Juxtaposed Designs Models:
A Method for Parallel Exploration in Parametric CAD

Halil Erhan \(^1\) and Naghmi Shireen \(^2\)

\(^1\,^2\)School of Interactive Arts and Technology
Simon Fraser University, Surrey, Canada

\(^1\)herhan@sfu.ca, \(^2\)naghmi.shireen@sfu.ca

Abstract. Computational tools mainly support authoring single-state models, which fall short in enabling designers to work with multiple solutions side-by-side. This is a natural design behaviour commonly observed when designers use other media or improvise digital tools to explore alternatives. In this paper we attempt to formalize a method that aims to help designers to create multiple design alternatives derived from a base parametric model and its controllers. The goal is to change alternative designs such that each alternative can respond to changes as their internal structures allow. We present five assumptions on the tools that this can be achieved and also a parametric design pattern to be used in similar situations. Despite the complexity of the models, we can demonstrate the possibility of working with multiple solutions in architectural design.

1 Introduction

The creative task of developing a design solution demands experimenting with alternative ‘what-if’ scenarios and reflecting on each solution generated. However, most tools today allow designers to interact with a single-state model hence provide little direct support for working with multiple solutions simultaneously. On one hand design research reveals the significance of working with alternative designs for informed exploration and better comparison; on the other hand, the tools available to support these activities fall short in meeting designers’ needs. To work with design alternatives, designers adopt ad-hoc techniques, such as opening two files side-by-side, layering for superimposed comparison; copy-pasting parts of the solutions into a new file; or saving versions of design files. None of these improvised techniques provide global operators for linking multiple design alternatives to work with them simultaneously without losing design states.

We aimed to develop a method for parametric modeling that can enable parallel editing of multiple design variations. The method we propose assumes that a design model can be replicated through a deep-copy function \([1\). This is the case in most CAD models particularly using propagation-based representations using graphs \([2\). By the deep-copy, we mean that the model elements in the graph can be partially or entirely replicated within the same design file. One way of achieving this is using
conventional copy-paste followed by renaming parameters. Although we observe improvised techniques by CAD modelers who duplicate models and edit them through ad-hoc solutions, each modeler’s technique differs, and there is no shared and common approach across different designs. To our knowledge, we believe that the method proposed in this paper is the first attempt towards a formalization and possibly can influence the software structure and functionality of the future CAD systems.

In this paper, we formalize how multiple subjunctive designs can be set together to respond the changes in parametric variations. Building our previous research on parametric modeling such as ViSA [1] and graphed-based editing [2], we propose using modular controllers that can be associated with local and global parameters and changes the respective models simultaneously. We used GenerativeComponents (GC) [3] as our test platform because of its capability of using multiple model-views with some degree of decoupling in a single design model. That is, in the same file multiple designs can be created partially-independent. Among possible controller types, we present continuous linear controllers to change the size and number of design features; continuous curvilinear controllers to change the non-unified spread of features; and discrete controllers to change the visibility. At the end of the paper, we present a ‘parametric design pattern’ that structures the method to be used in design.

2 Motivation

This study is part of a research program on representing and interacting with alternatives in design. The larger goals are related with modeling simultaneous independent scenarios, comparing them, and manipulating them independently or collectively [4] [5] [6]. Lunzer and Hornbæk [7] call such interfaces subjunctive, which term we use here. Briefly, a subjunctive model is a derived model from another to execute ‘what-if’ scenarios. A subjunctive model inherits all properties (nodes and dependencies) from a prototype model; can override these properties, and add new properties; define parametrically or structurally different solutions; can compose properties from several prototypes.

Parametric CAD (pCAD) systems receive a single-state limitation from the CAD tools on which they are largely based. This approach limits exploration and comparison of variations. Although there are studies on exploring variations using representations resembling the intended result [8] [9] [10] [11], to the best of our knowledge, there is limited research on how interaction with them can be achieved.

Smith et al. [12] focus on CAD interfaces for generating and managing multiple ideas. They report a set of suggestions based on their empirical findings to improve existing systems: a) Make it easy to switch between ideas, b) Provide a way to view multiple ideas at once, c) Allow users to adapt the interface to their needs and preferences, d) Provide ways to label the ideas both pictorially and textually, e) Provide multiple ways to group and classify the ideas, f) Provide an explicit means for capturing the situation, and g) Support fluid composition and decomposition of ideas.
Combined with the qualities of creativity support tools, these give an overall direction for development of the next generation CAD tools to better support design [13].

3 Proposed Method

The method is built on two perspectives to design model creation: structural and functional. In structural perspective, we made the following five assumptions for a system to support parallel exploration:

**Assumption 1:** Any given parametric design model $D_1$ should be controllable by a tuple of arbitrary number of parameters $DP_1 \{p_1, p_2, p_3... p_n\}$.

**Assumption 2:** The model’s structure can be deep-copied as described such that $D_2$ can be entirely or partially derived from $D_1 (D_2 \subset D_1)$. The derived $D_2$ should have its own parameters $DP_2$ that are equivalent and subset of $DP_1 (DP_2 \subset DP_1)$, i.e. $DP_2$ and $DP_1$ share the same semantics and exists independently from each other. The equivalency criterion allows changing parameter names without losing their function in the model. This and the first structural assumptions state that $n$ number of partially or entirely identical design models ($D_1, D_2, D_3...D_n$) can co-exist in the same editing environment.

**Assumption 3:** All independent parameters ($DP_1, DP_2, DP_3...DP_n$) are accessible in the editing environment simultaneously such that they can be together or selectively changed, e.g., through a function call or command. The third assumption is crucial for parallel editing that multiple models can be changed together through shared controllers.

**Assumption 4:** During the changes the models should keep their integrity, i.e. their constraints are not violated. In case of constraint violation, visualization elements on the model should signal such situations, or the model’s behavior must be obvious to deny such change.

**Assumption 5:** Each model in the same environment should be amenable to receive new features, drop and modify existing features as design elements, controllers, or visualizations. This is needed to enable adding new features to a design model independent from the others to ensure that a design is not a mere parametric variation of the original one. However, it must be noted that the new features added to a model can also be shared with others if desired so, in this case the corresponding parameters and controllers are needed for parallel editing. Similarly, dropping a feature from a model would require to drop the corresponding parameters and controllers from the parallel editing view. Among the five assumptions, this is the most challenging and difficult one as the complexity of design increases, the models’ structure may hinder this assumption.

The assumptions on structural perspective fundamentally guarantee modularity and decoupling the models, controller and views as in object-oriented and proto-type based software design. In our earlier work, we demonstrated that this can be possible in the current pCAD systems [1], and we formalized a method that can be followed to create modular parametric controllers and visualizations to plug-in and –out to a design model to meet the designer’s specific decision-making needs.
The functional perspective of the method assumes that the modeler can view all derived models, controllers and visualizations simultaneously in the same editing environment. Also, the modeler must be able to turn on or off a model’s response to changes in a parameter while other models respond the change. The required functional features are: (a) derive a model or part of model to create new variations, (b) activate a derived model to respond to changes globally or deactivate to take local changes only, (c) introduce new controllers and visualizations globally and locally, and (d) substitute controllers or visualizations with minimum effect on the integrity of the models. Below we briefly demonstrate how these can be manifested in a hypothetical design case.

4 Case Study: Conceptual Design of a Building

For demonstration purposes, we present a case study that demonstrates exploring conceptual designs of a residential building with three major spatial features: residential units, vertical circulation spaces, and terraces. Each of these features is modelled using parametrically defined boxes (Figure 1). The possible solutions that can be generated using this model can be highly diverse (Figure 2). Please note that, not all parametric combinations would result with a plausible design, but rather the goal here is to create a design space that can inspire form finding. We chose using this model in this case study for two reasons: the abstraction level of the model makes it open for interpretation and resembles mass modeling in the early phases of architectural design; and simplicity of the unit geometry (boxes) can reduce the complexity of the parametric model structure such that the method can be clearly described.

![Image](https://example.com/image.png)

**Fig. 1.** A building concept defined using 15 independent parameters. The size and position of each unit type can be parametrically adjusted based on the building size constraints.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Length (Max)</td>
<td>2 - 14</td>
<td>2.0</td>
</tr>
<tr>
<td>Building Width (Max)</td>
<td>2 - 14</td>
<td>2.0</td>
</tr>
<tr>
<td>Building Height (Max)</td>
<td>2 - 14</td>
<td>2.0</td>
</tr>
<tr>
<td>Cube-Grid (X, Y, Z)</td>
<td>2 - 10</td>
<td>2.0</td>
</tr>
<tr>
<td>Resident (Gray)</td>
<td>0.2 - 5</td>
<td>0.2</td>
</tr>
<tr>
<td>Circulation (Red)</td>
<td>0.2 - 5</td>
<td>0.2</td>
</tr>
<tr>
<td>Terraces (Green)</td>
<td>0.2 - 5</td>
<td>0.2</td>
</tr>
</tbody>
</table>
In an example scenario, the designers would be changing the unit dimensions and their spread in grid one unit type at a time. For example, if the designer wants to increase the size of the terraces, the corresponding parameters will be changed using the controllers associated with them. This is going to cause losing the previous design state and only working on the most current state. It is problematic as there is no mechanism that can enable designers to compare each state in a juxtaposed view, i.e. side-to-side. Also, in this scenario it is not possible to experiment with what-if scenarios with multiple states visible and editable independently at the same time.

**Fig. 2.** A small sample possible design variations that can be created using the base model

### 5 Parametric Structure of the Base Model

The parametric base model is composed of three types of boxes corresponding three different architectural units: residences, vertical circulation, and open spaces. Instances of these units are placed in a 3D orthogonal 4x4x4 grid (Figure 3). The number of each unit type and their size along with their spread can be controlled parametrically using a continuous linear function. The distance between the same type of features in the model is defined using continuous flexible function to allow non-uniform spread of consecutive features. Hence, each unit type can be significantly of different proportions, location, and spread. The overall building size can also be parametrically defined, which the units fill the bounding box spanning the corresponding grid. This can be analogous to a bounding box of a complex geometry. The inclusion and appearance of select features are decided based on a proximity
controller using a discrete function. The parametric structure we developed responds to these changes selectively or collectively, and is managed by the designer.

The independent parameters controlling the architectural features in the model are accessed through custom controllers we developed. The controllers are of three types: linear and flexible. The linear controllers change the parameter it is associated with by moving a point on a line, it has a fixed slope. The flexible controllers, on the other hand, are built using a spline curve with two control points, which the values assigned to a parameter can arbitrarily change in response to the curvature of the spline.

The parametric base model is composed of three types of boxes corresponding to different architectural units: residences, vertical circulation, and open spaces. Instances of these units are placed in a 3D orthogonal 4x4x4 grid (Figure 3). The number of each unit type and their size along with their spread can be controlled parametrically using a continuous linear function. The distance between the same type of features in the model is defined using continuous flexible function to allow non-uniform spread of consecutive features. Hence, each unit type can be significantly of different proportions, location, and spread. The overall building size can also be parametrically defined, which the units fill the bounding box spanning the corresponding grid. This can be analogous to a bounding box of a complex geometry.

The inclusion and appearance of select features are decided based on a proximity controller using a discrete function. The parametric structure we developed responds to these changes selectively or collectively, and is managed by the designer.

---

**Fig. 3.** Parametric structure of the base model

**Fig. 4.** Linear controllers (lines above), and flexible controllers as splines changed by two control points (boxes below)
6 Structure for Parallel Editing

The systems for editing a single model, by description, is not intended for creating multiple alternatives. For example, when working on a geometry on a vector graphics editor, the geometry is viewed only in its current state. Working on multiple states of the geometry, e.g. with different size properties, is not possible. The current pCAD tools are no exception to this. Therefore, the method we propose improvises the tool features of a parametric modeling system, namely GenerativeComponents from Bentley, to approximate to an environment that can edit multiple solutions parallel in juxtaposed views. This particular system enables creating up to eight model-views, each of which can contain a different model as part of a global model. The graph representation can be contained in one shared model-view. Each model-view can be named separately and represent an alternative for parallel editing.

Figure 5 shows the editing environment we configured for working with design alternatives in parallel. Due to the limitations in the number of model-views available in the parametric system we used, and rapidly increasing model complexity, we chose to work on three design alternatives. Each alternative used two model-views: one for the design model and another for placing controllers. One model-view is used to place controllers that can change the parametric values of the three alternative designs at the same time. However, this happens only if an additional global Boolean parameter to each alternative is switched to true, we discuss this below.

![Fig. 5. Editing environment of alternatives in juxtaposed views](image)

When creating the two alternative models from the base model, we applied the following process. First, we created a new base coordinate system in a new model view and copied the parametric model and pasted it into the graph. In the copied model, we replaced the base coordinate system with the coordinate system in the new model-view. Now that the new model can show on a separate view than the base model, we could rename all parameters it contained, e.g. from Base_height to Alt_2_height. These two moves are consistent with the second and third assumptions we stated earlier. We copied the original controllers and placed them into a separate model-view, similar to the deep-copy of a geometric model. We can do this as these controllers are treated same as any other parametric features as part of the internal data. Following renaming the controllers, we associated them their respective elements in the model by linking in the graph. In the same environment, we now have
two solutions that are independently changeable. Following the same process, we created the set up for editing the third alternative. To the best of our knowledge, a formalized modeling method has not been documented before, although similar techniques are shown on particular design cases.

The most crucial part of the method is how to link all these models and have them response to changes together, if and when needed. For this, we introduced a ‘global’ Boolean parameter as part of each alternative design. The role of this parameter is to switch if the model should response to global edits intended for parallel editing. For example, if this switch is on, the model will ignore the local parameters and response to changes triggered in the parallel controllers (Figure 5). Up to know, the entire parametric model has already become more complex than we expected. Figure 6 shows this complexity on the graph. The method’s applicability under the existing parametric software systems becomes highly difficult when the model’s complexity increases. We believe that these systems should consider new functions to alleviate their bottlenecks.

![Graph model generating three alternative design models and corresponding controllers as well as a model view for global controllers](image)

Although the structural criteria of the models were complete after these parametric modeling moves, we have yet to control the models: this required implementing a functional schema using an available technique in the selected parametric system: functional features and scripting. However, this was not as complex as the structure of the model. The following pseudo code presents the logic behind the functions:

```plaintext
ON_CHANGE (Global.<P_name>)
  FOR EACH a in Model //a: alternative with distinct baseC
    IF a.updateGlobally is true
      a.<P_name>.value = Global.<P_name>.value
      UPDATE (a.controller(<P_name>))
      UPDATE (a)
    END IF
END FOR
```

Please note, despite the simplicity in logic of this function, the implementation of this logic took more than 200 lines of code. We had to implement the following
subroutine to cover 6 different conditions for each parameter (permutation of two states for each alternative to set response to the changes in the global controllers). Although it was a labor-intensive task, we only had to copy-paste the code and edit them. Once again, it is worth to note the difficulty of using the system features that are not designed for editing multiple solutions.

Fig. 7. Snippet from the function that updates the corresponding parameter in each alternative following a change on a global parameter

7 Parallel Editor Pattern

Following the lessons learned when building the model for parallel editing, we decided to convert our experiences into a (parametric) design pattern that can be shared with other designers who wish to follow this method. A pattern—in our context—is essentially a generic solution to a recurring problem in a particular domain. For example, linking a geometric element to control its properties is a design pattern, which is commonly known as Controller Pattern. There are four salient features of a pattern [1413].

Explicitness: Explicitness aids reflection. Tools for advancing design skills. To write a pattern is to commit to a definite media for others to read in your absence.

Partiality: They are above nodes, below design. Properly written they are informal devices by which modules can be expressed in principle.

Problem focused: They state a problem with several clear solutions to it. They aid sketching by accelerating the creation of approximate models. They often combine geometrical, mathematical and algorithmic insights.

Abstract/Generic: Lastly they are abstract. To use them evidences mastery of the ‘divide’ part of divide and conquer.

Name: Parallel Editor

Related Patterns: This pattern can be implemented on top of all existing parametric patterns:
• Slider, Dial, Equalizer, Controller, Hyperboloid of one sheet, and Azimuth Altitude

**Intend:** Whenever there is a need to edit multiple models simultaneously to see the effects of a change side-by-side  
**Use When:** Exploring alternative solutions through applying what-if scenarios  
**Why:** Overcoming the challenges a single-state system poses when exploring design alternatives  
**How:**  
1. Build partial or complete base model enough to explore its variations  
2. Deep copy the base model with its controllers that the new model will have its own base coordinates.  
3. Rename the parameters and other model elements considering their unique place as an alternative solution  
4. Add a unique global Boolean parameter to the alternative to indicate its state for following local or global edits  
5. Do step 1-4 for each alternative  
6. Deep copy the base controllers to set the global controllers  
7. Implement a global update function that updates alternatives if their update switches are on  
8. Link function to the global controllers.  

**Global Samples:**  
1. A controller with multi-handles that can change the length of the line, changes the t-value for each controller point, hence the geometrical model attached to each controller point simultaneously.  
2. A controller with multi-handles plus a switch to turn updates on and off (two modes), such that all alternatives have an individual switch. When the switch is turned on, the models becomes active on parallel editing mode. Therefore, changing the length of the controller line, changes all models in parallel editing mode.  
3. A controller with multi-handles, a switch, and a parameter for retaining last value (recorder). When it comes back from its parallel editing mode to individual editing mode, it retains its last parametric value.  
4. Four-point law curve controller that can be used as flexible controller.  

## 8 Conclusion

The method we introduce in this paper is in its infancy. The main challenges include the lack of build-in functions in pCAD tools that can support direct deep-copy and functional feature libraries that be used in any given CAD model. Although the latter one can be overcome through a design community using ‘compiled’ or ‘generative’ components, the prior needs a direct involvement of the CAD companies. In this research, while we aim to demonstrate the utility of working on multiple solutions in
design exploration we also wish to discover the new functional and structural features of the future CAD tools. In our previous work, we presented some of such functions for graph-based editing [2]. In this method, we attempted to apply some of the ideas presented in an existing CAD tool.

References