TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Oliver Geisel

Die Programmiersprache Vala

Hauptseminar
28.4.2023

Inhalt

1. Allgemeine Informationen
2. Der Compiler „Valac“
3. Speicherverwaltung
4. Features
5. Benchmark
6. Zusammenfassung

Allgemeine Informationen

— Teil des GNOME Projects

— Sprache zum einfachen Umgang mit GTK

— Objektorientiert

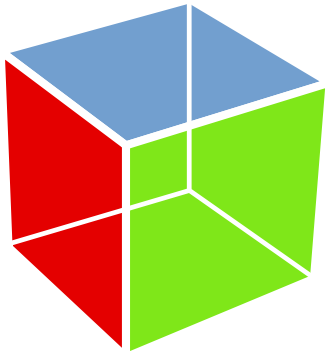
— Aktuelle Version 0.56

— Nutzt Glib und Gobject

— Syntax ähnlich zu Java, C#

— Start der Entwicklung 2006 von Jürg Billeter

Inoffizielles Vala Logo [B1]



GTK Logo

Projekte, die in Vala geschrieben sind

— elementary OS-Apps (77 Projekte)

—GNOME Projekte - 40 Vala; 194 C; 11 C++; 32 Python
  – Taschenrechner
  – Spiele

— Shotwell - Fotoverwaltung

— ev3dev (brickman)

— Gee – Bibliothek für Collections

Compiler - Valac

- In Vala geschrieben
- Benötigt C-Compiler
- Wandelt Vala-Code in C-Code mit Glib Code

```
void main(){
    print("Hello World! This is a Vala-Program!\n");
}
```

„valac –C 01_first.vala“

```
/* 01_first.c generated by valac 0.56.0, the Vala compiler
 * generated from 01_first.vala, do not modify */

#include <glib.h>

static void _vala_main (void);

static void
_vala_main (void){
    g_print ("Hello World! This is a Vala-Program!\n");
}

int
main (int argc, char ** argv){
    _vala_main ();
    return 0;
}
```

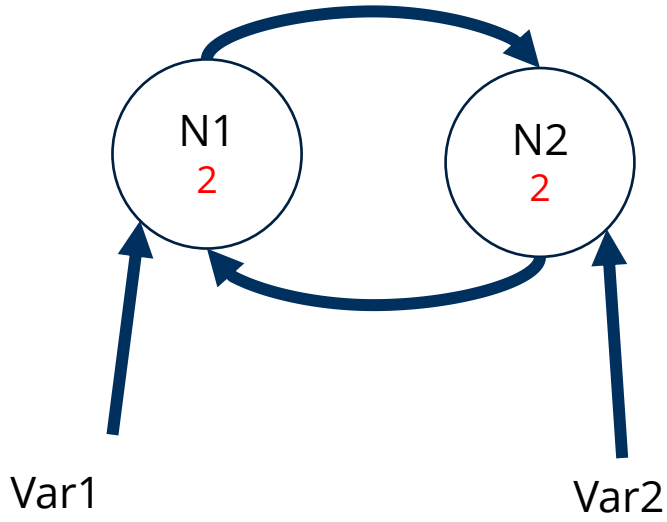
Speicherverwaltung

— Referenzzählend basiert

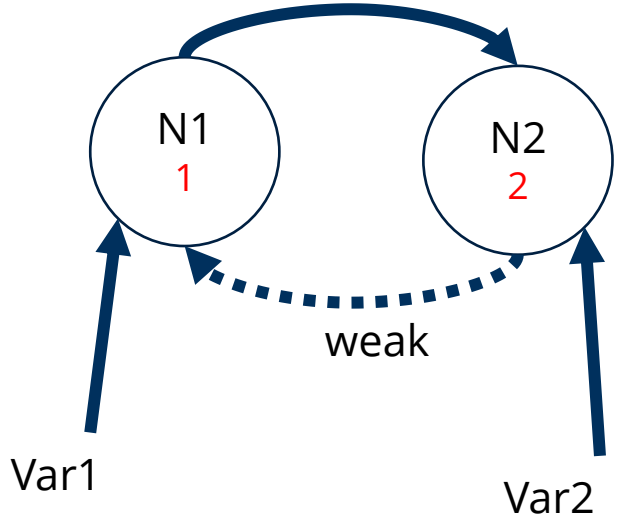
— Wird von GObject übernommen

— Kein Garbage Collector

— Zyklen können auftreten



- Auflösen von Zyklen durch Keyword **weak**
- **weak** erhöht den Zähler nicht



— Pointer und manuelle Speicherverwaltung können weiterhin genutzt werden

Features

— Contract Programming

— Signale

— Null Sicherheit

— Exception Handling

Feature – Contract Programming

— Funktionen können Vor- und Nachbedingungen haben

— Vorbedingung mir **requires (** condition **) -> Critical Log** bei Verletzung

— Nachbedingung mit **ensures (** condition **) -> Warning Log** bei Verletzung

— Wird in Logs festgehalten

— Kann zu Abbrüchen führen wenn es eingeschalten wird

```
int methodWithPreCondition(int a, int b) requires(a > 1) requires(b < 10){
    return a * b;
}
```

** (process:825): CRITICAL **: 18:23:10.224: methodWithPreCondition: assertion 'a > 1' failed

```
int methodWithPostCondition(int a, int b) ensures(result > 10){
    return a + b;
}
```

** (process:825): WARNING **: 18:23:10.225: (path/to/MethodContracts.vala.c:58): methodWithPostCondition: runtime check failed: (result > 10)

Feature – Signale

```
public class Event {
    public signal void mein_event (int i, string s);
}
```

— Mit „Observer“-Pattern vergleichbar

— Werden innerhalb von Klassen definiert

— Wird im GTK viel genutzt

```
public class EventHandler {
    public void mein_handler(Event e, int i, string s){
        print("I war: %d und S war: %s\n", i,s);
    }
}
```

```
void main(){
    Event e = new Event();
    EventHandler handler = new EventHandler();

    e.mein_event.connect(handler.mein_handler);
    print("Eventhandler wurde dem Event zugewiesen!\n");

    e.mein_event(5,"Hallo");
}
```

Feature – Null Sicherheit

- Null kann als Wert verboten werden
- Mit dem Flag **--enable-experimental-non-null** aktiviert
- Explizite Angabe von null möglich -> **?** nach Datentyp
- **A ?? B** gibt A zurück, wenn A nicht null ist sonst B

```
void main(){
    string s = "Hallo";
    string? null_string = null;
    string normal_string;
    int? j = null;
    int[]? b = null;

    // mit --enable-experimetal-non-null nicht möglich
    int[] a = null; // Fehler
    normal_string = null; // Fehler

    // Nullable Typen sind aber erlaubt
    int? nullable_int = null;
    int new_i = j ?? 1;
    print("%d\n", new_i);
}
```

Feature – Exception Handling

```
errordomain MathErrors{
    PI_IS_3,
    NOT_A_NUMBER
}
```

— Fehler müssen in einer Domäne definiert sein

— Nutzt **try-**, **catch-** und **finally-**Blöcke

— Werfen einer Exception mit **throw**

```
void testMethod() throws MathErrors.PI_IS_3{
    throw new MathErrors.PI_IS_3("Pi is now 3.0\n");
    stdout.printf("This section will not be
printed");
}
```

Weitere Features

— Parameter Directions

— Interfaces

— Generics

— Delegates/Lambdas

— foreach (Iteration)

— Type Inference

— Asynchrone Methoden

Benchmark

— Vergleich von Exception Handling in Vala, C und C++

— Vala mit try, catch

— C++ mit try, catch

— C mit if-Statement

—System:
- Intel Core i7 6700k @ 4GHz
- Ubuntu 22.04
- 32 GB RAM
- -O0

Benchmark

— 10.000.000 Wiederholungen; Warmup 10.000

— Zeiten werden im Array gespeichert

— Mittelwert, Varianz und Standardabweichung gebildet

— Zwei Varianten

5 Methoden Aufrufe (A->B->C->D->E)

Jede Ebene wirft eine Exception und fängt vorherige

5 Methoden Aufrufe (A->B->C->D->E)
E wirft Exception
B,C,D prüfen auf andere Exception
A fängt Exception von E

Ergebnisse

Alle werfen und fangen

Sprache	Durchschnitt	Varianz	Standardabweichung	Gesamt Zeit
Vala	0.887 µs	0.0 µs	0.651 µs	9.167 ms
C	0.028 µs	0.0 µs	0.135 µs	634 ms
C++	0.027 µs	8.9E-7 µs	0.140 µs	547 ms

Nur E wirft und A fängt

Sprache	Durchschnitt	Varianz	Standardabweichung	Gesamt Zeit
Vala	0.516 µs	0.0 µs	0.546 µs	5.376 ms
C	0.028 µs	0.0 µs	0.170 µs	634 ms
C++	2.416 µs	8.9E-10 µs	0.934 µs	24.394 ms

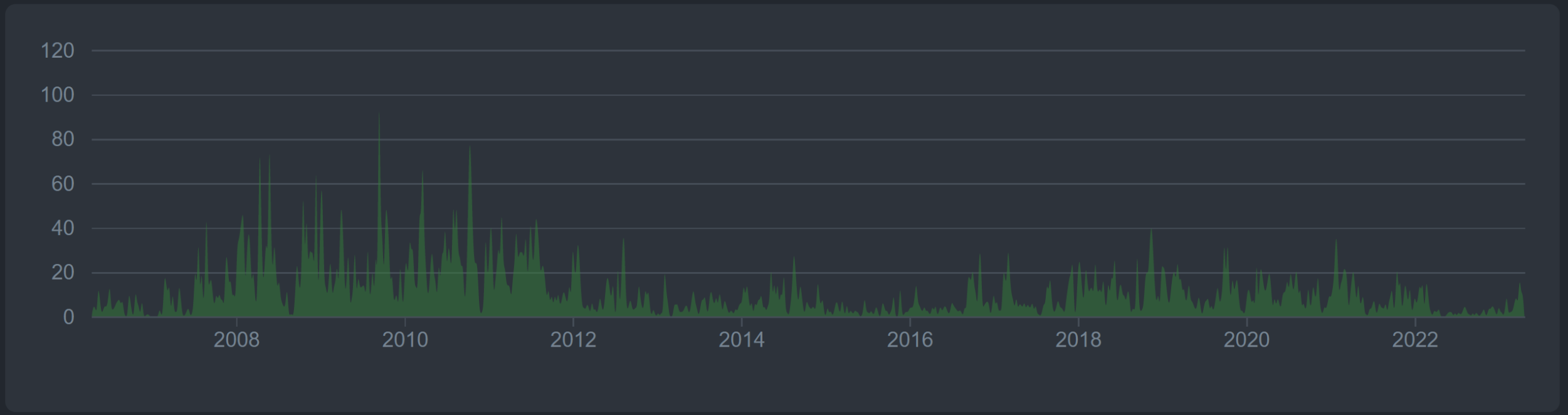
Herausforderungen bei Vala

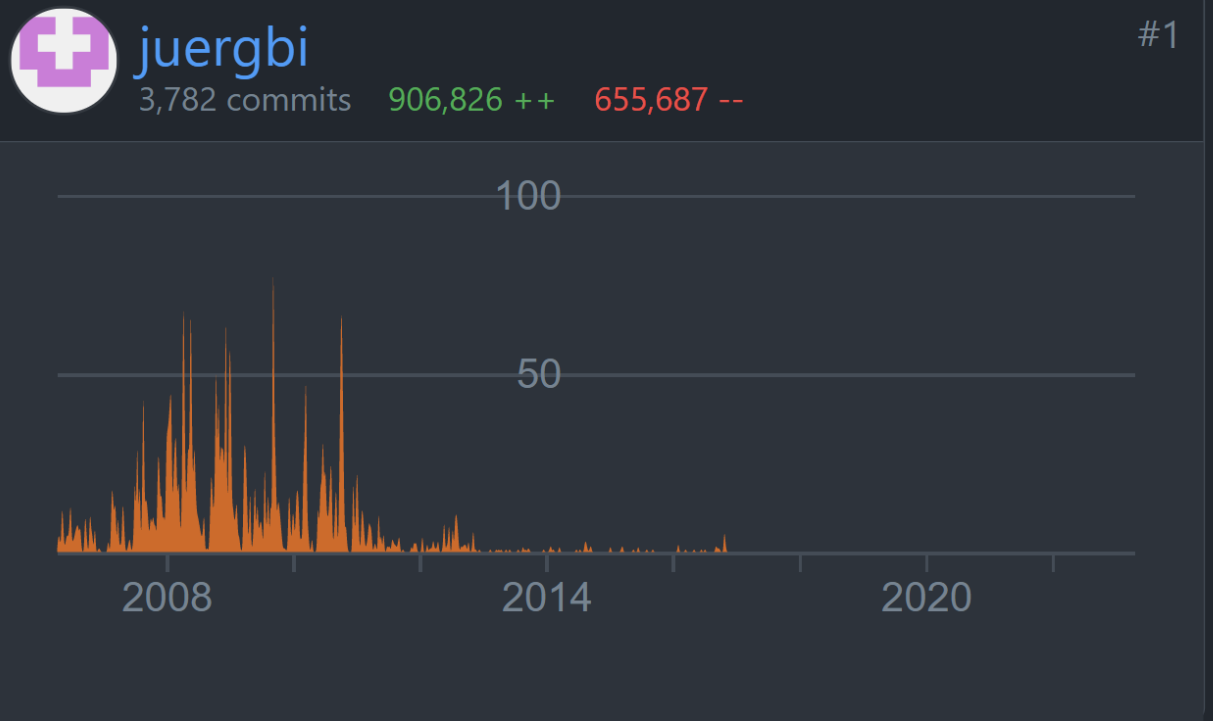
— Sprache wird nur von einer Person aktiv entwickelt

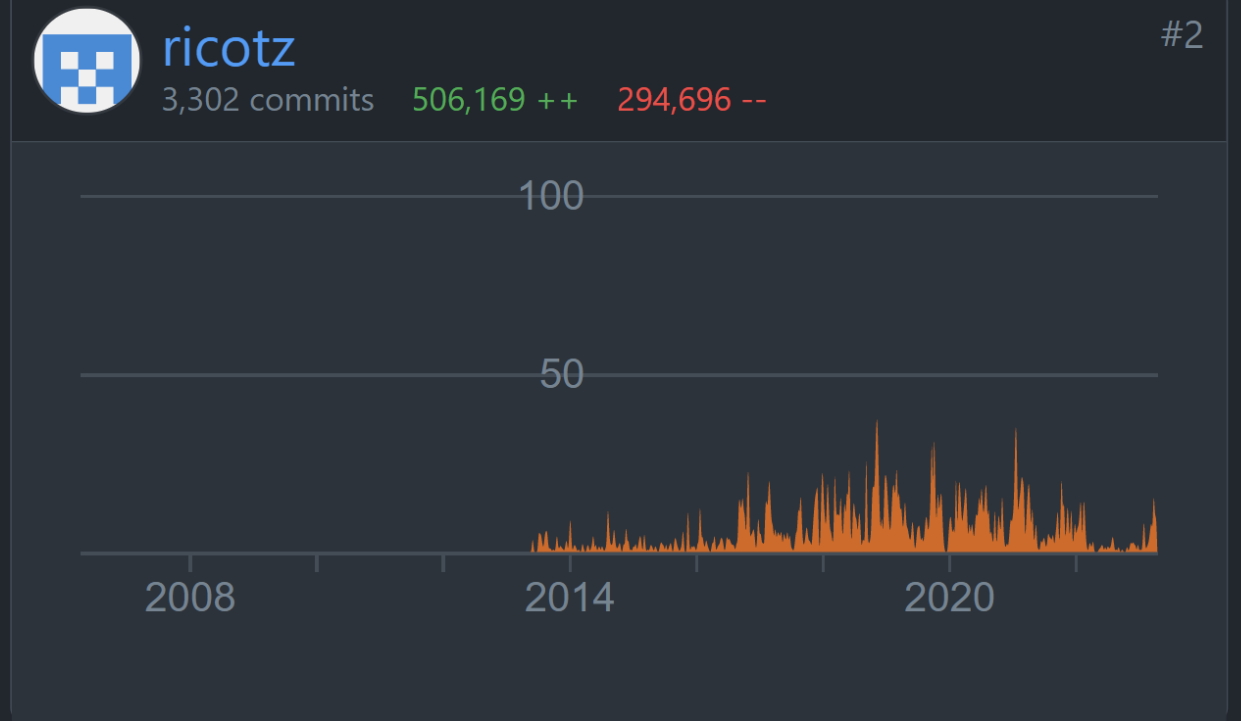
— Unvollständige Dokumentation

— TIOBE-Index nicht unter Top 50

— Eine „Tote Sprache“ – Emanuelle Bassi Gnome Entwickler - 2017


120
100
80
60
40
20
0
2008
2010
2012
2014
2016
2018
2020
2022


juergbi
3,782 commits
906,826 ++
655,687 --
#1
100
50
2008
2014
2020


ricotz
3,302 commits
506,169 ++
294,696 --
#2
100
50
2008
2014
2020

Zusammenfassung

— Programmiersprache für speziellen Einsatz -> GNOME, GTK und elementary OS

— Moderne Features, die Sicherheit bringen und Entwicklung vereinfachen

— Messbare Performanceeinbußen im vergleich zu C

— Wird langsam weiterentwickelt

Fragen?

Quellen

[B1] https://github.com/elementary/brand/blob/master/vala/vala-social.svg

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Weitere Folien

Feature – Lambda Funktionen

— Delegates bilden Grundlage

— Delegates: Datentypen für Methoden

— Syntax: **(** *Parameter* **) => {** *Statements* **}**

```
static delegate int MethodWithTwoInt(int a, int b);

void printResult(MethodWithTwoInt func, int a , int b ){
        stdout.printf("%d\n", func(a,b) );
}


int main(string[] args){
    printResult((first, second) => {return first * second + 1;} , 5 ,10); // 5*10+1 =51
    return 0;
}
```

Beispiel für GTK und Signale

```
int main ( string[] argv ) {
    // Create a new application
    var app = new Gtk.Application ( "de.oliver.ValaGTK_Example",
                                    GLib.ApplicationFlags.FLAGS_NONE );
    app.activate.connect (( ) => {
        var window = new Gtk.ApplicationWindow ( app );
        var text = new Gtk.Label("Willkommen bei Vala mit GTK!");
        var button = new Gtk.Button.with_label ( "Hallo!" );
        var box =  new  Gtk.Box(Orientation.VERTICAL, 2);

        button.clicked.connect (( ) => {
            button.set_label("Auf wiedersehen");
            Thread.usleep(1000000);
            window.close ();
        } );

        box.append(text);
        box.append(button);
        window.set_child ( box );
        window.present ();
```