisation of physical products require designing multiple variants of products, and additional requirements may be introduced when the product reaches subsequent stages (simulation, manufacturing, assembly, ...) in its engineering process.

The state-of-the-art 3D modelling software deals with variability in a mostly ad-hoc fashion. Designing products typically involves creating digital 3D models using Computer-Aided Design (CAD) software, and implementing variability requires duplication of entire models or parts thereof that then require changes without any identification of or distinction between the different requirements that caused them. Parametric CAD approaches do enable designing 3D models that contain modifiable parameters, but designers must still ensure that the 3D model with updated parameter values satisfies all requirements. It is therefore difficult or impossible with current approaches and tools to design variants of products in a structured and efficient manner.

In this work, we present VariMod, a 3D modelling approach that distinguishes between invariant requirements that each variant of a 3D model must satisfy, and variant-specific requirements that individual variants must satisfy. Hereby, VariMod enables the specification of 'generic' 3D models that satisfy invariant requirements, of which the parameter values can be optimised so that they also satisfy variant-specific requirements. To this end, VariMod represents both types of requirements as bidirectional constraints that are solved to find optimal parameter values that satisfy all constraints. VariMod features a constraint-solving process that aims to minimise the modifications made to parameter values when optimising a 3D model, thereby preventing unexpected modifications to the 3D model. We use PrintTalk, a programmatic CAD language for parametric 3D modelling, as a vehicle for implementing and validating VariMod by demonstrating how it can be used for designing variants of 3D models in a structured and efficient manner.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; • **Computing methodologies → Modeling methodologies**; *Shape modeling*.

KEYWORDS

Variational Design, Non-Functional Requirements, 3D Modelling, Parametric CAD, Constraints, PrintTalk, DFX

ACM Reference Format:
Jef Jacobs, Jens Nicolay, and Wolfgang De Meuter. 2023. VariMod: A Structured Approach to Variability in 3D Modelling. In *27th ACM International Systems and Software Product Line Conference - Volume A (SPLC '23), August 28-September 1, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3579027.3608987

SPLC '23, August 28-September 1, 2023, Tokyo, Japan
2023. ACM ISBN 979-8-4007-0091-0/23/08…$15.00
https://doi.org/10.1145/3579027.3608987

1 INTRODUCTION

1.1 Context

Today's manufacturing industry is confronted with an increasing demand for product variability [20]. Generally speaking, demand for variability stems from product customisation needs [14] and the manufacturing process [17]. We illustrate the sources of variability by describing the typical engineering process for manufacturing a chair by means of 3D printing (fig. 1).

In the first step, a chair is designed using *Computer-Aided Design* (CAD) software. CAD software enables the creation and modification of digital 3D models that serve as the 'blueprint' for an actual product that must be manufactured and assembled in later stages of the engineering process.

In this step, different customer demands and the mass-customisation of physical products require designing multiple variants of products. For example, multiple variants of a chair may be designed with varying dimensions for the legs or seat. As another example, multiple variations of a phone case are required to ensure compatibility with different phone models.

After the initial design step, additional requirements may be introduced when the product reaches subsequent stages in its engineering process. These requirements could be regarded as *non-functional requirements* (NFRs) [19], as they do not describe the 'functionality' of the product, but rather the conditions for ensuring that the product is manufacturable and embeddable in its target domain. A typical engineering process therefore also introduces variation, because the design of products must be modified when they do not satisfy one or more NFRs [9].

Continuing our example, after obtaining a digital 3D model representing a chair from the first step, this design is compiled into the form of an STL file. STL is a common file format that represents the surface of a 3D model as a triangulated mesh [11]. This STL file is then imported into simulation software for verifying that the chair is sufficiently strong to support a certain weight. If this is not the case, the design of the chair must be modified, which requires

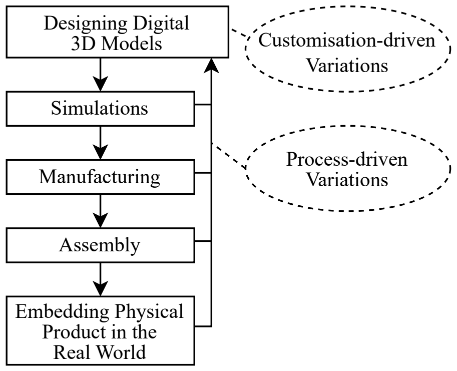
Designing Digital 3D Models
Customisation-driven Variations
Simulations
Manufacturing
Process-driven Variations
Assembly
Embedding Physical Product in the Real World

Figure 1: Typical engineering process of physical products.

going back to the initial design step. Otherwise, the chair can be manufactured in a next step.

If, as in our example, the chair is manufactured by means of 3D printing, its parts must meet the manufacturing constraints of a specific 3D printer, such as maximum print dimensions or maximum time required to print the parts. When not all parts satisfy these requirements, the chair's design must be modified, again moving back to the initial design step in the process. Otherwise, the manufactured parts can be assembled into a complete chair in the assembly step. However, here too the design may require adaptation when for example the assembly time is too long because of the number of parts, or the parts do not fit together due to imprecisions introduced throughout the process.

1.2 Problem

The previous example illustrated that variability may be introduced at many points during the design and engineering process of physical products. Creating variants of a product involves modifying its 3D model in CAD software [8], returning to the initial design step in the engineering process even when the need for the adaption is driven by later stages in the process.

State-of-the-art CAD software enables the design of *parametric* 3D models for dealing with variability. Parametric modelling enables the customisation of 3D models by modifying parameter values such as dimension and position. However, some modifications cannot be realised by modifying parameter values, or it would be cumbersome to do so. For example, it is not trivial to change aspects of a shape that do not directly correspond to adjustable parameters. In this case, designers are limited to creating variants by duplicating an original design and modifying each version independently. This limitation is partly solved by *programmatic CAD* (PCAD) languages such as OpenSCAD[1] and PrintTalk [15]. PCAD languages enable modifying 3D models by changing the program text by which the 3D model is constructed. Additionally, these languages enable a shape to be instantiated an arbitrary number of times, and each instance can be modified independently within a single program through (again) parameter changes. The programmatic approach to 3D modelling enables the application of *Software Product Line* (SPL) techniques for promoting the variability and reusability of 3D models. However, current PCAD languages lack an integration of these techniques [1].

- **No support for constraints** that express relations between parameters. These constraints are required for preventing errors resulting from incompatible parameter values. For example, constraints can express that the outer diameter of a ring must be larger than the inner diameter.
- **No composition of requirements** that are introduced at various points during the engineering process in such a way that additional requirements can be modelled 'on top of' an existing model and its requirements (which itself may represent a variant) without the need for changing the existing part.
- **No hierarchy between potentially incompatible requirements**, making it impossible to generate 3D models that satisfy 'more important' requirements but omit incompatible 'less important' requirements.

In summary, the state-of-the-art 3D modelling software deals with variability in a mostly ad-hoc fashion. Implementing variability requires duplication of entire models or parts thereof that then require changes without any identification of or distinction between the different requirements that caused them. It is therefore difficult or impossible to design variants of products in a structured manner.

1.3 Approach

In this paper, we present VariMod, an approach for designing variations of digital 3D models using parametric CAD software. To enable straightforward variational design, VariMod distinguishes between two categories of requirements:

- *Invariant Requirements* (IRs) for ensuring that each variant results in a valid 3D model. For example, each variant of our chair must have legs that are attached to the bottom of the seat, and the distance between them is constrained by the dimensions of the seat.
- *Variant-specific Requirements* (VRs) for ensuring that a variant either satisfies customer demands, or is manufacturable and compatible with subsequent steps of the engineering process. For example, a chair's legs must have minimal dimensions, determined by the weight that the chair is required to hold and the material out of which the chair will be manufactured.

VariMod improves upon the state of the art in three important ways.

- VariMod enables the *separation* of IRs and VRs of 3D models so that variational designs can be modelled efficiently and in a structured manner. A model that satisfies IRs can be instantiated multiple times, with each instance optimised so that it satisfies different VRs for meeting different customer demands or to be suitable for different means of manufacturing.
- In VariMod, both IRs and VRs are represented as *bidirectional* constraints. Instead of only testing whether they are satisfied, a constraint solver can help designers in finding optimal parameter values that satisfy bidirectional constraints.

[1]https://openscad.org/

For example, VRs can be used for optimising a hinge's dimensions to respect a 3D printer's size limitations while maximising its strength and load capacity.

- VariMod allows the assignment of *criticality levels* to constraints that express requirements, so that preferential requirements can be expressed. VariMod optimises the parameter values of a 3D model when preferential requirements contradict each other by prioritising requirements with higher criticality levels. For example, when requirements that ensure manufacturability are considered more important than requirements that are purely aesthetic, a higher criticality level can be assigned to the former such that VariMod can generate a manufacturable 3D model that satisfies as many of the aesthetic requirements as possible.

VariMod introduces *Invariant Constraints* for modelling invariant requirements (IRs) that ensure the validity of each variant of 3D models, and *Variant-Specific Constraints* that express variant-specific requirements (VRs). Invariant constraints may leave the model 'underconstrained', meaning that multiple values can be assigned to a parameter without invalidating any of the constraints on the parameter's value. VariMod uses variant-specific constraints for optimising these parameter values without invalidating the invariant constraints. In doing so, VariMod aims to modify the 3D model as little as possible in order to prevent unnecessary changes of parameter values that modify the 3D model in an unexpected manner. When parameter values satisfy all variant-specific constraints, they are left unchanged. Otherwise, VariMod attempts to find new parameter values that satisfy both invariant constraints and variant-specific constraints. VariMod's strategy for combining invariant constraints and variant-specific constraints is discussed in section 3.

We use the PrintTalk [15] programmatic CAD language as a means for implementing and validating our approach. PrintTalk textually describes 3D models using both imperative statements and constraints. The language features a powerful constraint mechanism that enables a clean integration of VariMod. We extend PrintTalk with a new component through which variant-specific constraints can be asserted. The implementation hereof is available in our software artefact at https://doi.org/10.5281/zenodo.8046217.

1.4 Overview

We discuss the state of the art and related work, and further motivate VariMod in section 2. A detailed description of VariMod is given in section 3. Section 4 describes how VariMod can be integrated into existing PCAD languages. VariMod is validated in section 5.

2 BACKGROUND AND STATE OF THE ART

We review the landscape of 3D modelling software that is used for designing digital 3D models and we describe the limitations of the state-of-the-art methods for designing variants of 3D models. The effectiveness of constraints for modelling requirements is also discussed in this section. Additionally, we provide an overview of existing tools for modelling the Non-Functional Requirements of 3D models and illustrate the limitations of these tools regarding variational design.

2.1 3D Modelling Software

Digital 3D models are commonly constructed using *Computer-aided Design* (CAD) software. While this software is typically GUI-based, 3D models can also be described programmatically through programmatic CAD languages such as OpenSCAD[2] and PrintTalk [15]. CAD software typically implements one of two main 3D modelling techniques.

- *Direct Modelling* can be thought of as sculpting a 3D model. Through a GUI, 3D models are manipulated by clicking and dragging. The resulting 3D model is free of parameters and constraints that express relations between parts of the 3D model. Direct modelling does not capture information on how the design was achieved. The lack of parameters and constraints makes direct modelling less suitable for the systematic designing of modifiable and reusable 3D models. Examples of direct modelling CAD software are PTC Creo Direct[3] and Shapr3D[4].
- *Parametric Modelling* forms a more systematic approach to design. Parametric 3D models are represented by a history tree. The nodes of this tree represent parameterised components such as primitive 3D shapes (e.g., cubes, spheres, and cylinders), and operations for modifying parameter values or combining components by means of set-theoretical operations such as union and difference.
  Some parametric modelling software supports *constraints* for declaratively specifying relations between components. In this case, 3D models can be modified through parameter changes, and requirements can be modelled as constraints on these parameters. Reusing 3D models is achieved by copying (parts of) the history-tree. Examples of parametric CAD software are FreeCAD[5], Siemens NX[6] and Solidworks[7].

This paper focuses on parametric modelling, as it provides a systematic way for adapting 3D models through parameter changes that can be conducted either manually or by an underlying constraint solver with the goal of satisfying requirements expressed as constraints. We now briefly discuss constraints in visual and programmatic parametric CAD approaches.

Constraints in Visual Parametric CAD Tools. Visual CAD software usually includes a limited set of constraints that can be asserted manually by the designer. Examples of constraints that are typically supported are perpendicularity, parallelism, and tangency. Existing visual CAD approaches do not distinguish constraints that express invariant constraints from variant-specific constraints, making it cumbersome to determine whether a constraint models one or the other.

Constraints in Programmatic CAD Languages. While many programmatic CAD languages and libraries that implement parametric modelling exist, most of them do not include support for constraints. We discuss the programmatic CAD approaches that do offer explicit support for constraints: OpenSCAD, CadQuery, and PrintTalk.

[2]https://openscad.org/
[3]https://www.ptc.com/en/products/creo/elements-direct
[4]https://www.shapr3d.com/
[5]https://www.freecad.org/
[6]https://www.plm.automation.siemens.com/global/en/products/nx/
[7]https://www.solidworks.com/

OpenSCAD[8] is one of the most popular CAD languages. Following the functional programming paradigm, the language is limited to `assert`-statements for constraining parameter values. There is no underlying constraint solver that can assign values to variables in order to satisfy constraints. When an assertion fails, the corresponding constraint is not satisfied and an error is raised.

CadQuery[9] is a Python library for parametric CAD that implements a limited set of geometric constraints. These constraints are bidirectional, as parameter values can be modified in order to satisfy constraints. An error is raised when not all constraints can be satisfied. However, because Python does not natively support constraint programming, the constraint solver must be invoked manually.

PrintTalk[15] is a programmatic CAD language for modelling 3D printable objects. Constraints represent requirements through declarative relations such as relative positions that must hold between elements of a design. An underlying constraint solver solves these constraints, and assign values to the constraint variables in a design. The language serves as a base for implementing and validating VariMod, and is further discussed in section 4.

2.2 SPL Practices Applied to 3D Modelling

The variability and reusability of software artefacts are prominent subjects of the *Software Product Line* (SPL) research field, but are also relevant to the field of physical product design. 3D modelling is mainly a software problem, and typically involves CAD software. While state-of-the-art 3D modelling tools lack support for SPL-like techniques for improving the variability and reusability of 3D models [1], the applicability of these techniques to the field of 3D modelling has been demonstrated by earlier research, which we briefly discuss next.

The work by Amand et al. [2] introduces a learning-based approach for detecting invalid parameter values of 3D models constructed by OpenSCAD scripts, and for deriving constraints on these parameters. The derived constraints improve the variability of 3D models by preventing errors resulting from incompatible parameter values. However, OpenSCAD and other PCAD languages provide no features for integrating these constraints into existing scripts. VariMod overcomes this limitation by featuring invariant constraints through which relations between the parameters of 3D models can be expressed.

The work by Hofmann et al. [13] presents a framework for managing 'design expectations' of 3D models with the goal of facilitating model reuse. The framework distinguishes between *assertions* and and *integrators.* Assertions check whether 'design expectations' are met, while integrators modify 3D models so that they meet expectations. VariMod provides similar functionality, as invariant constraints can specify conditions that each variant of the 3D model must meet, and an integrated constraint solver can determine parameter values that meet these conditions.

2.3 Variational Design of 3D Models

Earlier work on *variational modelling* explored the idea of using constraints for modelling variational geometry, so that the geometry of objects can be modified in a straightforward manner. The work by Lin et al. [18] describes a system through which objects can be modified by altering the constraints that describe their geometry. Later work by Kimura et al. [16] describes a framework that captures design requirements from different stages of the design process in the form of constraints. Modifications to an object's design are made by incrementally adding the constraints associated with each step in the design process to the set of constraints that describe the object's geometry. VariMod implements a similar strategy by distinguishing between invariant and variant-specific constraints to model an object's geometry.

2.4 Non-Functional Requirements in Product Design

In a traditional engineering process (fig. 1), only functional requirements are considered during the design phase of a product [17]. As products reach further stages in the development process, design flaws may arise. For example, products may be found to be impossible or expensive to manufacture. To account for this, non-functional requirements (NFRs) should also be considered during the initial design stage [19]. Taking NFRs into account during the design of digital 3D models using CAD is the topic of the active research on what is called the *Design For X* (DFX) methodology.

DFX. The goal of Design For X (DFX) [12] is to enable the design of 3D models to be optimised for a certain criterion X. The DFX methodology takes NFRs into account during the design stage with the goal of reducing the need for modifications when subsequent stages of the engineering process are reached.

Existing applications of DFX include 'Design For Materials' [4], 'Design For Manufacturing' [3], and 'Design For Assembly' [5].

DFX software exists both as standalone tools[10], and as plugins for CAD software[11]. Existing DFX tools are capable of analysing 3D models and making recommendations on how the models can be manually optimised in order to satisfy the DFX requirements. Yet, existing DFX tools are not capable of automatically optimising 3D models for compliance with DFX requirements. Instead, after analysis, the DFX tools only report which rules are satisfied and which are not. Designers still must manually modify 3D models and re-run the analysis, and this in an iterative manner until the 3D models are compliant with DFX requirements. This approach provides no distinction between the initial design and DFX optimisations, hampering the systematic specification of multiple versions of a design, each optimised for other DFX requirements, such as manufacturability on different machines.

DFX tools typically allow designers to assign a *criticality level* to each rule, so that the designer can prioritise more critical rules when trying to find the most optimal design through iterative modifications. As current DFX tools are not capable of automatically making modifications, these criticality levels are solely used for informing the designer, and it is up to the designer to ensure that rules with higher criticality levels are prioritised when modifying 3D models.

[8]https://openscad.org/
[9]https://github.com/CadQuery/cadquery
[10]https://www.dfma.com/
[11]https://www.dfmpro.com/

2.4.1 *Use Case: NFRs for 3D Printing.*

The work by Asadollahi-Yazdi et al. explores the DFX approach for *Additive Manufacturing* (AM), more commonly referred to as 3D printing. 3D printing is becoming a popular means of manufacturing and rapid prototyping. Yet, not all digital 3D models can be efficiently manufactured by means of layer-by-layer 3D printing. An example of a manufacturing constraint for layer-by-layer 3D printing is that each layer to be printed must be sufficiently supported by the layer underneath. Layers that are insufficiently supported by the 3D model itself require additional support structures that must be generated and printed to ensure the correct manufacturing of the 3D model.

The DFX approach can be applied to optimise an initial 3D model for satisfying non-functional requirements such as manufacturing constraints during the design stage [3]. Each type of 3D printer has different characteristics that influence its capability of printing unsupported parts. In order to make optimal use of each printer, the initial design must be optimised for multiple printers. With current DFX tools, this requires an independent instance of the original design per printer.

3 APPROACH

We present VariMod, a structured and efficient approach to variability in 3D modelling. Our approach enables instantiating multiple variants of a 3D model such that each variant satisfies different requirements. These requirements are expressed as constraints on the 3D model's parameter values, and VariMod relies on a constraint solving mechanism that combines invariant constraints that ensure the validity of all variants, together with variant-specific constraints for optimising parameter values. Additionally, criticality levels can be assigned to constraints, so that constraints with a lower criticality can be omitted in favour of constraints with a higher criticality when modifying 3D models.

3.1 Invariant vs. Variant-Specific Constraints

VariMod distinguishes between constraints for expressing invariant requirements (IRs) and variant-specific requirements (VRs). This distinction promotes the reusability of designs, as it facilitates designing multiple variations of a 3D model that each satisfy different VRs.

(1) **Invariant constraints** are part of a 3D model's design, and are meant for expressing IRs. Parameter values can be *derived from* and *limited by* invariant constraints. This implies that invariant constraints serve two purposes.
   (a) *Check* the validity of the values of object (shape, ...) parameters with respect to the conditions specified by the constraint. For example, invariant constraints can be used for expressing that shapes cannot have negative dimensions.
   (b) *Find* parameter values that satisfy the constraints placed on the parameters. To this end, an underlying constraint solver is used, alleviating the need for designers to provide a specific value for each parameter.

(2) **Variant-specific constraints** can be asserted onto a 3D model's parameters externally to the 3D model's design, and are meant for expressing VRs. Examples include customer demands, and requirements originating from the engineering process downstream from the design step. Variant-specific constraints also serve two purposes.
- *Check* whether a 3D model satisfies requirements implied by external factors such as manufacturability and other non-functional requirements. For example, variant-specific constraints can be used for checking whether a 3D model's dimensions do not exceed the maximum printable size of a 3D printer.
- *Optimise* a 3D model by further constraining its parameters. For example, variant-specific constraints can be used for expressing the maximum printable size of a 3D printer, so that the 3D model's parameter values can be optimised for making it compliant with these maximum dimensions.

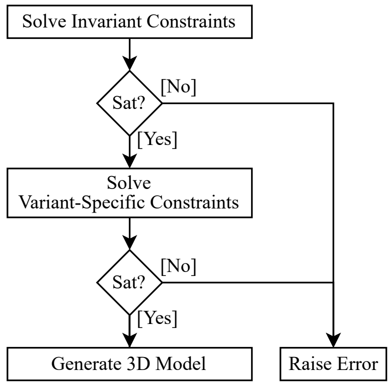

Figure 2: Two-round constraint solving process.

3.2 Preferential Constraints

Although some constraints are *required* for generating a valid 3D model that satisfies customer demands and that is compatible with the engineering process (simulation, manufacturing, assembly, ...), other constraints may be *preferential* instead. For example, requirements that ensure actual manufacturability are considered to be more important than requirements that are purely aesthetic. To account for this, VariMod distinguishes between required constraints and preferential constraints.

Furthermore, criticality levels can be assigned to preferential constraints, so that designers can introduce a hierarchy between constraints. This enables VariMod to optimise the parameter values of a 3D model when preferential constraints contradict each other, and this by prioritising constraints with higher criticality levels while omitting incompatible constraints with lower criticality levels. VariMod still raises an error when a required constraint can not be satisfied.

3.3 Constraint Solving Process

To generate 3D models that satisfy both invariant and variant-specific constraints, VariMod features a two-round constraint solving process, as illustrated in fig. 2.

In the first round, only invariant constraints are considered. The invariant constraints are sent to the solver, which attempts to find a set of satisfiable constraints using the process described in 4.3. When the solver finds a solution that satisfies the invariant

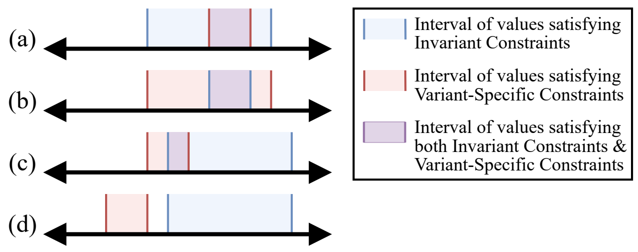
(a)
(b)
(c)
(d)
Interval of values satisfying Invariant Constraints
Interval of values satisfying Variant-Specific Constraints
Interval of values satisfying both Invariant Constraints & Variant-Specific Constraints

Figure 3: Possible scenarios when combining invariant constraints and variant-specific constraints.

constraints, VariMod proceeds to the second round. Otherwise, an error is raised.

In the second round, the variant-specific constraints, which can consist of both required and preferential constraints, are combined with the set of invariant constraints that were satisfied in the first round. This ensures that variant-specific constraints can only be added to the set of satisfied constraints when they do not invalidate previously satisfied invariant constraints. Preferential constraints are considered in decreasing order of criticality levels.

To ensure that every design can be exported to a valid 3D model file, VariMod requires parameters that are constrained by variant-specific constraints to also be constrained by invariant constraints. As such, 3D models cannot contain unbound parameters after the invariant constraints are solved. When a parameter is constrained by both invariant *and* variant-specific constraints, multiple scenarios are possible when both types of constraints are combined, depending on the actual constraints involved. Figure 3 visualises these different scenarios, which can be categorised as follows.

- **Invariant constraints and variant-specific constraints are (partially) compatible.** A set of values that satisfy both invariant and variant-specific constraints exists when variant-specific constraints are more general than invariant constraints (scenario a), when variant-specific constraints are equally or more specific than invariant constraints (scenario b), or when the values that satisfy the invariant constraints partially overlap with the values that satisfy the variant-specific constraints (scenario c).
  If the value that was determined from invariant constraints happens to also satisfy the variant-specific constraints, it remains unchanged in order to not unnecessarily change the 3D model, preventing any unexpected changes. Otherwise, an alternative value that also satisfies the variant-specific constraints is assigned to the parameter.
- **Invariant constraints and variant-specific constraints are not compatible.** No value exists that satisfies both the invariant and variant-specific constraints placed on a parameter (scenario d). In this case, an error is raised.

4 EXTENDING PCAD LANGUAGES TO SUPPORT VARIMOD

VariMod is general enough to be integrated with virtually all parametric CAD tools that support constraints. In this section we describe how VariMod can be integrated into PrintTalk, an existing PCAD language for modelling 3D printable products. We used the resulting implementation to validate our approach (section 5).

This section starts by providing a brief overview of the PrintTalk language, before describing the actual integration and implementation.

4.1 3D Modelling in PrintTalk

In PrintTalk, 3D components are called 'shapes'. The language contains a set of primitive shapes (cubes, spheres, cylinders, etc.) that can be combined into other, more complex shapes.

PrintTalk implements the *Constructive Solid Geometry* (CSG) modelling technique, where 3D models can be thought of as tree structures. The root of the tree represents the complete 3D model, while leafs represent primitive shapes. Shapes that compose other shapes are represented by internal nodes of the tree. In PrintTalk, shapes can be combined by means of union and difference operations.

PrintTalk features a powerful abstraction mechanism, as shapes can take two different types of parameters.

- *Construction parameters*, to which a value is assigned by the programmer upon instantiation of the shape.
- *Constraint variables*, to which a value is assigned by an underlying constraint solver. The value of a constraint variable is determined by the constraints that are asserted on it. Constraints on a shape's variables are asserted from within the shape when the constraints are part of a shape's definition. Additionally, constraints can be asserted externally, as part of the composition, when composing the shape with other shapes.

Shapes further contain a *script*, in which constituents of a composition can be instantiated and constraints asserted on the constraint variables of the shape and its constituents.

A constraint in PrintTalk is bidirectional, as the solver is used to both test whether the relation expressed by the constraint holds *and* to assign values to constraint variables to make the relation expressed by the constraint hold. PrintTalk constraints can either be primitive constraints that are natively supported by the underlying constraint solver, or user-defined constraints that combine other (possibly also user-defined) constraints.

Figure 4 illustrates a PrintTalk program and its resulting 3D model representing a hollow cube. First, a cube (line 4) is composed with a cylinder (line 5) that sticks out from two opposing sides. The resulting shape is named `cube-cyl`. Next, `cube-cyl` is cut (line 13) from a larger cube (line 12) to obtain the target `hollow-cube` shape, which is then exported as an STL file (line 19). The dimensions of the shapes are determined by constraints asserted as part of the shapes' definitions. For example, the `cyl-dia` constraint variable represents the diameter of the hole, and is specified to be 70% of the length of the edges of the hollow cube.

4.2 Integrating VariMod into PrintTalk

To integrate VariMod in PrintTalk, we introduce a new component named *variant* for asserting variant-specific constraints on shapes without modifying their definitions. A variant composes a root shape and its invariant constraints with a set of variant-specific constraints. Figure 5 provides an overview of the relations between

```
(shape: cube-cyl (cyl-dia cube-size)
 (cyl-length)
 (script:
  (cube 0 0 0 cube-size)
  (cylinder 0 0 0 cyl-dia cyl-length))
 (constraints:
  (assert: (>= cyl-length cube-size))))

(shape: hollow-cube (cube-size)
 (cyl-dia in-size)
 (script:
  (cube 0 0 0 cube-size)
  (cut: (named: cut-obj (cube-cyl cyl-dia in-size))))
 (constraints:
  (assert: (= in-size (* cube-size 0.9)))
  (assert: (= cyl-dia (* cube-size 0.7)))
  (assert: (= (cut-obj@cyl-length) cube-size))))

(print: (hollow-cube 100) "hollow-cube.stl")
```

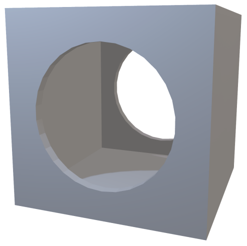

Figure 4: PrintTalk program that models a hollow cube.

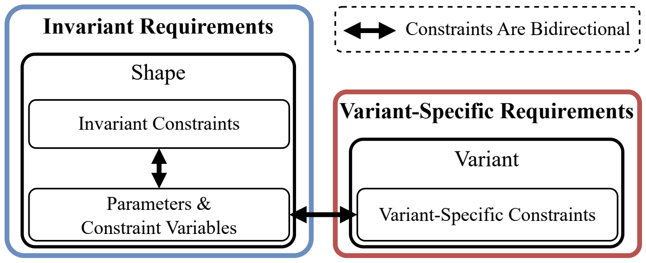
Invariant Requirements
Shape
Invariant Constraints
Parameters & Constraint Variables
Constraints Are Bidirectional
Variant-Specific Requirements
Variant
Variant-Specific Constraints

Figure 5: Overview of the relations between the newly introduced components.

existing PrintTalk components and the newly introduced `variant` component. We explain this structure in more detail in what follows.

A new variant is instantiated using the `variant:` construct.

```
(variant:
 (write-to: <filename>)
 (main: <main>)
 (constraints: <constraints>))
```

A variant consists of a filename, to which the 3D model is exported, a main shape, and a set of variant-specific constraints. The main shape represents the root of the 3D model's component tree. The constraints can operate on the constructor parameters and constraint variables of the main shape. When asserting constraints, the `main` pseudo-variable can be used for referencing the main shape.

In order to assign a criticality level to variant-specific constraints, a hierarchy of criticality levels can optionally be declared within *variant* components, as shown below.

```
(variant:
 ...
 (preferential-constraints: (id0 id1 ... idn)
  <preferential-constraints>))
```

A hierarchy is a list that consists of tags id_0 to id_n, representing criticality levels in descending order. This design allows for new criticality levels to be introduced by adding them to the list. Additionally, the relative hierarchy between criticality levels can be altered by changing the positions of their corresponding tags in the list. An asserted constraint is preferential if it is preceded by a criticality level.

```
(assert: <criticality-level> <constraint>)
```

Constraints that are asserted without a criticality level are considered to be required constraints.

```
1   (shape: cube-constrained-size ()
2    (cube-size)
3    (script:
4     (cube 0 0 0 cube-size))
5    (constraints:
6     (assert: (>= cube-size 10))))
7
8   (variant:
9    (write-to: "cube-cs-ext.stl")
10   (main: (cube-constrained-size))
11   (constraints:)
12   (preferential-constraints: (strong weak)
13    (assert: weak (= (main.cube-size) 12))
14    (assert: strong (= (main.cube-size) 20))))
```

Listing 1: `variant` Component in PrintTalk.

Listing 1 illustrates how the different types of constraints can be used together in PrintTalk extended VariMod. In this program, shape `cube-constrained-size` is defined with an invariant constraint (line 6) that specifies that its size must be equal or greater than `10`. Next, a variant of `cube-constrained-size` is declared by asserting two preferential variant-specific constraints on constraint variable `cube-size` of the main shape.

Although compatible with the invariant constraint, the variant-specific constraints contradict each other. Therefore, VariMod suggests 20 as the value for `cube-size`, satisfying the variant-specific constraint with the higher criticality level on line 14, while omitting the constraint on line 13.

4.3 Solving Constraints

In addition to the newly introduced variant component, we also adapt PrintTalk's constraint solving process to allow variables to be constrained by a combination of invariant constraints and variant-specific constraints, thereby supporting VariMod's two-stage constraint solving process as described in section 3.3.

The first stage remains identical to PrintTalk's original constraint solving process, as the solver only has to consider invariant constraints. When the invariant constraints do not contradict each other, variant-specific constraints are considered. First, required variant-specific constraints are considered, followed by preferential variant-specific constraints in descending order of criticality level. If at any point a constraint cannot be satisfied, an error is raised.

Our implementation of PrintTalk integrates the Z3 Solver [7]. Z3 offers no explicit support for constraint hierarchies [6] used to express preferential constraints. We overcame this limitation by making *multiple* calls to the constraint solver with monotonically growing sets of constraints following the solving pattern outlined above, essentially implementing the *locally-predicate-better* solving mechanism from Wilson and Borning [21]. In the first call, the set of invariant constraints is sent, in the second call this set is extended with the required variant-specific constraints, and so on.

5 VALIDATION

This section validates VariMod and demonstrates how it overcomes modern-day CAD software's limited support for variational design by distinguishing between invariant and variant-specific constraints. Additionally, we compare VariMod to the functionality of current CAD tools for modelling variant-specific requirements (VRs). All our scenarios start from the initial design of a hinge modelled in PrintTalk, to which different VRs expressed as variant-specific constraints are added.

Invariant Requirements of Shapes. We first define a PrintTalk shape that models the invariant requirements (IRs) of a hinge using invariant constraints. Upon instantiation, this shape represents a valid 3D model that does not model VRs yet. We design multiple variants of the hinge using `variant` components that assert different variant-specific constraints throughout this section. The PrintTalk program modelling the IRs of a hinge is provided in listing 2. The program defines two shapes. The first shape represents the leaf of a hinge, and is reused twice as part of the second shape for modelling the two leaves of the hinge. Both shapes contain invariant constraints that model IRs for ensuring that the resulting 3D model is valid, and can function as a hinge. For example, invariant constraints ensure that the diameter of the screw-holes does not exceed the dimensions of a leaf (lines 12–13). While the code produces a valid 3D model, the diameter of the screw-holes and the distance between them, and the diameter of the pin are intentionally left 'underconstrained'. Therefore, additional variant-specific constraints are required for the hinge to be practical.

```
1   (shape: leaf (x y z w t h knuckle-dia)
2    (screw-dia c-dist screw-dist)
3    (script:
4     (cuboid x y z w t h)
5     (cut: (rotate: 90 0 0 (cylinder x y (+ z c-dist) screw-dia t)))
6     (cut: (rotate: 90 0 0 (cylinder x y (- z c-dist) screw-dia t)))
7     (cuboid (+ x (/ w 2) (/ knuckle-dia 4)) y (- z (/ h 4))
8      (/ knuckle-dia 2) t (/ h 2))
9     (cylinder (+ x (/ w 2) (/ knuckle-dia 2)) y (- z (/ h 4))
10     knuckle-dia (/ h 2)))
11   (constraints:
12    (assert: (< screw-dia w))
13    (assert: (< screw-dia (/ h 2)))
14    (assert: (= screw-dist (* 2 c-dist)))
15    (assert: (> screw-dist screw-dia))
16    (assert: (< screw-dist (- h screw-dia)))
17    (assert: (>= knuckle-dia t))))
18
19  (shape: hinge (l w h)
20   (screw-dia screw-dist pin-dia knuckle-dia pin-rim clearance)
21   (script:
22    ;; Leaf with pin
23    (named: leaf1 (leaf 0 0 0 l w h knuckle-dia))
24    (named: pin (cylinder (+ (/ l 2) (/ knuckle-dia 2)) 0 (/ h 4)
25     (- pin-dia clearance) (/ h 2)))
26    ;; Leaf with hole
27    (named: leaf2 (leaf 0 (* knuckle-dia 2) 0 l w h knuckle-dia))
28    (cut: (named: pin-hole (cylinder (+ (/ l 2) (/ knuckle-dia 2))
29     (* knuckle-dia 2) (/ h -4) pin-dia (/ h 2)))))
30   (constraints:
31    (assert: (= screw-dist (leaf1.screw-dist)))
32    (assert: (= screw-dist (leaf2.screw-dist)))
33    (assert: (= screw-dia (leaf1.screw-dia)))
34    (assert: (= screw-dia (leaf2.screw-dia)))
35    (assert: (< pin-dia knuckle-dia))
36    (assert: (= pin-rim (/ (- knuckle-dia pin-dia) 2)))
37    (assert: (>= pin-rim (/ w 8)))
38    (assert: (= clearance (/ w 8)))))
```

Listing 2: PrintTalk program modelling the IRs of a hinge.

5.1 Variation through Variant-Specific Constraints

In a first experiment, we validate VariMod for designing variants that satisfy the same IRs, but varying VRs. Through VariMod's `variant` components, variant-specific constraints representing VRs can be asserted on a shape's variables without making modifications to the shape's definition itself. A clear distinction between IRs and VRs is maintained, as they are asserted in a distinct place in the program text. This makes it straightforward to determine which constraints model VRs, and can therefore be safely modified without invalidating IRs. The example below describes a scenario in which a variant of an existing 3D model is designed, while ensuring that modifications do not cause the 3D model to invalidate IRs. In this scenario, *variant-specific* constraints influence the diameter of the hinge's pin and knuckle, and the size of the rim. Additional constraints specify the diameter of the screw-holes and the distance between these holes.

```
(variant:
 (write-to: "hinge.stl")
 (main: (hinge 25 4 50))
 (constraints:
  (assert: (>= (main.pin-dia) 5))
  (assert: (<= (main.knuckle-dia) 8))
  (assert: (>= (main.pin-rim) 1))
  (assert: (= (main.screw-dia) 5))
  (assert: (= (main.screw-dist) 20))))
```

Listing 3: Variant component for modelling the variant-specific requirements of a hinge.

Optimising Parameter Values. The example below illustrates VariMod's capabilities for optimising parameter values. We consider a scenario in which the design of the hinge in modified so that it can be manufactured and used to support a door. To obtain a hinge that is manufacturable, and that satisfies the requirements imposed by its targeted use, VRs originating from different steps of the engineering process must be considered. In this scenario, we focus on the size and position of the mounting holes of the hinge. Next to IRs for ensuring that the resulting 3D model represents a functional hinge, the following VRs, originating from different steps in the engineering process, must be considered for finding an optimal value for their corresponding parameters in the digital 3D model:

- To be manufacturable, the diameter of the screw-holes must be at least 5mm.
- For assembly, the hinge will be mounted using screws with a diameter of 6mm.
- The door to which the hinges are to be mounted foresees that the screw-holes are 30mm apart.

Listing 4 illustrates a VariMod `variant` component that models the VRs described above, originating from different steps of the engineering process.

```
(variant:
 (write-to: "hinge-engineering-process.stl")
 (main: (hinge 25 4 50))
 (constraints:
  (assert: (>= (main.screw-dia) 5))
  (assert: (= (main.screw-dia) 6))
  (assert: (= (main.screw-dist) 30))))
```

Listing 4: Variant component modelling VRs originating from different steps of the engineering process.

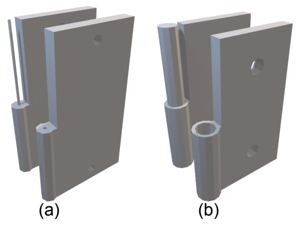

Figure 6: Observing the effect of variant-specific constraints on 3D models.

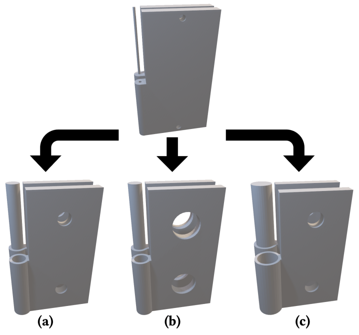

Figure 7: Multiple variants that satisfy different variant-specific constraints.

Comparing Variants of Designs. As VariMod distinguishes invariant constraints and variant-specific constraints, the effect of variant-specific constraints can be observed by comparing the 3D models resulting from variants that assert different variant-specific constraints. Furthermore, VariMod ensures that the invariant constraints of a 3D model are satisfiable, so that an initial 3D model can be generated before it is optimised using variant-specific constraints. Hereby VariMod enables comparing initial 3D models with their optimised versions. Figure 6 illustrates the differences between these two versions for the hinge described above. Figure 6 (a) represents the 3D model that results from only considering IRs, while fig. 6 (b) represents the 3D model that results after also considering VRs.

Scalable Variations on the Design. This example illustrates VariMod's capabilities for modelling multiple variants of a design in a scalable manner. VariMod enables the specification of VRs through the assertion of variant-specific constraints in a `variant`, without requiring modifications to the initial shape that models IRs. This approach enables the straightforward creation of variations on a design, as multiple variants can be instantiated for modelling different VRs through variant-specific constraints. Modifications made to the original shape that only models IRs affect all variants of that shape, while modifications made through a `variant` component only affect the variant that is represented by that `variant` component. Figure 7 illustrates three variants of a `hinge` that are designed through three `variant` components, without modifying the initial shape that models IRs, depicted at the top of the figure. Modifications made to this initial shape would affect each variant of the shape.

5.2 Support for Preferential Constraints

This final example illustrates VariMod's capabilities for optimising 3D models when not all requirements are compatible, but some requirements are considered to have a higher priority over others. VariMod supports preferential variant-specific constraints, and uses the criticality level for determining an optimal set of parameter values, prioritising the satisfaction of constraints with a higher criticality level over constraints with a lower criticality level. Listing 5 demonstrates how several variant-specific constraints with different criticality levels are used for determining the optimal diameter for the pin and knuckle of the hinge.

```
(variant:
 (write-to: "hinge.stl")
 (main: (hinge 25 4 50))
 (constraints:
  (assert: (= (main.pin-dia) 5))
  (assert: (= (main.screw-dia) 5))
  (assert: (= (main.screw-dist) 20)))
 (preferential-constraints: (strong medium weak)
  (assert: medium (= (main.knuckle-dia) 8))
  (assert: strong (= (main.pin-rim) 2))
  (assert: weak (> (main.pin-dia) (/ (main.knuckle-dia) 2)))))
```

Listing 5: Assigning criticality levels to variant-specific constraints.

6 LIMITATIONS AND FUTURE WORK

We identify four items of future work for overcoming current limitations of VariMod.

Querying Multiple Conceptual Representation Layers. Digital 3D CAD-drawings can be represented on multiple conceptual levels [10]. In PrintTalk, for example, 3D models have three conceptual levels.

- The front-end PrintTalk program representing the 3D model.
- The back-end representation of the 3D model in the underlying geometric modelling kernel, for calculating the geometric properties of the 3D model.
- A triangulated approximation of the 3D model in the form of an STL file, that can be visualised and imported into other software suitable for applications such as 3D printing.

The integration of VariMod into PrintTalk only supports constraints that operate on the attributes of the front-end PrintTalk-code. This front-end representation contains no information about attributes that require calculations on the back-end level, such as volume or intersecting lines. To constrain these lower-level attributes, future work may extend VariMod with support for constraints that operate on attributes present in other model representation layers.

Supporting Modifications beyond Parameter Changes. VariMod currently only supports the modification of parameter values of 3D models. Future work could explore an extension of VariMod that is capable of optimising 3D models through modifications that cannot be expressed as parameter changes. For example, 3D models may be modified by substituting components, or by adding new features such as chamfers and fillets.

Variants of Variants. Currently, we only investigated the creation of variants of 3D models that contain just invariant constraints (i.e., root shapes without variations). Although there is nothing particular in our current approach that disallows creating variants of *variants*, this is something we have not yet explored and implemented.

Learning-Aided Parameter Configurations. As argued by Amand et al., machine learning can be used for deriving constraints that prevent errors resulting from incompatible parameter configurations of 3D-models. Because VariMod supports the specification of such constraints, future work could enable automatic insertion of these constraints into PrintTalk shapes so that errors resulting from faulty parameter configurations of underconstrained shapes can be prevented.

7 CONCLUSION

This paper describes VariMod, an approach for designing variants of 3D models in parametric CAD in a structured manner. VariMod distinguishes between *invariant constraints* that ensure validity of 3D models, and variant-specific *variant-specific constraints* for optimising the parameter values of each variant. By distinguishing *invariant constraints* and *variant-specific constraints*, VariMod promotes the reusability of 3D models, as an initial 3D model that satisfies all invariant constraints can be instantiated multiple times to create variations that each satisfy the same invariant constraints, but different variant-specific constraints. This way, VariMod facilitates instantiating different variants of a 3D model in a structured and efficient manner.

Both invariant and variant-specific constraints are composed and solved by VariMod's two-stage constraint solving process, which first determines initial parameter values using invariant constraints and then optimises these values based on variant-specific constraints. VariMod also differentiates between required constraints and preferential constraints. 3D models must satisfy required constraints, and VariMod raises an error when no parameter values that satisfy all required constraints exist. Preferential constraints have a criticality level assigned to them, and VariMod prioritises constraints with a higher criticality level over constraints with a lower hierarchy level. When constraints are incompatible, VariMod attempts to still generate a 3D model that satisfies as many constraints as possible by omitting incompatible constraints with low criticality levels.

We implemented VariMod as an extension to PrintTalk, and validated our approach by using VariMod to design 3D models. Comparing its capabilities with those of the state-of-the-art parametric CAD software, we demonstrated how VariMod can effectively by used for designing variants of 3D models in a structured and efficient manner.

REFERENCES

[1] Mathieu Acher, Benoit Baudry, Olivier Barais, and Jean-Marc Jézéquel. 2014. Customization and 3D Printing: A Challenging Playground for Software Product Lines. In *Proceedings of the 18th International Software Product Line Conference - Volume 1* (Florence, Italy) *(SPLC '14)*. Association for Computing Machinery, New York, NY, USA, 142–146. https://doi.org/10.1145/2648511.2648526

[2] Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, and Jean-Marc Jézéquel. 2019. Towards Learning-Aided Configuration in 3D Printing: Feasibility Study and Application to Defect Prediction. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems* (Leuven, Belgium) *(VaMoS '19)*. Association for Computing Machinery, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/3302333.3302338

[3] Elnaz Asadollahi-Yazdi, Julien Gardan, and Pascal Lafon. 2017. Integrated Design for Additive Manufacturing Based on Skin-Skeleton Approach. *Procedia CIRP* 60 (2017), 217–222. https://doi.org/10.1016/j.procir.2017.02.007 Complex Systems Engineering and Development Proceedings of the 27th CIRP Design Conference Cranfield University, UK 10th – 12th May 2017.

[4] Jean-Bernard Bluntzer, Egon Ostrosi, and Jérémy Niez. 2016. Design for Materials: A New Integrated Approach in Computer Aided Design. *Procedia CIRP* 50 (2016), 305–310. https://doi.org/10.1016/j.procir.2016.04.153 26th CIRP Design Conference.

[5] G. Boothroyd and L. Alting. 1992. Design for Assembly and Disassembly. *CIRP Annals* 41, 2 (1992), 625–636. https://doi.org/10.1016/S0007-8506(07)63249-1

[6] Alan Borning, Robert Duisberg, Bjorn Freeman-Benson, Axel Kramer, and Michael Woolf. 1987. Constraint Hierarchies. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications* (Orlando, Florida, USA) *(OOPSLA '87)*. Association for Computing Machinery, New York, NY, USA, 48–60. https://doi.org/10.1145/38765.38812

[7] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[8] Ram Prasad Diwakaran and Michael D. Johnson. 2012. Analyzing the effect of alternative goals and model attributes on CAD model creation and alteration. *Computer-Aided Design* 44, 4 (2012), 343–353. https://doi.org/10.1016/j.cad.2011.11.003

[9] Claudio Favi, Federico Campi, Michele Germani, and Marco Mandolini. 2022. Engineering knowledge formalization and proposition for informatics development towards a CAD-integrated DfX system for product design. *Advanced Engineering Informatics* 51 (2022), 101537. https://doi.org/10.1016/j.aei.2022.101537

[10] Daniel Fuchs, Ronald Bartz, Sebastian Kuschmitz, and Thomas Vietor. 2022. Necessary advances in computer-aided design to leverage on additive manufacturing design freedom. *International Journal on Interactive Design and Manufacturing (IJIDeM)* 16, 4 (2022), 1633–1651. https://doi.org/10.1007/s12008-022-00888-z

[11] Julien Gardan. 2016. Additive manufacturing technologies: state of the art and trends. *International Journal of Production Research* 54, 10 (2016), 3118–3132. https://doi.org/10.1080/00207543.2015.1115909

[12] David A. Gatenby and George Foo. 1990. Design for X (DFX): Key to Competitive, Profitable Products. *AT&T Technical Journal* 69, 3 (1990), 2–13. https://doi.org/10.1002/j.1538-7305.1990.tb00332.x

[13] Megan Hofmann, Gabriella Hann, Scott E. Hudson, and Jennifer Mankoff. 2018. Greater than the Sum of Its PARTs: Expressing and Reusing Design Intent in 3D Models. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173875

[14] Shixuan Hou, Jie Gao, and Chun Wang. 2022. Design for mass customisation, design for manufacturing, and design for supply chain: A review of the literature. *IET Collaborative Intelligent Manufacturing* 4, 1 (2022), 1–16. https://doi.org/10.1049/cim2.12041

[15] Jef Jacobs, Jens Nicolay, Christophe De Troyer, and Wolfgang De Meuter. 2021. PrintTalk: A Constraint-Based Imperative DSL for 3D Printing. In *Proceedings of the 18th ACM SIGPLAN International Workshop on Domain-Specific Modeling* (Chicago, IL, USA) *(DSM 2021)*. Association for Computing Machinery, New York, NY, USA, 11–20. https://doi.org/10.1145/3486603.3486778

[16] Fumihiko Kimura, Hiromasa Suzuki, and Toshio Sata. 1986. Variational Product Design by Constraint Propagation and Satisfaction in Product Modelling. *CIRP Annals* 35, 1 (1986), 75–78. https://doi.org/10.1016/S0007-8506(07)61842-3

[17] Tsai-C. Kuo, Samuel H. Huang, and Hong-C. Zhang. 2001. Design for manufacture and design for 'X': concepts, applications, and perspectives. *Computers & Industrial Engineering* 41, 3 (2001), 241–260. https://doi.org/10.1016/S0360-8352(01)00045-6

[18] V. C. Lin, D. C. Gossard, and R. A. Light. 1981. Variational Geometry in Computer-Aided Design. *SIGGRAPH Comput. Graph.* 15, 3 (aug 1981), 171–177. https://doi.org/10.1145/965161.806803

[19] Prabhu Shankar, Beshoy Morkos, Darshan Yadav, and Joshua D. Summers. 2020. Towards the formalization of non-functional requirements in conceptual design. *Research in Engineering Design* 31, 4 (2020), 449–469. https://doi.org/10.1007/s00163-020-00345-6

[20] I.A.R. Torn and T.H.J. Vaneker. 2019. Mass Personalization with Industry 4.0 by SMEs: a concept for collaborative networks. *Procedia Manufacturing* 28 (2019), 135–141. https://doi.org/10.1016/j.promfg.2018.12.022 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2018).

[21] Molly Wilson and Alan Borning. 1993. Hierarchical constraint logic programming. *The Journal of Logic Programming* 16, 3 (1993), 277–318. https://doi.org/10.1016/0743-1066(93)90046-J