entwickler.press
shortcuts

Sprachen-kompendium

Vala, Go und Rust

Tam Hanna

Tam Hanna

Sprachenkompendium

Vala, Go und Rust

ISBN: 978-3-86802-548-4

Ein Imprint der Software & Support Media GmbH

1 Mit C objektorientiert programmieren? Vala macht’s möglich!

Bjarne Stroustrups C++ gilt heute als Goldstandard in Sachen OOP: Die Spracherweiterungen für C fanden ob ihrer hervorragenden Abwärtskompatibilität bald jede Menge Anhänger. Dank Apples tatkräftiger Unterstützung lebt sein Vorgänger Objective-C ebenfalls munter weiter – unter anderem in der seit 2006 entwickelten Sprache Vala.

Vergleichsweise wenige Entwickler wissen, dass man auch mit C objektorientiert programmieren kann. Die dazu notwendige Trickserei mit Structs und Funktions-Pointern ist das täglich Brot all jener Entwickler, die Applikationen auf Basis von GObject entwickeln. GObject ist eine Bibliothek, die anfangs Teil des in C gehaltenen GTK-Frameworks war. Nach dem Erscheinen der zweiten Version beschlossen die Entwickler, dass ihr Typ- und Klassensystem auch ohne die diversen UI-Widgets lebensfähig ist: Die Standalone-Version von GObject war geboren.

Unaufmerksame Beobachter neigen dazu, die Bibliothek als Spielerei für eine Randgruppe von konservativ-reaktionären Freaks abzutun. Das ist grundfalsch: Im Laufe der Jahre entstanden einige beeindruckende Programme. Neben dem auch in Ubuntu verwendeten Gnome-Desktop sei an dieser Stelle auch die Bildbearbeitung GIMP erwähnt.

Aufgrund der für Quereinsteiger verwirrenden Bedienung entschieden sich Jürg Billeter und Raffaele Sandrini im Jahr 2006 dazu, mit einer neuen Sprache Abhilfe zu schaffen. Die nach einer skandinavischen Prophetin Völva benannte Programmiersprache Vala sollte die Nutzung der GObject-

Bibliothek ermöglichen und eine an C# angelehnte Syntax aufweisen.

Wer mit Qt gearbeitet hat, kennt das in **Abbildung 1.1** gezeigte Diagramm mit Sicherheit. Ein als moc bezeichnetes Werkzeug wandelt den Qt-Code vor der eigentlichen Kompilation in normales C++ um, das daraufhin zum nativen Compiler der Plattform wandert. Das Resultat ist eine Anwendung, die – von außen betrachtet – nicht von einem normalen C++-Programm zu unterscheiden ist.

Vala nutzt einen ähnlichen Ansatz. Der als *valac* bezeichnete Compiler nimmt *.vala*-Dateien entgegen, die daraufhin in normale *.c*-Dateien umgewandelt werden. Daraus folgt, dass die Performance von in Vala gehaltenem Code mit dem einer nativen C-Applikation vergleichbar ausfällt.

Abbildung 1.1: Schematische Darstellung der Umwandlung von Qt-Code in normales C++ vor der eigentlichen Kompilation

Vala herbei

Aufgrund der vergleichsweise geringen Verbreitung der Programmiersprache lässt sich das ohnehin nicht für schnelle Updates bekannte Ubuntu-Team beim Aktualisieren des per *apt-get* erhältlichen Vala-Compilers besonders viel Zeit.

Zum Zeitpunkt der Drucklegung ist v0.26 aktuell – im Repository wartet die veraltete v0.22. Die in den folgenden Schritten beschriebene Kompilationsmethode führt unter einer 64-Bit-Version von Ubuntu 14.04 zu einer lauffähigen Version des *valac*-Tools. Die unter anderen Distributionen notwendige Vorgehensweise unterscheidet sich mitunter minimal.

Extrahieren Sie den unter *https://wiki.gnome.org/Projects/-Vala/Release* zum Download bereitstehenden Quellcode-Tarball

an eine bequem zugängliche Stelle. Führen Sie danach das *configure*-Programm in einer Kommandozeile aus – es wird Fehlermeldungen nach dem folgenden Schema emittieren:

```
tamhan@TAMHAN14:~/vala-0.26.0$ ./configure
checking for bison... no
configure: error: bison not found but required
```

Diese lassen sich – normalerweise – durch Nachinstallieren der passenden Bibliothek beheben:

```
sudo apt-get install bison
```

Im nächsten Schritt folgen die von der Kompilation klassischer Unix-Pakete bekannten Befehle *make* und *make install*. Besitzer von stark parallelisierenden Computern sollten *make* durch den *-jX*-Kommandozeilenschalter zur Nutzung mehrerer Rechenkerne animieren. Auf einer achtkernigen Maschine mit einem AMD FX 8320 ist die Kompilation binnen rund 40 Sekunden abgeschlossen.

Falls der *valac*-Compiler auf Aufrufe mit dem in Listing 1.1 gezeigten Fehler reagiert, müssen Sie das Betriebssystem durch Aufruf von *ldconfig* über die neuen Bibliotheken informieren. Nach dem Abarbeiten des am Ende im Folgenden gezeigten Befehls sollte das Werkzeug ohne Probleme funktionieren:

```
tamhan@TAMHAN14:~/vala-0.26.0$ valac
valac: error while loading shared libraries: libvala-0.26.so.0: cannot open shared
object file: No such file or directory

tamhan@TAMHAN14:~/vala-0.26.0$ sudo ldconfig /usr/local/lib
```

Unter Windows arbeitende Entwickler müssen vor der Kompilation MinGW installieren. Die vom Vala-Projekt angebotenen fertigen Pakete sind auf Stand der Version v0.11 – viele hier gezeigte Beispiele funktionieren mit ihnen nicht.

Hello Vala

Nach der vergleichsweise arbeitsaufwändigen Installation ist es an der Zeit, ein erstes realistisches Programmbeispiel auf den Weg zu bringen. Ernsthafte Nutzer sollten ihre Programme in

MonoDevelop erstellen, um von IntelliSense zu profitieren. Wer die hier vorgestellten Beispiele nur ausprobieren möchte, kann dies stattdessen mit gedit und der Kommandozeile tun. *HalloSUS.vala* sieht wie in Listing 1.1 aus.

```
class SUS.HelloWorld : GLib.Object
{
  public static int main(string[] args)
  {
    stdout.printf("Hallo Welt!\n");
    return 0;
  }
}
```

Listing 1.1

Die meisten in Vala enthaltenen Klassen sind direkt oder indirekt von *Glib.Object* abgeleitet: Es handelt sich dabei um ein Mutterobjekt, das – ähnlich wie das in Qt enthaltene *QObject* – die für viele Features notwendige Logik mitbringt. Unsere im Namespace *SUS* sitzende Klasse *HelloWorld* enthält eine statische *main*-Methode, die im Rahmen der Programmausführung aufgerufen wird. Vala zeigt sich bei der Positionierung der Einsprungfunktion flexibel: Sie kann, muss aber nicht in einer Klasse liegen.

Über den Rest des trivialen Codes müssen wir uns an dieser Stelle nicht weiter unterhalten. Für mit der Kommandozeile unerfahrene Entwickler hier noch ein Snippet, das die Kompilation samt nachfolgender Ausführung zeigt:

```
tamhan@TAMHAN14:~/SUSVala$ valac HelloSUS.vala
tamhan@TAMHAN14:~/SUSVala$ ./HelloSUS
Hallo Welt!
```

Falls Sie den Intermediärcode „behalten“ möchten, so führen Sie *valac* mit dem Parameter *--save-temps* aus. Die resultierenden Codedateien sind nicht „selbsterklärend“, mit den Konzepten von GObject unerfahrene Entwickler kommen beim spontanen Durchschauen nicht sonderlich weit.

Eine Frage der Klasse