ABSTRACT

Designers devise and explore alternatives as they work. Current parametric modeling tools can work against this essential feature of design work: they require advanced skills in constructing hierarchies of elements and defining relationships among them. Successful exploration is contingent on designers’ ability to interact with complex parametric representations. As a consequence, designers often make hard commitments to design decisions at an early stage of the design process, when they have not yet found potentially viable designs, nor have they received enough feedback about the alternatives they are exploring. Parametric modeling shares many activities with text-based content creation such as writing, for example, combining elements in different sequences, composing hierarchies of elements, varying properties, binding entities through relationships, replacing content, cross-referencing elements, and assembling wholes from parts. The similarities in both domains, and the simpler context of text, inspired us to build a text-editing prototype that focuses on developing a hierarchical, multi-state, data model that supports parallel exploration of alternatives. Unobstructed by the complexity of parametric modeling, we developed Alt.Text. We hypothesize that the features of Alt.Text are portable to other domains, not only that of parametric modeling.
INTRODUCTION AND BACKGROUND

Research in design space exploration focuses on strategies and systems to amplify designers’ abilities in creating and managing alternative solutions (Woodbury and Burrow 2003; Woodbury and Burrow 2006). Expert designers explore solutions by a breadth-first-depth-next search strategy (Akın 2009), allowing them to gain better understanding of the design space that they are working with. They manage the complexity of search by merging smaller, more manageable, solutions to formulate larger ones. This is both an iterative and a recursive problem-solving strategy. It is well-described in Simon’s theory of complexity (Simon 1962) and is widely accepted in the computational design field. It aims at finding “satisficing” paths, ones that will “permit satisfaction at some specified level of all requirements” (Simon 1956).

The ability to formulate solutions by combing wholes or parts of other solutions hinges on the representational model being used. It needs to accommodate creating, mixing, and reusing existing descriptions to capture the hierarchy of newly formed solutions. In order for designers to assess progress, they need to be able to compare the difference between what is required and what is attained. Comparison is a key mechanism for decision making (Krishnamurti 2006). This mechanism can be carried over multiple alternatives in parallel to attain higher degrees of success (Woodbury and Burrow 2006). Lunzer and Hornbæk (2008) assert the value and effectiveness of simultaneously viewing multiple results and of separate feedback visualizations for what-if scenarios.

Designers’ ability to explore and compare alternatives is sharply bound by their memory limitations, for which the only relief is to be found in external media. Yet current media are almost exclusively single-state, that is, they support working with one solution at a time. Further, they hinder designers’ ability to combine or “mix and match” solutions from different explorations by elaborate operations. Despite their inherent ability to carry variations, parametric models are nearly universally presented through single-state interfaces.

LIMITATIONS IN PARAMETRIC MODELING TOOLS

Parametric modeling tools bring a wide range of possibilities to any design exploration. However, they tend to be used mostly by those with high technical skill. This can be attributed to their steep learning curves, the elaborate process by which parametric models are made, and the low degrees of flexibility found when changing developed models. Further, their display systems are limited to showing only a single instance or configuration at a time, even though the element being displayed is parametric and is capable of permitting multiple changes—each of which is distinct from others. All of these limit the ability to explore the solution space and compare solutions.

These limitations, and many others, are not only due to computational costs coming from the parametric models’ representations (Woodbury and Burrow 2006), but are also due to the user-interactions accompanying them, and the data-model being used.

SCOPE OF ALT.TEXT

We tackle these issues in the context of writing that is, authoring text. While writing and design differ, the goals behind exploration in either one are similar: being able to compose solutions using other sub-solutions, view and compare content at any point, branch out in the exploration process without having to commit to decisions, and achieve the required results through simple and non-destructive interactions within the content, cross reference elements, focus-in-context to work with solution-subsets while keeping the rest constant, reduced setup time prior to commencing content creation, and among others. Our attempt is presented in this paper as a software prototype that we call Alt.Text.

PARAMETRIC DESIGN AND TEXT-DOCUMENT DESIGN: A COMPARISON

The processes of constructing parametric models are deeply similar to creating text documents. In essence, a text-document is an assembly of passages, the content of which vary in style, composition, and substance. Writers, or document-designers, constantly explore activities such as:

1) inclusion or omission of passages;
2) embedding of certain phrases and varying their wording;
3) changing transitions between passages; and
4) changing visual arrangements and styles.

PARAMETRIC MODELING EXAMPLE

(Figure 1) depicts a very simple, yet typical, parametric modeling approach as argued in (R. Woodbury 2010)—a designer setting controllers and jigs to create surfaces. In the figure below, there are eight possible paths of exploration and an unlimited number of variations within a given path. However, currently available parametric modeling tools provide display for only a single state, a set of parameter values, at a time.
WORK-AROUND METHODS TO EXPLORE ALTERNATIVES

Common workflow methods for exploring alternatives in today’s tools include the following: use of layers, duplicating content or whole files, creating clones, using the “Undo” command to recover past states, and assembling multiple instances of the same artifact.

While such methods allow progress in exploration, they impose restrictions in different ways. For example, duplicated content has no connection to the sources from which it was duplicated; cloned content is not directly editable, relying on the “Undo” stack of operations requires paying careful attention to the order of operations to avoid loss of work.

All methods consume cognitive resources that are better spent on designing. (Yamamoto and Nakakoji 2005) describe this situation from a user-interaction stand point, advocating that designers should be dealing with content, that is, with interfaces that reveal the structure of that content, rather than an imposed, external structure as is common in most extant systems. Further, the authors provide a number of guidelines for developing tools that support creative exploration focusing on the interaction and the use of multiple representations that can inherit different meanings throughout the exploration process. This is also argued in (Woodbury and Burrow 2006).

A more advanced workflow is the use of Version Control Systems (VCS) such SVN, CVS, or Git. While these systems support multiple-states, they still require designers to work in a linear fashion due to the way they handle file versions. Users can work on one version at a time and, depending on the VCS, use versions as restore points, or as a way to create a tree with branches—a model that limits the exploration to one branch at a time.

RELATED WORKS

(Chen, Wei, and Chang 2011) demonstrate the value of implementing a non-linear version control system where designers can view multiple states of a given element in parallel. (Terry et al. 2004) report the effectiveness of user-interactions that support simultaneous editing, heads-up display, and parallel viewing of alternatives. (Yamamoto et al. 2005) stress the need to separate representations used to create and interact with content from the representations used to show the output of the exploration. (Shireen et al. 2012; R. Woodbury et al. 2013) show that parallel editing and simultaneous viewing of alternatives is a means to creative design exploration.

FRAMEWORK AND STRATEGY

The samples of related works mentioned above, the limitations we find as computational designers using advanced CAD systems, and similarity of processes in text-editing and parametric modeling inspired us to develop Alt.Text.

FRAMEWORK

The framework we follow is based on the understanding that:

1) designers construct hierarchies;
2) they rely on comparison to make decisions;
3) they benefit from being able to create, edit, manage, view and compare multiple alternatives in parallel; and finally
4) they need to directly interact with their work—interfaces should be domain proximate.

STRATEGY

We develop Alt.Text with the following strategies: a weak-representation as defined in (Woodbury and Burrow 2006) to allow for easy restructuring of hierarchies; the implementation of a multi-state data model serving as the back end to support explorations of alternative artefacts (parts) as well as hierarchies (assemblies) in parallel; the reduction of typical user activities into fewer gestures or mouse clicks to boost productivity; and the implementation of subjunctive user interfaces for simultaneous viewing and interaction with content.
as described by (Lunzer and Hornbæk 200). A subjunctive interface has elements that present alternatives to users. Further, because the work we introduce investigates a relatively new area of parallel user interaction with multiple artefacts, where possible we employ familiar graphical user interface approaches, for example, contextual menus, menu bar commands, shortcut commands, buttons, sliders, and toggle and push buttons. Finally, to widen our user-base and to conduct more comprehensive user-studies of the prototype, the native file format of Alt.Text is HTML-compatible. It is possible to view the native files in any Web-Browser. We also support writing and running Processing programming language code: this requires the Processing application to be installed. We also support compiling Latex code. In addition, we support the simple ASCII text using the .TXT file extension.

ALT.TEXT REPRESENTATION

REPRESENTATION COMPONENTS
Alt.Text is designed with a flexible hierarchy that implements weak representation comprising four components:
1) content, as styled-text characters,
2) containers that hold other containers or content;
3) sequence-links defining ordering between containers; and
4) reference-links establishing references across containers.

REPRESENTATION HIERARCHY
Using the representation components described above, Alt.Text provides four hierarchical levels to enable designers to describe and organize their explorations: Document, Outline, Section, and Passage. The outcome of the choices of across these levels is defined as an Alternative. Alternatives form the concrete outputs of Alt.Text, and not a level in the hierarchy. An example of this would be a drawing that is used to describe a specific state of an assembly in a parametric model.

This breakdown is analogous to the element hierarchy found in many parametric modeling tools:
1. Document <=> Collection of Product or Assembly files;
2. Outline <=> Assembly or Product;
3. Section <=> A conceptual Part;
4. Passage <=> An alternative implementation of a Part;

A Document is a container for Sections, represented as a graph, connecting those Sections, each representing a part of a document. The graph need not be connected, that is, a Document can contain multiple, independent graphs. Designers can create relationships (sequence-links or reference-links) between any two nodes on the in a Document.

Outlines are paths in a Document’s graph. They originate in a user-defined Root node in the graph and comprise an ordered sequence of Sections starting from that Root and ending at a system- or user-defined Leaf node. It is possible to build and explore multiple Outlines in parallel, thus Outlines make Documents subjunctive. Further, users can define subsets within paths as new Outlines. For example, this allows users to explore Outlines that include a preface or not, or that include an appendix or not.

The graph nodes in a Document are its Sections. They hold containers called Passages, which, in turn, hold actual text content. Designers can assign different meanings to Sections. For example, a Section may be an entire book, a chapter, a paragraph, a line, a word, or even a character. This enables authors to structure their work in different ways at any point during the writing process. For simplicity in this first implementation of Alt.Text, Sections are not hierarchical: they cannot contain other Sections.

A Section can be labeled as a Root or a Leaf. An Outline starts at a Root and ends at a Leaf. Roots (and Leaves) are either user-defined or implicitly defined as nodes having no predecessors (successors) in the graph. A Section can be both Root and Leaf. Thus any Section is in one of four states: Root (it can start a Outline), Leaf (it can end a Outline), Root and Leaf (it is a Outline in and of itself), or it is unlabeled (and thus can occur only within a Outline).

Passages make Sections subjunctive: Sections can hold multiple distinct Passages, each of which is an alternative implementation of the Section. Similar to Sections, users can assign different meanings to Passages. They can use them to represent alternative contents, or variations of an alternative, or versions of a written body of text.

REPRESENTATION OF MULTIPLE-STATES
Alt.Text supports design space exploration as follows:
1) Outlines as sequences of Sections—a Document can be composed of multiple alternative Outlines. Further, User-defined Roots and Leaves enable Outlines that are parts of other Outlines.
2) Passages enable Sections to have multiple definitions.
3) Alternatives comprise a Outline and a choice of Passage for each contained Section. They thus specify the alternatives forming the design space.
Alt.Text supports parallel editing and viewing of content both at the interface layer and at the file system layer.

COLLABORATION

The implementation of hierarchy in Alt.Text enables collaboration among different users. Alt.Text stores separates files for each Section and Passage element. Thus, it is possible to exchange and co-develop content concurrently. And because the final outcome is an assembly of Sections (which can be empty, thus acting as place holders), users can stagger the development of content as fitting to their production schedule.

PARALLEL VIEWING AND EDITING

Alt.Text offers subjunctive interfaces, ones that offer distinct views to different what-if scenarios pointing to different states for a given content. Thus, it is possible to perform parallel editing across different Passages; and display difference-visualizations on multiple versions of the same entity. Displaying alternatives as overlays adds depth to the text-editing space. Replacing entities within a Outline or a Passage in-place allows designers to work in using ad-hoc rather than predefined processes. Finally, galleries, enable users to organize or group representations of their content.

ALT.TEXT GRAPHICAL USER INTERFACES

The graphical user interface (GUI) of Alt.Text comprises three types: a Graph Editor for creating Sections and defining Outlines; Text-Editors, one for each Section to allow users to work on the content of each, separately; and Outline Viewers, one for each outline to display the assembled outcome of a Outline.
THE GRAPH EDITOR INTERACTIONS:

(Figure 3) shows the Graph Editor GUI with a project made of five Sections. The Graph Editor offers proximity groups, a novel real-time grouping mode, controlled by a proximity sensor, enabling setting multiple relationships among graph nodes in modes: one-to-one, one-to-many, many-to-one, and many-to-many, as shown in (Figure 4). When enabled, proximity groups are formed by placing graph nodes in close proximity to each other.

These four interactions operate are set in reference to the leading-node: the node that the user is interacting with. Relationships are set by choosing one of three rules: Add, Synchronize, and Override. The Add rule performs a one-way synchronization of relationships from the leading-node (source) to nodes in the proximity group (destination). The Synchronize rule performs a two-synchronization of relationships across all nodes in the proximity group. The Override rule overrides the relationships in the proximity group by those from the leading-node. Proximity groups enable setting multiple relationships on the fly.

Alt. Text also introduces the concept of Listeners as a means to form clones interactively. A Listener node mimics the relationships of the nodes it is listening to. It is possible to listen to multiple nodes simultaneously. User can activate and deactivate nodes’ listening at any time without loss of existing relationships. This feature enables the propagation of multiple graph relationships instantly.

Additionally, Alt. Text can store the states of a graph as Snapshots, which are collections of all relationships among graph nodes. Snapshots are displayed in a gallery with interactive previews. Users can also cycle through Snapshots from within the graph editing area.

Finally, the Alt. Text graph editor offers a command line that can process multiple commands sequentially. Commands facilitate creating Sections and Outlines (nodes and sequence links), and clearing and setting sequence-links among Sections with supports for the one-to-one, one-to-many, many-to-many, and many-to-one creation modes.

THE TEXT EDITOR INTERACTIONS

Each Section has a dedicated Text-Editor, within which users create and edit Passages. The editor supports viewing and working in multiple Passages in parallel through operations such as typing or applying character styles.

It is also possible to perform operations such as splitting the context of a Passage to other Passages by two ways: automatically at predefined character-symbols or manually by selecting bodies of text to export as new Passages. The Editor also enables users to push content from the active text-editing area to other Sections. Further, users can perform visual comparison in standard text-difference visualizations over multiple Passages, or by overlaying the content of multiple Passages. (Figure 5) shows the Text Editor GUI.

In addition, users can preview character styles for a text-selection through head-ups display allowing them to compare what-if scenarios without making changes to the text. Further, Alt. Text offers novel interactions that allow users to scroll through saved character styles or saved text-snippets over selected-text. Text selection can be user-defined, or automatically detected by identifying the range of homogeneous character style at the caret position.

THE DOCUMENT VIEWER INTERACTIONS:

A dedicated viewer is provided for each Outline. The viewer supports the creation of multiple views to the same outline, allowing users to load different Passages for each Section as shown in (Figure 6).

In addition to parallel viewing of Outlines, Alt. Text provides two difference visualizations. The first is for the text-character difference operating at the Passage level, showing words to be inserted or deleted shown in the text display area. The second is for the choices of Passages operating at the Section level, shown in a tree-view. The difference is visualized by colors. A reference view must be selected for either difference visualization to be available.

Alt. Text works over simple text. It is thus nearly trivial to connect it to tools that employ text such as the Processing programming language and the LaTeX document preparation system. This we have done.
SUMMARY AND DISCUSSION

Current parametric modeling tools impose many limitations on the exploration of design alternatives. As computational designers, we attempt to identify and address these limitations through a prototype supporting text-editing, which demonstrates similar structure and exploration processes. We developed a tool, Alt.Text, in which we attempt to address the identified limitations. We hypothesize that the approach for developing Alt.Text and the features offered can be portable to the parametric modeling domain.

Alt.Text is pervasively subjunctive, through the devices of Outlines (in Documents) and Passages (in Sections). This stands in sharp contrast to parametric systems such as Grasshopper and Generative Components, with the sole exception of Grasshopper’s Save State command, which saves parameter values only. With Alt.Text a user can define alternative structures at multiple hierarchical levels; and can simultaneously view compare and edit multiple end-states.

We plan to run user studies to evaluate the effectiveness of the offered features and study the particulars of re-creating them for manipulating CAD models. We also plan to work on expanding our Section hierarchy model to include containment (parent-child) structures.
REFERENCES


IMAGE CREDITS


Figure 2. Elkhaldi, Maher and Woodbury, Robert (2014). Document Hierarchy. Interacting with Alternatives: Alt.Text.

Figure 3. Elkhaldi, Maher and Woodbury, Robert (2014). The Graph Editor GUI. Interacting with Alternatives: Alt.Text.

Figure 4. Elkhaldi, Maher and Woodbury, Robert (2014). Interacting with proximity groups. Interacting with Alternatives: Alt.Text.

Figure 5. Elkhaldi, Maher and Woodbury, Robert (2014). The Text Editor GUI. Interacting with Alternatives: Alt.Text.


MAHER ELKHALDI is a PhD student at the School of Interactive Arts and Technology of Simon Fraser University. He combines professional experience in managing and delivering technology solutions on large-scale design and construction projects, with academic training in architecture, BIM, and computational design. Maher attended the American University of Sharjah, The Massachusetts Institute of Technology, and The Georgia Institute of Technology; where he taught courses on design automation and parametric modeling.

ROBERT WOODBURY BArch (Carleton), MSc and PhD (Carnegie Mellon) served on the faculty at CMU and Adelaide University and is now at Simon Fraser. He is currently the Director, Highly Qualified Personnel of the Graphics, Animation and New Media Canada research network. His research is in computational design, visual analytics and sustainability. He has over 200 publications, including his book Elements of Parametric Design. In 2009 he chaired Team North, in the 2009 Solar Decathlon. In 2008 he was awarded the Innovative Research Award from ACADIA and the Tee Sasada Award from CAADRIA. He is a former Olympian in sailing.