

Computational Composition and Creativity

Wesley Smith

Media Arts and Technology Program
University of California Santa Barbara
Santa Barbara CA 93106

whsmith@mat.ucsb.edu

Graham Wakefield

Media Arts and Technology Program
University of California Santa Barbara
Santa Barbara CA 93106

wakefield@mat.ucsb.edu

ABSTRACT

In this paper we put forward a notion of computational composition: an exploratory approach to creativity uniquely available by means of computation. The implications for an expanded overlap of the intellectual and computational are discussed and developed into a design strategy. Finally we describe progress on our implementation.

Categories and Subject Descriptors

J.5 [Computer Applications]: Arts and Humanities; F.1.1 [Theory of Computation]: Computation by Abstract Devices—Models of Computation

General Terms

Creativity, Composition, Theory

Keywords

Computational Composition, Philosophy, Software Design

1. INTRODUCTION

Computational composition may be briefly defined as a domain of *computationally embodied, exploratory creativity*. In general terms, it indicates an elevated aesthetic role of computation beyond mere assistance or transmission. Computational composition software belongs to the category of creativity support rather than productivity support, indicating increasingly user- and project-defined tasks and constraints (“making software more soft” [5]). The domain of computational composition blurs architect and composer, artist and engineer, placing user as meta-creator. In this paper we trace the principal features and conditions of possibility of this domain as we put forth design requirements for a supporting software environment, and report on the current state of our implementation.

To help clarify, we can distinguish computational composition from related applications of computing in creative work (Figure 1). On the one hand, there are computer-assisted applications (such as Adobe Photoshop, Apple Final Cut Pro, Digidesign ProTools). Such applications impose a mode of interaction through a prescribed role represented by a tool/canvas, that in turn controls an opaque sequential schema of computation. The focus

is on productivity within a particular domain, so very little of computational engine is exposed to the user. On the other hand, there are autonomous, generative applications (such as music visualizers, screensavers [4]) where parametric control over a computational engine is exposed to the user but the structure of the engine itself is not.

In both of these cases, the overlap of the intellectual and computational domains is relatively constrained. In contrast, computational composition is characterized by an expansive overlap where the computational engine is exposed in such a way that it can be extended and transformed by the user during operation.

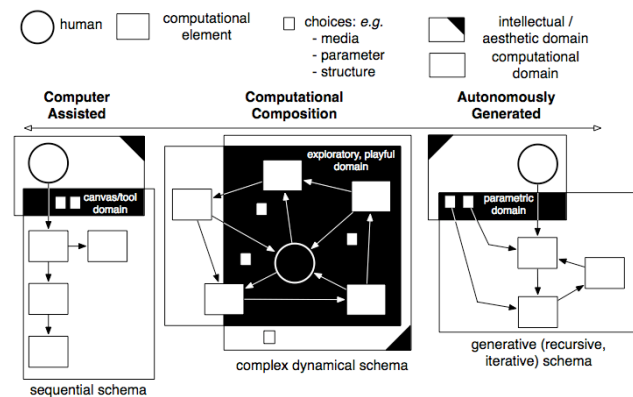


Figure 1. Composition application schema.

1.1 Epistemological Significance

The enduring issues of creative disciplines (including values, purposes and implications for culture and humanity) are expressed through the available models of representation (such as tools and methodologies), yet these models are never comprehensive. Computation is no exception. Not only must an artist know the limitations and capacities of models to prevent sterile work, but a discipline must also actively evolve the models available to incorporate novel developments.

By instrumenting a domain or discipline with computation a vast territory of possibility is opened up: information and knowledge can be processed in a qualitatively different manner [8]. This derives from the orders of magnitude more work that computational devices can perform over ordinary human faculties and in a sense provides a kind of epistemological emergence where novel forms of knowledge develop on top of the strata of current knowledge and computation.

1.2 Computational Embodiment

When computation becomes the subject of a software environment, as opposed to acting in service of it, the results

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MAST 09, January 2009, Santa Barbara, CA, USA.

produced become secondary to the morphological behavior of the environment itself. In other words, the space of exploration moves from that of a particular paradigm to that of how computational paradigms are generated and embodied in general. The key point is that computation is not hidden or glorified but instead brought to bear in a malleable form, within the realm of the aesthetic and algorithmic, as a kind of meta-material itself.

1.3 Exploratory Creativity

Creative exploration is open-ended by nature: all of the information at hand required to complete a task is not available without experience [5], as knowledge is constructed in the process of solving problems.

The desire is for a system that enables the interactive evolution of known models to encompass that of the possible and improbable. As a meta-material, computation enables creative exploration, where the design problem is a meta-problem whose open, dynamic, and adaptable requirements must be accounted for computationally.

2. DESIGN CONDITIONS

An elegant solution must find a trajectory that navigates the potential conditions of failure [6] while maximizing the purposes of the task. Through a deeper analysis of the task, critical conditions of a solution emerge.

Computational composition calls for creative approaches to synthesize conceptual models and tools (theories, heuristics, speculations, inferences, desires, etc) with the characteristic qualities of computation (procedures, functions, maps, structures, filters, etc).

It becomes apparent that the overlap of aesthetic and algorithmic domains calls for the cognitive and computational to be intertwined with each other at multiple levels of hierarchy: a meshwork of ends and means with a qualitatively fractal structure. Maximizing the available co-extension of interface and process has a greater priority than ease-of-use¹.

2.1 Low-level, modular primitives

The user design space is thus bounded by the range and granularity of the available *primitives*: units, components, features and types. An emphasis on creative process calls for generalized primitives of a sufficiently low level that they do not enforce representational allegiance but function as a kind of computational alphabet from which composition-specific material may be constructed. Avoiding a priori schemas calls for flexibility in assembly of such primitives, including modularity of interconnection and encapsulation. The open-ended and initially blank canvas of the Max family of composition environments draws principally upon this condition [12, 15].

2.2 Dynamic reconfigurability

Appropriate primitives and modularity cannot be readily derived for open-ended tasks; the key question of granularity and how to circumscribe it becomes one not of a priori derivation, but one of actively transporting concepts across the threshold from the cognitive to the computational in a dynamic fashion.

In other words, reconfiguration of the system as the need arises is a necessarily first-class user activity. To what degree this may be possible is delimited by the *dynamic flexibility of the runtime environment*. Reducing the complexity and latency of the generate-and-test cycle is a corollary desirable feature.

2.3 Creative augmentation

We suggest that maximizing the creative potential of a computational environment calls for the facility to not only reconfigure active processes, but also augment them with new primitives. In this respect we call upon Cariani's distinction of combinatoric and creative emergence [3], where the latter distinguishes (emergent) phenomena in which new qualitative primitives must be added to a model.

2.4 Efficiency

The remaining challenge is to provide this flexible and expressive dynamism such that efficiency is realized. Hardware imposes implementation limits of processing, distribution and storage capacity, leading to the question of flexibility versus efficiency. This trade-off is directly expressed for example in interpreted versus compiled execution strategies.

3. IMPLEMENTATION

One might productively compare the conditions identified above to a vocabulary, a grammar, and the means by which they can change. Programming languages already draw upon such insights, and dynamic interpreted languages are a popular solution for problems that are not well-defined before initiation.

Over recent years an increasing number of dynamic language-based computational composition environments have been developed such as SuperCollider [11], Fluxus [7], Impromptu [1] etc. Our own contribution extends the dynamic language Lua [9].

3.1 LuaAV

LuaAV [14] is a stripped down platform upon which users can define custom application environments and execute and manage Lua scripts. These scripts make use of grammar and vocabulary of the Lua language along with extensions we have provided for temporal, spatial and sonic composition. LuaAV is the focal point for our explorations in computational composition as well as a medium for research within the audiovisual arts. Basing our work upon the dynamic language Lua satisfies many of our design requirements due many of its inherent features.²

Dynamic typing, a universal container model, prototype/attribute patterns (rather than strict class hierarchies), meta-mechanisms and first-class functions support great modularity in structural and behavioral description. Our media-domain extensions tend toward low-level interfaces, meshing as deeply as possible with existing Lua paradigms.

Dynamic memory allocation and incremental garbage collection, runtime exception handling, dynamic code generation/evaluation and dynamic loading/linking allow runtime flexibility, reconfiguration and extension.

In addition, the language may be augmented with domain-specific capabilities, linking to external libraries, services or protocols as

¹ Clearly, increased freedom demands increased effort by the user.

² Many points are equally applicable to other dynamic languages such as Scheme, Ruby, JavaScript, Python etc.

required through a well-designed low-level C API. By this means we add extensions to LuaAV for accurate temporal scheduling (extending the language's coroutine construct), 3D graphics (OpenGL with extensions), audio synthesis, MIDI and OSC inter-processes communication and more.

3.2 Native code generation

Unfortunately, the dynamic flexibility comes with an attendant cost of efficiency. Though Lua is recognized as a particularly fast dynamic language, it remains typically an order of magnitude slower than C/C++. The lack of determinism is harder to predict and optimize, and code paths suffer indirection through the interpreter and virtual machine.

A potential solution is runtime native code generation or JIT compilation. We have embedded the LLVM platform (Low-Level Virtual Machine [10], a set of library tools for building compiler infrastructures) within LuaAV to this end. The issue of granularity is raised once more, in the appropriate choice of mapping from IR (Intermediate Representation) instructions, values, structures and functions to compositional components. A brief outline is detailed in Figure 2.

4. DISCUSSION

We have outlined the significance of augmentation in the extension of knowledge to access the unprecedented and improbable, and the ways in which computation may provide this capacity with respect to the arts.

In creative work, the unexpected can also be a desirable characteristic of the process itself. We note that Boden's notions of creativity [2] - a hierarchy of combinatorial (reconfiguring elements) through exploratory (combinations of rules within a concept-space) to transformational (redefining the space itself) - offer a remarkably similar vector of increasing dynamism and modularity (or reducing granularity) as identified in this paper. Of course this movement can be recursively applied: what we take as primitives and features in our design today, may need to become supple and fluid in the future.

Acknowledgments. The authors would like to acknowledge the Allosphere Research Lab and transLAB for their support and creative stimulation.

5. REFERENCES

- [1] A. R. Brown, A. Sorenson. Dynamic media arts programming in impromptu. Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition. Washington, DC, USA. 2007.
- [2] M. A. Boden. What is creativity? In M. A. Boden (Ed.), Dimensions of creativity (pp. 75-118). Cambridge, MA: MIT Press. 1994.
- [3] P. Cariani. On the design of devices with emergent semantic functions. PhD thesis, State University of New York at Binghamton, 1989.
- [4] S. Draves. The electric sheep screen-saver: A case study in aesthetic evolution. In Applications of Evolutionary Computing, LNCS 3449. Springer Verlag, 2005.
- [5] G. Fischer et. al. Meta-design: a manifesto for end-user development. Communications of the ACM, 30(2), 2004.
- [6] M. Fuller. Elegance. Software Studies, pages 87-92, 2008.
- [7] D. Griffiths. Fluxus. <http://www.pawfal.org/Software/fluxus/>, 2008.
- [8] P. Humphreys. Extending Ourselves. Oxford University Press, 2004.
- [9] R. Ierusalimschy, L. H. de Figueiredo, W. Celes, "Lua - an extensible extension language", in Softw& Experience 26 #6 (1996) 635-652, 1996.
- [10] C. Lattner, V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. International Symposium on Code Generation and Optimization (CGO'04), 2004
- [11] J. McCartney, "Rethinking the Computer Music Language: SuperCollider," Computer Music Journal 26, 4 (2002), 61-68.
- [12] M. Puckette, "Pure Data," in Proceedings of the 1997 International Music Conference (ICMC '97) (1997), Computer Music Association, pp. 224-227.
- [13] B. Shneiderman. Creativity support tools: Accelerating discovery and innovation. Communications of the ACM, 50(12):20-32, 2007.
- [14] W. Smith, G. Wakefield. 2008. Computational audiovisual composition using Lua. Transdisciplinary Digital Art. Sound, Vision and the New Screen, CCIS:7. Springer Verlag, 2008.
- [15] David Zicarelli. How I learned to love a program that does nothing. Computer Music Journal, 26(4):44-51, 2002

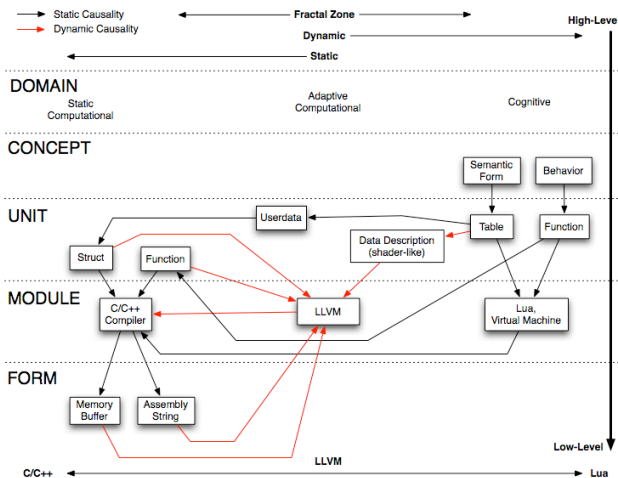


Figure 2. Hierarchies of computation.