BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Erlang

Eine interessante Programmiersprache

Vortrag vor der Fachgebietsgruppe Software-Entwicklung
am Di 12.01.2010

von Ulrich Grude

Erlang-Programme:

- **nebenläufig**
- **verteilt**
- **funktional**
- **ungetypt** (Wikipedia: Typing discipline: dynamic, strong ?)
- Module **zur Laufzeit austauschbar** (engl. hot swap)
- Schnittstellen zu **Java**, **C** und anderen Sprachen

Erlang Geschichte

- Entwickelt von der Firma **Ericcson**
- Erste Version **1986** (von Joe Armstrong)
- Eine Vorversion war in **Prolog** geschrieben
- Seit **1998** Open Source
- Name:
  - nach Agner Krarup **Erlang**
    (dänischer Mathematiker, 1878-1929)
  - oder **Er**iccson **Lang**uage
  - **nicht** nach der Stadt Erlang (二郎镇) in Zentral-China!

Zugänglichkeit von Erlang (1)

- Open Source
- **Gute Distribution** für Windows (wohl auch für Unix)
- Enthält eine eigene **Erlang-Shell**
- Enthält mehr als 50 **zusätzliche Module**:
  - otp Open Telecom Platform
  - wx für Grabos (engl. GUIs)
  - parsetools zum Parsen
  - eunit zum Testen
  - crypto zum Ver-/Entschlüsseln
  - washdog to wash the family dog
  - **...**

Zugänglichkeit von Erlang (2)

- **Umfangreiche Dokumentation** leicht erreichbar
- Als **HTML** und **man pages**
- Zwei gute Bücher (die sich überlappen und ergänzen)
- Was ich noch nicht gefunden habe (und vermisse):
  - Eine etwas **formalere Spezifikation**
  - Was ist ein Ausdruck?
  - Was ist ein Term?
  - Was heisst "globally unique"?
  - Wie weit ist ein registrierter Prozess sichtbar?
  - ...

Erlang ist intensiv praxiserprobt und -bewährt

- **CouchDB**: Eine dokumentenorientierte DB für JSON-Dokumente (z.B. in Ubuntu)
- **AXD301 ATM Switch**: 99,999% Verfügbarkeit, 1,5 Millionen Zeilen Erlang-Code
- **Mnesia**: Verteiltes DBMS
- Ein **SIP-Stack** ("ISDN über IP") und vieles mehr von Ericcson
- 2007 **Erlang**-Programme sind zuverlässiger, schneller und kürzer als **C++**-Programme [Nyström et.al., Concurrency and Computation, Practice & Experience, 20(8), 2008]
- Server **Yaws**, hat 2002 gegen eine Apache-Version gewonnen
- (Teile von) **Facebook**, **Twitter**, **Yahoo**

Warum gerade jetzt Erlang?

- Weil Prozessoren zur Zeit nicht mehr **schneller**, aber **zahlreicher** werden.
- Erlang: **Bewährtes Werkzeug** für die Realisierung verteilter Anwendungen
- Interessanter Vergleich: **Scala** und **Erlang** (Welches ist die schönste, einfachste, schnellste, beste, sympathischste, coolste, ... Sprache im Land?)

Was ist eine funktionale Programmiersprache? (1)

- Es gibt keine **Zustände**
- Es gibt keine **Variablen** (die man verändern kann)
- Es gibt keine **Zuweisungen**
- Es gibt keine **Seiteneffekte**
- Es gibt keine **Anweisungen** (nur Ausdrücke)

Was ist eine funktionale Programmiersprache? (2)

- Es gibt **Zustände.**
- Es gibt **Variablen** (die man verändern kann).
- Es gibt **Zuweisungen.**
- Es gibt **Seiteneffekte.**
- Es gibt **Anweisungen** .

  Aber man verwendet diese **prozeduralen Mittel** nur sehr **zurückhaltend!**

- Man verwendet vor allem **Funktionen**, **Funktionale**, **Funktionsabschlüsse**, **unveränderbare Variablen**, **Ausdrücke**, **Terme**, **Musterabgleiche**, ...

Wo kann man auf prozedurale Mittel kaum verzichten?

- Beim **Einlesen und Ausgeben** von Daten
  - Die Variable **Tastatur** wird verändert, wenn man von ihr liest
  - Die Variable **Bildschirm** wird verändert, wenn man zu ihr ausgibt
  - **...**
- Beim **Ändern von großen Datenstrukturen**
  - z.B. beim **Einfügen** eines Elements in eine sortierte Liste, beim **Ändern** oder **Entfernen** eines Elements, ...

Nebenläufige Einheiten (1)

- Weit verbreitet: **Fäden** (engl. threads).
- Haben **gemeinsamen Speicher** (engl. shared memory).
- Der ermöglicht **Nebenläufigkeitsfehler.**
- Abhilfe: Reservierende **Monitore** (in Java: `synchronized`).
- Die ermöglichen **Verklemmungen** (engl. deadlocks).
- Abhilfe: Reservierbare Betriebsmittel nummerieren, nur in einfachen Fällen möglich.

Nebenläufige Einheiten (2)

- In Betriebssystemen: **Prozesse.**
- **Kein** gemeinsamer Speicher, eigene **Adressräume.**
- Das **erschwert** Nebenläufigkeitsfehler.
- und macht Prozesse **schwerfällig** (engl. heavyweight).

- **Warum** sind Prozesse so **schwerfällig**?

Nebenläufige Einheiten (3)

- Prozesse sind **schwerfällig**, weil
  - sie **misstrauisch** gegen Programme sein müssen
  - Programme **fehlerhaft** oder **bösartig** sein können
  - Programme vor ihrer Ausführung **nicht geprüft** werden

Nebenläufige Einheiten in Erlang

- werden **Prozesse** genannt.
- entsprechen dem **Actor Modell** (Hewitt, Bishop, Steiger, 1973).
- kommunizieren über **Nachrichten** (engl. messages).
- haben **keinen** gemeinsamen Speicher.
- sind sehr **leichtfüßig** (nicht schwerfällig).

Prozesse und Seiteneffekte (1)

```
next(N) ->
    receive
        {'N?', Absender} ->   % Eine 2-Tupel-Meldung
            Absender ! N,     % N an den Absender schicken
            next(N);          % Endrekursion
        'incN!' ->            % Eine Atom-Meldung
            next(N+1);        % Endrekursion
        halt ->               % Eine Atom-Meldung
            ok                % Rekursion wird beendet
    end.
```

Jeder Prozess, der diese Funktion ausführt, ist eine Art **Variable**, deren Wert von anderen Prozessen **gelesen** und **verändert** werden kann.
`'N?'` bedeutet: Welchen Wert hat **N** momentan?
`'incN!'` bedeutet: Erhöhe den Wert von **N** um 1!

Prozesse und Seiteneffekte (2)

Jeder Erlang-Prozess hat eine **Abbildung** (engl. map, process dictionary)

Folgende Befehle bearbeiten diese Abbildung
(**S** wie **Schlüssel** und **W** wie **Wert**):

```
put(S,W)  Fügt einen Eintrag {S,W} ein, liefert den alten
          Wert zum Schlüssel S (evtl. undefined).
get(S)    Liefert den aktuellen Wert zu S (evtl. undefined)
erase(S)  Liefert den aktuellen Wert zu S und verbindet
          S dann mit undefined.
...
```

Prozesse und Seiteneffekte (3)

Einen **Faden / Prozess erzeugen** und starten dauert (in **Mikrosekunden**):

Java	250
Erlang	5

Simple und informelle Zeitmessung auf einem PC mit Windows XP,
einem 2,6 GHz Pentium 4 Prozessor, Java 6 und Erlang 5.7,
je 30 Tausend Fäden/Prozesse erzeugt und gestartet.

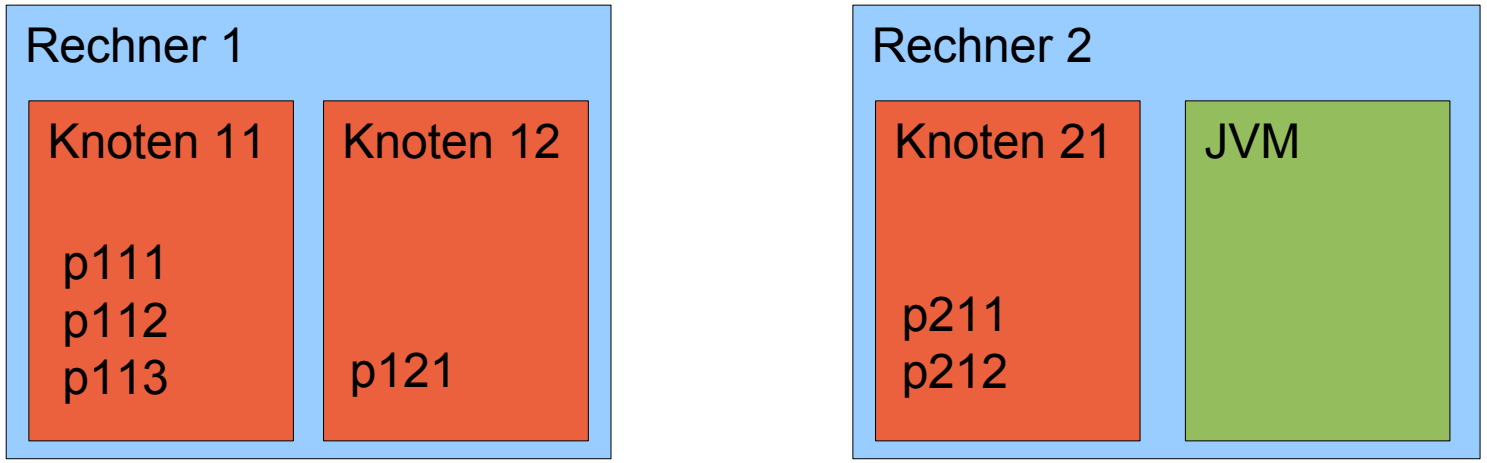
Statische Struktur von Erlang-Software

- Eine **Erlang-Quelldatei** enthält einen (Erlang-) **Modul.**
- Ein **Modul** enthält **Funktionen.**
- Ein Modul kann beliebig viele seiner Funktionen **exportieren.**
- Andere Module können nur exportierte Funktionen aufrufen.

```
%% Datei meinModul01.erl
-module(meinModul01).
-export([hallo/0, hallo/2, start/0]).
-import(lists, [reverse/1]).
```

Dynamische Struktur eines Erlang-Systems

- Erlang **Knoten** (engl. nodes)
  ("eine Erlang-Maschine", vergleichbar mit einer JVM)
- Auf einem Knoten laufen **Prozesse**
- Jeder Prozess hat eine eindeutige **PID** und
  einen **Briefkasten**, in dem er Meldungen empfangen kann.



Funktionale Konzepte in Erlang

- **Funktionale** (engl. higher order functions)
  (d.h. Funktionen mit Funktionen als Parameter oder Ergebnis)
- **Musterabgleich** (engl. pattern matching)
- **Fun**-Ausdrücke (griechisch λ - Ausdrücke)
- **Funktionsabschlüsse** (engl. closures)
- **Komprehension** (Beschreibung von Listen durch Ausdrücke)
- **Strikte** Auswertung (engl. **eager** evaluation)
  - **Nicht-strikte** (engl. **lazy**) Auswertung leicht programmierbar

Musterabgleich (1): Vereinbarung von Funktionen

```
fibA(N) ->                  % A wie "Anfaenger"
    case N of
        0 -> 1;
        1 -> 1;
        _ -> fibA(N-1) + fibA(N-2)
    end.


fibB(0) -> 1;               % B wie "schon besser"
fibB(1) -> 1;
fibB(N) -> fibB(N-1) + fibB(N-2).
```

Musterabgleich (2)

Namen von Variablen beginnen mit einem **Groß**buchstaben.

Muster		**Ausdruck**	**gelingt?**
`{ok, {A, B}}`	=	`{ok, {17, "ABC"}}`	ja
`{`**`ok`**`, {A, B}}`	=	`{`**`xx`**`, {17, "ABC"}}`	nein
`{ok, `**`{`**`A, B`**`}`**`}`	=	`{ok, `**`[`**`17, "ABC"`**`]`**`}`	nein
`{ok, {A, B}}`	=	`{ok, {17, "ABC", `**`3`**`}}`	nein
`{ok, {A, B}}`	=	`{ok, {"ABC", 17}}`	ja
`C`	=	`...`	immer!

Muster: Darf "**alte**" Variablen (mit Wert) und "**neue**" (ohne Wert) enthalten.
Ausdruck: Darf nur "**alte**" Variablen mit Wert und Literale enthalten.

Ein Funktional:

Vereinbarung:

```
tst(_, [])     ->
   io:format("Test fertig~n"); % nicht-rekursiver Fall
tst(Fun, [E1|Rest]) ->
   io:format("(~w, ~w)~n", [E1, Fun(E1)]),
   tst(Fun,Rest).                % Endrekursion
```

Zwei Aufrufe:

```
tst(fun meineFunC/1,    [0, 5, -3, 23])
tst(fun(N) -> N*N end,  [0, 5, -3, 23])
```

Warum ist C keine funktionale Sprache?

- Die Sprache C hat doch:
  - Funktionen als **Parameter** von Funktionen
  - Funktionen als **Ergebnis** von Funktionen
  - Konstanten
  - Ein Ausdrucks-if-then-else (`... ? ... : ...`)

Was **fehlt** der Sprache C zum funktionalen Glück?

C hat keine Funktionsabschlüsse (engl. closures)

Beispiel für Abschlüsse in Erlang:

```
machMal(M) ->
    fun(N) -> M * N end.

wendeAn() ->
    Mal3 = machMal(3),      % fun(N) -> 3 * N end.
    N    = Mal3(7),         % fun(7) -> 3 * 7 gleich 21
    ...
```

Die Funktion **machMal(3)(N)** (alias **Mal3(N)**) greift nicht nur auf ihren Parameter **N** zu, sondern auch auf die (für sie globale) Variable **M**. Jedes Ergebnis der Funktion **machMal(M)** muss den Wert von **M** "irgendwie enthalten".

Listen-Komprehension: Zwei Beispiele

```
[X+1 || X <- [1,2,3,17]]
```

Die Liste aller X+1, wobei X aus [1,2,3,17] stammt:
[2,3,4,18]

```
[{X,Y} || X <- [7,3,5],
          Y <- lists:seq(10,13), Y rem 2 == 0]
```

Die Liste aller {X,Y}, wobei
X aus [7,3,5] stammt und
Y aus [10,11,12,13] stammt und gerade ist:
[{7,10},{3,10},{5,10,{7,12},{3,12},{5,12}]

Listen-Komprehension: Quicksort

```
qsort([]) -> [];
qsort([E1|Rest]) ->
   qsort([X || X <- Rest, X  < E1])
   ++ [E1] ++
   qsort([X || X <- Rest, X >= E1]).
```

Der Operator `++` konkateniert zwei Listen zu einer Liste.

Erlang-Datenstrukturen

Ganzzahlen: beliebig groß, auch mit Basen 2 bis 16, sonst wie üblich

Bruchzahlen: 64-Bit IEEE-754-1985, wie üblich

Atome: `breite hÖHE 'Tiefe' '<- +OXO+ ->' true false`
(jeder Zugriff kostet nur einen Maschinenbefehl)

Tupel: `{123,abc} {ab,{cd, ef}, 34} {a,{b,{c,1},2},3}`
`{'Hallo', 'Sonja', 'wie_geht es?'}`

Listen: `[65,66,67] [$A,$B,$C] "ABC" [] [[],[],[]]`
`[alfa,12,beta,3.5,"ABC"] [a,[b,[c,1],2],3]`

Listen und **Tupel** sind beliebig kombinierbar:
`[ab,{ok,3},{error,5}] {list,[di,mi,do]}`

Warum ist Erlang eine gute nebenläufige Sprache ?

- Weil Erlang funktional ist ?

Warum ist Erlang eine gute nebenläufige Sprache ?

- Weil Erlang funktional ist ? **Nein!**
- Weil Prozesse ausschließlich durch **Meldungen** kommunizieren und **keinen gemeinsamen Speicher** haben!

Abschließende Thesen

- Seiteneffekte
  - sind schrecklich

Abschließende Thesen

- Seiteneffekte
  - sind schrecklich
  - und unverzichtbar (auch in funktionalen Sprachen)

Abschließende Thesen

- Eine gute Programmiersprache
  - hat einen **funktionalen** Teil
  - und eine **prozeduralen** Teil

Abschließende Thesen

- Eine gute Programmiersprache
  - hat einen **funktionalen** Teil
  - und eine **prozeduralen** Teil
  - und **trennt** die beiden Teile klar und offensichtlich