Competitors to Java: Scripting languages

Judith Bishop and Riaan Hurter
Computer Science Department
University of Pretoria
Pretoria, 0002
jbishop@cs.up.ac.za

1. INTRODUCTION

As time progresses, we see more and more new programming languages of all kinds appearing on the market [Trott 1997]. Some are developed by a tremendous work force, while others spring forth from a single person's work. All usually have one goal, which is to provide either new or improved functionality. The main advantage of new languages lies in the fact that they always make sure language functionality closely matches hardware and software innovations. A good example of this phenomenon are Java applets, which now make web browsers much more attractive, but which were unavailable until a few years ago. To provide applets, C++ underwent a considerable metamorphosis and re-emerged as Java [Gosling 1997].

It is rare that a single language causes a complete revolution and becomes a *lingua franca* for industry and academia. Languages which achieved this status in the 1980s could be said to be Pascal (with the arrival of microcomputers and home computing) and C, with the rise of Unix and client-server programming. Java is certainly well on the way to becoming such a *de facto* standard for everything, and it is therefore a good time to look at it careful, in the light of other modern languages, which we shall call its competitors. In this paper we shall look at one specific category of languages, the scripting languages. The big three in this group are Tcl/Tk, Perl and Python. Are these languages being sidelined because they simply do not have the muscle of Sun Microsystems' backing? Are they genuinely not as good as Java overall, but do they shine in their own area? In examining these issues, we aim to answer the questions: What do they provide that perhaps Java does not? How do they compare in general terms to Java? And finally, how should they fit into the computing curriculum?

2 WHAT ARE SCRIPTING LANGUAGES

2.1 Origins

Scripting languages have their origins in the job control languages of operating systems, such as JCL, REXX and Unix shell. They gradually evolved to include parameterisation, macros, control structures and support for a graphical user interfaces. So were born the modern scripting languages, such as Perl, TCL and Python, which we examine in this paper.

The factors that influenced the development of more sophisticated scripting languages were the increased speed of processors (which balanced out interpretation), the demand for graphical user interfaces, the ubiquity of the Internet, and the overwhelming need for object orientation. Although not all scripting languages started off with OO as an integral part (Python did), they do make an effort to provide it as an add-on, for example Perl provided OO from version 5 and Tcl has an au extension (itcl) [van Rossum 1998, Ousterhout 1998].

2.2 Common features

Scripting languages such as Perl, Tcl and Python represent a different style of programming to system programming languages such as Java. Scripting languages are designed for glueing applications, that is, they use mostly typeless approaches to achieve a higher level of programming and more rapid application development than system programming languages.

A good scripting language is a high-level software development language that allows for quick and easy development of trivial tools while having the process flow and data organisation necessary to develop complex applications. It must be fast while executing. It is efficient when calling system resources such as file operations, interprocess communications, and process control. It runs on every popular operating system, is tuned for the efficient processing of free form text and yet is reasonable at processing numbers and raw, binary data. It is embeddable, and extensible. The following contrasts with system programming languages apply:

1. System programming languages were designed for building data structures and algorithms from scratch, starting from the most primitive computer elements such as bytes and words of memory. In contrast, scripting languages assume the existence of a set of powerful components and are intended primarily for connecting components together.
2. System programming languages are strongly typed to help manage complexity, while scripting languages are typeless to simplify connections between components and provide rapid application development. In fact, Tcl has only one data type – the string.
3. Successful scripting languages distinguish themselves by the ease with which they call and execute operating system utilities and services. To reach the next level, and function as general-purpose languages, they must be robust enough that an entire complex application program can be built. The scripting language is used to prototype, model, and test. If the scripting language is robust and fast enough, the prototype evolves directly into the application.
4. Scripting languages speed up development by leaving the details for later. Since scripting languages are generally good at calling system utilities if one doesn't exist and is necessary, it will be easy to write in a compiled language. System languages on the other hand need to have everything spelled out. This makes compilation of complex data structures easier, but programming a more drawn out process. Scripting languages make as many assumptions as they can, so they are easier to learn and faster to write in. The price to be paid is difficulty in developing complex data structures and algorithms [Ousterhout 1998].
5. Scripting languages are interpreted, which makes them slower in execution than compiled languages. The advantage of interpreted languages is that programs are portable to any system that the interpreter will run on. Some scripting languages are now compiled to an intermediate code such as byte code which makes them valuable cross platform application languages.

In summary, the main advantage of scripting over general-purpose programming is cost. Development time is today usually more expensive than fast hardware and memory. Scripting languages are seen as mostly easier to learn, and simpler and faster to use. [Ousterhout 1998, Masse 1996].

2.2 Tcl/Tk

The Tool Command Language with the widget Toolkit (Tcl/Tk) is a programming system developed by John Ousterhout at the University of California, Berkeley. Tcl is the basic programming language, while Tk is a Tool Kit of widgets, which are graphical objects similar to those of other GUI toolkits [Harrison and McLennan 1998]. Tcl is a high-level scripting language for creating small and medium-sized applications quickly as well as easily glueing together existing ones. It has a simple syntax and almost no structure, which makes it good for scripting. However, Tcl is an interpreted language and it may not perform well for very large tasks that involve highly interactive calculations. One can think of Tcl as something like UNIX shell, except that it is embeddable and portable and can be used for Internet scripting.

Tcl supports many of the features of conventional procedural languages, including variable assignment, procedure calls, control structures, and in addition it has very easy to use access to graphical widgets through Tk. Tcl is less strict in its handling of data than many conventional languages, and does not provide their data structuring or modular namespaces. In fact, Tcl is purely a string-substitution language, which affects its performance. Tcl does not store its data or code in internal form, which means it has to re-parse it again and again. Thus it may not be an appropriate language in which to write large and complex programs, which are required to be highly reliable and robust.

A simple Tcl program to display a button which when pressed will cause the script to exit would be:

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

As is typical with scripting programs, there is no introductory fuss, just statements. Tcl has glued together a button with an exit command and one to display a window. A more substantial example of Tcl, this time using Tk, is given by an on-screen stop watch [Laird and Soraiz 1997]:

```
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 was originally developed by Larry Wall as a scripting language for Unix, aiming to blend the ease of use of the Unix shell with the power and flexibility of a system programming language like C. Perl quickly became the language of choice for Unix system administrators, and with the advent of the world wide web, Perl usage exploded. The Common Gateway Interface (CGI) provided a simple mechanism for passing data from a web server to another program, and returning the result of that program interaction as a web page. Perl quickly became the dominant language for CGI programming.

Despite all the press attention to Java and ActiveX, the real job of "activating the Internet" seems to belong to Perl. Perl is not well known in the world of computer scientists but which is most often used by everyone else, such as webmasters, system administrators and programmers whose daily work involves building custom web applications or glueing together programs for purposes their designers had not quite foreseen. Hassan Schroeder, Sun's first webmaster, once remarked: "Perl is the duct tape of the Internet." As a practical example, consider Amazon.com, one of the biggest web businesses in the world. It provides an information front-end to a back-end database and order-entry system, with Perl as a major component tying the two together. Perl access to databases is supported by a powerful set of database-independent interfaces called DBI. Perl + fast-CGI + DBI is probably the most widely used database connector on the world wide web at the moment [Hurter 1998].

An important part of Perl's information-handling power comes from regular expressions which give Perl enormous power to perform actions based on patterns that it recognises in a body of free form text, such as HTML. Other languages support regular expressions as well (there is even a freeware regular expression library for Java), but no other language integrates them as well as Perl. Unfortunately, Perl’s power is also a drawback, since the language can sometimes be hard to understand and use. Suppose we wish to return the indices of the sorted version of a list of names. The Perl “one-liner” to accomplish this would be [Hall and Schwartz 1998]:

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

A more realistic example showing Perl’s strengths as a network based language is a TCP/IP ps daemon [Hall and Schwartz 1998]:

```
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;
}
```

A Java version of the same server in [Bishop 1998] is nearly four times as long.

2.4 Python

Python is a newer language than the other two. It was released by its designer, Guido van Rossum, in 1991 while he was working for the research institute CWI in the Netherlands. Many of Python's features originated from an interpreted language called ABC. Van Rossum was working on the Amoeba distributed operating system group and was looking for a scripting language with a syntax like ABC but with the access to the Amoeba system calls, so he decided to create a language that was generally extensible. Since he had some experience with using Modula-2, he decided to talk with the designers of Modula-3. The modular features thus incorporated make Python suitable for “programming in the large”, unlike Tcl and Perl [Laird and Soraiz 1997].

Python is an interpreted, byte-compiled, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It is a largely procedural (imperative) programming language that is good at text processing and sometimes even general-purpose or GUI programming, just as Perl is. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as en extension language for applications that need programming interfaces. Finally, Python is portable across all major hardware and software platforms [Python website 1999].

Python allows the writing of very compact and readable programs, more similar in style to general-purpose languages than Tcl or Perl. Programs written in Python are typically much shorter than equivalent C or Java programs, for several reasons:

- The high-level data types allow you to express complex operations in a single statement.
- Statement grouping is done by indentation instead of begin-end brackets.
- No variable or argument declarations are necessary.

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
```

There is more on Python in the next section.

3 SCRIPTING LANGUAGES AND JAVA

Java has already had a considerable influence on the scripting languages themselves. Examples are:

- Tcl 8.0 address the speed issue by providing a compiler (Tcl Blend) to Java bytecode, and adds limited data types and namespaces [van Rossum undated].

- Jacl is a 100% Java re-implementation of the Tcl interpreter, but not producing bytecode [van Rossum 1998];
- A new Perl compiler back-end (JPL) has been created that could compile Perl applications to Java bytecode, thus creating a standard way to incorporate Perl into Java without embedding the Perl interpreter [Hurter 1998];
- JPython is a Python interpreter producing bytecode and written entirely in 100% Java. It has its own following and website [JPython website, van Rossum 1998].

JPython's integration between Java and Python is remarkably seamless. As van Rossum [1998] says: " A Python script can import any Java class, call static and instance methods, create instances, and even create subclasses of Java classes. Instances of these subclasses can be passed back to Java and their methods (implemented in Python) can be called by Java code." Masse [199] gives the following JPython applet which shows a "Starfield" demo. The point is that the equivalent Java code is at least three times as long, thus strengthening the case for a Java-Python mix.

```
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)
```

4 SCRIPTING IN THE CURRICULUM

Scripting languages include Unix shell, Visual Basic and JavaScript, which all have their place in a computer science curriculum. However the main three languages discussed in this paper are remarkable by their absence [McCauley and Manaris 1998]. At the two main computer science education conferences [SIGCSE 1998 and ITiCSE 1998], Java appeared in the title of many papers, but scripting in none. Given the importance of scripting in the computer systems we run today, it would seem that there is a place for covering scripting in operating systems courses, as well as courses on systems integration, advanced programming, comparative programming languages, softwre engineering and databases. It could also be claimed that scripting in its own right should be seen as a postgraduate course, preparing graduates to become system engineers.

In his seminal paper on scripting, Ousterhout [1998] predicts that in the next decade, more and more new applications will be written entirely in scripting languages, with system languages being used mainly for creating components. We tend to disagree. Since Lewis [1997] posed his provocative "If Java is the answer, what was the question?" Java has developed out of all recognition, with hundreds of classes and APIs being added to the core language which is supported by the JVM and run by all browsers. There are facilities to access databases, run servers, embed code in mobile devices and link to components in other languages from Fortran to Corba – the strengths of Perl and Tcl. Moreover, contrary to van Rossum [1998], Java does have built in hash tables, dictionaries and other sophisticated data structures, which Python is proud of. There is certainly a case for scripting languages complementing C or C++. The case for Java needing scripting is not as convincing as made out by van Rossum [1998] and Masse [1996], both Python devotees.

5 CONCLUSION

Scripting languages are becoming ever more powerful, and the lines between them and general-purpose system programming languages are now blurred. This is more so with Java than any other language (even C) since bytecode has become the a target for scripting language interpreters. Java does not do everything a scripting language does, nor does it have its speed of development. On the other hand, Java does so much that there is a case to be made that scripting will not increase its market share. On balance, we agree with [Laird and Soraiz 1997] who end their comparative paper with: "Finally, be good to yourself. Learn a scripting language. Teaching legacy components to play together nicely will come easier than you imagined, and you'll also prototype serious standalone applications in less time than it used to take just to figure out how to describe them. Once you start scripting, you won't stop."

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