# Developing a New Computer Music Programming Language in the 'Research through Design' Context

Hiroki Nishino

NUS Graduate School for Integrative Sciences & Engineering, National University of Singapore
*CeLS, #05-01, 28, Med. Drive, Singapore 117456*
g0901876@nus.edu.sg

## Abstract

The development of computer music languages seems to be considered from outside the computer music community just as contributions in practice rather than in research. Yet, the emerging approach of 'Research through Design (RtD)' in HCI also casts a significant question as to how the academic contribution can be made through the design of such DSLs. We describe our practice in the development of a computer music language from the perspective of the RtD.

***Categories and Subject Descriptors*** D.3.3 [**Programming Languages**]: Language Classifications – Specialized application languages

***Keywords*** computer music, research through design

## 1. Motivation

Computer music programming languages have been playing a significant role both in research and artistic creation in the field. However, the development of such DSLs itself has been considered as the development of practical tools rather than academic contributions from outside the computer music community; at the same time, the development of programming languages with such domain-specific needs is becoming a topic of interest in HCI as end-user programming gets more popular. Blackwell argues, "*the study of unusual programming contexts such as Laptop music may lead to more general benefits for programming research*", as seen in the spreadsheet, which "*was invented in response to the needs of business school students*" [1].

Such a situation for computer music languages also corresponds well to the concept of 'Research through Design (RtD)', an approach in HCI research that recently emerged within the community. In RtD, designers and researchers are expected to develop "*a product that transforms the world from its current state to a preferred state*" and "*the artifacts produced in this type of research become design exemplars, providing an appropriate conduit for research findings to easily transfer to the HCI research and practice communities"* and RtD focuses on "*producing a contribution of knowledge"* as academic research rather than *"more immediately inform the development of a commercial product"* through design practices [14].

Copyright is held by the author/owner(s).
*SPLASH'12,* October 19–26, 2012, Tuscon, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

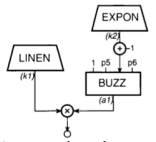
This doctoral symposium proposal emphasizes this aspect of 'Research through Design' in the development of a new computer music language.

## 2. Background and Problems

In this section, we briefly describe the background knowledge in computer music, which we discuss later in the problems that we aim to solve in our RtD practice.

### 2.1 The Unit-Generator Concept

The unit-generator concept, one of the most significant abstractions in computer music, was established around 1960 and even today many computer music languages are still based on these abstractions. A unit-generator is a software module considered to perform "*conceptually similar functions to standard electronic equipment used for electronic sound synthesis*"[7], e.g. oscillators and filters. A sound synthesis model is described as a network of such unit-generators; figure 1 below is a pictorial representation of the unit-generators and the synthesis model definition in *CSound* computer music language [4].




```
        instr   115
k1      linen   p4, p7, p3, p8
k2      expon   p9, p3, p10
a1      buzz    1, p5, k2+1, p
        out     k1*a1
```

**Figure 1.** A synthesis model definition in *CSound* [4,p.27]

### 2.2 Microsound Synthesis Techniques

Generally speaking, microsound synthesis techniques [10] involve many short sound particles (microsound) that overlap-add each other to constitute the entire sound output.

Such concept significantly differs from the traditional unit-generator concept, which originates in electronic sound synthesis that assumes the signal s is a function *s(t)* of time *t*. Figure 2 describes an example of *synchronous granular synthesis*, a kind of microsound synthesis that schedules microsounds, which are called *grains*, with a regular interval. Figure 2(a) shows 3 grains scheduled with relatively large intervals. As the intervals get shorter from (a) to (f), the overlap-add of microsounds results in the different waveforms with various pitches, even when they all schedule the identical grains with the regular interval.

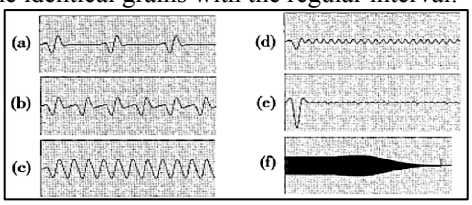



**Figure 2**. Influence of grain density on pitch (a) 50 grains/sec (b) 100 grains/sec (c) 200 grains/sec (d) 400 grains/sec (e) 500 grains/sec (f) Plot of a granular stream sweeping from the infrasonic frequency of 10 grains/sec to the audio frequency of 500 grains/sec over 30 seconds [10, p.95].

### 2.3 Problems Related to Microsound Synthesis

While there are a number of different microsound synthesis techniques, it is necessary to schedule each microsound in precise timing with sample-rate accuracy for the accurate sound rendering; yet many computer music languages are incapable of such precise timing behavior as seen in most general purpose programming languages. Furthermore, some previous works argue difficulty in realizing microsound synthesis techniques solely in computer music languages and how to implement microsound synthesis has been one of significant concerns in computer music [5,12].

## 3. Approach

### 3.1 RtD and Computer Music Language Design; Beginning wit the Existing Problems

RtD aims to produce academic contributions through designing a product that results in 'a preferred state' to users. In the development of a new computer music language as a RtD practice, we interpreted 'a preferred state' as an issue in the usefulness of a programming language; usefulness "*concerns the degree to which a product enables a user to achieve his or her goals*" [11, p.4]; thus, we begin with identifying the existing problems in computer music as the problems in usefulness, and expect the design process for solving such problems to lead to academic contributions.

In our practice, we primarily focus on the difficulty in implementing microsound synthesis techniques, which we consider '*unpreferable*' in computer music programming. We interpreted this difficulty as the issues in usability and functionality of computer music languages; thus this RtD practice contextualizes this problem in HCI (usability) and software engineering/programming language design (functionality). Such contextualization enables to the investigation of the problem in the frameworks provided by the previous study in each field.

### 3.2 The usability issue and incompatible abstraction in the sound synthesis frameworks

The first problem in this RtD practice is how to discuss such 'usability' problems such as readability and simplicity in programming language design. Markstrum even argues that a language designer throw "*together some convincing looking examples and skirt the effort involved in administering a user study or a long-term development study*" [6].

As for usability aspects of this RtD practice, our interest is in reducing difficulty in implementing microsound synthesis techniques in a computer music language. We consider this as a problem in the abstraction of the underlying sound synthesis software framework in a computer music language design, as it is generally considered that incompatible abstractions with the users' conceptualization of the domain can lead to significant usability problems. Blandford provides a valuable view to such issues in her CASSM (Concept-based Analysis of Surface and Structural Misfits") framework, "*the purpose of which is in the identification of misfits between the way the user thinks and the representation implemented within the systems*" [3].

We borrowed such perspectives and frameworks from the previous study in HCI so to better analyze and discuss the usability problems for a new language design. We primarily used the Cognitive Dimensions of Notations framework (CDs) developed for "*broad-brush analysis*" of usability problems in notation [2] and analyzed the usability problems in the existing languages regarding to microsound synthesis, and then assessed the gap between the user's conceptualization of computer music sound synthesis and the representation implemented within the language designs, as CASSM approach suggests. For this assessment, we consider nine musical time-scales by Roads [10] as the conceptualization of time in computer music. We found that the traditional unit-generator-based synthesis frameworks entirely lack the counterpart entities and the manipulations for microsounds, which exist in the user's conceptualization. Thus, the usability issues led to a new abstraction of sound synthesis framework that integrates microsound objects and manipulations, which we developed for our new sound synthesis language. The details are described in [8].

### 3.3 The functionality issue and mostly-strongly-timed programing concept

The other problem proposed in computer music languages related to microsound synthesis is imprecise timing behav-

iors. Generally speaking, lack of precise timing behaviors can easily lead to inaccurate sound rendering in computer music, which is audible to human ears; Wang's *strongly-timed programming* is a variation of synchronous programming that integrates explicit control of logical synchronous time into an imperative programming language and thus achieves precise timing behaviors in logical time [13]. Such a feature of *strongly-timed programming* is suitable for timing issues in microsound synthesis. Together with the novel abstraction of sound synthesis framework described above, our new computer music language can provide precise timing behaviors with terse programming model.

However, since as a variation of synchronous programming, strongly-timed programming is based on the *synchrony hypothesis*, which is *"interpreted to imply the system must execute fast enough for the effects of the synchronous hypothesis to hold"* [6]. In practice, the presence of time-consuming tasks that invalidate this assumption can cause a system to fail to meet the deadline for real-time sound rendering; yet time-consuming tasks can be often seen in computer music, e.g. loading and processing of the large amount of sound data. This problem also led us to propose *mostly-strongly-timed programming*, a programming concept that integrates asynchronous behaviors and the explicit switch between a synchronous context and asynchronous context, as an extension of strongly-timed programming. The underlying scheduler can suspend and resume time-consuming tasks so as not to let them cause the failure in meeting the dead-line for real-time sound rendering; such extension can help solving the practical problem as above in audio programming. More information on *mostly-strongly-timed programming* can be seen in [9].

## 4. Evaluation Methodology

Basically, we follow Markstrum's classification of three types of claims in programming language design, which are *novel features*, *incremental improvement* (*of existing features)* and *desirable language properties* [6]. We can follow the established evaluation methods for the claims in the first two types, such as comprehensive survey of the previous study, performance analysis and comparative approaches with the existing languages.

Yet, as Markstrum also argues, programming language study has been significantly lacking an appropriate framework for the claims in *desirable language properties*, such as readability and simplicity; since we integrated the HCI frameworks such as Cognitive Dimension of Notations in the design phase for the identification and assessment of the usability problems, these frameworks are also introduced in the evaluations phase and used for the heuristic evaluations by external experts; thus, by introducing the same HCI frameworks during the design and evaluations, we can provide good *claim-evidence correspondences* for desirable language properties by comparing the designer's claims and the evaluation result by external during in the iterative design process.

## 5. Conclusion

We described our 'Research through Design (RtD)' approach in the development of a new computer music language, with a focus on the problems in microsound synthesis in the existing languages, which resulted in a novel abstraction of sound synthesis framework and programming concept for computer music. As is expected of a RtD practice, our study led to 'contribution of knowledge' through the design process of a product.

## References


[1] A.F. Blackwell and N. Collins. The Programming Language as Musical Instrument, In *Proc.of PPIG'05*, 2005

[2] A.F. Blackwell et al. Notational Systems–the Cognitive Dmensions of Notation Framework. In *HCI Models, Theories and Frameworks: Toward a Multidisciplinary Science,* Morgan Kaufmann, 2003

[3] A. Blandford et al. Evaluating System Utility and Conceptual Fit Using CASSM. In *Int'l J. of Human-Computer Studies*, *Vol.66*, 2008

[4] R. Boulanger et al. *The CSound Book*, The MIT Press, 2000

[5] E. Brandt. *Temporal Type Constructors for Computer Music Programming*. Ph.D Thesis, Carnegie Melon University, 2002

[6] S. Markstrum. Staking Claims: A History of Programming Language Design Claims and Evidence. In *Proc. of PLATEAU'10*, 2010

[7] M.V. Mathews. et al. *The Technology of Computer Music*. The MIT Press, 1969

[8] H. Nishino. and N. Osaka. LCSynth: A Strongly-Timed Synthesis Language that Integrates Objects and Manipulations for Microsounds. In *Proc. of Sound and Music Computing'12*, 2012

[9] H. Nishino. Mostly-Strongly-Timed Programming. In *Proc. of SPLASH'12, 2012*

[10] C. Roads. *Microsound*, The MIT Press, 2004

[11] J. Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Wiley, 1994

[12] G. Wakefiled and W. Smith. Using Lua for Audio Visual Composition, In *Proc. of ICMC'07*, 2007

[13] G. Wang. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality*. PhD Thesis, Princeton University, 2008.

[14] J. Zimmerman et al., Research through design as a method for interaction design research. In *Proc. of SIGCHI'07*, 2007