

The Essence of Design

Paul Hudak
Yale Department of Computer Science
paul.hudak@yale.edu

October 1, 2003

I have been fascinated with the process of design for a long time. In my twenty-one years at Yale I have had the privilege of meeting a very diverse set of scholars: from artists and musicians, to architects and economists, and to scientists and mathematicians. In discussions with them and studies of their achievements, it is not surprising to discover an amazing variety of creative ideas. What *is* surprising, however, is that there is also a great deal of common ground. In particular, I believe that there are common principles of *creative design* that underly these diverse domains.

I am interested in capturing the *essence* of the creative design process that underlies a diverse set of disciplines, including mathematics, engineering, science, music, and the visual arts. I use the term “designer” to refer to someone engaged in the creative design process, regardless of area. And I use the phrase *creative design* in a fairly loose sense to encompass many kinds of problem solving in these diverse domains. This includes, for example:

- Problems in mathematics and engineering that are well defined, even if informally, and whose discovery involves a fairly directed search of the design space.
- Software artifacts for, say, user interfaces, that are weakly defined, and whose construction involves a search for tasteful features constrained by specifications for correct functionality.
- Artistic artifacts (in music, art, dance) whose final form and substance are initially completely undefined (and even if known, embody aesthetics that are subjective and impossible to codify), and whose creation thus involves a free-wheeling exploration of the design space.

The call for white papers states that the emphasis of this NSF program is on *software* design. But the design of software is dictated by *applications*. In that sense, the software is just a vehicle for expressing an artifact of the application domain. Therefore, *I believe that it is vital that we understand the process of design at a more abstract level*, and not become bogged down with traditional details of software engineering.

Of course, we must begin somewhere. My approach is to ask what the *language* of design is. What language do the domain experts speak? Is there commonality between these languages? And can we capture them, either independently or as a whole, in a formal way?

I have made some progress in answering these questions through the design of *domain specific languages* for graphics and animation [EH97, Hud00], robotics [PHH99], music [HMGW96] and dance [HH03]. A domain-specific language (or *DSL*) is a programming language adapted to a specific setting, allowing non-programmers to write interesting and useful programs within the domain of interest without the need to master a complex programming language [Hud98]. Well-designed DSLs allow students and practitioners to explore large regions of a design space, discovering and defining new patterns of design within a particular field of study. A DSL is thus the “ultimate abstraction” of a domain, reflecting the core intuitions and meanings (i.e. semantics) of the domain, with no extraneous detail and no loss of expressiveness.

It may seem strange, or at best naive, to think that common principles of design exist across a variety of disciplines. But I am not the first to suggest this possibility. Connections between visual arts and mathematics (see [Emm93]), and music and mathematics (see [Rot95]), are well-known. Perhaps the best embodiment of this thesis is Douglas Hofstadter’s seminal book *Gödel, Escher, Bach: An Eternal Golden Braid* [Hof79] that draws strong parallels between the creative design processes found in mathematics, art, and music.

In my own work I have tried to formalize these commonalities by defining a polymorphic representation of these languages, together with an axiomatic semantics whose axioms themselves are polymorphic

[Hud03]. To see this more concretely, consider the following hypothetical (and thus rather over-simplified) scenario. Suppose there are three combinators: one for parallel composition ($|$), one for sequential composition ($.$), and one for scaling (**trans**). Now consider the simple expression:

$(p1 | p2) . \text{trans } k \ p3$

1. In the domain of music, this might represent the simultaneous (i.e. parallel) playing of the phrases $p1$ and $p2$, followed (sequentially) by the transposition of the phrase $p3$ by k semitones.
2. In the domain of animations, this might represent the simultaneous (i.e. parallel) animation of clips $p1$ and $p2$, followed (sequentially) by the clip $p3$ scaled in size by a factor k .
3. In the domain of graphics, this might represent the simultaneous (i.e. parallel) display of graphic images $p1$ and $p2$, followed (left to right) by the linear translation of image $p3$ by k units.
4. In the domain of robotics, this might represent the simultaneous (i.e. parallel) execution of robotic tasks $p1$ and $p2$, followed (sequentially) by the execution of task $p3$ at rate k .

Although this scenario is over-simplified, we have in fact designed and implemented DSLs for these four domains that do have this kind of commonality. They also share useful algebraic properties. For example:

$(a | b) | c = a | (b | c)$ -- associativity
 $a | b = b | a$ -- commutativity
 $\text{trans } k (a | b) = \text{trans } k a | \text{trans } k b$ -- distributivity

I believe that DSLs can be used to magnify the creative design process in a variety of different domains, including software design. We need to study a wide spectrum of disciplines in an attempt not only to discover common underlying principles, but also to build a common infrastructure to support them. To be able to use the same methods and common notation to express creative ideas in mathematics, engineering, science, music, art, and software development would arguably be a monumental and far-reaching (if ambitious) achievement.

References

- [EH97] Conal Elliott and Paul Hudak. Functional reactive animation. In *International Conference on Functional Programming*, pages 163–173, June 1997.
- [Emm93] Michele Emmer, editor. *The Visual Mind*. MIT Press, Cambridge, 1993.
- [HH03] Liwen Huang and Paul Hudak. Dance: A language for humanoid robot motion. Technical Report YALEU/DCS/RR-1253, Yale University, Department of Computer Science, July 2003.
- [HMGW96] Paul Hudak, Tom Makucevich, Syam Gadde, and Bo Whong. Haskore music notation – an algebra of music. *Journal of Functional Programming*, 6(3):465–483, May 1996.
- [Hof79] Douglas R. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, New York, 1979.
- [Hud98] Paul Hudak. Modular domain specific languages and tools. In *Proceedings of Fifth International Conference on Software Reuse*, pages 134–142. IEEE Computer Society, June 1998.
- [Hud00] Paul Hudak. *The Haskell School of Expression – Learning Functional Programming through Multimedia*. Cambridge University Press, New York, 2000.
- [Hud03] Paul Hudak. Polymorphic temporal media, August 2003. Submitted to POPL 2004.
- [PHH99] John Peterson, Gregory Hager, and Paul Hudak. A language for declarative robotic programming. In *International Conference on Robotics and Automation*, 1999.
- [Rot95] Edward Rothstein. *Emblems of Mind: The Inner Life of Music and Mathematics*. Times Books, New York, 1995.