Konzeptionelle Einführung in Erlang

24C3

Ben Fuhrmannek <ben@fuhrmannek.de>
Stefan Strigler <steve@zeank.in-berlin.de>

Ziel des Vortrags ist es, einen kleinen Einblick in Erlang/OTP zu gewähren, allerdings weniger in der Form "Wie programmiere ich was mit Erlang?" als eher eine Antwort auf Fragen zu liefern wie "Was macht Erlang besonders, was kann es was andere Sprachen nicht oder nicht so gut können?". Es soll mehr um den Einsatz von Erlang in der Praxis gehen, als eine Einführung in das Arbeiten mit Erlang zu geben (sorry, kein 'Hello World' today).

HISTORIE

Erlang was created by the Computer Science Laboratory at Ellemtel (now Ericsson AB) around 1990. It originates from an attempt to find the most suitable programming language for telecom applications. Characteristics for such an application include:

- *Concurrency - Several thousand events, such as phone calls, happening simultaneously.*
- *Robustness - An error in one part of the application must be caught and handled so that it does not interrupt other parts of the applications. Preferably, there should be no errors at all.*
- *Distribution - The system must be distributed over several computers, either due to the inherent nature of the application, or for robustness or efficiency.*

(Quelle: http://www.ericsson.com/technology/opensource/erlang/)

Open Source ist Erlang seit 1998. Die Sprache wurde nach dem dänischen Mathematiker Agner Krarup Erlang benannt, wobei die Doppeldeutigkeit mit Ericson-Language (ErLang) gewollt ist.

PROZESSORIENTIERTE PROGRAMMIERUNG

Joe Armstrong: "The world is parallel."

In Erlang besteht die Welt aus Prozessen, die mit einander Nachrichten austauschen. Dieses Konzept ist für uns sehr leicht zu verstehen, denn wir agieren auf ähnliche Weise: Eine Ampel signalisiert grün, dann fahren wir los. Oder wir fragen die Auskunft nach einer Telefonnummer und sie wird uns genannt. Jede Person und jedes Objekt, das irgendwie interagieren möchte, wird so einfach als Prozess abgebildet. Eine kleine Erweiterung zur Realität stellt die Tatsache dar, dass Prozesse, die sich erwartet oder unerwartet beenden, noch die Ursache preisgeben; z.B. eine Ampel fällt aus, dann sagt sie als Letztes noch 'Glühbirne durchgebrannt'. Falls ein anderer Prozess sich dafür interessiert, dann kann die Ampel passend repariert werden.

In der objektorientierten Entwicklung werden Daten als Objekte und Abläufe als Use-Cases mit Methodenaufrufen von Objekten modelliert. In aktuellen Diskussionen wird das leider allzu oft als Gegensatz aufgegriffen, was wohl daher rührt, dass klassische objekt-orientierte Sprachen Parallelisierung nur mittels Threads unterstützen. Erlang dagegen aber keine Klassen und Objekte kennt. Im Prinzip widersprechen sich die Ansätze aber nicht. So lassen sich Prozesse auch als Objekte begreifen. In Python werden Methodenaufrufe sowieso Nachrichten genannt und sind ohnehin von jeher konzeptionell dasselbe.

Threads teilen Speicher miteinander, dessen Zugriff zum Schutz vor Inkonsistenzen mit Locks abgesichert wird. Sollte während eines bestehenden Locks ein Fehler auftreten, muss explizit sichergestellt werden, dass das Lock wieder freigegeben wird, ansonsten wäre der Programmablauf beim nächsten Zugriff auf das Lock gestoppt.

Erlang dagegen kennt keinen Shared-Memory und keinen globalen Variablen, sondern Prozesse kommunizieren über Nachrichten.

SPRACHLICHE BESONDERHEITEN

- Erlang ist eine sequentiell[1] funktionale[2] Programmiersprache.
- Variablen können nur einmal assoziiert werden, z.B.

```
X = 1.
X = 2 (ERROR)
```

[1] sequentiell: a, b, c

[2] funktional: f(e(d()))

und müssen vorher nicht deklariert werden. Es gibt keine globalen Variablen und keinen von mehreren Prozessen gemeinsam genutzten Speicher.

- Die nahezu platformunabhängige Laufzeitumgebung (footnote: läuft unter Linux, ...) interpretiert Byte-Code.
- Anstatt Threads gibt es Prozesse, die von der Laufzeitumgebung verwaltet werden und daher sehr leichtgewichtig (footnote: sowohl RAM als auch Startdauer) sind.
- Inter-Process-Communication (IPC) ist sehr einfach durch asynchrone Nachrichten abbildbar, z.B.

```
Pid ! nachricht.
```

- Dabei stellt Pid eine Prozess-ID dar, die in einem verteilten System auch auf einen anderen Erlang-Node verweisen kann.
- Erlang unterstützt Hot-Code-Replacement.

ERLANG OTP (OPEN TELECOM PLATFORM)

Äquivalent zu den Standardbibliotheken in anderen Programmiersprachen bietet Erlang die Open Telecom Platform:

- große Bibliotheksklassen für den Programmiereralltag
- integrierte Anwendungen wie Mnesia (Verteiltes Datenbanksystem)
- vordefinierte Archtitekturmuster wie gen_server für Client-Server Architekturen oder gen_fsm für endliche Automaten
- Debugging- und Deployment-Tools

WAS KANN ERLANG FÜR DICH TUN?

Erlang zeigt sein volles Potential, wenn ein oder mehrere der folgenden Kriterien besonders wichtig sind:

Parallelisierung

z.B. typisch für Client-Server-Architektur und um Multi-Core-Systeme auslasten

Es folgt ein vergleichendes Beispiel mit vielen Prozessen/Threads mit Erlang, dann Python:

```
-module(processes).
-export([max/1]).

max(N) ->
     Max = erlang:system_info(process_limit),
     io:format("Max. processes: ~p~n", [Max]),
     statistics(runtime), statistics(wall_clock),
     L = for(1, N, fun() -> spawn(fun wait/0) end),
     {_, Time1} = statistics(runtime),
     {_, Time2} = statistics(wall_clock),
     lists:foreach(fun(Pid) -> Pid ! die end, L),
     U1 = Time1 * 1000 / N,
     U2 = Time2 * 1000 / N,
     io:format("time for ~p processes: ~p/~p (runtime/real)~n", [N,
U1, U2]).

wait() ->
     receive
          die -> void
     end.

for(N, N, F) -> [F()];
for(I, N, F) -> [F()|for(I, N-1, F)].

%% Beispiel aus 'Programming Erlang'
```

output:

```
1> processes:max(32000).
Max. processes: 32768
time for 32000 processes: 1.56250/3.71875 (runtime/real)
```

```
import sys,os
from threading import Thread, Lock

gl = Lock()
class TestThread(Thread):
     def run(self):
          gl.acquire()
          gl.release()

t1 = sum(os.times())

N = int(sys.argv[1])
threads = []
gl.acquire()
for i in range(N):
     t = TestThread()
     t.start()
     threads.append(t)

gl.release()
for t in threads:
     t.join()
t2 = sum(os.times())
print "elapsed cpu time: " + str(t2-t1) + "s"
```

Skalierbarkeit durch Verteilheit (Cluster)

Verfügbarkeit durch Fehlertoleranz und Hot-Code-Replacement

99,999% Verfügbarkeit

KILLER-APPLICATIONS

Ejabberd

- High-Performance Jabber/XMPP-Server,
- clusterbar,
- Komponenten für JUD, Groupchat, IRC und PubSub integriert,
- Web-Administration,
- Leicht erweiterbar durch Erlang-Module (ejabberd-modules)
- In-House Benchmarks: Ein Node auf dual Xeon 2.8GHz und 8GB Ram bedient ca. 150.000 c2s Connections.
- MXit Südafrika betreibt Ejabberd-Cluster mit 4.8M registrierten User, 9M logins und 200M pro Tag.

Tsung

- Benchmark-Tool für HTTP und XMPP
- Clusterbar

Yaws

- High-performance Webserver für dynamischen generiertent Content
- embedable

KRITIK

- Useability der Dokumentation nicht auf der Höhe der Zeit - wer mit manpages umgehen kann, kommt aber gut zurecht
- Community noch etwas unorganisiert
- Für Fragen, Hilfe, Support existiert (nur?) eine Mailingliste mit mittlerweile doch sehr hohem Traffic. Dort schreiben aber eben auch Leute aus dem Ericsson Entwicklerteam sowie Joe Armstrong selbst.

GETTING STARTED

- Download und Doku unter [http://www.erlang.org http://www.erlang.org]
- Community-Site: [http://www.trapexit.org Trapexit]

LITERATUR

- Joe Armstrong, Robert Virding, Cleas Wikström, Mike Williams: Concurrent Programming in Erlang, Second Edition, Prentice Hall, 1996
- Joe Armstrong: Programming Erlang - Software for a Concurrent World, The Programatic Programmers, 2007
- http://www.thinkingparallel.com/2007/03/20/ten-questions-with-joe-armstrong-about-parallel-programming-and-erlang/ Ten Questions with Joe Armstrong about Parallel Programming and Erlang
- http://armstrongonsoftware.blogspot.com/2006/08/concurrency-is-easy.html Concurrency is easy
- http://armstrongonsoftware.blogspot.com/2006/09/why-i-dont-like-shared-memory.html Why I don't like shared memory
- http://armstrongonsoftware.blogspot.com/2006/09/pure-and-simple-transaction-memories.html Pure and simple transaction memories
- http://weblogs.mozillazine.org/roadmap/archives/2007/02/threads_suck.html Threads suck
- http://en.wikipedia.org/wiki/Erlang_%28programming_language%29 Wikipedia: Erlang (programming language)
- http://de.wikipedia.org/wiki/Erlang_%28Programmiersprache%29 Wikipedia (de): Erlang (Programmiersprache)
- http://en.wikipedia.org/wiki/Declarative_programming Wikipedia: Declarative programming
- http://en.wikipedia.org/wiki/Functional_programming Wikipedia: Functional programming
- http://lambda-the-ultimate.org/node/2533 Generative Code Specialisation for High-Performance Monte Carlo Simulations