研究報告

新しいコンピュータ音楽言語LCとその３つの特徴
LC: A NEW COMPUTER MUSIC LANGUAGE AND ITS THREE CORE FEATURES

西野 裕樹
シンガポール国立大学
Keio-NUS Cute Centre

小坂 直敏
東京電機大学
情報メディア学科

中津 良平
シンガポール国立大学
IDM Institute

概要

This paper gives a brief overview of the three core features of LC, a new computer music language we prototyped: (a) prototype-based programming, (b) mostly-strongly-timed programming, and (c) the integration of objects and manipulations for microsound synthesis in its sound synthesis framework. After addressing three problems in existing computer music languages, which are (1) insufficient support for dynamic modification, (2) insufficient support for precise timing behavior and other features with respect to time, and (3) difficulty in microsound synthesis programming, we describe how the three core features of LC correspond to these problems, together with several code examples. Since the design of LC is significantly motivated by the problems that hinder creative practices in computer music of our time, such a language design can contribute to both academic research and creative practices, at least as a design exemplar.

本稿は著者らがプロトタイピングした新しいコンピュータ音楽用言語であるLCについて概説するものである．LCは (a) prototype-based programming(b) mostly-strongly-timed programming とその他時間に関連する機能，および (c) マイクロサウンド・シンセシスの為のオブジェクト及び操作の音響合成フレームワークへの統合という３つの特徴を有する．本稿では，まず既存のコンピュータ音楽言語における (1) 動的変更のサポートの不十分さ，(2) 正確な時間的振る舞いと時間に関する特徴の不十分さ，(3)　マイクロサウンド・シンセシスのプログラミングの難しさの３つの問題を挙げた後，LCのそれぞれの特徴がどのようにこの３つの問題に関わっているかを記述する．LCの設計はこのように現代のコンピュータ音楽の創造的実践を妨げる問題によって動機づけられているため，そのような言語設計は コンピュータ音楽における研究と実践の両面にたいして，すくなくとも デザインの一例として有益であろう．

1. はじめに

本稿では著者らが開発した新しいコンピュータ音楽用プログラミング言語LCの概要を述べる．コンピュータ音楽は様々なコンピュータ技術の発展やプログラミング言語の研究の進展などに影響されつつ，研究され続け現在に至っているが，一方でこのような外部的な要因だけではなく音楽的・創造的な要求や創作の現場で生じた問題も，コンピュータ音楽言語の発展に大きな役割を果たしてきた．同言語の設計過程において著者らは，このような音楽的・創造的な要求もその発展に寄与して来たという視点から，まず現在のコンピュータ音楽の創作活動においてプログラミング言語側の不備により起っていると考えられる問題を同定し，それを新しい言語の設計の際に解決すべき課題と捉えた．

本稿での以降の部分では，まず現在の既存のコンピュータ音楽言語における主要な問題，(1) 動的変更のサポートの不十分さ，(2) 正確な時間的振る舞いと時間に関する特徴の不十分さ，(3)　マイクロサウンド・シンセシスのプログラミングの難しさという３つの問題をあげた後，LCの３つの特徴，(a) prototype-based programming，(b) mostly-strongly-timed programming とその他時間に関連する機能，および (c) マイクロサウンド・シンセシスの為のオブジェクト及び操作の音響合成フレームワークへの統合の３つの特徴に関連させ解説する．なお，本論文は過去に公表された著者らによる論文に記述されている研究の日本向けの要約を意図して書かれたものである．

2. 既存のコンピュータ音楽における３つの問題

2.1. 動的変更のサポートの不十分さ

例えば，近年注目されている Live-coding [4] の演奏では，単にプログラムをステージ上で実行するだけではなく，既に実行中のプログラムの作曲アルゴリズムを変更するなどの動的な変更がコンピュータ音楽のシステムに加えられる事が多い．Kaltenbrunner [12] らの reacTable システムでは，音響生成モジュールのネットワークが演奏中に動的に変更される．

このようにシステムの動的な変更は作曲アルゴリズムと音響生成の両レベルのにおいて必要とされている．しかしながら多くの既存のコンピュータ音楽言語においては，少なくとも作曲アルゴリズムや音響生成レベルの両方もしくはどちらかにおいて動的変更が不可能であったり，様々な制限がかせられている事が多い．動的な変更に対するサポートの不十分さは，コンピュータ音楽言語の設計において，重視すべき問題となってきている．

2.2. 正確な時間的振る舞いと時間に関する特徴の不十分さ

時間的な振る舞いの正確さはリズムのレベルだけではなく，音響合成のレベルにおいてマイクロサウンド・シンセシスなどを行う際にも重要な要素であるが，サン

プル・レートの単位での正確な時間的振る舞いを保証するインタラクティブかつリアルタイムなコンピュータ音楽言語は，いまだそれほど多くなく[1]，多くのコンピュータ音楽言語やシステムで時間的な振る舞いの不正確さが問題となっている．

コンピュータ音楽システムにおいてサンプルレート単位（もしくはコントロールレート単位のものもあるが）で，論理時間にもとづく正確な時間的振る舞いを実現するときには，一般的にはいわゆる the ideal synchronous hypothesis[2] に基づく設計で実現される．これはコンピュータ音楽システムの実装においては，ある時間にスケジュールされたタスクが全て処理が終わるまで論理時間を進めずにブロックし，するべき処理がない場合に限って論理時間が進められ，出力サンプルの計算を行うというかたちで解釈されている．しかし，リアルタイム音響生成を滞りなく行うには，次の出力サンプルを実時間上のデッドラインに間に合うように生成し出力デバイスへと送らねばならないという制限がかせられているため，このような synchronous なアプローチのみに基づくコンピュータ音楽システム上で，時間のかかるタスクが実行された場合，音響処理のデッドラインに間にあわず音響生成が一時的にサスペンドすることになる．

また，正確な時間的振る舞いを保証する言語においても，start-time constraint や execution-time constraint など，時間に関する好ましい特徴が考慮されていなかったり，考慮されていても不十分なものとなっている．FORMULA [1] 等，過去の MIDI シンセサイザとコンピュータで構成された言語・システムの一部においては仕様として含まれていたものであるが，現状でサンプル・レート単位の正確な振る舞いとともに，このような機能が十分な形で実装されているものはない．

このため sychronous な振る舞いによるサンプルレート単位での正確な時間的振る舞いを保ちつつ，このような時間のかかるタスクをどうあつかうかとうことは，現代のコンピュータ音楽言語に提示された問題であり，同様にそこに時間に関する好ましい特徴をどのように組み込みかというのもひとつの考慮すべき課題である．

2.3. マイクロサウンド・シンセシスのプログラミングの難しさ

ユニット・ジェネレータのコンセプトが，それが発明された当時に主流であったアナログ回路による音響合成をもとに行われた抽象化であると見方は，その発案者である Mathews らの記述をみても妥当なものであると言える．[3] 実際にデジタルにマイクロサウンド・シンセシス [8] が行われたのは，ユニット・ジェネレータの登場より遥かに遅い．アナログ回路による音響合成を抽象化した側面が非常に強いユニット・ジェネレータのコンセプトと，小さな音の粒子が互いに重なり合い全体の音響を構成するマイクロサウンド・シンセシスのコンセプトにはかなりの開きがある．このようなギャップは，何らかのマイクロサウンド・シンセシスの技法をユニット・ジェネレータ言語で実装する際にユーザのコードのレベルにおいては，abstraction inversion[2] という，より高レベルの抽象を組み合わせて，低レベルの抽象を表現するという，ソフトウェア開発におけるアンチ・パターンの一つとして現れ，ユーザに対してエキスパート・レベルのプログラミング・スキルを必ずしも期待できないコンピュータ音楽言語においては，マイクロサウンドの領域における創造的な探求を阻害し好ましくないと考えられる．このようなユニット・ジェネレータ言語にマイクロサウンド・シンセシスの技法の実装におけるプログラミングの困難さは解決されるべき課題である．

[1] 例えば ChucK [10] や LuaAV [9] が等がある

[2] “Ideal systems produce their outputs synchronously with their inputs” and “all computation and communications are assumed to take zero time (that is, all temporal scopes are executed instantaneously)” [3, p.360]

[3] 例えば，Mathews らはユニット・ジェネレータについて “conceptually similar functions to standard electronic equipment used for electronic sound synthesis” を持つと記述している [6, p.15].

3. LC における３つの特徴

3.1. プロトタイプ・ベース・プログラミング

既存言語における動的変更のサポートの不十分さにを考慮しするため，LC はプロトタイプ・ベースとしてデザインされた．プロトタイプ・ベース・プログラミング [5] は，オブジェクト指向に属するプログラミング・コンセプトである．クラスというテンプレートからオブジェクトを生成するクラス・ベースの言語と違い，プロトタイプ・ベースの言語においては，まずオブジェクトを無から (ex nihilo) もしくは他オブジェクトのクローニングにより生成し，その後そのオブジェクトにたいして自由に属性やメソッドの追加や変更を，他のオブジェクトに影響を与えずに行える．このような特性によってプロトタイプ・ベースの言語は実行時における動的な変更により柔軟に対応ができる．

LC では，作曲アルゴリズムに関わる部分において，一般のプロトタイプベース言語でのオブジェクトと同様の役割を果たす *Table* を用意しているが，音響合成に関わる部分においては，ユニット・ジェネレータ言語におけるパッチの役割をはたす別のオブジェクト *Patch* を用意している．これは両者の間で想定されるオブジェクトの動作が違うことによる．例えば，オブジェクトをクローンするとき，一般のプロトタイプ・ベース言語のオブジェクトでは，浅いコピー (shallow copy) が行われ，そのオブジェクトが参照しているオブジェクトまではコピーされず，単におなじ参照の値をもつことが多い．また，delegation の機能等は必須とされている．一方，音響合成においてあるパッチをコピーする際は，そのパッチが参照しているユニット・ジェネレータやサブパッチもコピーしなければ意味がなく深いコピー（deep copy）が必要であるが、一方で delegation の機能は特に必要とされない．この様な要求される性質の違いから別のオブジェクトをそれぞれ用意した．Figure 1 と Figure 2 にそれぞれ Table オブジェクト，Patch オブジェクトの使用例を示す．

3.2. Mostly-strongly-timed programming とその他時間に関する機能

既に述べたようにサンプルレート単位の正確な動作を synchronous な振る舞いによって論理時間上では保証ができる一方，時間のかかるタスクによる次回の音響生成の実時間上のデッドラインに間にあわなくなり，音響生成に支障が生じる事態が発生しうる．この問題は「次回の音響処理のデッドラインの前に，全てのタスクの処理がおわり論理時間を進める準備ができていること」という，the ideal synchronous hypothesis の実装時における解釈に違反してしまうことにより起るものなので，単純に synchronous な振る舞いのみを行うシステムでは回避が不可能である．

prototype-based programming example (1)

```
01:  //create a Table object
02:  var obj = new Table();
03:
04:  //then, attach values and functions to its slots.
05:  obj.name = "John";
06:  obj.age = 34;
07:  obj.print = function(var self){
08:    println("name :" ..  self.name ..
09:            ", age:" ..  self.age);
09:  };
10:
11:  //calling the method.
12:  obj.print(obj);
13:  //alternatively, use —> operator
14:  //to abbreviate `self'.
14:  obj->print();
```

The output from the above example (1)

```
name :John, age:34
```

図 1. A Table object example in LC.

prototype-based programming example (2)

```
01:  //create a Patch object.
02:  var p = new Patch();
03:
04:  //store unit-generator objects to its slots.
05:  p.src = new Sin∼(440);
06:  p.dac = new DAC∼();
07:
08:  //build a connection between ugens.
09:  //then, compile the patch to update the graph
10:  p->connect(\src, \defout, \dac, \defin);
11:  p->compile();
12:
13:  //start playing the patch immediately.
14:  p->start();
15:
16:  //wait for 1 second and change the frequency.
17:  now += 1::second;
18:  p.src.freq = 880;
19:
20:  //swap a sine wave osc with a phasor.
21:  var tmp = p.src; //store a sinewave osc to tmp.
22:  p.src = new Phasor∼(440);
23:  p->compile();
24:
25:  //restore a sinewave osc after 1 sec.
26:  now += 1::second;
27:  p.src = tmp;
28:  p->compile();
```

図 2. A Patch object example in LC.

そこで，LC では synchronous programming の一種である Wang らが提案した strongly-timed programming [10] を拡張し，論理時間にたいて synchronous な振る舞いと asynchronous な振る舞いの間で明示的にコンテキストをスイッチできる mostly-stronlgy-timed programming を提案し実装した．mostly-strongly-timed なプログラムでは，スレッドが asynchronous なコンテキストに有る場合，音響生成のデッドラインに間にあうように明示的な論理時間の進行が行われずとも，スケジューラは任意のタイミングでスレッドをプリエンプト可能であり，時間のかかる処理が音響生成に支障をもたらすことを避けられるものとなる．一方で，asynchronous コンテキストではこのように明示的な論理時間の進行が指定されていなくとも，論理時間が進みうるということになる．このようにサンプルレート単位の正確な動作を synchronous コンテキストにおいて行い，時間のかかるタスクは asynchronous コンテキストに行うという，プログラム側での明示的コンテキストの切り替えにより，時間的な振る舞いの正確さを保持しつつ，時間のかかるタスクもバックグラウンドで実行可能とした．Figure 3 のコード例に示すように sync と async の２つの予約語によりコンテキストのスイッチが明示的に行われる．

また LC には，start-time constraint, execution-time constraint, timed message communication など，時間に関連した機能も実装されており，これらも論理時間においてサンプル単位での正確な振る舞いが保証されている．

A mostly-strongly-timed programming example

```
01:  //create an array with 16 elements.
02:  var wsarray = new Array(16);
03:
04:  //load 16 sound files (snd0.wav-snd15.wav)
05:  //and then perform waveset analysis.
06:  //this can be a time-consuming task and
07:  //may suspend real-time DSP temporarily.
06:  for (var i = 0; i < 16; i += 1){
08:    LoadSndFile(i, "snd" ..  i ..  ".wav");
09:    wsarray[i] = ExtractWavesets(i);
10:  }
11:
12:  //`async' block switches to the asynchronous
13:  //context.  The current thread can be preempted
14:  // even without explicit advance of logical time.
15:  async {
16:    //the below code performs the same task again,
17:    //but doesn't suspend real-time DSP this time.
18:    for (var i = 0; i < 16; i += 1){
19:      LoadSndFile(i, "snd" ..  i ..  ".wav");
20:      wsarray[i] = ExtractWavesets(i);
21:    }
22:
23:    //switching to the synchronous context.
24:    //`sync' and `async' can be nested freely.
25:    sync {
26:      //perform some task that requires
27:      //precise timing behaviour here.
28:    }
29:    //the async context (lines 15- ) is recovered.
30:  }
31:  //the end of the asyn context (lines 15-30).
```

図 3. A mostly-strongly-programming example in LC.

3.3. マイクロサウンド・シンセシスの為のオブジェクト及び操作の音響合成フレームワークへの統合

前述のように本研究では既存のユニットジェネレータ言語におけるマイクロサウンド・シンセシスのプログラミングの困難さを abstraction inversion によるものとか分析している．この分析に基づき，マイクロサウンドに直接該当するオブジェクトと関連する操作を直接表現できるライブラリを用意した．Mostly-storngly-timed programming による簡便な論理時間の制御に加え，これらのオブジェクトやライブラリによる簡潔なマイクロサウンドの操作とスケジューリングが可能となり，実現したいタスクよりも高レベルの抽象を組み合わせてプログラムを書く必然性がなくなり，対応する抽象を直接つかって、マイクロサウンド・シンセシスの様々な技法を簡潔にプログラミングすることが可能となった．

実例として，waveset harmonic distortion を行うプログラムを Figure 5 に示す．Waveset harmonic distortion とは，Figure 4 に図示されているように，基音となるもとの Waveset[4] のハーモニクスに重み付けしたのちに，もとの Waveset の上に重ね合わせるものである [8, p.207]．このコード例ではスペースの関係から第２ハーモニクスまでを利用したが，LC ではマイクロサウンドと関連した操作を直接表現できるオブジェクトとライブラリ関数・メソッドを用意した結果，単純で理解しやすいコードとして記述できるようになっている．同様のコードを SuperCollider 等で記述した場合この約２倍の長さとなり[5]，*Pbind* その他複数の SuperCollider 特有の知識を必要とするパターン・オブジェクトを利用

[4] a waveset is defined as a “distance from zero-crossing to a 3rd zero-crossing” whereas wavecycle is defined as“wavelength of sound, where clearly pitched” [11, p.50].

[5] 具体的なコード例は [7, pp.49-50] に示されている．

するためコードも複雑になりユーザにも相応の知識が要求される．

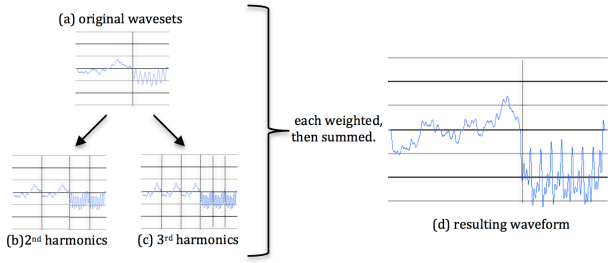

図 4. A pictorial representation of waveset harmonic distortion.

```
01:  LoadSndFile(0, "/sound/sample1.aif");
02:  var wavesets = ExtractWavesets(0);
03:
04:  //weights for 2nd harmonics
05:  var weight2 = 0.5;
06:
07:  for (var i = 0; i < wavesets.size; i +=1 ){
08:    //create the 2nd harmonics
09:    //from the original waveset.
10:    var ws = wavesets[i];
11:    var harm2 = ws->resample(ws.size / 2)
12:                  ->amplify(weight2);
13:
14:    //schedule the original waveset for output.
15:    PanOut(ws, 0.0); //0.0 is the center (stereo).
16:    //then, the 2nd harmonics (two in series)
17:    PanOut(harm2, 0.0);
18:    PanOut(harm2, 0.0, offset:harm2.dur);
19:
20:    now += ws.dur;
21:  }
```

図 5. A waveset harmonic distortion example in LC.

4. 結論

本稿では，既存のコンピュータ音楽言語における３つの問題，(1) 動的変更のサポートの不十分さ，(2) 正確な時間的振る舞いと時間に関する特徴の不十分さ，(3)　マイクロサウンド・シンセシスのプログラミングの難しさを提示・分析し，これを新しいプログラミング言語の設計開発の機会としてとらえ設計された LC 言語において，それらを解決する３つの特徴，(a) prototype-based programming，(b) mostly-strongly-timed programming とその他時間に関連する機能，および (c) マイクロサウンド・シンセシスの為のオブジェクト及び操作の音響合成フレームワークへの統合の概要を述べた．言語設計に関する問題については様々な解決が存在するであろうが，LC におけるこれらの特徴はコンピュータ音楽における創作支援および言語設計の研究にたいして，少なくともデザインの一例として有益であると考える．

5. その他

本稿ではページ数の制限から，より深く議論することはできなかったが，公表済みの論文（[7] などが `http://nus.academia.edu/HirokiNishino` からダウンロード可能である）においてより詳細に記述されているため，必要であればそちらを参照していただきたい．

6. 参考文献

[1] Anderson, D. P. and Kiuvila, R. “Formula: A programming language for expressive computer music”. *Computer Vol.24 (7)*, 1991.

[2] Baker, T. “Opening up ada-tasking”. *Proceedings of the 4th International Workshop on Real-time Ada Issues*, 1990.

[3] Burns, A. and Wellings, A. J. *Real-time systems and programming languages: Ada 95, Real-time Java and Real-time Posix*, Addison Wesley, 2001.

[4] Collins, N. et al. “Live coding in laptop performance.” *Organised Sound Vol.8 (03)*, 2003.

[5] Noble. J et al. *Prototype-based Programming: Concepts, Languages and Applications*, Springer-Verlag, 1998.

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

[7] Nishino, H. *LC: A Mostly-strongly-timed Prototype-based Computer Music Programming Language that Integrates Objects and Manipulations for Microsound Synthesis*. Ph.D. thesis, National University of Singapore, 2014.

[8] Roads, C. *Microsound*, 2004.

[9] Wakefield, G., Smith, W., and Roberts, C. “LuaAV: Extensibility and Heterogeneity for Audiovisual Computing.” *Proceedings of the Linux Audio Conference*, 2010.

[10] Wang, Ge. *The chuck audio programming language. a strongly-timed and on-the-fly environ/mentality*. Ph.D. thesis, Princeton University, 2008.

[11] Wishart, T. *Audible Design: Appendix 2*, Orpheus Books LTD, 1994.

[12] Kaltenbrunner, M. et al. “Dynamic patches for live musical performance.” *Proceedings of NIME*, 2004.