Duale Hochschule

Baden-Württemberg Mannheim

Portfolio-Prüfung

Dokumentation: Erlang

Verfasser(innen):	Babett Müller (2696346), Jost-Tomke Müller (9974531), Mathis Neunzig (2240587), Nathalie Möck (7163124)
Kurs:	WWI 2020 SE B
Modul:	Entwicklung verteilter Systemen
Betreuer(in):	Prof. Dr. Hans-Henning Pagnia
Bearbeitungszeitraum:	4. Theoriesemester (15.06.2022 – 29.07.2022)
Abgabedatum:	29.07.2022

Inhaltsverzeichnis

1. Einleitung

Diese Ausarbeitung ist im Zuge der Portfolio-Prüfung des Moduls „Entwicklung verteilter Systeme“ des Kurses WWI 2020 SE B der Dualen Hochschule Baden-Württemberg Mannheim entstanden, welches von Prof. Dr. Pagnia unterrichtet wird.

Sie dient als Zusammenfassung über das wichtigste Basiswissen sowie einen ersten Überblick über die Rolle von Erlang in der verteilten Programmierung. Hierbei im Folgenden genauer auf die Besonderheiten und Grundlagen von Erlang eingegangen sowie den Installations- und Einrichtungsprozess. Des Weiteren werden kurz die Vor- und Nachteile von Erlang beleuchtet wie auch seine Art und Weise die verteilte Programmierung zu realisieren.

2. Verteilte Programmierung

2.1 Allgemeine Definition

Verteilte Programmierung wird die Programmierung von Anwendungen, Programmen, et.c innerhalb verteilter Netze oder Systeme bezeichnet. Hinter dem Begriff „verteiltes System“ können sich jedoch verschiedene Definitionen verbergen. Jedoch lässt es sich allgemein als definieren als eine „*Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen*“ (Tanenbaum & Steen, 2008, p. 19). Es handelt es sich hierbei also um ein System, welches einzelne autonome Funktionseinheiten, z.B. Computer, miteinander mittels eines Transportsystems verbinden, sodass diese gemeinsam Tätigkeiten ausführen können. Dieser Aufbau ermöglicht eine hohe Flexibilität sowie eine optimale und gezielte Funktionsverteilung auf angepasste Komponenten. Dies wiederum führt auf Grund des perfekten Zuschnitts zu einer Effizienz- und Leistungssteigerung bei gleichzeitig hoher Anpassbarkeit an bestimmte Lösungen (vgl. Bengel, 2014, pp. 4, f.).

2.2 Realisation der verteilten Programmierung in Erlang

Das Prinzip der verteilten Programmierung wird in Erlang mittels Aktoren umgesetzt, Aktoren im Kontext der Programmiersprache Erlang sind *Prozesse*. Im Gegensatz zur allgemeinen Definition von Prozessen handelt es sich hierbei nicht um eine Abbildung auf Systemprozesse oder -threads, sondern lediglich um eine getrennte Verarbeitung innerhalb einer virtuellen Maschine. Diese können auf Grund ihrer geringen Speichergröße von wenigen Bytes und ihrer Leichtgewichtigkeit schnell gestartet werden und in einer Vielzahl parallel erstellt werden (vgl. Adermann, 2010, p. 7). Dies ist darauf zurückzuführen, da die Prozesse durch das Erlang-Laufzeitsystem kontrolliert werden und nicht durch das zugrundeliegende Betriebssystem (vgl. Armstrong, 2003, p. 3).

Um die nebenläufige Programmierung in Erlang realisieren zu können, ist der einzig mögliche Mechanismus das sog. *Aktormodell* bzw. *Aktorsystem*. Diese Systeme beinhalten eine Menge von Aktoren, welche untereinander kommunizieren. Sie sind in der Lage ihr Verhalten abhängig von den erhaltenen Nachrichten zu verändern (vgl. Agha, 1986). D.h. ein Aktor beeinflusst nicht direkt einen anderen, sondern er sendet eine Nachricht an einen 2. Aktor mit jener Nachricht, wie dieser beeinflusst werden soll. Daher ist jede Kommunikation explizit, nachvollziehbar und sicher (vgl. Hebert, 2013). Jeder dieser involvierten Aktoren besitzt eine eigene eindeutig identifizierbare Adresse, welche eine Kommunikation mit diesem Aktor ermöglicht (vgl. Agha, 1986). Diese wird als *Prozess-Identifikator* bezeichnet (vgl. Adermann, 2010, p. 7).

Durch die Tatsache, dass die gesamte Kommunikation zwischen den Aktoren mittels sog. „Message Passing“ realisiert werden, existieren keine Zustände oder Daten, welche die

Aktoren gemeinsam teilen. In diesem Verfahren werden den Aktoren asynchron Kopien der entsprechenden Nachrichten zukommen gelassen anstatt von Referenzen oder Zeiger auf entsprechende Speicherebereiche, auf welche auch andere Aktoren eventuell zugreifen können. Dies wird als „Shared Nothing“ Architektur bezeichnet (vgl. Agha, 1986).

3. Erlang

3.1 Grundlagen

Um eine neue Programmiersprache kennenzulernen, sollte sich ein Leser zuerst mit den Grundlagen derselbigen auseinanderzusetzten. Dies ist auch hier im Zusammenhang mit Erlang geschehen. Zuerst einmal handelt es sich bei Erlang um eine funktionale Programmiersprache, welche in ihrer Anwendung auf nebenläufige und verteilte Programmierung fokussiert ist. Die verteilte Programmierung wurde bereits zuvor grundlegend erklärt, was genau es im Zusammenhang mit Erlang hier auf sich hat folgt entsprechend an späterer Stelle. In den Grundlagen bleibt zu betrachten was es mit einer funktionalen Programmiersprache auf sich hat. Hierbei ist vor allem auf einen Grundsatz aufmerksam zu machen: Den der unveränderbaren Variablenwerte. Hier lehnt sich die funktionale Programmierung stark an die Mathematik in dem Prinzip, dass eine Variable, der zu einem Zeitpunkt ein Wert zugewiesen wird, diesen für die gesamte Dauer einer Rechnung beziehungsweise eines Programmablaufes beibehält (vgl. Hebert, 2013). Der Fokus liegt also anders als bei der objektorientierten Programmierung nicht auf der Situation eines Objektes beziehungsweise einer Variablen, da sich diese nicht verändert, sondern darauf welche Aktionen durchgeführt werden. Die Frage, “Was gemacht werden soll?”, steht also bei Erlang im Vordergrund (vgl. Peter, 2017). Im Rahmen von Erlang gibt es bei tieferer Betrachtung zwar einige Ausnahmen zu dieser strikten Definition, jedoch gibt es einen guten ersten Eindruck, worauf sich der Leser einstellen sollte. Nachdem geklärt ist, dass Erlang sich im Programmierparadigma der funktionalen Programmierung bewegt, fällt der nächste Blick auf die genaue Arbeit mit Erlang (vgl. Hebert, 2013).

Hierbei muss zuerst eine Programmierumgebung eingerichtet werden auf der Erlang programmiert und ausgeführt werden kann. Dazu sind einige Besonderheiten zu beachten. Zuerst teilt sich der Prozess in zwei Phasen die “build” und die “installation” Phase. Beginnend in der “build” Phase muss der Nutzer zuerst die nötigen Werkzeuge installieren, um später ein lauffähiges Erlang-Programm zu besitzen. Nachdem die nötigen Werkzeuge heruntergeladen und installiert wurden, kann nun die Erlangdatei unpacked werden, damit ein erster build konfiguriert werden kann. Anschließend wird die $ERL_TOP Variable gesetzt. Damit der erste build richtig konfiguriert ist wird als nächstes $ ./configure [options] ausgeführt. Abschließend wird der erste build durchgeführt mit $ make als Befehl in der Kommandozeile. Erst nach erfolgreichem Test dieses builds wird Erlang an sich installiert und die zweite Phase beginnt. Hierzu wird $ make install ausgeführt, wodurch der neuste Release von Erlang installiert wird. Mit diesem Schritt ist der Installationsprozess abgeschlossen und das erste Programm kann umgesetzt werden (vgl. Ericsson AG, 2022).

Nachdem sichergestellt ist, dass Erlang an sich funktioniert, stellt sich die Frage wofür es am besten genutzt werden kann. Wie die meisten Programmiersprachen besitzt Erlang seine Stärken für die es sich besonders gut eignet. Durch die Art seiner Entwicklung ist Erlang vor allem für Dienste die eine hohe Verfügbarkeit und Skalierbarkeit benötigen nutzbar. Ursprünglich aus der Telekommunikationsdienstleistungsbranche stammend wurde es hierfür eigens entwickelt. Zudem muss in der Branche eine gute Parallelität von Verbindungen aufrechterhalten werden. Daher ist die Sprache nach wie vor besonders für das Aufbauen von verteilten Systemen geeignet. Ein weiterer essenzieller Aspekt war der Sprachenumfang, welcher auf ein Minimum reduziert wurde (vgl. Adermann, 2010, pp. 1-3). Neben den Eigenschaften, die sich direkt auf die Nutzung beziehen ist die Sprache auch bekannt dafür eine besonders hohe Fehlertoleranz mitzubringen (vgl. Armstrong, et al., 1996, p. 1).

Es wurde geklärt für welchen Umfang Erlang sich grundlegend eignet und in welches Programmierparadigma es eingeordnet werden kann. Was genau es aber mit der funktionalen Programmierung auf sich hat, wurde bisher nicht im Detail erläutert. Daher wird nun genauer darauf eingegangen was neben den nicht veränderbaren Variablen die funktionale Programmierung ausmacht. Ihren Ursprung nimmt die funktionale Programmierung aus der Mathematik, welche sie versucht besonders akkurat darzustellen. Hier findet auch das Phänomen der starren Variablen seinen Ursprung, denn eine Variable x in der Mathematik wird nie ihren Wert innerhalb einer Rechnung ändern. Dies wurde also auch in die funktionale Programmierung übernommen. In der Programmierung umgesetzt wird das Konzept durch die Definition von Funktionen innerhalb eines Programmes. Diese Funktionen werden in Erlang zudem deterministisch betrachtet, sollten also für gleiche Eingaben stets gleiche Ausgaben erzeugen. In Erlang selbst wird dieses Paradigma entsprechend angepasst nach dem Prinzip ihm auf oberster Ebenen zu folgen und dies genau so lange wie es für reale Probleme sinnvoll ist (vgl. Herbert, 2013).

3.2 Die Rolle von Erlang in der verteilten Programmierung

„*Die Sprache Erlang eignet sich hervorragend für verteilte Systeme in denen Ausfallsicherheit und Skalierbarkeit von großer Bedeutung sind*“ (Adermann, 2010, p. 9). Erlang ist daher perfekt zugeschnitten auf die Verwendung in der verteilten Programmierung und erhält auch seine grundlegenden Charakteristiken sowie Stärken aus diesem Verwendungszweck. Daher sind Eigenschaften wie asynchrone Kommunikation über Computergrenzen hinweg, die nebenläufige und verteilte Programmierung mittels vorgegebener Konstrukte und die Robustheit von Systemen direkter Bestandteil der Sprache (vgl. Meiser, 2010, p. 1).

3.2.1 Vorteile

Ein Vorteil von Erlangs *Shared Nothing Architektur* und dem *Message Passing* ist, dass das Aktorsystem auf einem Einprozessorsystem oder in einem verteilten System ohne weitere Probleme oder Anpassungen verwendet werden kann (vgl. Adermann, 2010, p. 6). Die Umwandlung kann einfach durchgeführt werden, indem die verschiedenen parallel ablaufenden Prozesse auf unterschiedliche Maschinen zugewiesen werden (vgl. Armstrong, 2003, p. 3). Des Weiteren macht Erlang hierbei auch keinen Unterschied, ob es in einem verteilten System mit geteiltem oder getrenntem Speicher angewendet wird. Daher wird die Verteilung der Aktoren als transparent angesehen und für viele Entwickler vom Vorteil, da ihnen dabei kein Mehraufwand entgegenkommt, deren Verwendung jedoch entsprechende Vorteile mit sich bringen. Um dies so umsetzen zu können, müssen jedoch Aktoren in allen Bereichen der programmierten Anwendungen konsequent verwendet werden (vgl. Adermann, 2010, p. 6).

Ein weiterer Vorteil stellt Erlang auch im Bereich der Fehlertoleranz und Robustheit dar. Die Aktoren von Erlang können auch in Programmteilen eingebaut werden, welche nicht nebenläufig ausgeführt werden müssen. Jedoch führt eine entsprechende Trennung der Systemkomponenten durch jene Aktoren zu einer gesteigerten Fehlertoleranz. Dies ist darauf zurückzuführen, dass, sollte ein Aktor (oder „Worker“) einen Fehler aufweisen, der Rest des Systems unabhängig weiterarbeiten kann, während der betroffene Aktor z.B. neu gestartet wird (vgl. Adermann, 2010, p. 6). Hierbei kümmern sich die sog. „Observers“, um die Lauffähigkeit sowie die Wiederherstellung der Aktoren im Fehlerfall (vgl. Armstrong, 2003, p. 3). Dadurch kann eine sehr hohe Verfügbarkeit gewährleistet werden.

Erlangs Aktoren kapseln interne Zustände, welche durch Nachrichten oder Methodenaufrufe verändert oder manipuliert werden können. Dieses Aktorsystem ist auch in der Lage Polymorphie oder andere objektorientierte Konzepte umzusetzen und bietet somit ein ähnliches Prinzip für Erlang an, wie objektorientierten Programmiersprachen es zur Verfügung stellen (vgl. Adermann, 2010, pp. 6, f.).

Neben den bereits erwähnten Funktionalitäten und Performanz erweist sich auch die Skalierbarkeit als Vorteil von Erlang. Durch das Hinzufügen weiterer Prozessoren und das Verschieben von Prozessen auf andere Prozessoren ist eine einfache Skalierung möglich (vgl. Armstrong, 2003, p. 3).

Weitere Vorteile, welche besonders von den Entwicklern von Erlang hervorgehoben werden, sind Folgende (vgl. Armstrong, et al., 1996, pp. 1, f.):

- eine deklarative Syntax, welche frei von Nebenwirkungen ist.
- ein schnelle Reaktionszeit, da Erlang für die Programmierung von Echtzeitsystemen entwickelt wurde.

- ein kontinuierlicher Betrieb, welche es ermöglicht, Code in einem laufenden System zu ersetzen sowie alte und neue Codeversionen gleichzeitig ausführen zu können.
- eine effektive Speicherverwaltung mit einem Echzeit-Garbage Collector, welche dahingehende typische Programmierfehler verhindert.
- leichte Integration und Interaktion mit Programmen oder Code anderer Programmiersprachen.

3.2.2 Nachteile

Da es sich bei Erlang um eine Programmiersprache konzipiert für die verteilte Programmierung handelt, kann sie nicht als Allzweckprogrammiersprache verwendet werden. Dies liegt u.a. an der Tatsache, dass Erlang nur eine beschränkte Unterstützung der Textverarbeitung sowie eine begrenzte Möglichkeit für benutzerdefinierte Datenstrukturen vorzuweisen hat (vgl. Adermann, 2010, p. 9).

Des Weiteren ist der Support durch die Community in Form von Foren, Tutorials, o.Ä. eingeschränkt. Dies ist damit zu begründen, dass Erlang zwar weit verbreitet noch eingesetzt wird, jedoch die aktive Community durch seine eher unscheinbare Verwendung recht klein ist. Demnach kann bei komplexeren Bugs nicht im großen Umfang auf die Hilfe von anderen zurückgegriffen werden.

Außerdem, wie im vorherigen Unterkapitel bereits beschrieben, ist der Prozess der Installation und Einrichtung für Erlang umständlich und aufwendig im Vergleich zu anderen alltäglichen Programmiersprachen. Darunter fällt außerdem die Bereitstellung von Erlang-Anwendungen, welche für manch andere Programmierer auf Grund der Syntax schwer zu verstehen sein kann.

Der größte Nachteil von Erlang ist schließlich das Fehlen eines Typsystems, da der Compiler den Code nicht auf dessen Typisierungen prüft. Dies hat zur Folge, dass z.B. Tippfehler im Quellcode oft nicht zu einem Fehler, sondern zu einem Deadlock führen und somit die Fehlerbehandlung zur Laufzeit sehr schwierig macht (vgl. Huch, 2001, p. 4).

4. Aufteilung

Aufgabenaufteilung:

a) bearbeitet von: Nathalie Möck, Babett Müller
   vertont von: Mathis Neunzig

b) bearbeitet von: alle
   vertont von: Jost-Tomke Müller

c) bearbeitet von: Jost-Tomke Müller
   vertont von: Babett Müller

d) bearbeitet von: Mathis Neunzig
   vertont von: Nathalie Möck

Literaturverzeichnis

Adermann, N., 2010. *Seminar Sprachen für die Parallelprogrammierung: Erlang.* Berlin: o.V..

Agha, G., 1986. An overview of actor languages. *OOPWORK '86: Proceedings of the 1986 SIGPLAN workshop on Object-oriented programming*, 9.-13. Juni, pp. 58-67.

Armstrong, J., 2003. *Concurrency Oriented Programming.* Kista: Distributed Systems Laboratory, Swedish Institute of Computer Science.

Armstrong, J., Virding, R., Wikström, C. & Williams, M., 1996. *Concurrent Programming in ERLANG.* 2. Hrsg. New Jersey: Prentice Hall.

Bengel, G., 2014. *Grundkurs Verteilte Systeme: Grundlagen und Praxis des Client-Server und Distributed Computing.* 4. Hrsg. Wiesbaden: Springer Fachmedien.

Ericsson AG, 2022. *ERLANG, Installation Guide, User's Guide, 2 Building and Installing Erlang/OTP.* [Online]
Available at: https://www.erlang.org/doc/installation_guide/install
[Zugriff am 25. Juli 2022].

Hebert, F., 2013. *Learn You Some Erlang for Great Good!, A Beginner's Guide.* San Francisco: No Starch Press.

Huch, F. G., 2001. *Verification of Erlang Programs using Abstract Interpretation and Model Checking.* Aachen: Rheinisch-Westfälische Technische Hochschule Aachen.

Meiser, S., 2010. *Einsatz von Erlang in der Lehre - Eine Untersuchung für das Fach "Verteilte Systeme".* Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.

Peter, K., 2017. *Codeadventurer, Funktionale Programmierung mit Hilfe von Java Lambda Ausdrücken.* [Online]
Available at: http://www.codeadventurer.de/?p=3166
[Zugriff am 24. Juli 2022].

Tanenbaum, A. S. & Steen, M. v., 2008. *Verteilte Systeme: Prinzipien und Paradigmen.* 2. Hrsg. München: Pearson Education Deutschland GmbH.

Ehrenwörtliche Erklärung

Hiermit bestätigen wir, Babett Müller, Jost-Tomke Müller, Mathis Neunzig und Nathalie Möck, dass wir das Programmteile sowie die dazugehörige Dokumentation selbstständig nur unter Verwendung der erlaubten Hilfsmittel ohne die Hilfe anderer entwickelt und implementiert haben.

27.07.2022, Mannheim, B. Müller
Datum, Ort, Unterschrift

27.07.22, Mannheim, T. Müller
Datum, Ort, Unterschrift

27.07.2022, Walldorf, Neunzig
Datum, Ort, Unterschrift

27.07.2022, Mannheim, N. Möck
Datum, Ort, Unterschrift