






Comparing Digital Tools for Implementing a Generative System for the Design of Customized Tableware

Eduardo Castro e Costa¹ , Joaquim Jorge²  and José Duarte³ 

¹The Pennsylvania State University, erc15@psu.edu

²University of Lisbon, jorgej@tecnico.ulisboa.pt

³The Pennsylvania State University, jxp400@psu.edu

Corresponding author: José Duarte, jxp400@psu.edu

Abstract. In typical mass customization design processes, designers encode design processes into configurators that are subsequently used by end-users to customize products. Using the case study of ceramic tableware, we explore how to extend the role of designers beyond the deployment of such configurators. In this paper we describe the development of a generative design system for the mass customization of ceramic tableware, focusing on the implementation of generic shape grammar rules encoded into parametric models. The design system enables designers to develop customizable tableware collections, later to be customized using configurators. Two prototypes are presented and compared, each encoding the design system through different programming environments: Grasshopper and Unity.

Keywords: mass customization, generative design, shape grammars, parametric modeling, Grasshopper, Unity.

DOI: <https://doi.org/10.14733/cadaps.2019.803-821>

1 INTRODUCTION

Design is traditionally the *métier* of designers. However, in the last decades we have witnessed the emergence of approaches such as user-centered design, participatory design and co-design, in which end-users, for whom design is performed, are brought closer to the design process with various degrees of proximity [1]. Such transfer of design responsibility occurs in many creative fields such as architecture and product design, and in different contexts, namely mass customization.

In mass customization, computer-implemented configurators engage non-expert end-users in co-design by enabling them to craft customized products and services according to their needs and preferences. In many current configurators, the customization process consists in having the end-user directly exploring a constrained design space, which corresponds to the set of all possible design solutions automatically generated by the configurators [2].

The definition of such design space covers only part of the problem of defining the mass customization solution space, which must take into consideration the real individual needs of potential customers [3,4]. In fact, the design space that is provided by a particular configurator does not necessarily correspond to such solution domain, and should be narrowed down in order to eliminate or at least omit solutions that do not appeal to its users. Consequently, user feedback is essential to identify undesired designs and validate a configurator, before its deployment into the market.

Another important aspect that needs addressing is that, for the end-user, the quality of a mass customization experience depends on exploring a balanced design space that provides enough design variety for the process to become interesting, but at the same time ensures the end-user is not overwhelmed by a myriad of choices. Such a balance is difficult to ensure with traditional configurators [5,6]. To this end, we suggest an innovative Design Participation Model (DPM) towards a more flexible solution space management, by modulating the interactions among participants in the creative process, namely system designers, designers and end-users, Figure 1. According to the proposed model, the system should allow designers to create customizable domains in the form of parametric models and, subsequently, allow end-users to manipulate the shape of such customizable creations. In this way, complex solution spaces can be simplified by designers and more easily manipulated by end-users [7].

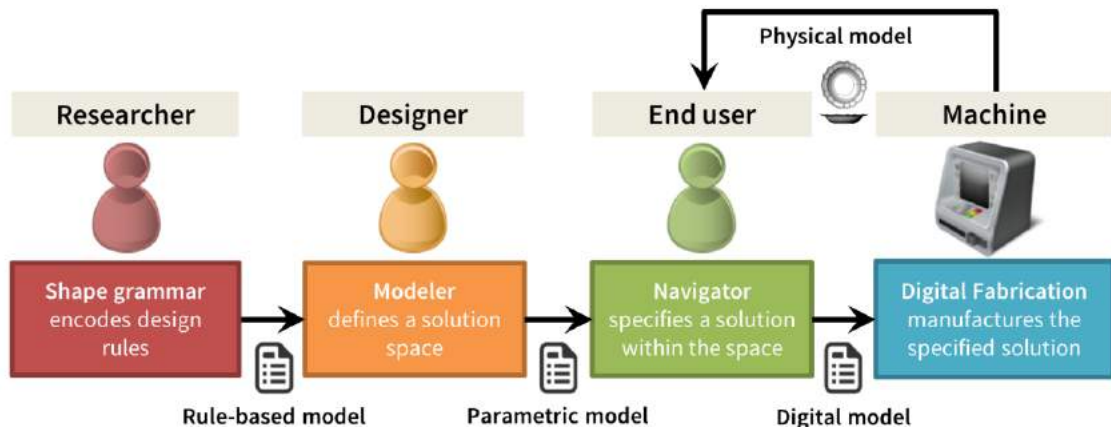


Figure 1: Design Participation Model for the design system.

To test the proposed Design Participation Model, we developed a prototype of a mass customization system focused on ceramic tableware. Reasons for choosing ceramic tableware as a case study include its reduced production cost and the sector's need to be competitive against companies manufacturing low-cost products [8]. Borrowing from an existing model for implementing mass customization in architectural design [9], our system comprises a generative design technique that encompasses design rules for spawning customized designs, as well as a production framework capable of manufacturing such models using digital fabrication technology, while using a computational framework to articulate both systems. Implementation of the generative design system implied the development of generic shape grammars for encoding the rules for creating ceramic tableware and parametric models for implementing those rules. We developed the design system using a case study from a Portuguese tableware manufacturer, Matceramica (henceforth referred to as Manufacturer), who supported our research by providing information about their tableware designs and feedback from their design team on the resulting prototype.

Our Design System (DS) is supported by the *Tableware Shape Grammar* (TSG), a parametric formalism that encodes rules for designing tableware collections. TSG was developed according to Duarte's model for inferring generic shape grammars [10]. Inferring the generic grammar began by

analysing existing tableware collections produced by the Manufacturer, focusing on aspects such as the shape of ceramic tableware elements, or the relationships among elements within a collection. We developed a specific shape grammar for each analysed collection, containing the design rules deemed necessary for its generation. The resulting set of specific shape grammars was subsequently abstracted into a generic grammar, containing parametric rules [11,12]. The resulting TSG incorporates the rules that govern the design of customized tableware (Figure 2) and serves as a framework for encoding design knowledge and implementing the computational tools that enable designers and end-users to customize the shape of tableware collections.

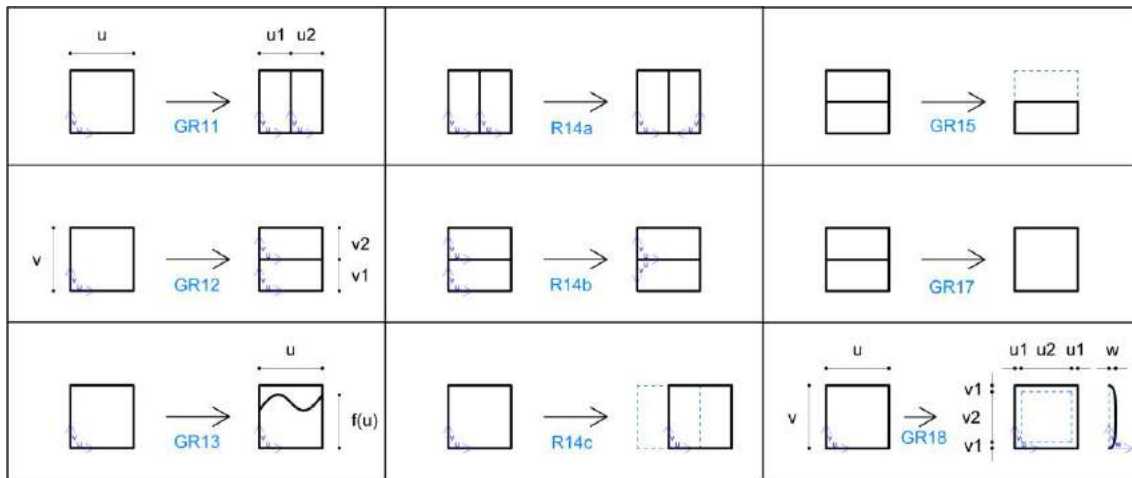


Figure 2: Sample of generic rules of the Tableware Shape Grammar for decorating tableware elements.

We studied ceramic tableware collections both as product and process. Besides analyzing collections and inferring specific and generic grammars, we also used Contextual Inquiry for observing the task of creating tableware collections as performed by tableware specialists, namely a team of in-house designers working for the Manufacturer, to identify typical conceptual moves when designing a new collection. This inquiry was performed after developing the TSG and its results were taken into account in the implementations of the design system.

In this paper, we present and discuss the implementation of the DS into computational tools, taking advantage of the computer's ability to quickly generate visual representations of many solutions. We targeted two main user groups foreseen in the Design Participation Model (DPM), designers and end-users, which resulted in two different, complementary components, Modeller and Navigator, respectively. Section 2 presents the tools used in the implementation of different prototypes of the design system. Section 3 describes the strategy and design for the implementation of the first prototype developed using Grasshopper. Section 4 describes the adaptation of the initial prototype into the final one implemented in Unity. In Section 5, we discuss to what extent the presented prototypes implement the rules of the TSG. Finally, Section 6 concludes this paper with its main takeaways and issues for future research.

2 IMPLEMENTATION TOOLS

In the implementation of the design system, three prototypes were developed consecutively, using three different programming environments: (1) Grasshopper, (2) Racket, and (3) Unity. The transition between environments was induced by our goal to develop the DS as a Web-based

standalone application. This enabled the system to be tested and used by a large number of users with diversified backgrounds.

Difficulties in using the first two prototypes online led to the implementation of the third one. Nevertheless, each implementation demonstrated strengths and weaknesses for the task of implementing a design system based on shape grammars with mass customization in mind. All three prototypes played important roles in the implementation of the generative design system. However, in this paper we will focus on the initial Grasshopper prototype, since it is illustrative of the interplay between shape grammars and parametric modeling, and on the final Unity prototype, since it is the one chosen for the final implementation of the design system.

2.1 Grasshopper

The first prototype of the DS served to quickly test the TSG while being developed and it was implemented using Grasshopper and Rhinoceros (Figure 3). Grasshopper (GH) is what its developers describe as 'a graphical algorithm editor' [13] and it enables developing parametric models through visual programming. In GH, parametric models are implemented by articulating components that typically perform geometric operations, allowing users to manipulate the shape of designs by changing the values of the components' input parameters.

GH has been progressively embraced by the creative community, namely designers and architects, mostly because it is a visual language, enabling them to create generative design algorithms by visually introducing and connecting components [14]. Designs encoded in GH may be generated as digital models in CAD software Rhinoceros [15], whose capability of representing complex geometry such as NURBS surfaces makes it particularly suitable for representing the double-curved shapes of ceramic tableware elements. Despite its advantages, GH was deprecated for implementing the DS for tableware because of its limitations in running as a standalone program or as a web application.

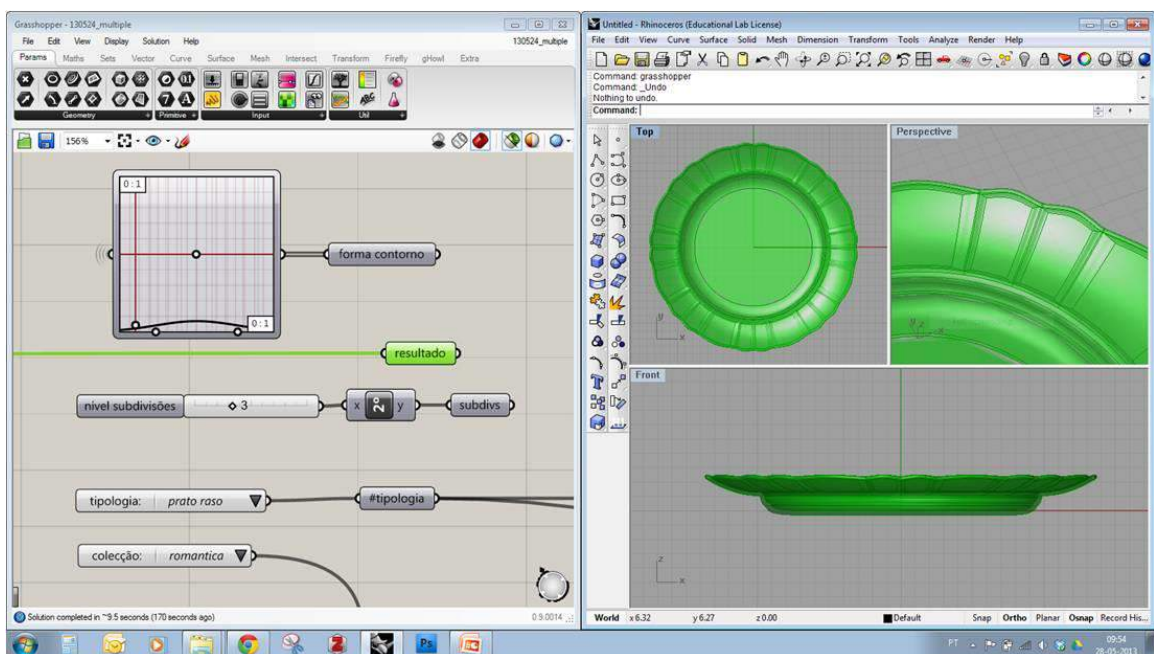


Figure 3: Implementation of the Grasshopper prototype.

2.2 Unity

The limitations in deploying the prototype as a web app led to the implementation of a final prototype of the DS using the Unity game engine (Figure 4). Applications developed in Unity can run in many different platforms for both workstations and mobile devices, namely Windows, MacOS, Android, iOS and Web browsers. From Unity's version 5 on, projects can be compiled in HTML5, the current standard platform for Web applications [16]. This means that Web browsers can run Unity applications directly as if they were Web pages.

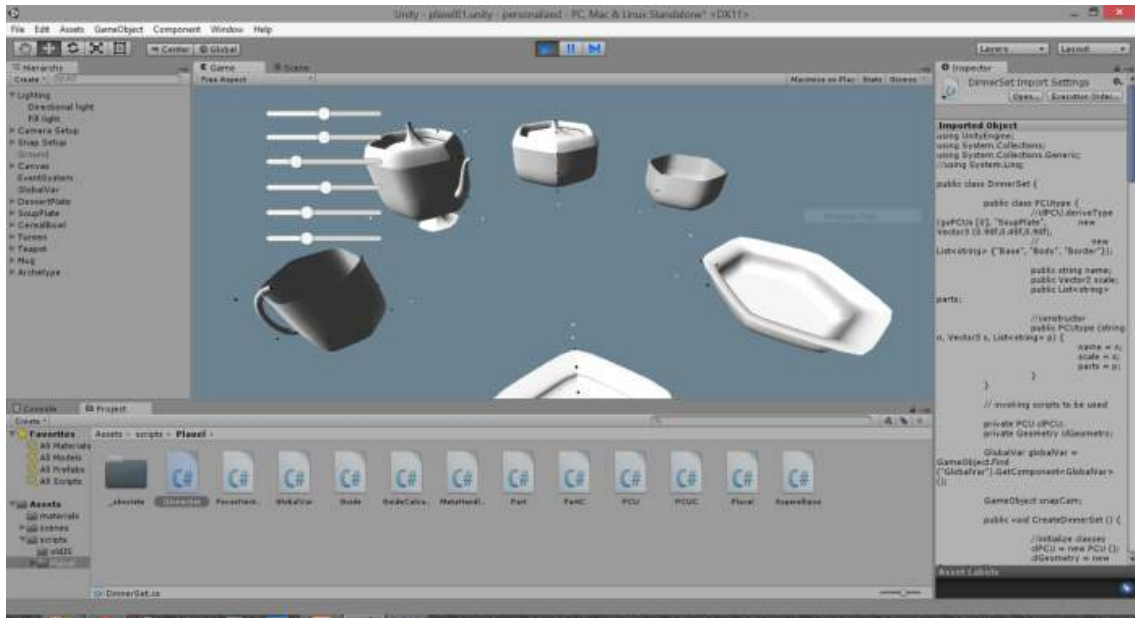


Figure 4: The Unity prototype of the Tableware Design System.

Unity's versatility rendered it as a suitable tool for implementing the design system as a web application. However, Unity bears the disadvantage of not being equipped with a native CAD library, which constrains its generative capabilities to a small set of geometric primitives such as cubes, spheres, or cones. In fact, in a typical development workflow, 3D models are created using a third-party modeling application like Rhino, and then imported into Unity.

Therefore, it was necessary to implement 3D modelling functions from the ground up, namely lofts and sweeps. The selected language was C#, since it is the most widespread of the languages available in Unity: C#, Boo, and Unityscript.

3 GRASSHOPPER: FROM SHAPE GRAMMARS TO PARAMETRIC MODELS

For understanding the process of translating shape grammar rules into parametric models, it is important to understand the relationships among the different parts of the Tableware Design System.

3.1 Hierarchical Structure of Tableware Collections

Tableware collections were deconstructed into hierarchical levels: collections are composed of dishware types, which are composed of pieces, which are composed of parts (Figure 5). According to the strategy adopted in the shape grammar development, all types of a collection can be initially described as variations of the same initial shape. Therefore, the parametric model into which the collection is translated can generate different dishware types: charger, dinner and soup plates, cereal bowl, mug and cup. One parametric model representing different types means that these types correspond to parametric variations of a more generic meta-type.



Figure 5: Hierarchical levels for designing tableware collections.

The adopted hierarchical relationships can be identified in the design rules of the TSG. For example, Rule 2a (Figure 6) subdivides a quadrilateral that represents a type of a tableware collection element which is composed of a single piece. Such quadrilateral, which we call 'envelope', is subdivided into three other envelopes, each representing a different part of the type. Envelopes correspond to the boundary of sections of elements on any level of the hierarchical structure. For example, as shown in Figure , both the higher-level type and each of its parts in the subsequent level are represented by envelopes. Despite representing objects of different levels, envelopes are similar in shape, which is determined by the object's properties, namely its dimensions such as height and width. Therefore, in the Tableware Shape Grammar (TSG), the left side of rules often features a similar envelope, suggesting that they can be subjected to similar operations. This led to the conceptualization of such generic envelopes as a class of objects named Plaxel (capitalized), which stands for PLATE Element, similarly to terms used in computer graphics such as pixel (PIcture ELement) or voxel (VOLUME ELement). Objects or instances of this class are called plaxels (non-capitalized) and they can represent the base shape of any element of the design system, from the top to the bottom of the hierarchical structure.

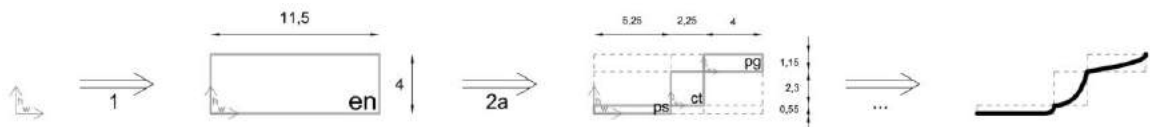


Figure 6: Initial steps in the derivation of a soup plate: envelope creation and functional partitioning.

Thinking about these objects as instances of a class, as well as developing a symbolic representation for the Plaxel class, helped rationalizing the DS and, thereby, simplifying its implementation into GH parametric models. Although GH does not provide direct support for creating classes, or other custom data types, it allows users to create 'custom components' via scripting using textual programming languages. Natively, these custom components can be written in either VB.NET or C#.NET, and additional languages are supported via third party plug-ins, such as Python, for example.

3.2 Implementing Shape Grammar Rules

As mentioned previously, the Tableware Shape Grammar (TSG) holds rules for the design of tableware collections. Despite being considered computational models, shape grammars can be initially developed and tested using pencil and paper. However, as the TSG grew more complex, it became impractical to test it by hand, requiring a computer implementation. To implement a shape grammar on a computer is not a trivial task. Such implementations, called shape grammar

interpreters, need to perform two different tasks: a) to recognize shapes, and b) to operate on those shapes [17]. Since automatic shape recognition and shape emergence was beyond the scope of the current research, the TSG was developed as a set grammar. Such approach facilitated translating the TSG into parametric models in Grasshopper (GH), enabling us to test the shape grammar by selectively applying its rules.

The process of converting the shape grammar into parametric models was adopted from a research project called Digital Alberti [18], which produced a shape grammar encoding the rules for designing churches according to rules of classical architecture set by Leon Batista Alberti [19]. In order to implement Alberti's shape grammar, each of its rules was translated into a small parametric model and implemented as a modular GH definition [20].

In shape grammars, designs can be generated by recursively applying grammar rules, creating what is called a derivation [21]. A derivation of tableware collection designs according to TSG rules is shown in Figure . Since it is a parametric shape grammar, each derivation generates a family of designs that may vary according to parameter values in the rules. To this extent, a derivation corresponds to a parametric model, and consequently can be translated into a GH definition. Following the approach in Digital Alberti, each rule of the TSG was translated into a modular GH definition and enclosed into a clustered component (Figure 7). Articulating such components into different combinations creates different parametric models, each capable of generating a different family of designs.

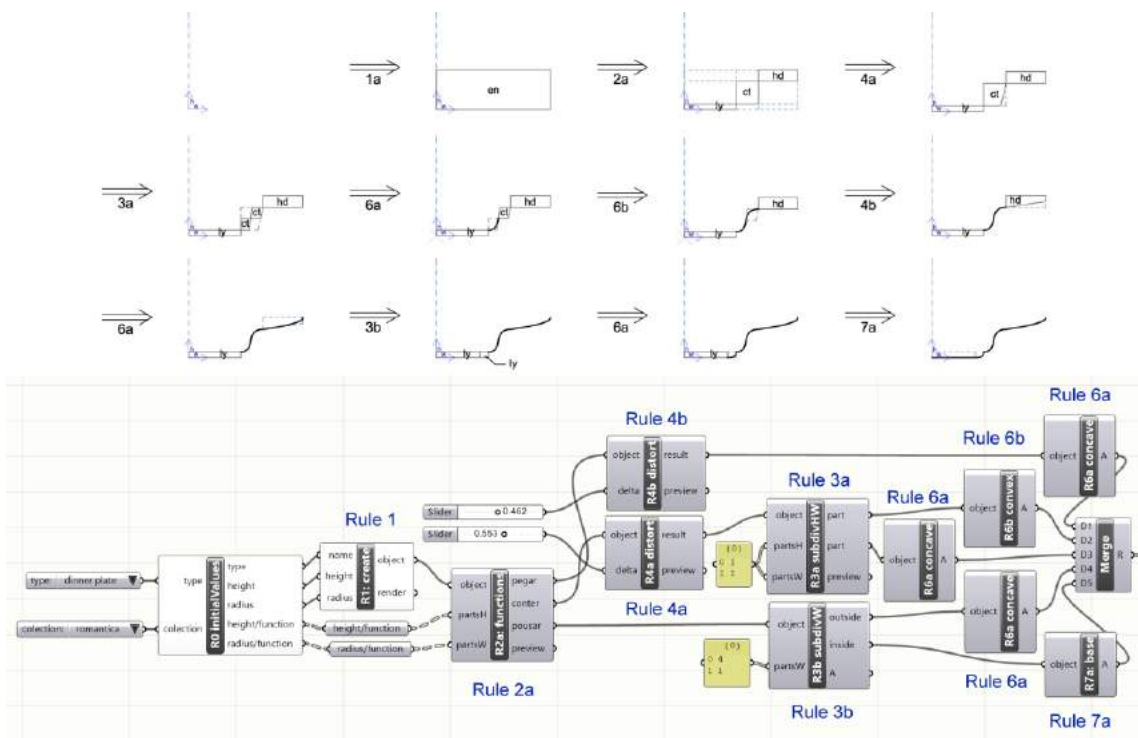


Figure 7: Tableware Shape Grammar: derivation of a soup plate's base shape: section view (top) and parametric model of a tableware element, consisting of clustered components corresponding to rules (bottom).

The set of rule-corresponding clustered components constitutes a preliminary design system that enables designers to generate derivations of tableware collections by combining those components,

and subsequently to generate designs by assigning different values to its parameters. Since the implemented design system does not perform shape recognition, the task of recognizing when and to which design elements to apply the grammar rules is delegated to the human designer. Therefore, we can consider the parametric models produced using the system to be implementations of derivations that follow the TSG rules [20].

3.3 Geometry and Behaviour of a Plaxel Object

The implementation of shape grammar rules into GH clusters begins in the lowest levels of the hierarchical structure mentioned above, namely the Plaxel class. This class defines the behaviour of instantiated plaxels through its properties, which are related to geometric attributes, and its methods. Regarding the geometry of a plaxel's shape, although its description can be divided into base shape and decoration, the Plaxel pertains exclusively to the base shape of a given element. The shape of a plaxel corresponds to a three-dimensional object, namely a set of double-curved Bézier surfaces. Each surface is generated by and described in terms of its 'profile', which can be better visualized in the section view, such as in the base shape derivation summarized in Figure , and its 'contour', which corresponds to its horizontal shape, and can be better visualized in the plan view, such as in the decoration derivation in Figure 8.

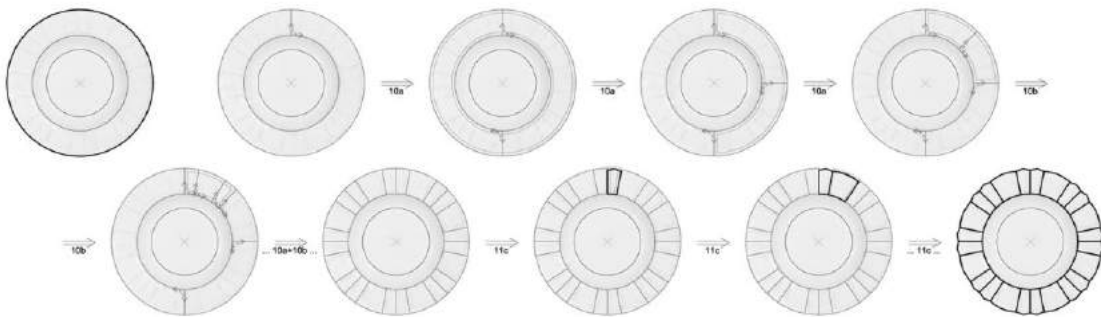


Figure 8: Shape grammar derivation of a soup plate's decoration: plan view [11].

For illustrating the generation of a plaxel's double-curved surface, consider a plate with a round contour, such as the one depicted in Figure . Although the shape of such round plate could be generated by revolving its profile around the plate's radial axis, such an approach would limit the potential of the design system to tableware designs with a circular contour, ruling out many other interesting design solutions.

Therefore, rather than revolving vertical profiles, the double-curved surface corresponding to a plaxel is generated by lofting horizontal closed curves, which we call 'guides' (Figure , left). Note that the loft operation used in Rhino for generating a plaxel uses the 'loose' option. Despite not being necessarily circular, a plaxel presents symmetry around a central axis. Therefore, its radial section, or 'profile', corresponds to a quadratic Bézier curve inscribed in an envelope (represented by dotted lines in Figure 9, right). The vertices of the envelope correspond to the guides that are used for lofting the plaxel's surface.

Regarding its behaviour, the Plaxel class defines a set of methods that can perform specific operations on an instantiated plaxel, namely: a) creating an instance of the Plaxel class ('createPlaxel'), b) changing specific attribute of a given instance ('editPlaxel'), c) replacing a given plaxel by a number of smaller plaxels ('subdivPlaxel'), or d) rendering a plaxel visible by generating a lofted surface according to the plaxel's dimensional attributes ('renderPlaxel').

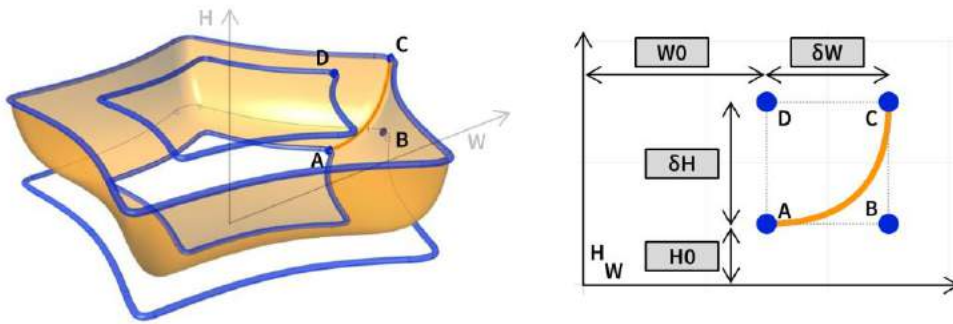


Figure 9: A plaxel's three-dimensional surface (left), and its attributes in profile view (right).

Despite having been implemented as an abstract class in VB.NET, Plaxel instances and methods can be manipulated visually in GH through the custom components mentioned previously. Custom components therefore act as containers for Plaxel methods and can be subsequently connected to other GH components. We could then make the most out of the visual nature of Grasshopper, while at the same time taking advantage of the class approach.

Finally, we can translate rules into GH clustered components. Figure 10 shows the implementation of Rule 2a of the Tableware Shape Grammar. The grammar rule divides an envelope into three smaller envelopes placed diagonally, with particular functions. A corresponding GH cluster divides a plaxel using a custom component containing the 'subdivPlaxel' method. Additional clustered components perform additional operations towards generating a tableware collection, namely by selecting the diagonally placed plaxels via standard GH 'List Item' components, then editing the attributes of the resulting plaxels via custom components containing the 'editPlaxel' method, and finally generating the corresponding surfaces via custom components containing the 'renderPlaxel' method (Figure 11).

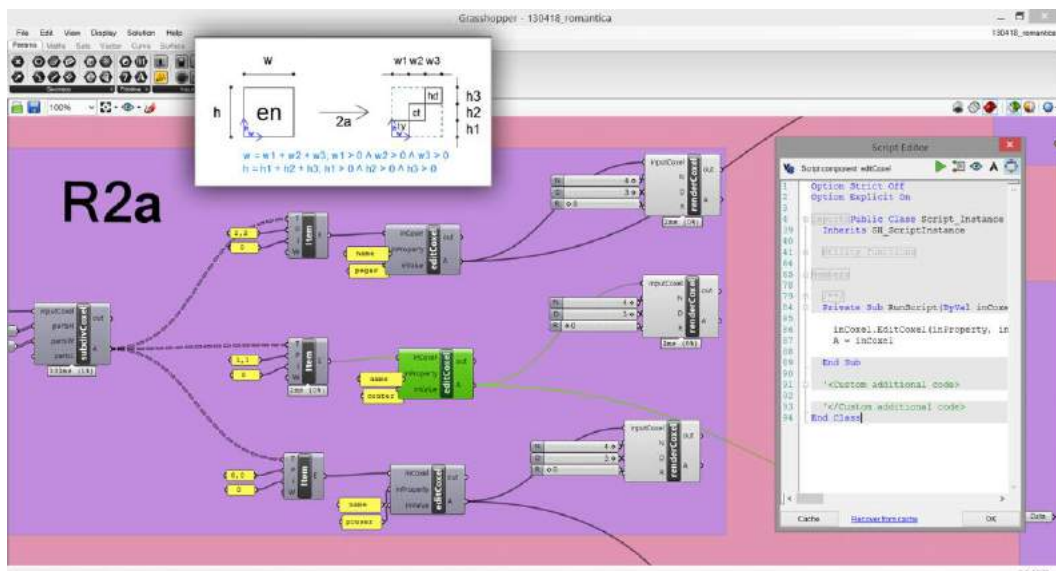


Figure 10: Grasshopper VB.NET component operating the editCoxel method (renamed editPlaxel), used for implementing Rule 2a of the Tableware Shape Grammar as a clustered component.

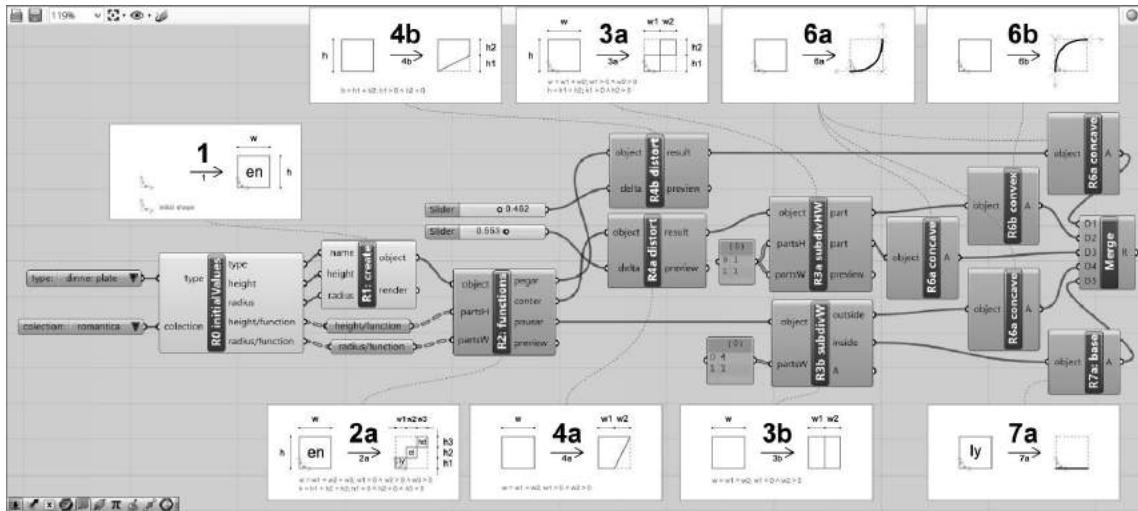


Figure 11: Parametric model of the base shape of a soup plate, with corresponding shape grammar rules.

Looking back at the Grasshopper implementation of the design system, we can identify the following hierarchical relationships among the different design elements, which reflect the hierarchical nature of the taxonomy that has been adopted for tableware design (Figure 12):

- The Plaxel class was written in VB.NET using Visual Studio 2010, defining properties and methods for a new type of object, the plaxel;
- Plaxel class methods for manipulating instantiated plaxels are implemented in Grasshopper through its custom components into elemental design operations;
- Such elemental operations are grouped and clustered together into more abstract, higher order operations, which correspond to shape grammar rules;
- Finally, such shape rules are combined into parametric models corresponding to types and meta-types.



Figure 12: Hierarchical levels of the TSG implementation.

Following subsequent developments of the shape grammar, through further analysis of additional collections [12], the GH prototype was extended with the implementation of new parametric models corresponding to rules inferred from the analysis of other collections. Such parametric models can generate the analysed collections but also new collections (Figure 13).

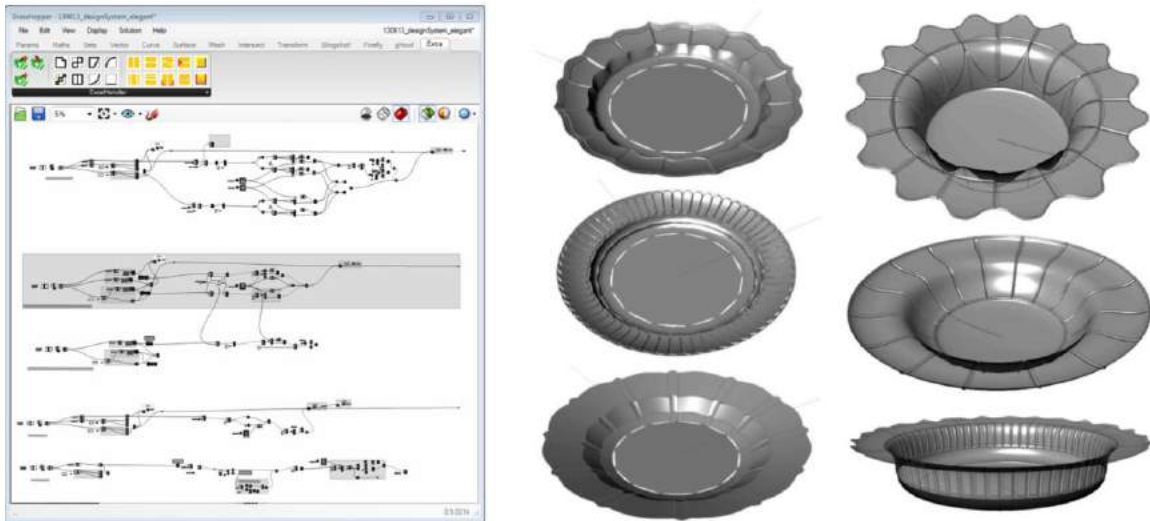


Figure 13: Implemented parametric models (left) and resulting digital models of existing (middle) and original collections (right).

4 UNITY: FROM PARAMETRIC MODELS TO DESIGN TOOLS

In this section, we present the current state of the Tableware Design System (DS), which underwent significant changes from its first iterations in Grasshopper (GH) to its current implementation in Unity and C#. C# is a textual programming language (TPL), as opposed to a visual programming language (VPL) like GH. Migrating to a TPL implied waiving the advantages of a VPL and required translating the VB.NET code that supports the Plaxel class, as well as GH's graphical components and connections into actual lines of code. However, it also meant to take advantage of the scalability provided by TPLs [14].

Besides being written in a TPL, the Unity prototype incorporates additional changes relatively to its predecessors, namely in the way the shape of tableware types is represented and manipulated. Moreover, the current prototype features new functionalities that were considered necessary, or at least useful, during subsequent implementations, such as non-circular contours or collection-wise editing features. The main motivation behind these changes derived from the insights gathered during the Contextual Inquiry performed with tableware designers. It should be mentioned that most of these changes addressed the base shape and decoration outline of tableware types.

The Unity implementation of the DS served as the foundation for developing two distinct applications related to the Design Participation Model presented earlier: Modeller (Figure 14) targets tableware designers and enables them to create customizable tableware collections, whereas Navigator targets end-users and enables them to customize tableware collections elaborated by designers. We focus on the implementation of Modeller in this paper and plan to describe Navigator in future publications.

4.1 Plaxel Geometry in the Unity Implementation

A new functionality included in the Unity implementation was the direct manipulation of guides. In the preceding prototypes, a user could change the profile shape of a plaxel indirectly, by applying operations that manipulated coordinates of its guides according to numeric values. The Unity prototype enables the user to directly and visually manipulate the coordinates of guides, by dragging and dropping handles in 3D space (Figure 15). The movement of these handles is constrained to a

plane that contains the vertical axis of the designated type, which we call the handle plane, and dragging is accomplished by projecting the mouse position onto that handle plane.

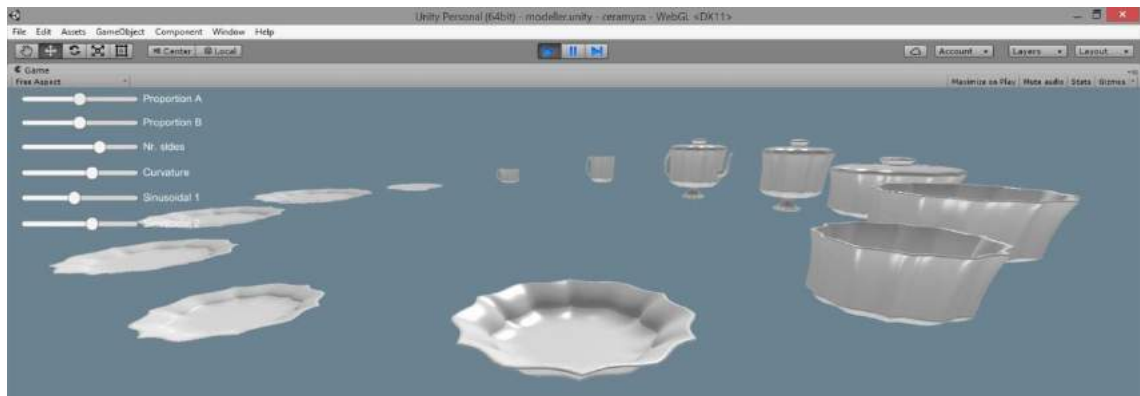


Figure 14: The Modeller application running in Unity.

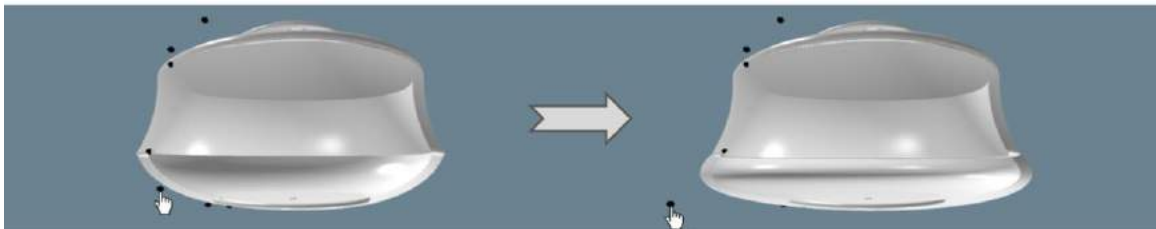


Figure 15: Defining the base shape's profile by directly manipulating guides.

The Unity implementation enabled horizontal contours other than circular by using a more versatile shape, the superellipse. The superellipse corresponds to a wide variety of shapes defined by the following equation, dubbed 'superformula' by its inventor [22]:

$$r(\theta) = \left[\left| \cos\left(\frac{1}{4}m\theta\right)/a \right|^{n_2} + \left| \sin\left(\frac{1}{4}m\theta\right)/b \right|^{n_3} \right]^{-1/n_1} \quad (1)$$

The superellipse presented many advantages for the design system, namely:

- it can generate a wide range of shapes, which include circles, ellipses and regular polygons such as a square, by varying a reduced number of parameters;
- it can be described parametrically as a function of the angle, rendering it appropriate for a representation in polar or cylindrical coordinates;
- it can produce curves with different degrees of radial symmetry; and
- it can produce curves that are continuous and periodic.

The superellipse was implemented in the DS at the guide level. Instead of corresponding to a horizontal circle, a guide is henceforth a horizontal superellipse. In the final prototype, the designer controls the superellipse of a collection by manipulating a set of sliders that define its parameters a through n_3 (see Figure 16).

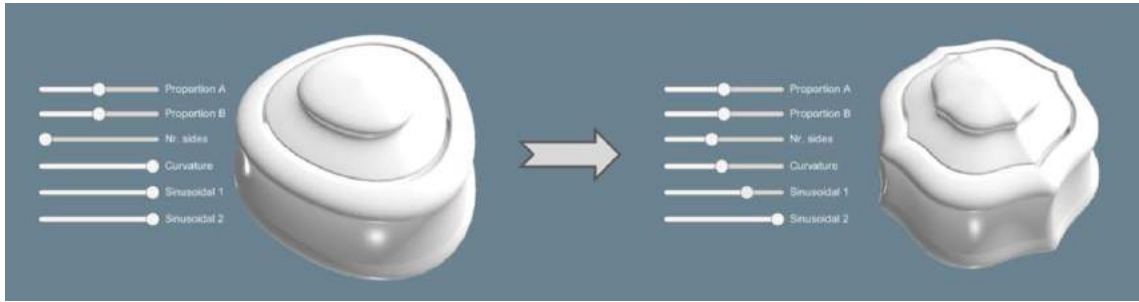


Figure 16: Defining the base shape's contour by manipulating the superellipse's parameters.

4.2 Generating Plaxels in Unity

Unity's default limitations in generating geometry in run-time can be overcome through scripting, by using mesh objects [23]. Unity allows the creation of custom polygonal meshes through scripting by means of the Mesh class in its API. The geometrical information necessary for describing a mesh follows the face-mesh data structure, consisting of an ordered list of vertices, each represented by its 3D coordinates, and a list of faces, each represented by three integers, corresponding to vertices in the vertex list [24].

To enable run-time generation of double-curved geometry in Unity, an algorithm was implemented to generate the mesh corresponding to a loft using the plaxel's horizontal guides. Contrary to the NURBS representation, the number of guides that define a plaxel's shape is not enough to guarantee a smooth mesh. Therefore, it was necessary to determine a set of additional curves called sub-guides, through which the surface is lofted. The coordinates for each sub-guide are interpolated from the coordinates of the actual guides (Figure 17).

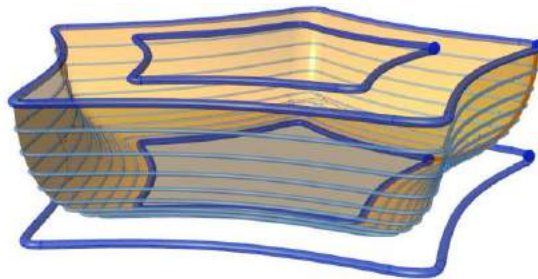


Figure 17: Sub-guides for generating a plaxel.

Instead of tracing the profile curve of the plaxel using the obtained set of interpolated coordinates, these will be used to generate sub-guides that, like guides, correspond to super-elliptical shapes. Then, points in each superellipse are sampled according to the superformula. In summary, the points of the plaxel's lofted surface follow the following cylindrical representation:

$$\begin{cases} x = W * \left[\left| \cos\left(\frac{1}{4}m\theta\right) / a \right|^{n_2} + \left| \sin\left(\frac{1}{4}m\theta\right) / b \right|^{n_3} \right]^{-1/n_1} * \cos \theta \\ y = H \\ z = W * \left[\left| \cos\left(\frac{1}{4}m\theta\right) / a \right|^{n_2} + \left| \sin\left(\frac{1}{4}m\theta\right) / b \right|^{n_3} \right]^{-1/n_1} * \sin \theta \end{cases} \quad (2)$$

in which H and W are the guides' height and width respectively. Note that H is associated to y instead of z , because in Unity the Y axis is associated to the vertical direction. By feeding the equation with a set of values for the angle θ , corresponding to the division of the trigonometrical circle by the number of desired sampled points, we obtain the set of Cartesian points needed for generating the mesh. The number of sub-guides and sampled points in each sub-guide will determine the resolution of the generated surface and, consequently, its smoothness.

4.3 Collection-Wise Design

Guides and plaxels are considered the building blocks of the DS. Let us now look at how they can be articulated into a customizable tableware collection. Recalling the taxonomy of tableware design presented earlier, a collection corresponds to the set of all available types in a ceramic tableware dinner set, such as the dinner plate, the cereal bowl, the tea cup, and so on. Each type comprises a set of parts and each part corresponds to a surface, which results from lofting its guides.

As an improvement to the GH implementation, in which it was only possible to combine plaxels into individual types, the final prototype of the DS in Unity generates a collection of tableware types. Moreover, it allows the user to change the design of the whole collection simultaneously, so that changes in the shape of a part in a specific tableware type are propagated to all the other types (Figure 18). Such feature, dubbed 'collection-wise editing', guarantees formal coherence among the collection's different elements.

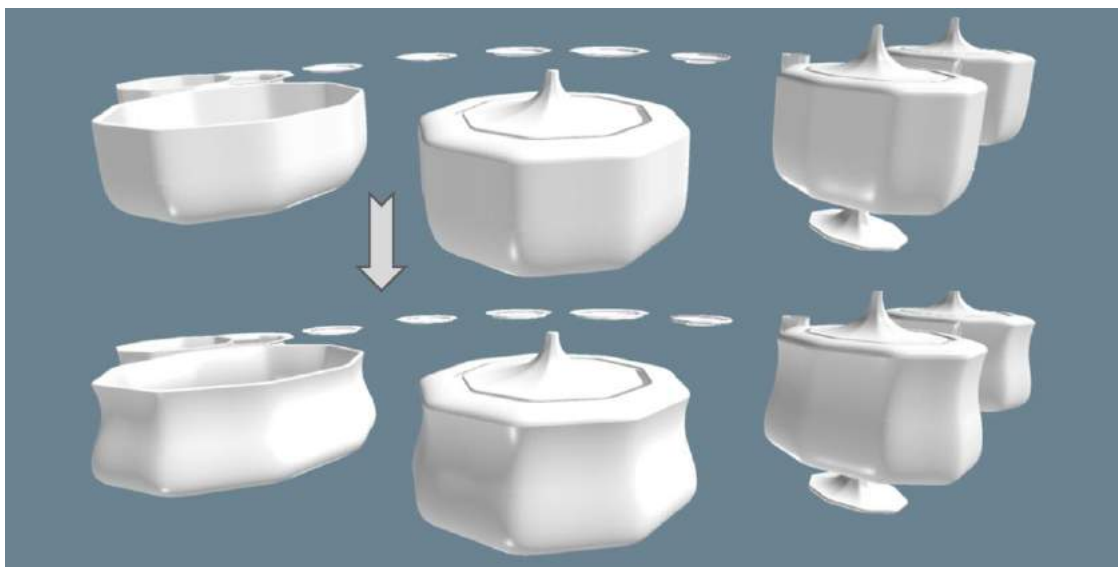


Figure 18: Collection-wise editing: changes made in one element are reflected in all the other elements.

Implementation of collection-wise editing required a unifying mechanism to enable variation in a manageable fashion. The shape grammar development suggested that the shapes of a tableware collection's types varied according to: a) the parts each type comprises, and b) the proportions of these parts. In the final prototype, this approach was taken further by adopting an Archetype, a meta-type that features all possible parts, from which it is possible to derive all types. A type is derived by hiding parts that are not featured in that type, as well as changing the proportions of the parts that are visible, according to previous observation and analysis of the case-study collections.

For example, a soup plate corresponds to a 'squashed' version of the Archetype, in which only the base, body and border parts are visible (Figure 19).



Figure 19: Types of a tableware collection as variations of the Archetype.

A collection thus comprises a set of instances of the Archetype, in which each instance corresponds to a type. The collection-wise paradigm simplified serialization, allowing storing and retrieving information about collections manipulated in the DS, namely the parametric configuration of the archetype. The archetype's shape depends on two factors: its contour shape, which is determined by the associated superellipse, and its profile shape, which is determined by the position of the guides of all its composing parts. Since these factors depend on numeric parameters, namely the superellipse parameters and the guides' coordinates, a collection's parametric configuration can be easily translated into XML format for later reproduction. Encoding the design of a collection into a parametric configuration proved particularly useful for the subsequent implementation of Navigator.

5 DISCUSSION

In this paper we described the implementation of two prototypes for the Tableware Design System, whose objective was to enable designers to create customizable tableware collections. Reporting back to the proposed Design Participation Model (DPM) presented earlier, a customizable collection corresponds to a parametric model that is handed to an end-user as part of the mass customization experience. Such parametric model is created by a designer *a priori* using the Modeler component. According to the DPM, the Modeler component receives a rule-based model as input that corresponds to the Tableware Shape Grammar (TSG). Therefore, Modeler enables the designer to create parametric models by manipulating design rules encoded in the TSG.

In the Grasshopper prototype, such workflow was implemented literally. The designer is handed a set of custom components that have a direct correspondence to TSG rules, each performing different design operations. By combining these operations, the designer creates parametric models that represent families of tableware collections. In the Unity prototype, however, correspondence to the TSG rules is less obvious and, therefore, it is pertinent to make such a correspondence explicit. In Unity, TSG rules are implemented in three different ways:

(1) embedded in automatic features of the design system; (2) translated into parameters that can be directly manipulated by the user; or (3) implemented as design operations that can be invoked by the designer, similarly to the Grasshopper prototype.

An example of productions embedded in automatic features is the initialization rules, which create a tableware type and divide it into functional parts according to its type (Figure 20). Such rules are implicit in the automatic generation of every collection types when the Unity prototype is run, according to the proportions and visible parts of each type (Figure 23). Another example is the

replacing rule (Rule 6a, Figure 21) in the base shape definition of the initial shape grammar. This makes sense while using the shape grammar, where the envelopes representing plaxels are drawn, so that other rules can be applied. The replacing rule is also explicit in the Grasshopper implementation, being encoded in the renderPlaxel method. On the other hand, in the Unity prototype, the surfaces that correspond to visible parts are always automatically generated and, therefore, the replacing rule is embedded as an automatic feature.

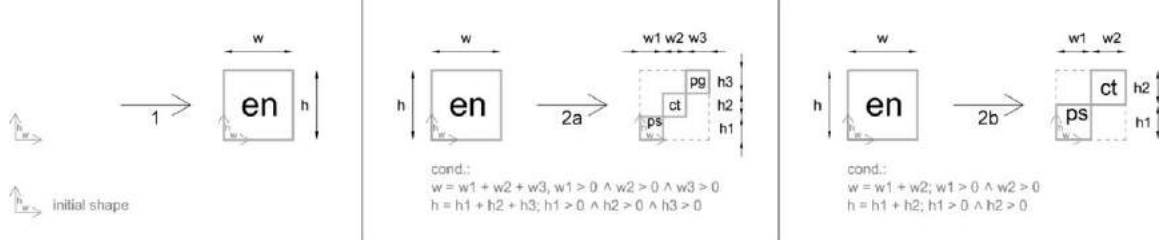


Figure 20: Tableware Shape Grammar: initialization rules.

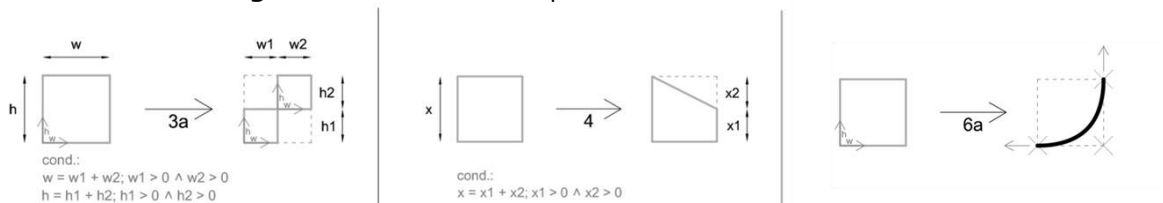


Figure 21: Tableware Shape Grammar: base shape rules.

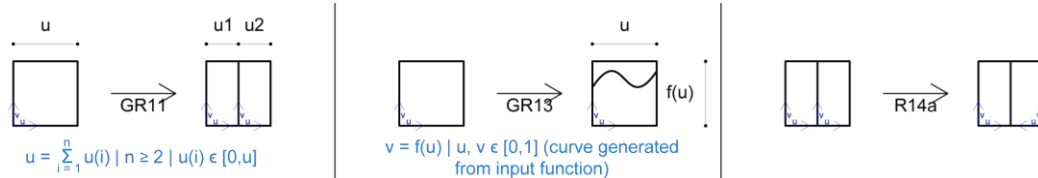


Figure 22: Tableware Shape Grammar: decoration rules related to the contour.

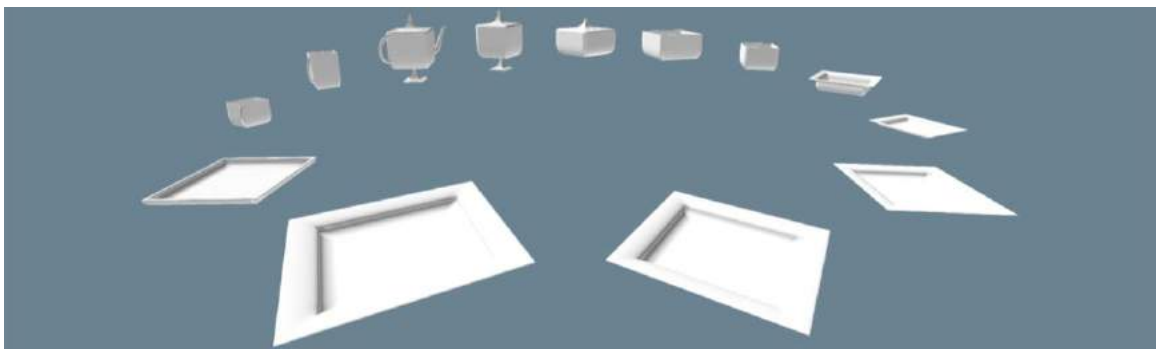


Figure 23: Default tableware collection, generated at the start of the Modeller application.

A good example of productions translated into parameters is the distortion rule (Rule 4, Figure), whose effects can be replicated by manipulating the guide-controlling handles, and subsequently altering the shape of the corresponding plaxel. Another example lies in the superellipse parameters, whose manipulation applies decoration design rules related to contours (Figure 22). Rule GR11 subdivides a plaxel along the u direction according to the number of user-specified u values, whereas the m parameter of the superellipse determines the m -fold symmetry. At the same time, the m parameter determines a secondary symmetry in alternating surfaces, mimicking Rule 14a (Figure). Moreover, the result of applying GR13 can be reproduced by manipulating parameters n_2 and n_3 , respectively dubbed 'Sinusoidal 1' and 'Sinusoidal 2'.

Finally, functionalities that replaced the original plaxel subdivision operation (Rule 3a, Figure 21), namely adding or subtracting guides, are examples of rules that were implemented as design operations. When applying these operations, the designer changes the complexity of the shape of a tableware collection and so its topology, going beyond parametric manipulation. Therefore, these operations bear a more direct correspondence to rules of the TSG. In summary, the design rules inferred in the earlier development of the design system were implemented both in the Grasshopper and the Unity prototypes. While the former corresponds to a more literal implementation of the productions, the latter embeds most of these inside semi- or fully-automated features.

6 CONCLUSION

The presented work explored in detail the strengths and weaknesses of two different development environments to implement a generative design system, in terms of both the potential of each prototype to support the shape grammar describing the system and the simplicity of user interaction. Let us now look back at the proposed Design Participation Model (DPM) in the light of the implemented prototypes. In their own ways, both prototypes contribute to demonstrating the applicability of the DPM, particularly towards extending the role of designers beyond the deployment of a mass customization platform. We can look at both prototypes from the perspectives of designer and end-user.

Starting from the end-user's perspective, we can only speculate how the Grasshopper prototype would perform for them since it was never implemented as a Navigator component. However, taking into consideration the present development of new tools like ShapeDiver and Speckle, we might hypothesize that the Unity prototype provides more design freedom than either of those tools, where user interaction is limited to selecting parameter values using a slider. In the Unity prototype, shape is determined both by sliders and by dragging control vertices. However, from a usability point of view, it is important to assess whether or not such added freedom for designing tableware is actually needed or desired by end-users.

On the other hand, from the designer's perspective, Grasshopper provides more design freedom, by allowing them to manipulate shape grammar rules directly. Although such added design freedom would certainly be more appreciated by designers than by end-users, it would require from them at least to possess some expertise on shape grammars. Nevertheless, we consider the Grasshopper prototype to be closer to the original concept of the proposed DPM, suggesting that returning to Grasshopper in future implementations of the DPM would be advantageous, namely when online publishing of Grasshopper parametric models becomes more flexible.

As this work testifies, there was not a single tool that could satisfy all implementation requirements. While Grasshopper provided a flexible CAD environment with a visual interface that could eventually be used by more advanced designers for manipulating shape grammar rules, it lacked methods to articulate the developed models with a web-based interface. On the other hand, although the Unity prototype can be deployed as a web-based interface with the potential to be tested and used by end-users, such implementation entailed additional tasks, such as implementing geometric modeling functions like lofting. Also, it implied a simplification of the TSG into a modifiable parametric model, an approach taken in previous research using shape grammars [18,25]. Other approaches using semi-immersive virtual environments could also be extended by using shape

grammars to customize generic shapes [26]. Nonetheless, beyond its suitability for interaction, the potential of the Unity implementation becomes more evident with the design system in the Navigator component that allows end-users to customize collections created using Modeler, and which will be addressed in future publications.

The framework presented in this paper will enable answering an essential question: is there a market for the mass customization of ceramic tableware? While the concept seems to be welcome by this industrial sector, judging by the interest and support provided by the Manufacturer, we cannot provide a definitive answer without testing the developed applications in a real world market scenario. Such an experiment will provide information about the actual interest of the customer base in mass customizing such type of product, as well as allow fine-tuning the delimitation of its solution space, which is one of the most important enablers for successful mass customization strategies [3]. Other interesting research directions include exploring Machine Learning Techniques to simplify grammar acquisition and generalization.

7 FUNDING

This work was supported by Fundação para a Ciência e a Tecnologia [PhD grant number SFRH/BD/88040/2012, research grant UID/CEC/50021/2013], by the Stuckeman Center for Design Computing and by the Stuckeman School of Architecture and Landscape Architecture.

Eduardo Castro e Costa, <http://orcid.org/0000-0002-3113-9270>

Joaquim Jorge, <http://orcid.org/0000-0001-5441-4637>

José Duarte, <http://orcid.org/0000-0002-3826-3987>

8 REFERENCES

- [1] Sanders EB-N; Stappers PJ: Co-creation and the new landscapes of design, *CoDesign*, 4, 2008, 5–18. <https://doi.org/10.1080/15710880701875068>
- [2] Heiskala M; Tiihonen J: Mass customization with configurable products and configurators: a review of benefits and challenges, *Mass Customization Information Systems in Business*, 2007, 1–32. <https://doi.org/10.4018/978-1-59904-039-4.ch001>
- [3] Salvador F; Martin de Holan P; Piller F: Cracking the Code of Mass Customization, *MIT Sloan Management Review*, 50, 2009, 71–78.
- [4] Piller F; Salvador F: Design Toolkits, Organizational Capabilities, and Firm Performance, In: Harhoff D, Lakhani KR, editors. *Revolutionizing Innovation: Users, Communities, and Open Innovation*. 1 edition. Cambridge, MA; London, England: The MIT Press, 2016.
- [5] Piller FT; Schubert P; Koch M; et al.: From Mass Customization to Collaborative Customer Co-Design, *Proceedings of the European Conference on Information Systems (ECIS) 2004*. Turku, Finland, 2004.
- [6] Franke N; Piller F: Value Creation by Toolkits for User Innovation and Design: The Case of the Watch Market, *Journal of Product Innovation Management*, 21, 2004, 401–415. <https://doi.org/10.1111/j.0737-6782.2004.00094.x>
- [7] Castro e Costa E: Managing Complexity in Mass Customization through Design Space Subdivision: A Case Study in Ceramic Tableware Design [PhD in Architecture], [University Park, PA]: Pennsylvania State University; 2018.
- [8] Ken Research: Europe Ceramic Industry Outlook to 2018 - Rising Consolidation and Favorable Government Regulations to Lead Towards Industry Transformation [Internet], Ken Research; 2014. p. 188. Report No.: KR228. Available from: <https://www.kenresearch.com/manufacturing-and-construction/construction-materials/europe-ceramic-market-research-report/554-97.html>.
- [9] Duarte JP: Synthesis Lesson - Mass Customization: Models and Algorithms [Habilitation Candidacy Exam], [Lisboa]: Faculdade de Arquitectura, Universidade Técnica de Lisboa; 2008.

- [10] Duarte JP: Mass Customization: models and algorithms, Ljubljana, Slovenia, 2011.
- [11] Castro e Costa E; Duarte JP: Tableware Shape Grammar, In: Stouffs R, Sariyildiz S, editors. Computation and Performance – Proceedings of the 31st eCAADe Conference [Internet]. Faculty of Architecture, Delft University of Technology, Delft, The Netherlands; 2013, 635–644. Available from: http://cumincad.scix.net/cgi-bin/works/Show?_id=ecaade2013_126.
- [12] Castro e Costa E; Duarte JP: Generic shape grammars for mass customization of ceramic tableware, In: Gero JS, editor. Design Computation and Cognition DCC'14 [Internet]. Springer; 2014, 437–454. Available from: http://link.springer.com/chapter/10.1007%2F978-3-319-14956-1_25.
- [13] Rutten D: Grasshopper [Internet], 2007 [cited 2016 Feb 4]. Available from: <http://www.grasshopper3d.com/>.
- [14] Leitão A; Santos L; Lopes J: Programming Languages for Generative Design: A Comparative Study, International Journal of Architectural Computing, 10, 2012, 139–162. <https://doi.org/10.1260/1478-0771.10.1.139>
- [15] McNeel B: Rhinoceros [Internet], McNeel; 1998 [cited 2016 Feb 4]. Available from: <http://www.grasshopper3d.com/>.
- [16] Krill P: HTML5 finally reaches official status [Internet], InfoWorld. 2014 [cited 2017 Jun 26]. Available from: <http://www.infoworld.com/article/2839557/application-development/html5-finally-reaches-official-status.html>.
- [17] Chau HH; Chen X; McKay A; et al.: Evaluation of a 3D Shape Grammar Implementation, In: Gero JS, editor. Design Computing and Cognition'04. Dordrecht, The Netherlands: Kluwer Academic Publishers; 2004, 357–376. https://doi.org/10.1007/978-1-4020-2393-4_19
- [18] Duarte JP; Figueiredo B; Castro e Costa E; et al.: Alberti Digital: investigando a influência de Alberti na arquitetura portuguesa da contra-reforma, In: Brandão C, Furlan F, Caye P, et al., editors. Na gênese das racionalidades modernas: em torno de Alberti. Editora UFMG. Belo Horizonte, Brazil, 2013.
- [19] Figueiredo B: Descodificação do De re aedificatoria de Alberti: Gramáticas de forma para a análise e geração de edifícios sagrados [PhD], Univesidade do Minho, 2016.
- [20] Figueiredo B; Castro e Costa E; Duarte JP; et al.: Digital Temples: a Shape Grammar to generate sacred building according to Alberti's theory, Future Traditions, Rethinking Traditions and Envisioning the Future in Architecture Through the Use of Digital Technologies [Internet]. Porto: FAUP Publicações; 2013, 63–70. Available from: http://cumincad.scix.net/cgi-bin/works/Show?_id=ecaade2013r_004.
- [21] Stiny G: Shape: talking about seeing and doing, MIT Press, 2006. <https://doi.org/10.7551/mitpress/6201.001.0001>
- [22] Gielis J: A generic geometric transformation that unifies a wide range of natural and abstract shapes, Am. J. Bot, 90, 2003, 333–338. <https://doi.org/10.3732/ajb.90.3.333>
- [23] Unity: Unity - Manual: Meshes [Internet], 2015 [cited 2016 Apr 14]. Available from: <http://docs.unity3d.com/Manual/class-Mesh.html>.
- [24] Smith C: On Vertex-vertex Systems and Their Use in Geometric and Biological Modelling [PhD], [Calgary, Alta., Canada, Canada]: University of Calgary, 2006.
- [25] Barros M: Mass customisation in the furniture design industry: the case of Thonet chairs [Internet] [PhD], [Lisbon, Portugal]: University of Lisbon, 2015. Available from: <https://www.repository.utl.pt/handle/10400.5/10732>.
- [26] De Araújo BR; Casiez G; Jorge JA; et al.: Mockup Builder: 3D modeling on and above the surface, Computers & Graphics, 37, 2013, 165–178. <https://doi.org/10.1016/j.cag.2012.12.005>