

LuaによるLive Programming環境「LuaLive」の開発

情報・通信工学科 学籍番号:1211156 成見研究室 福澤 優

1 はじめに

プログラミングは編集、コンパイル、実行のサイクルで行われる。ソースコードへの変更をプログラム上で確認するためには、コンパイルを行い、変更した場所までプログラムの実行を進める必要がある。このため、変更がパラメータの調整のような僅かなものであっても、何度もサイクルを繰り返す必要があり、プログラム開発には多大な時間を要する。この問題を解決する手法にLive Programmingがある。これは、プログラムを実行したままコードを編集し、その変更を直ちに反映し変化を観察可能にする技術である[1]。

また、Lua[2]は組み込み可能なスクリプト言語であり、ゲーム開発に用いられることが多い[3]。ゲーム開発は、プロトタイピングやパラメータ調整などで修正を何度も繰り返し行うこと、出力にサウンドやグラフィックスを伴うため変更が用意に確認できることなどから、Live Programmingが威力を發揮する分野の1つであり、LuaによるLive Programmingは利用価値が高いと考えられる。

本研究では、Luaスクリプト言語によるLive Programming環境「LuaLive」を開発を目指す。

2 関連研究

Dynamic Software Updatingは、プログラムを実行した状態で更新を可能にする技術である[4]。これは容易に停止できないプログラムに対して、動作させたまま修正や機能追加を行うための技術であり、開発の効率化を主眼としたLive Programmingとは目的が異なるものである。

また、浜中は、実行時に更新可能なLuaスクリプトを記述するプログラミングテクニックを紹介している[3]。しかし、この方法は、特定な方法でスクリプトを記述する必要がある。更新可能な部分が限られている。更新単位から更新単位外のローカル変数を使用できないなど、本研究と比較してやや制約の強いものである。

3 LuaLive

本研究では、LuaによるLive Programmingの実現を目的とし、Live Programming環境「LuaLive」を開発する。LuaLiveは、スクリプトを実行時に更新可能にするライブラリ「lualive-lib」とプログラミングを支援するエディタ拡張「lualive-code」からなる。

3.1 使用方法

LuaLiveを用いたLive Programmingの流れについて説明する(図1')。

1. lualive-libライブラリを読み込む。
2. スクリプトの実行を開始する。
3. 実行中のスクリプトを編集し、内容を保存する。
4. lualive-codeが更新リクエストを送信する。
5. 更新リクエストを受け取り更新を行う。

なお、スクリプトの実行開始と更新にはlualive-libの提供する関数lualive.loadとlualive.reloadをそれぞれ用いる。図中のホストプログラムはLuaを使用する対象のプログラムであり、Luaを呼び出せるのであれば任意のプログラミング言語で開発することができる。

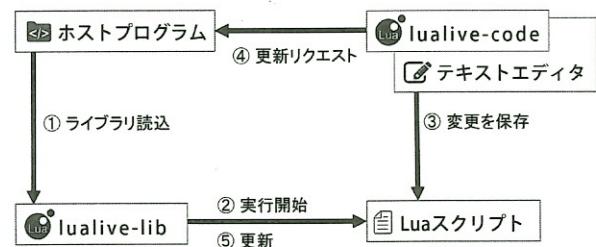


図1 LuaLiveの動作の流れ

3.2 lualive-lib

lualive-libは、Luaで書かれたLuaのライブラリである。以下のように関数やループ文などの構文にラベルを

付加することで、その構文を更新可能にする。

```
function foo() ::label1::  
    ... (略) ...  
end  
  
while true do ::label2::  
    ... (略) ...  
end
```

更新が行われると、次にその部分が先頭から開始されると変更された内容が実行される。

lualive-lib はスクリプトを直接実行するのではなく、更新可能な形に変換して実行する。これにより、実行時に各部分を新しいものに差し替えることで、更新を適用することができる。

3.3 lualive-code

lualive-code は、Visual Studio Code[5] の拡張機能である。スクリプト編集時に、lualive-lib が用いるラベルを自動で付加するほか、保存時に UDP で編集リクエストを送信する、数値の入力を補助するなどの機能を提供する。

4 評価実験

LuaLive 使用時の実行時オーバーヘッドとスクリプトの更新時間について評価実験を行った。各実験には、The Computer Language Benchmarks Game[6] から n-body, spectral-norm, binary-trees の 3 つのベンチマークスクリプトを用いた。

実行時オーバーヘッドの評価は、スクリプトを、lualive-lib を使う場合と使わない場合でそれぞれ実行して、実行時間の比を求めた。LuaLive を使った場合、実行速度は 20~50 倍低下した（表 1）。更新時間は、lualive-code がホストプログラムに更新リクエストを送信開始してから、更新が完了するまでの時間を測定した。この際、ホストプログラムは更新リクエストを受け取ったらただちに更新を行い、更新開始までの待ち時間は無いものとした。更新時間は、100 行程度のスクリプトで、20~50 ミリ秒となった（表 2）。spectral-norm と binary-trees の 2 つは行数が近いにも関わらず、更新時間に大きな差が現れた。これは、spectral-norm が変換に多くの行数を必要とする for 文を多く含んでおり、変換後のスクリプトの行数が binary-trees より多くなるためである。

表 1 実行時オーバーヘッド

スクリプト名	実行時間比
n-body	46.8
spectral-norm	36.1
binary-trees	22.4

表 2 更新時間

スクリプト名	更新時間 [ms]	行数
n-body	44	107
spectral-norm	50	33
binary-trees	24	36

5 まとめと今後の課題

本研究では、lualive-lib と lualive-code を開発し、両者を連携させて Live Programming 環境が構築できることを示した。今後の課題としては、実行時間が著しく低下することが挙げられる。これは公式の Lua 処理系より高速な Luajit[7] を用いることで解消できると考えられる。また現在の実装は、更新時に全ての更新単位を更新するため無駄に時間が掛かっている。これを変更された更新単位だけ更新するように改良して更新時間を短縮したい。

参考文献

- [1] Sebastian Burckhardt, Manuel Fahndrich, Peli de Halleux, Sean McDermid, Michal Moskal, Nikolai Tillmann and Jun Kato. "It's alive! continuous feedback in UI programming." 95-104 PLDI (2013).
- [2] The Programming Language Lua
<http://www.lua.org/>
- [3] 浜中誠 (2008). スクリプト言語による効率的ゲーム開発 C/C++へのLua組込み実践 ソフトバンククリエイティブ
- [4] Seifzadeh H, Abolhassani H, Moshkenani MS (2013) A survey of dynamic software updating. J Softw Evol Process 25(5):535568
- [5] Visual Studio Code - Code Editing. Redefined
<https://code.visualstudio.com/>
- [6] The Computer Language Benchmarks Game
<http://benchmarksgame.alioth.debian.org/>
- [7] The Luajit Project
<http://luajit.org/>