Good performance and easy coding

In the Bottle

Vala and Genie yield the high performance of compiled languages with the ease and flexibility of scripts. *By Ankur Kumar*

Every programmer wants maximum performance with minimal effort. Compiled languages like C and C++ offer high performance and a high degree of control, but they also require a high level of programming effort. On the other hand, scripting languages like Python and Ruby are easy and flexible, but often at the expense of performance. In the case of managed languages like C# and Java, the overhead and bloat of the virtual machines can hamper performance.

Programmers have developed many tools to combine the benefits of compiled and scripting languages, but these solutions are mostly add-ons or alternative forms. No single alternative addresses the needs of the programmers in one complete solution. As always, the open source technology angels are working to address this challenge with a pair of promising new programming languages known as Vala and Genie.

Understanding

Vala [1] and Genie [2] are modern programming languages designed from the ground up for the Gnome desktop environment. The main programming language of Gnome development is C, based primarily on an object system provided by the GObject library. Gnome is heavily dependent on the GLib and GObject libraries. GLib provides a portable wrapper for many lower level functions and data types and offers various advanced data structures. In most cases, it is the C counterpart to the C++ standard template library. GObject [3] is the type and object system that provides portable object orientation in C and offers memory management for automatic objects through reference counting.

In Vala and Genie, the source code is converted to a GObject-based C code representation first, then to the host platform-specific executable code. A Vala compiler does the conversion then invokes the platform-specific C compiler to produce the final executable. This design deviates from the prevalent *one compiler, one syntax* phenomenon used with other programming languages. The syntax of Vala is inspired by C# and Java, and the syntax of Genie is inspired by Python; your choice of which language is just a matter of taste and convenience. Figure 1 shows the scheme for creating executables from Vala and Genie sources.

Language bindings are available to several third-party libraries for Vala and Genie. The only prerequisites for working with Vala and Genie are the Gnome development libraries and C run-time libraries, which are available wherever the Gnome development environment is present. So, you don't need any extra performance-dampening run-time components like virtual machines or extra run-time libraries. This hybrid development provides the productivity and flexibility of managed languages with the run-time performance of compiled languages. The run-time performance of both Vala and Genie is at or near the performance of C.

MY ENVIRONMENT

I used Puppy Linux 5.0 and Ubuntu 9.10 64-bit desktop edition to test the code presented in the article.

LISTING 1: hellovala.vala

```
01 int main()
02 {
03
04      stdout.printf("\n Hello to
          FLOSS from Vala!!!\n");
05      return 0;
06
07 }
```

Installing

To work with both Vala and Genie, you need the Vala compiler, valac. If you don't care about the latest version, issue the following command to install valac on Ubuntu:

```
sudo apt-get install valac
```

Both Vala and Genie are in continuous development. For the latest version of valac, build and install it from the latest source tarball using bison, flex, GCC, and GLib sources. On Puppy Linux, all the prerequisites are fulfilled when you set up the development environment with the devx SFS module. On Ubuntu, issue the command

Mikhail Dudarev, 123RF

```
sudo apt-get install gcc ➥
   libglib2.0-dev bison flex
```

then download the latest Vala compiler source tarball from the Vala homepage and issue the following commands to build and install valac:

```
tar jxvf vala-version.tar.bz2 ➥
   && cd vala-version
./configure && make && sudo make install
```

If everything goes well, `valac --help` will show the valac help text.

Playing with Vala

To compile the `hellovala.vala` file (Listing 1) enter `valac hellovala.vala`, then type `./hellovala`. By default, valac gives the executable the same name as the Vala source file, but you can name the executable yourself by appending the `-o executablename` option.

To generate only the intermediate C source file (instead of the executable), pass the `-C` option to valac. If you examine the generated C source file, you will notice the GObject-based C programming. With the `-v` option, valac invokes the C compiler and links to the GLib and GObject libraries to generate an executable from the intermediate C file.

Vala provides the various basic data types, depending on the underlying platform. It also provides ways to know the maximum and minimum values represented by various data types. The `stdin` and `stdout` objects give you console-mode functionalities.

A `string` data type handles immutable strings. If you want to work with mutable strings, you have to instantiate `StringBuilder`, and you can use various methods, such as `append`, `prepend`, and `insert`, on the mutable strings. Vala also contains a *type inference* mechanism that lets you define a local variable with `var` instead of providing a type if the variable is unambiguous. Listing 2 shows data types, string types, and the type inference in action. `main` returns void, so the return value is `0` on execution. The `/** */`-style comments are not specific to Vala, but the Vala documentation generation utility Valadoc [4] treats these comments in special ways.

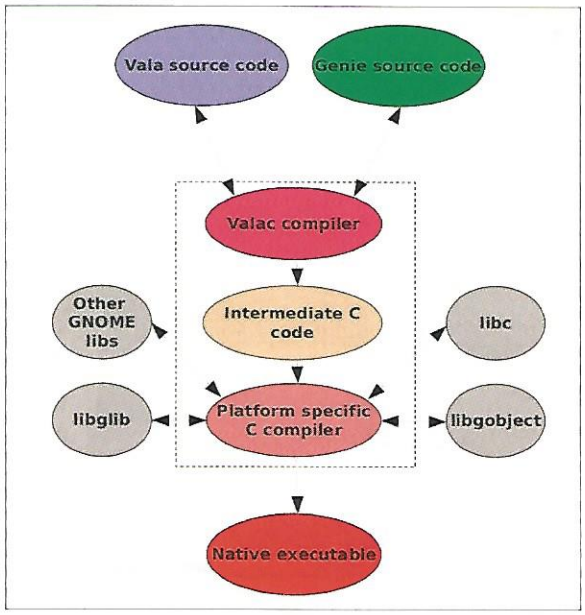

Figure 1: Executable generation from Vala and Genie.

LISTING 2: datatypes.vala

```
01 /** Data types in Vala */
02
03 void printwln(string sMsg) {
04
05     stdout.printf("\n %s", sMsg);
06
07 }
08
09 void main() {
10
11     var   ulSizeInt = sizeof(int);
12     string sSizeInt = @"sizeof(int) : $ulSizeInt";
13     printwln(sSizeInt);
14
15     int iMax    = int.MAX;
16     var iMin    = int.MIN;
17     var sMaxInt = @"int.MAX : $iMax";
18     var sMinInt = @"int.MIN : $iMin";
19     printwln(sMaxInt);
20     printwln(sMinInt);
21     printwln("");
22
23     ulong ulSizeInt8 = sizeof(int8);
24     string sSizeInt8 = @"sizeof(int8) : $ulSizeInt8";
25
26     printwln(sSizeInt8);
27     stdout.printf("\n uint8.MAX : %u"  , uint8.MAX);
28     stdout.printf("\n uint8.MIN : %u\n", uint8.MIN);
29
30     ulong ulSizeInt64 = sizeof(int64);
31     string sSizeInt64 = @"sizeof(int64) : $ulSizeInt64";
32     printwln(sSizeInt64);
33     printwln("");
34
35     var sVerbatim = """\nThis is the example of verbatim
                          string
36  that goes three lines down\\
37  \tand then closes.""";
38     stdout.printf("\n %s\n", sVerbatim);
39
40     var      sGnu = "GNU is a recursive acronym.";
41     string sFloss = "Free Libre Open Source Software
                        rulz!!!";
42     string sCmbnd = sGnu + " " + sFloss;
43     stdout.printf("\n sGnu.up()         : %s", sGnu.up());
44     stdout.printf("\n sFloss.reverse() : %s\n",
                   sFloss.reverse());
45     stdout.printf("\n sCmbnd.len()  : %ld", sCmbnd.len());
46     stdout.printf("\n sCmbnd.size() : %Zd",
                   sCmbnd.size());
47
48     printwln("\n");
49
50 }
```

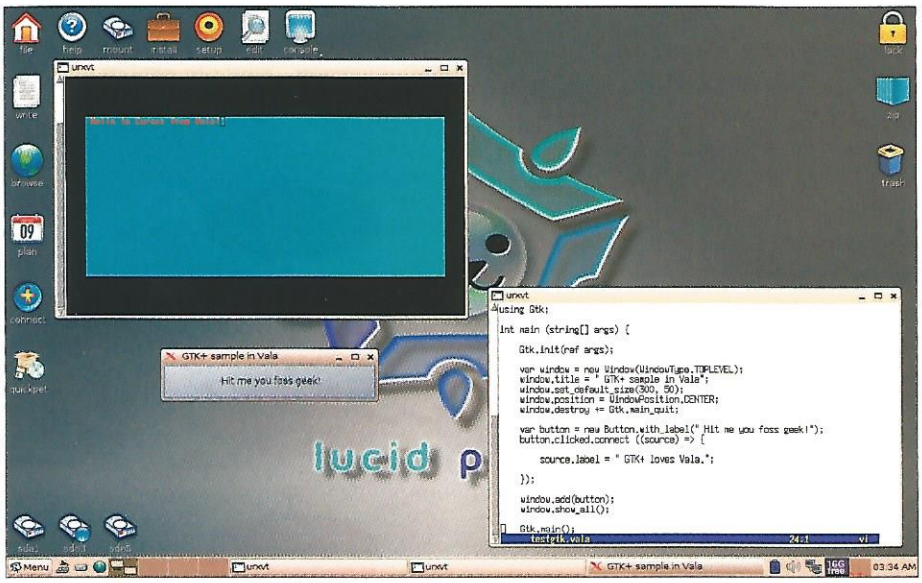

Figure 2: Curses and Gtk applications created with Vala.

If you want to use various collection objects like lists, hash maps, and sets, you have to install libgee. To build and install it, download the latest source tarball from the homepage [5] and issue the command:

```
tar jxvf libgee-version.tar.bz2 ➘
   && cd libgee-version
```

To build and install libgee, type:

```
./configure && make && sudo make install
```

If everything compiles without error, you are ready to add collection classes in your Vala programs. To see Vala collections in action, compile the `collections.vala` code provided in the Article Code archive [6] with

```
valac --pkg=gee-1.0 collections.vala
```

and note the similarity to C++ containers and C# or Java collections.

To debug Vala code snippets, use GDB. Compile your Vala code with additional `-g --save-temps` switches, then launch the executable with GDB to debug. To help with debugging, the generated intermediate C files (from the `--save-temps` switch) contain a line-by-line mapping of Vala and C code.

The Vala language supports powerful object-oriented programming around various Gnome-based objects. The various classes derived from GLib's `Object` class can take full advantage of GObject. All the objects are instantiated in Vala through the `new` keyword – as in C# and Java – and Vala automatically cleans the various objects by reference counting. Vala doesn't provide multiple inheritance.

Properties in a class defined with the `get` and `set` methods can work as accessors and mutators while keeping the class data private:

```
private int _iVal;
public int iVal {
  get { return _iVal; }
  set { _iVal = value; dVal *= 2; }
}
```

`value` in the `set` mutator is the value assigned to public property `iVal`.

The type of object at run time becomes known, with the `name` method of the `Type` reference returned by the `get_type` method inherited in the object:

```
public static void main() {
  var voop1 = new VOop1(3.14);
  Type type = voop1.get_type();
  stdout.printf(➘
   "\n voop1.get_type() : %s\n", ➘
   type.name());
}
```

Vala doesn't provide method overloading, so you can't declare and define multiple functions and constructors with the same name and different signatures, but *named constructors* provides overloaded constructors, which lets you give different name additions to overload various constructors. Also, you can chain con-

LISTING 3: testcurses.vala

```
01 using Curses;
02
03 int main(string[] args) {
04
05     initscr();
06
07     start_color();
08     init_pair(1, Color.RED,
                 Color.CYAN);
09
10     var win = new Window(LINES - 8,
         COLS - 8, 4, 4);
11     win.bkgdset(COLOR_PAIR(1) |
                 Attribute.BOLD);
12     win.addstr(" Hello to Curses
                 from Vala!");
13     win.clrtobot();
14     win.getch();
15
16     endwin();
17     return 0;
18
19 }
```

LISTING 4: testoop1.gs

```
01 [indent = 4]
02
03 class GOop1 : GOop
04
05   prop iV : int
06   prop dV : double
07
08   init
09     this.iV = 8
10     this.dV = 7.890
11
12   construct(iVal : int, dVal :
                 double)
13     iV = iVal
14     dV = dVal
15
16
17   def getState1()
18     print "\n iV : %d, dV : %f\n",
           iV, dV
19
20   def setState1(iVal : int, dVal :
                 double)
21       setState(iVal, dVal)
```

structors to one another. Vala also provides destructors for the very rare cases when you manage memory manually with pointers. Vala does a *nullability check* on function parameters and return values. This feature is like static checking to prevent run-time errors, like dereferencing a null reference. If you want any function parameter or return value to be null, use `modifier ?` with the parameter or the return value. When you work with multiple Vala source files to create executables, if you supply just the file names to valac, it figures out a way to connect different pieces of code to generate the executable.

The curses example in Listing 3 requires the ncurses headers. To see the system bindings available by default in Vala, compile with the command

```
valac --pkg curses -X ➥
   -lncurses testcurses.vala
```

then run the executable (Figure 2).

A basic Gtk programming sample in Vala (Figure 2, lower right) can give you some feeling for Gnome programming.

Genie Time

Now it is time to rub the valac magic lamp and call Genie. If you prefer Python and want to finish big programs with little effort, you will love Genie, which provides the ease of Python syntax and the performance of GLib- and GObject-based C code.

Although many Vala features are equally applicable to Genie, Genie does a few things differently [7]. For instance,

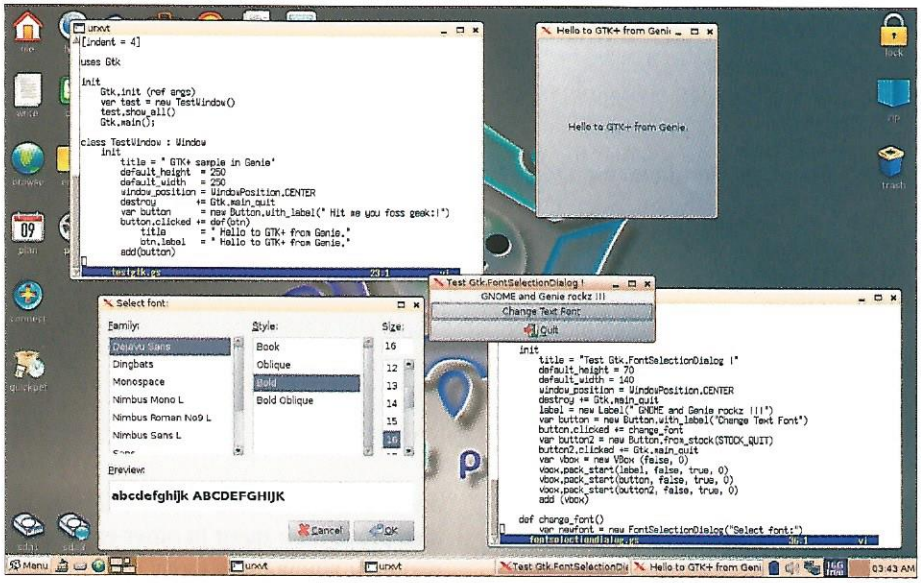

Figure 3: Gtk applications created with Genie.

Genie doesn't use curly braces to create code blocks; by default, it uses Tabs. Tab behavior can be changed with indentation (as in Python) by putting `[indent=number of spaces]` at the top of Genie source files.

In Genie, execution starts in the `init` block, not in `main`. To see the Python-like compactness of Genie code compared with Vala code, compile and run the following code:

```
[indent=4]

init
    print "Hello to FLOSS from Genie!!!"
```

Genie syntax for variable and function declarations is slightly different from Vala, with the variable format `variable: type` – if you don't use a type inference – and the format for functions is `function (parameter:type, ...) : return type`. Genie classes can have only one `init` block as a constructor. The `init` block outside of a class works as `main`. If you precede class members with an underscore or the `private` keyword modifier, they become private; otherwise, class members are public by default.

A `construct` block sets various properties of a class by supplying parameters at construction time. To try the object-oriented concepts provided by Genie, compile the programs shown in Listings 4 and 5 with

```
valac -o testoop testoop1.gs ➥
   testoop2.gs
```

and run the `testoop` executable.

Listings 4 and 5 also demonstrate some concepts introduced earlier with Vala, such as singular superclass inheritance, properties, and code in multiple files. To see the Genie version of a basic Gtk+ application, compile `testgtk.gs`

```
valac --pkg gtk+-2.0 testgtk.gs
```

LISTING 5: testoop2.gs

```
01 [indent = 4]
02
03 class GOop : Object
04
05   prop _iV : int
06   prop _dV : double
07
08   init
09     _iV = 7
10     _dV = 3.14
11
12   def getState()
13     print "\n_iV : %d, _dV : %f\n",
             _iV, _dV
14
15   def setState(iVal : int, dVal :
                    double)
16     _iV = iVal
17     _dV = dVal
18
19 init
20   var goop = new GOop()
21
22   goop.getState()
23
24   goop.setState(89, 56.789)
25   goop.getState()
26
27   var goop1 = new GOop1(
       56, 89.45657)
28   goop1.getState1()
```

LISTING 6: classgenie.gs

```
01 [indent=4]
02
03 class Ggenie : Object
04
05   def prntInfoGenie(
       info : string)
06     print "\n Ggenie :: %s\n", info
```

LISTING 7: classvala.vala

```
01 class Vvala : Ggenie
02 {
03
04   void prntInfoVala(string info) {
05
06     stdout.printf("\n Vvala ::
         %s\n", info);
07
08   }
09
10   public static void main() {
11
12     var ovala = new Vvala();
13
14     ovala.prntInfoVala("Calling
         method of vala class from vala
         class.");
15
16     ovala.prntInfoGenie("Calling
         method of genie class from
         vala class.");
17
18   }
19
20 }
```

and run the `testgtk` executable (Figure 3, upper left).

The next Gtk+ example, taken from the Genie examples site [8], explores the font selection dialog functionality of Gtk+ in Genie (Figure 3, lower right).

Mixing Vala and Genie

Because both Vala and Genie are converted into the same intermediate C code by valac, you can seamlessly mix them in one application. Therefore, you can write a function in Genie and use it in Vala, or you can write a class in Vala and use its properties and methods in Genie. A small example to derive a class defined in Vala from a superclass defined in Genie and invoke inherited and class methods is shown in Listings 6 and 7. To compile, enter

```
valac -o valagenie classvala.vala ➥
                    classgenie.gs
```

and run the `valagenie` executable.

Missing Bytes

Although I have focused primarily on the functionalities of the GLib, GObject, and Gnome libraries, Vala and Genie also prove their merit in other environments. If you want to target only the C run-time library with Vala and Genie, use the valac `--profile=posix` option to create the executables. Vala and Genie can replace direct C programming in embedded systems or in systems that require very small executables. If you generate intermediate C code or executables for some Vala and Genie programs without using Gnome resources, you will see that valac generates very efficient C code. With the use of alternate light versions of libc and static linking, you can create very tiny and self-contained native executables in Vala and Genie.

Performance

The performance of Vala and Genie programs is equal or similar to equivalent C programs. Although Vala syntax is similar to C# and Java, it doesn't come with the run-time penalties associated with Mono/CLR or JVM. Similarly, Genie has marvelous syntax inspired from Python, but it does not suffer the performance overhead of CPython or Mono/CLR. An independent Google Code project implements *The Computer Language Benchmarks Game* in Vala [9] and compares performance against C, C++, and C#. Table 1 shows the results published on the benchmarks project page using

```
Gcc-4.3.4, Mono C# compiler 2.4.2.3
```

and

```
valac-0.7.6.
```

Clearly, Vala provides a performance level equivalent to C in all the benchmarks [9].

TABLE 1: Performance Comparison of Vala

Benchmark	C++	C#	C	Vala
mandelbrot	14.48	47.40	12.38	13.05
partialSums	32.68	56.58	35.11	34.92
recursive	12.87	28.15	8.38	8.61
binaryTrees	27.53	42.62	21.56	30.75
sumFile	16.90	23.70	14.18	15.24
fannkuch	11.18	26.59	12.26	14.23
spectralNorm	32.77	46.58	32.83	36.84
nsieve	25.49	29.31	26.09	25.70
nBody	28.33	43.55	26.02	28.06

Conclusion

Vala and Genie are modern programming languages that promise high productivity, flexibility, and native performance without a large number of run-time libraries or run-time engines like virtual machines. They were primarily designed around the Gnome libraries and provide Gnome developers excellent programming alternatives. However, Vala and Genie are suitable for non-Gnome areas targeting libc as well. ■■■

INFO

[1] Vala homepage: *http://live.gnome.org/Vala*

[2] Genie homepage: *http://live.gnome.org/Genie*

[3] GObject Wikipedia page: *http://en.wikipedia.org/wiki/GObject*

[4] Valadoc homepage: *http://live.gnome.org/Valadoc*

[5] libgee homepage: *http://live.gnome.org/Libgee*

[6] Code used in this article: *http://www.linux-magazine.com/Resources/Article-Code*

[7] Genie tutorial from the founder of Puppy Linux: *http://bkhome.org/genie/index.html*

[8] Vala and Genie examples: *http://code.valaide.org*

[9] Vala benchmarks: *http://code.google.com/p/vala-benchmarks*

AUTHOR

Ankur Kumar Sharma is a software developer and researcher. He likes to play and croon classic rock songs on his guitar, read self-help books, write, and explore many interests in his spare time. He blogs at *www.richnusgeeks.com*.