The Importance of Revision in Computational Models of Design

Paul Reinerfelt (Paul.Reinerfelt@cs.lth.se)
Department of Computer Science, Lund University; Box 118
SE-221 00 Lund, Sweden

Abstract

A new, more cognitively plausible, model of design, the Generate-Revise* model, is proposed. This model focuses on the importance of continuous revision in design work. Preliminary work on an implementation is also described.

Design as an activity

Design is a complex activity with connections to many cognitive mechanisms. It is a dynamic and challenging process. Design can be seen as a "purposeful, constrained, decision making, exploration and learning activity" (Gero, 1994). This activity is purposeful in the sense that the designer has a goal to achieve, whether this goal is to designing a building, a spoon or a book cover. It is constrained because the chosen domain provides functional, structural and behavioural restrictions that must be obeyed. Learning can be an aspect of revision, as will be explained later. The designer must also explore the possible design space and be open to possible transformations of this space. At the same time, many of the constraints and affordances of this design space are based on the designers perception. During the development of the design, these perceptions change and with them does the design space. The designer is therefore exploring a design space that changes because of this exploration. While it is outside the scope of this article to deal with representational issues in detail it is nevertheless important to recognise that this requires a great deal of flexibility in the underlying representation of domain information.

Context-sensitive representations

Flexible, context-dependent representations have been developed by, among others, Kokinov (1994b), Mitchell (1993), McGraw (1995) and Rehling (2000). The common theme among these is a hybrid approach somewhere between symbolic and connectionist systems.

These systems are flexible, not only with respect to the context, but also in external appearance. They can be viewed as belonging to several different paradigms all at once.

From one perspective they can be seen as ordinary semantic networks (like the slipnets used by Mitchell (1993), McGraw (1995) and Rehling (2000)) where nodes are connected according to their semantic proximity. This sematic perspective focuses on the systems function as a knowledge base.

They can also be seen as traditional connectionist systems, using spreading activation to select relevant parts of the symbolic parts or even to determine the speed of execution as in the DUAL system (Petrov and Kokinov, 1999). The background state of activation is determined by prior executions as well as perceptual activations, thereby gaining a context-dependent activation pattern.

It is important to note that flexible, context-sensitive memory representations easily becomes inefficient. This makes it difficult to generalise these models. Care should therefore be taken to ensure efficiency. Further discussion about this is given by Kokinov (1994a).

The importance of revision

So far, the dominant model of creative design has been the so called Generator-Test model. This model was proposed by Simon (1969) as a model of problem-solving but Dennett (1978) and others have used it as a model for creativity and design as well. Basically, it consists of two modules, one generator of ideas and one module selecting the best from the output of the first module. Lots of variations of this idea have been created over the years but the fundamental dichotomy of the model is always apparent.

A major drawback of the Generator-Test model is that it basically is a one-pass system. That is, first the generator does its thing, and only then does the selection take place. There is no feedback to the generator and the generator part has normally no knowledge about the criteria used by the selector (or even that a selector exists). Even in the cases where some feedback does exist, it is usually only to enable the selector to restart the generator in case it didn't find any acceptable candidates in the current output. Typically, information about the reason for this failure is not given to the generator.

That model is akin to the case of an artist who paints several canvases while wearing a blindfold and then selecting the best from them (or even letting someone else choose), which would be a very unusual style. An ongoing process of review during the work is central and this process influences the work to be done. There is, in essence, an continuous process of revision. This is most

obvious in the case of drawers, sketching a new drawing. New strokes are added to old to move lines, soften curves, sharpen corners, etc., all with the purpose of improving on the first attempt. When satisfied, the drawer then fixates the end result with bolder lines, perhaps even using ink. Writers are another group of individuals where this revision process is particularly evident. Manuscripts are rewritten, several times, in varying sizes, all the way from chapters down to fiddling with individual sentences. The end result is much better that the first rough draft. Revision is inherent in all creative work, but this has rarely been included in artificial models of design and creativity.

In response to this, I propose another model of design, called the Generate-Revise* model (the star comes from its use in computer science to denote repetition). This model is not new, neither is it my own (though the name is), but it has received very little attention in the field of artificial design. This is amazing as revision is so prevalent in every kind of creative work.

Two variants of the model can be conceived. In the first variant some subsystem creates its output, much like in the Generator-Test model, and this output is then judged by some other component. Again, so far it is identical to the Generator-Test model. The difference is that the judgement is then *passed back to the generator* to be used to revise the output just created. The new version is then judged again and so on, continuing until the output is of sufficient standard to be considered the output of the whole system. A partial example of this model is Letter Spirit (McGraw, 1995; Rehling, 2000), where the Drafter is the generator and the judging is carried out by the Adjudicator. However, no direct feedback is given to the Drafter, instead the top-level program decides whether or not to replace the old (if any) version of the current letter with the new one based on the Adjudicator's judgements.

The second variant, is a system where the feedback is given *while* the generator is working. That is, the generator is no longer working "blindly" as even unfinished designs are evaluated by whatever evaluation-module the system has and that evaluation can immediately influence the ongoing work. Such a system would very much be like a sketcher, who has immediate access to visual feedback about the emerging drawing created.

It is important to note that both variant must include some form of learning capacity even if it is in the most rudimentary form. At the very least it must avoid making the same mistake twice. While there are many flavours of learning algorithms and this is not the place to review them, it would be an interesting prospect if the system could get a long term benefit from the revision process.

Typographical design as an example

As an example of this model, a prototype implementation is under development. This implementation works in the area of typeface design. The basic idea of this implementation is the same as in the Letter Spirit project (McGraw, 1995; Rehling, 2000), to develop a computer program that, given a small set of *seed letters*, can design new letters in the same style as the seeds and to be a demonstration of how creative design might work.

An overview of typographical design

While the shapes of letters differ wildly between typefaces, some common structural parts can be identified. Some examples of parts are vertical *stems* (like the backbone of an upper case 'B'), horizontal *arms* or *crossbars* and diagonal *strokes*. Enclosed *bowls* are also common (the letter 'O' consists of a single bowl). In some typefaces, parts can have their endpoints decorated by *serifs* (like the letters you are now reading). The basic decomposition of letters into parts are usually the same between typefaces but the realisation of these parts can vary.

These terms are in the language of typographers which is important because a human typographer working on a new typeface does not think in terms of bezier segments (the usual representation for curves in digital typefaces) but in terms of stems, arms, serifs and so on and of attributes of these parts, such as widths, heights, and so forth. Therefore, this model should do this as well.

Thus it is important to decompose and organise the shape of a glyph (which is normally given as a contour) into parts so that relevant features can be identified. The implementation described here contains *models* of characters in terms of these abstract parts (compare the use of *roles* in McGraw et al. (1994)). The use of typographical parts also allows the system to handle constraints like the fact that the three stems of a lower case 'm' should have equal width, a fact that would be very difficult to express generally when dealing with only a contour.

On characters, glyphs and typefaces

It is important to have the distinction between characters and glyphs clear. A character is an abstract object having a single and unique semantic or phonetic meaning. A glyph represents the visual, graphical appearance of a character. For example, glyphs in different typefaces, such as Times Roman or Zapf Chancery, may be used to represent the character 'A'. Figure 1 is an example of a kanji glyph which corresponds to characters in several different languages.

Figure 1: A kanji glyph (image taken from chapter 6 of the TrueType Reference Manual (Apple Computer, Inc., 1999))

A typeface is a collection of glyphs with some element of design consistency, such as the use of serifs or consistent stroke thickness. Typefaces usually incorporate other information, such as which glyphs represent ligatures or contextual forms.

Many glyphs do not have a one-to-one relationship to characters: one or more glyphs may make up a single character (for example, an 'i' could be rendered as a dot accent over a dot-less-i 'ı'), and a single glyph can represent two or more characters ('f' and 'i' could be represented by a 'fi' ligature). Context may also determine which glyph is used to represent a character. For example, in a cursive typeface four glyphs may represent the same character: a separate glyph for the character at the beginning, middle, and end of a word; and a glyph for the character in isolation (these variants of the same letter are sometimes called *allographs*).

The implementation

The main parts of the system are shown in figure 2. As can be seen, the system consists of three main subsystems or processes (shown as rectangles) and three kinds of data-storage areas or memories (shown as ellipses). The "Raw Font Data" ellipse at the bottom is not really part of the system but indicates where the seed letters enters.

A run of the system begins with the system using its Perceiver to identify and convert those glyphs of an external font-file that are designated as seed letters, thereby entering them into the glyph memory. At this point the system enters into a cycle of analysis, design and (re-)evaluation where it analyses the existing glyphs for stylistic properties, uses these properties to design new glyphs for missing characters, evaluates them for conformance to the character model and re-analyses the developing style as a whole. This cycle ends when the system has created glyphs for all characters it knows (or was asked for) and these glyphs have been judged good representatives of the given style. The design phase itself consists of a loop where the Designer frequently requests feedback from the Analyser and tries to remedy the flaws it discovers.

If the system, as part of its reevaluations, finds that it deviates too much from the specified design (for example because one of the seed letters is no longer judged to be a good member of the developing style), it goes back and modifies earlier designs to bring the style back on track.

The system also gathers "experience" by remembering glyphs and typefaces it has encountered. This allows it to fill in missing information from similar typefaces, with appropriate adjustments based on the differing dimensions of the two typefaces.

Perceiver The Perceiver performs several related functions. At the beginning of a run it is responsible for importing the seed glyphs from an external font file. Importing glyphs consists of analysing the outline representation in the file, extracting the typographic features, like stems, bars and arcs, from it and creating an internal representation of it. It also uses the character models to identify what character the glyph represents. If the Perceiver can not identify a glyph, it asks for assistance from the operator and based on the information given by the operator it creates a new model to help it recognise this character better in the future.

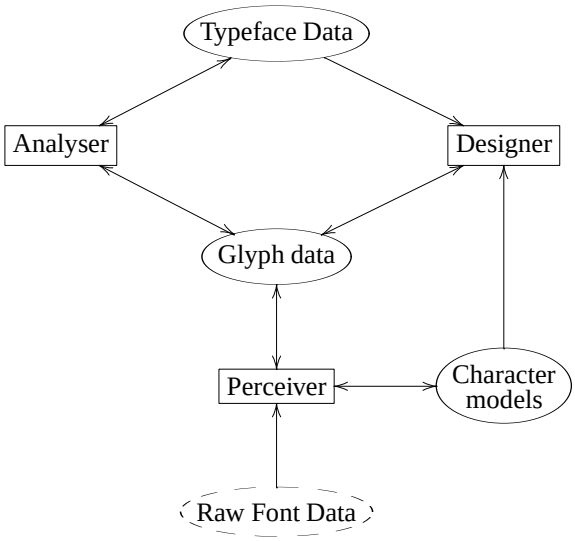

Figure 2: Main architecture

Currently, several different algorithms for typographic feature extraction are investigated. Related work describing interesting algorithms include Herz and Hersch (1993, 1994); Park and Maeng (1993) and Shamir and Rappoport (1996).

Character models This is where the system keeps its knowledge about the abstract shapes of different characters. Each character is associated with one or more character model (some characters, 'g' for example, commonly occur in more than one shape).

Analyser The purpose of the Analyser is to gather information from the glyphs belonging (at the moment) to the typeface under development. It uses this information to calculate the properties of the typeface as a whole in a process similar to that of the PANOSE system (Bauermeister, 1988; Hewlett-Packard, 1995). These properties are then used by the Designer while working on new glyphs.

The analyser also has a second responsibility. It judges the stylistic appropriateness of the Designer's work thus providing the necessary feedback to the Designer for revision. This stylistic judging is also periodically carried out on the finished glyphs (including the seed letters). The purpose of this is to keep an eye out for stylistic drift. If the seed letters looks like they are starting to creep towards the edge of the acceptable region in the design space, then this is a sign that the region is moving because accumulated design choices are moving it in a direction away from the original intent. If this is discovered, the Analyser sends the most offending glyphs back to the Designer for revision.

Glyph Data The Glyph Data area can be viewed as the working memory of the system. This is the place where all data about glyphs is collected. This information is

based on the data from the Character Models but is enhanced with actual shape information as well as values for different attributes of the parts.

Typeface Data This area contains information about the typeface as a whole as calculated by the Analyser. It contains things like the capital-height and x-height of the glyph, as well as things like the general thickness of letter parts (the *weight* of the typeface).

Designer As mentioned in the section on the Generate-Revise* model, the design of Letter Spirit does not include any direct feed back to the Drafter. It is, in essence, working blindly (this is acknowledged by Rehling (2000, chap 7)). In contrast to this, the Designer module of the present system is designed to utilise the skills of the Analyser as often as possible. To do a fair job, however, the Analyser requires a complete character (all features present). Therefore, the Designer makes a rough sketch of the character. This sketch is basically just a fill-in of a Glyph Data structure with the data from the Character Model. This data lacks serifs and all features are average, non-descript members of their type. This rough glyph is then handed to the Analyser for evaluation. The resulting feedback is then used to improve on the first rough draft. It continues like this in consecutive cycles of review and revision until the Analyser gives it good enough ratings. The glyph is then released to the main loop, to be inspected by the Perceiver and Analyser (this time the Analyser will also reevaluate the whole style).

Conclusion

The Generate-Revise* model of design have been proposed. This model is felt to be cognitively closer to the design process as performed by humans. Preliminary work towards studying it in practice in the domain of typeface design have also been reported.

Future work includes completing the implementation and studies of the feedback process inheritent in revision.

References

Apple Computer, Inc. (1999). TrueType reference manual. `http://developer.apple.com/fonts/TTRefMan/index.html`.

Bauermeister, B. (1988). *A Manual of Comparative Typography, The PANOSE System*. Van Nostrand Reinhold.

Dennett, D. C. (1978). *Brainstorms: philosophical essays on mind and psychology*. Bradford Books.

Gero, J. S. (1994). Creativity and design. In Dartnall, T., editor, *Artificial Intelligence and Creativity: an interdisciplinary approach*, pages 259–267. Kluwer Academic Publishers.

Herz, J. and Hersch, R. D. (1993). Analysing character shapes by string matching techniques. *Electronic Publishing—Origination, Dissemination, and Design*, 6(3):261–272. `http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume6/issue3/hersch.pdf`.

Herz, J. and Hersch, R. D. (1994). Towards a universal auto-hinting system for typographic shapes. *Electronic Publishing—Origination, Dissemination, and Design*, 7(4):251–260. `http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume7/issue4/ep125jh.pdf`.

Hewlett-Packard (1995). PANOSE classification metrics guide. `http://www.fonts.com/hp/panose/greybook`.

Kokinov, B. (1994a). Flexibility versus efficiency: The DUAL answer. In Jorrand, P. and Sgurev, V., editors, *Artificial intelligence: Methodology, systems, applications*. World Scientific Publ., Singapore. `http://www.nbu.bg/cogs/personal/kokinov/flex_vs_.pdf`.

Kokinov, B. N. (1994b). The context-sensitive cognitive architecture DUAL. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Erlbaum. `http://www.nbu.bg/cogs/personal/kokinov/dual_ctx.pdf`.

McGraw, G., Rehling, J., and Goldstone, R. (1994). Roles in letter perception: Human data and computer models. Technical Report CRCC-TR 90, Center for Research on Concepts and Cognition, Indiana University.

McGraw, Jr., G. E. (1995). *Letter Spirit (part one): Emergent High-Level Perception of Letters Using Fluid Concepts*. PhD thesis, Indiana University.

Mitchell, M. (1993). *Analogy-Making as Perception*. The MIT Press.

Park, S. W. and Maeng, S. R. (1993). Structure extraction and automatic hinting of Chinese outline characters. *Electronic Publishing—Origination, Dissemination, and Design*, 6(2):67–91. `http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume6/issue2/OLD/ep076sp.pdf`.

Petrov, A. A. and Kokinov, B. N. (1999). Processing symbols at variable speed in DUAL: Connectionist activation as power supply. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 846–851. Morgan Kaufman. `http://www.nbu.bg/cogs/personal/kokinov/energy2.pdf`.

Rehling, J. A. (2000). *Letter Spirit (part two): Modeling Creativity in a Visual Domain*. PhD thesis, Indiana University.

Shamir, A. and Rappoport, A. (1996). Extraction of typographic elements from outline representations of fonts. *Computer Graphics Forum*, 15(3):259–268.

Simon, H. A. (1969). *The sciences of the artificial*. MIT Press.