Javaの競争相手 スクリプト言語

Judith Bishop and Riaan Hurt
er コンピュータサイエンス学
部 プレトリア大学 プレトリア、
0002 jbishop@cs.up.ac.za

1. INTRODUCTION

時代が進むにつれて、あらゆる種類の新しいプログラミング言語がどんどん市場に登場している [Trott 1997]。膨大な数の労働力によって開発されたものもあれば、一人の人間の仕事から生まれたものもある。どの言語もたいていの場合、新しい機能を提供するか、あるいは機能を向上させるというひとつの目標を持っている。新しい言語の主な利点は、言語の機能が常にハードウェアやソフトウェアの技術革新と密接に一致するようにするという事実にある。この現象の良い例がJavaアプレットで、今ではウェブ・ブラウザをより魅力的なものにしているが、数年前までは利用できなかった。アプレットを提供するために、C++はかなりの変態を遂げ、Javaとして再登場した [Gosling 1997]。

一つの言語が完全な革命を起こし、産業界と学界の共通語になることは稀である。1980年代にこのような地位を獲得した言語は、マイクロコンピュータとホームコンピューティングの登場によるPascalと、Unixとクライアント・サーバー・プログラミングの台頭によるC言語と言えるだろう。Javaは確かに、このようなあらゆるもののデファクト・スタンダードになりつつある。本稿では、スクリプト言語という特定のカテゴリーを取り上げる。このグループのビッグ3は、Tcl／Tk、Perl、Pythonである。これらの言語が敬遠されているのは、単にサン・マイクロシステムズの後ろ盾がないからなのだろうか？Javaに比べると全体的に劣っているが、独自の分野では輝いているということなのだろうか？これらの問題を検証することで、私たちは次のような問いに答えることを目指している： Javaにないものは何か？一般論として、Javaと比較してどうなのか？そして最後に、これらはコンピューティング・カリキュラムにどのように組み入れるべきか？

第2章 スクリプト言語とは何か

2.1 Origins

スクリプト言語の起源は、JCL、REXX、Unixシェルなどのオペレーティングシステムのジョブ制御言語である。スクリプト言語は徐々に進化し、パラメータ化、マクロ、制御構造、グラフィカル・ユーザー・インターフェースのサポートを含むようになった。そこで生まれたのが、本稿で取り上げるPerl、TCL、Pythonのような最新のスクリプト言語である。

より洗練されたスクリプト言語の開発に影響を与えた要因は、プロセッサの高速化（これは解釈のバランスに影響を与えた）、グラフィカル・ユーザー・インターフェースの需要、インターネットのユビキタス化、そしてオブジェクト指向の圧倒的な必要性であった。すべてのスクリプト言語がOOを不可欠なものとしてスタートしたわけではないが（Pythonはそうだった）、例えばPerlはバージョン5からOOを提供し、Tclはauの拡張機能（itcl）を持っている[van Rossum 1998, Ousterhout 1998]。

2.2 Common features

Perl、Tcl、Pythonのようなスクリプト言語は、Javaのようなシステム・プログラミング言語とは異なるスタイルのプログラミングである。スクリプト言語は、アプリケーションをグルーイングするために設計されている。つまり、システム・プログラミング言語よりも高度なプログラミングと、より迅速なアプリケーション開発を実現するために、ほとんど型付けのないアプローチを使用している。

優れたスクリプト言語とは、複雑なアプリケーションを開発するのに必要なプロセスフローとデータ整理を備えながら、些細なツールを素早く簡単に開発できる高水準のソフトウェア開発言語である。実行速度が速くなければならない。ファイル操作、プロセス間通信、プロセス制御などのシステムリソースを呼び出す際に効率的である。

あらゆる一般的なオペレーティング・システム上で動作し、自由形式のテキストを効率的に処理できるように調整されており、数値や生のバイナリ・データを処理するのにも適している。組み込み可能で、拡張性もある。システム・プログラミング言語との対比を以下に示す：

1. システム・プログラミング言語は、データ構造やアルゴリズムをゼロから構築するために設計された。対照的に、スクリプト言語は、強力なコンポーネントのセットが存在することを前提とし、コンポーネント同士を接続することを主な目的としている。

2. システムプログラミング言語は、複雑さを管理するために強く型付けされる # 一方、スクリプト言語は、コンポーネント間の接続を単純化し、迅速なアプリケーショ ン開発を提供するために型付けされない。実際、Tclのデータ型は文字列だけである。

3. 成功するスクリプト言語は、オペレーティングシステムのユーティリティや サービスを簡単に呼び出して実行できることで区別される。次のレベルに到達し、汎用言語として機能するためには、複雑なアプリケーションプログラム全体を構築できるほど堅牢でなければならない。スクリプト言語は、プロトタイプ作成、モデル化、テストに使用される。スクリプト言語が十分に堅牢で高速であれば、プロトタイプはそのままアプリケーションに発展する。

4. スクリプト言語は、詳細を後回しにすることで開発をスピードアップする。スクリプト言語は一般的に、システム・ユーティリティが存在せず必要な場合に呼び出すのが得意なので、コンパイル言語で書くのは簡単だろう。一方、システム言語はすべてを明示する必要がある。そのため、複雑なデータ構造のコンパイルは容易になるが、プログラミングはより時間がかかる。スクリプト言語は、できる限り多くのことを仮定するため、習得が容易で、書くのも速い。その代償として、複雑なデータ構造やアルゴリズムの開発が難しくなる [Ousterhout 1998]。

5. スクリプト言語はインタプリタ型であるため、コンパイル型言語よりも実行速度が遅い。インタプリタ型言語の利点は、インタプリタが動作するシステムであれば、プログラムが移植可能であることである。いくつかのスクリプト言語は、現在ではバイトコードなどの中間コードにコンパイルされ、クロスプラットフォームのアプリケーション言語として重宝されている。

要約すると、汎用プログラミングに対するスクリプトの主な利点はコストである。今日、開発時間は高速なハードウェアやメモリよりも高価である。スクリプト言語は、習得が容易で、よりシンプルで高速に使用できると考えられている。[Ousterhout 1998, Masse 1996].

2.2 Tcl/Tk

ウィジェットツールキット付きツールコマンド言語（Tcl／Tk）は、カリフォルニア大学バークレー校のJohn Ousterhoutによって開発されたプログラミングシステムである。Tclは基本的なプログラミング言語であり、Tkは他のGUIツールキットと同様のグラフィカルオブジェクトであるウィジェットのツールキットである[Harrison and McLennan 1998]。Tclは、中小規模のアプリケーションを素早く作成したり、既存のアプリケーションを簡単につなぎ合わせたりするための高レベルのスクリプト言語である。Tclはシンプルな構文を持ち、ほとんど構造を持たないので、スクリプトを書くのに適している。しかし、Tclはインタプリタ型言語であり、高度にインタラクティブな計算を伴うような非常に大規模なタスクには向かないかもしれない。TclはUNIXシェルのようなものだと考えることができるが、組み込み可能で移植性があり、インターネット・スクリプトに使用できる点が異なる。

Tclは、変数代入、プロシージャ呼び出し、制御構造など、従来の手続き型言語の機能の多くをサポートしており、さらにTkを通じてグラフィカルウィジェットへのアクセスが非常に使いやすくなっている。Tclは従来の多くの言語よりもデータの扱いが厳密ではなく、データ構造化やモジュール式の名前空間を提供しない。実際、Tclは純粋な文字列置換言語であり、パフォーマンスに影響を与える。Tclはデータやコードを内部形式で保存しないので、何度も再パースする必要がある。したがって、高い信頼性と堅牢性が要求される大規模で複雑なプログラムを書くには、適切な言語ではないかもしれない。

ボタンを表示し、そのボタンを押すとスクリプトが終了するような # 単純なTclプログラムは、次のようになる：

```
button .b -text "Press to end" -command exit
pack .b
```

スクリプティング・プログラムにありがちなように、入門的な大騒ぎはなく、ただステートメントがあるだけです。Tclは、終了コマンドを持つボタンとウィンドウを表示するボタンをくっつけた。Tclのより本質的な例は、今度はTkを使った、画面上のストップウォッチ [Laird and Soraiz 1997]です：

```
set seconds 0.0; set stopped 1

label .stopwatch_display -text $seconds
button .start -text Start -command {
    if $stopped {
        set stopped 0
        tick
    }
}
button .stop -text Stop -command {set stopped 1}
pack .stopwatch_display -side bottom -fill both
pack .start .stop -side left

proc tick { } {
    global second stopped
    if $stopped return
    after 100 tick
    set seconds [expr $seconds + .1]
    .stopwatch_display config -text [format "%.1f" $seconds]
}
```

2.3 Perl

Perlは、Unixシェルの使いやすさと、C言語のようなシステム・プログラミング言語のパワーと柔軟性の融合を目指し、ラリー・ウォールによってUnix用のスクリプト言語として開発された。コモン・ゲートウェイ・インターフェイス（CGI）は、ウェブ・サーバーから別のプログラムにデータを渡し、その結果をウェブ・ページとして返すシンプルなメカニズムを提供した。Perlは瞬く間にCGIプログラミングの主流言語となった。

JavaやActiveXが注目されているにもかかわらず、「インターネットを活性化する」本当の仕事はPerlにあるようだ。Perlはコンピュータ科学者の世界ではあまり知られていないが、ウェブマスター、システム管理者、プログラマーなど、カスタムウェブアプリケーションを構築したり、設計者がまったく予見していなかった目的のためにプログラムをつなぎ合わせたりするような仕事を日常的に行う人たち以外にはよく使われている。サンの初代ウェブマスターであるハッサン・シュローダーはかつてこう言った：「Perlはインターネットのダクトテープだ」。実例として、世界最大のウェブビジネスのひとつであるAmazon.comを考えてみよう。Amazon.comは、バックエンドのデータベースとオーダーエントリーシステムへの情報フロントエンドを提供しており、Perlはこの2つを結びつける主要なコンポーネントである。Perlによるデータベースへのアクセスは、DBIと呼ばれる強力なデータベース非依存インターフェースのセットによってサポートされている。Perl + fast-CGI + DBIは、おそらく現在ワールドワイドウェブで最も広く使われているデータベースコネクタです [Hurter 1998]。

Perlの情報処理能力の重要な部分は、HTMLのような自由形式のテキストから Perlが認識したパターンに基づいてアクションを実行する巨大な力を与える 正規表現から来ています。他の言語でも正規表現はサポートされているが（Java用のフリーウェアの正規表現ライブラリもある）、Perlほど正規表現をうまく統合している言語は他にない。残念ながら、Perlの強力さは欠点でもある。ある名前のリストをソートしたインデックスを返したいとしよう。これを実現するPerlの 「ワンライナー 」は [Hall and Schwartz 1998]である：

```
@rank[sort {$x[$a] cmp $x[$b]} 0..$#x] = 0..$#x
```

ネットワークベースの言語としてのPerlの強みを示す、より現実的な例は、 TCP／IP psデーモン[Hall and Schwartz 1998]です：

```
use strict;
use Socket;
my $port = 2001;
my $proto = getprotobyname 'tcp';
my $ps = 'usr/ucb/ps';

socket SERVER, PF_INET, SOCK_STREAM, $proto
    or die "socket: $!";
```

```
bind SERVER, sockaddr_in($port, IADDR_ANY)
    or die "bind: $!";

listen SERVER, 1 or die "listen: $!";
print "$0 listening to port $port\n";
for (;;) {
    accept CLIENT, SERVER;
    print CLIENT '$ps';
    close CLIENT;
}
```

Bishop 1998]の同じサーバーのJavaバージョンは4倍近く長い。

2.4 Python

Pythonは他の2つの言語よりも新しい。Pythonは他の2つの言語よりも新しい言語であり、設計者のGuido van Rossumがオランダの研究機関CWIで働いていた1991年にリリースした。Pythonの機能の多くは、ABCと呼ばれるインタプリタ言語に由来している。ヴァン・ロッサムはアメーバ分散オペレーティング・システム・グループの仕事をしており、ABCのような構文を持ちながらアメーバのシステムコールにアクセスできるスクリプト言語を探していた。彼はModula2を使った経験があったので、Modula-3の設計者と話をすることにした。こうして組み込まれたモジュール機能により、PythonはTclやPerlとは異なり、「大規模なプログラミング」に適している[Laird and Soraiz 1997]。

Pythonはインタプリタ型、バイトコンパイル型、対話型、オブジェクト指向のプログラミング言語です。モジュール、例外、動的型付け、非常に高度な動的データ型、クラスが組み込まれている。Perlがそうであるように、Perthonもまた、テキスト処理や、時には汎用的なGUIプログラミングを得意とする、主に手続き型（命令型）プログラミング言語である。多くのシステムコールやライブラリ、さまざまなウィンドウシステムとのインターフェイスを持ち、CやC++で拡張可能です。また、プログラミングインターフェースを必要とするアプリケーションのための拡張言語としても使用できる。最後に、Pythonはすべての主要なハードウェアおよびソフトウェアプラットフォームに移植可能である [Python website 1999]。

Pythonは、TclやPerlよりも汎用言語に近いスタイルで、非常にカンプアクトで読みやすいプログラムを書くことができます。Pythonで書かれたプログラムは、CやJavaで書かれたプログラムよりもはるかに短いのが一般的です：

- 高水準のデータ型により、複雑な操作を1つの文で表現できる。
- Statement grouping is done by indentation instead of begin-end brackets.
- 変数や引数の宣言は不要。

For example, consider a Python function to invert a table [van Rossum 1997]:

```
def invert (table);
    index = { }
    for key in table.keys():
        value = table[key]
        if not index.has_key(value):
            index[value] = []
        index[value].append(key)
    return index
```

Pythonについては、次のセクションで詳しく説明します。

3 スクリプト言語とJava

Javaはすでにスクリプト言語自体にかなりの影響を与えている。例としては以下のようなものがある：

- Tcl 8.0は、Javaバイトコードへのコンパイラ(Tcl Blend)を提供することで、速度の問題に対処し、制限されたデータ型と名前空間を追加した [van Rossum undated]。

- Jacl はTclインタプリタを100%Javaで再実装したものだが、バイトコードは生成しない [van Rossum 1998]；

- 新しいPerlコンパイラのバックエンド(JPL)は、Perlアプリケーションを Javaバイトコードにコンパイルすることができる；

- JPythonはバイトコードを生成するPythonインタプリタで、100%Javaで書かれている。JPythonは、100%Javaで書かれたバイトコードを生成するPythonインタプリタであり、独自のフォロワーとウェブサイトを持っている [JPython website, van Rossum 1998]。

JPythonのJavaとPythonの統合は驚くほどシームレスです。van Rossum [1998]が言うように: ”Pythonスクリプトは、Javaクラスをインポートし、静的メソッドやインスタンスメソッドを 呼び出し、インスタンスを生成し、Javaクラスのサブクラスを生成することもできます。これらのサブクラスのインスタンスはJavaに渡すことができ、（Pythonで実装された）そのメソッドをJavaコードから呼び出すことができる。」 Masse[199]は、「Starfield 」のデモを示す以下のJPythonアプレットを提供している。ポイントは、同等のJavaコードは少なくとも3倍の長さであり、JavaとPythonの混合のケースを強化するということです。

```
from Tkinter import *
import string
from whrandom import random

class Starfield: # Copyright (c) 1996 CNRI
    def _init_(self, master, width=400, height=100, numstars=50):
        self.width = width
        self.height = height
        self.canvas = Canvas(master, width=width, height=height, background='black')
        self.canvas.pack()
        self.star = []
        for i in range(numstars):
            x = int(random()*width)
            y = int(random()*height)
            z = 1+ int(random()*10)
            c = 35 + z*20
            color = "#%02x%02x%02x" % (c, c, c)
            tag = self.canvas.create_oval(x,y, x+4, y+4, fill=color)
            self.stars.append((tag, z, [x]))
        self.update()

    def update(self):
        try: self.canvas['width']
        except: return
        for tag, z, xlist in self.stars:
        x = xlist[0]
        self.canvas.move(tag, z, 0)
        x = x + z
        if x > self.with:
            self.canvas.move(tag, -self.width, 0)
            x = x- self.width
        xlist[0] = x
        self.canvas.after(10, self.update)
```

カリキュラムの4つのスクリプト

スクリプト言語には、Unixシェル、Visual Basic、JavaScriptがあり、いずれもコンピュータサイエンスのカリキュラムに含まれる。しかし、本稿で取り上げる主要な3つの言語が存在しないことは注目に値する [McCauley and Manaris 1998]。つの主要なコンピュータサイエンス教育会議 [SIGCSE 1998とITiCSE 1998] では、Javaは多くの論文のタイトルに登場したが、スクリプティングは皆無であった。今日のコンピュータ・システムにおけるスクリプティングの重要性を考えると、オペレーティング・システムのコースだけでなく、システム統合、高度なプログラミング、比較プログラミング言語、ソフトウェア工学、データベースなどのコースでも、スクリプティングを取り上げるべき場所があるように思われる。また、スクリプティングはそれ自体、卒業生がシステム・エンジニアになるための大学院のコースとみなされるべきであると主張することもできる。

Ousterhout [1998]は、スクリプティングに関する彼の代表的な論文の中で、今後10年間で、より多くの新しいアプリケーションが、完全にスクリプティング言語で書かれるようになり、システム言語は主にコンポーネントを作成するために使われるようになるだろうと予測している。私たちはこれに同意しない。Lewis [1997]が 「Javaが答えなら、質問は何だったのか？」という挑発的な質問を投げかけて以来、Javaはまったく認識されずに発展してきた。Javaは、JVMによってサポートされ、すべてのブラウザで実行されるコア言語に、何百ものクラスやAPIが追加され、あらゆる認識から発展してきた。データベースへのアクセス、サーバーの実行、モバイル機器へのコードの埋め込み、FortranからCorbaまで、PerlやTclの強みである他の言語のコンポーネントへのリンクなどの機能がある。さらに、van Rossum [1998]に反して、JavaにはPythonが誇るハッシュテーブル、辞書、その他の洗練されたデータ構造が組み込まれている。スクリプト言語がCやC++を補完するケースは確かにある。Javaがスクリプトを必要とするケースは、Python信奉者であるvan Rossum [1998]やMasse [1996]が主張するほど説得力はない。

5 CONCLUSION

スクリプト言語はますます強力になり、汎用システムプログラミング言語との境界線は曖昧になっている。バイトコードがスクリプト言語インタプリタのターゲットになったので、これは他のどの言語よりも(Cでさえも)Javaで顕著である。Javaは、スクリプト言語がすることすべてをできるわけではないし、その開発速度もない。その一方で、Javaは非常に多くのことを行うので、スクリプトがその市場シェアを拡大することはないだろうというケースもある。バランスから言えば、われわれは[Laird and Soraiz 1997]に同意する： 「最後に、自分自身を大切にしましょう。スクリプト言語を学びなさい。レガシーコンポーネントがうまく協調して動作するように教えるのは、想像していたよりも簡単なことです。また、本格的なスタンドアロンアプリケーションのプロトタイプを作成するのも、記述方法を理解するのにかかっていた時間よりも短時間で済むでしょう。一度スクリプトを始めたら、やめられなくなるだろう。」

REFERENCES

Bishop J, **Java Gently 2nd ed**, Addison Wesley 1998

Gosling J, *The feel of Java*, IEEE Computer **30** (6) 53–58 June 1997

Hall J N and Schwartz R L, **Effective Perl programming**, Addison-Wesley 1998

Harrison M and MacLennan M, **Effective Tcl/Tk programming**, Addison-Wesley 1998

Hurter R, *Competitors to Java*, Computer science honours project, University of Pretoria, 1998

ITiCSE, Proceedings of the 4th International Conference on Integrating Technology into the Computer Science Curriculum, Dublin Ireland, June 1998, in SIGCSE Bulletin **30** (2) June 1998

JPython website at www.JPython.org, visited on 16 May 1999

Laird C and Soraiz K, *Choosing a scripting language*, SunWorld, October 1997 on www.sunworld.com/swol-10-1997/swol-10-scripting.html, visited 16 May 1999

Lewis T, *If Java is the answer, what was the question?*, IEEE Computer **30** (3) 133–136 March 1997

Masse R E, *An analysis of two next-generation languages: Java and Python*, presented at the Fifth Python Workshop, Washington, November 1996, and available on www.python.org/~rmasse/papers/java-python96.html, visited on 16 May 1999

McCauley R and Manaris B, *Computer science degree programs: what do they look like?*, in Proc. 29th ACM SIGCSE Technical Symposium on Computer Science Education, Atlanta USA, February 1998, in SIGCSE Bulletin **30** (1) 15–19 March 1998

Ousterhout J K, *Scripting: higher-level programming for the 21st century*, IEEE Computer **31** (3) 23-30 March 1998

Python website at www.python.org, visited on 16 May 1999

SIGCSE, Proceeding of the 29th ACM SIGCSE Technical Symposium on Computer Science Education, Atlanta USA, February 1998, in SIGCSE Bulletin **30** (1) March 1998

Trott P, *Programming languages: past present and future*, ACM SIGPLAN Notices **32** (1) 14–87, January 1997

van Rossum G, *Comparing Python to other languages,* undated, on www.python.org/doc/essays/comparison.html, visited on 16 May 1999

van Rossum G, *Java and Python: a perfect couple*, Developer Journal, August 1998, on www.developer.com/journal/techfocus/081798_jpython.html, visited on 16 May 1999