[[1]](#footnote-1)

Redesign einer Chat-Anwendung als verteiltes System.

C. Eidelloth, D. Sauter, F. Stützinger and M. Auch, Team <Winner>

*Abstract*—These instructions give you guidelines for preparing papers for IEEE TRANSACTIONS and JOURNALS*.* Use this document as a template if you are using Microsoft *Word* 6.0 or later. Otherwise, use this document as an instruction set. The electronic file of your paper will be formatted further at IEEE. Define all symbols used in the abstract. Do not cite references in the abstract. Do not delete the blank line immediately above the abstract; it sets the footnote at the bottom of this column.

*Index Terms*—About four key words or phrases in alphabetical order, separated by commas. For a list of suggested keywords, send a blank e-mail to [keywords@ieee.org](mailto:keywords@ieee.org) or visit the IEEE web site at [http://www.ieee.org/web/developers/webthes/index.htm](http://www.ieee.org/organizations/pubs/ani_prod/keywrd98.txt).

# Motivation und Problemstellung

Vernetzte Rechnersysteme haben sich in letzter Zeit rasch entwickelt, dazu zählen auch die Verteilten Systeme, welche sich aus verschiedenen unabhängigen Bestandteilen zusammensetzen um ein Vollständiges System zu bilden.[1]

In dieser Arbeit soll der Umbau eines Vorhandenen nachrichtenbasierten Java Programms in ein Verteiltes System realisiert werden. Dabei gilt es im Rahmen der Aufgabenstellung, neue und bereits erlernte Technologien zu nutzen um eine Verteilte Anwendung mit dem Fokus auf Skalierbarkeit, Transaktionssicherheit und Performance zu entwickeln.

Im Zuge einer Umsetzung ergeben sich eine Reihe von Komplikationen aufgrund der gestiegenen Anforderungen, welche auf die Integration bzw. Kopplung von Wildfly und MariaDB der Chatanwendung zurückzuführen sind. Diese Erweiterung soll es ermöglichen die Anwendung, über die Rechnerumgebung hinaus, betreiben zu können.

# Einführung

## Transaktionen in verteilten Systemen

Eine Transaktion ist eine zusammengefasste Abfolge von Ereignissen, die alle erfolgreich ausgeführt werden müssen, um ein Ergebnis zu erzielen.[2] Verteilte Transaktionen zeichnen sich im Wesentlichen dadurch aus, dass

### Eigenschaften

Um die Erfüllung des Zwecks einer Transaktion sicherzustellen, müssen Transaktionssysteme, die für die Verarbeitung von Transaktionen eingesetzt werden, die Einhaltung der ACID-Prinzipien gewährleisten. Zu diesen Prinzipien zählen die Unteilbarkeit (Atomicity), Konsistenz (Consistency), Isolation (Isolation) und Dauerhaftigkeit (Durability).[3] Wichtig für das Verständnis von Transaktionen sind zudem die Phasen, die während der Verarbeitung einer Transaktion eintreten können.

begin of transaction

doing

end of transaction

abort

rollback

( -> Abbildung!)

### Herausforderungen

Für die Realisierung von verteilten Transaktionen werden Commit-Protokolle wie das Zwei-Phasen Commit Protokoll eingesetzt.

## Message-Service-Architektur

### Kriterien für die Anwendung, Vorteile und Herausforderungen

Erzeuger (Sender) von Nachrichten benötigt für die weitere Verarbeitung keine synchrone Antwort (fire-and-forget).

Integrationsszenarien: Es müssen mehrere unterschiedliche Systeme zusammenarbeiten.

Sender erzeugt Nachrichten erheblich schneller, als sie von Empfängern verarbeitet werden können. Eine Message-Queue als Puffer kann hier ausgleichen.

Vorteile

Sender und Empfänger von Nachrichten können in völlig unterschiedliche Technologien und Programmiersprachen erstellt werden.

Message-Queues, insbesondere solche mit zuverlässiger Zustellung (reliable messaging), können die Verfügbarkeit und Rubstheit von Systemen erheblich steigern. Aus diesem Grund werden MQ-Systeme insbesondere im Bereich Finanz- und Kontodaten häufig eingesetzt.

Nachteile

Message-Queues (ob kommerziell oder Open-Source) sind in sich komplexe Systeme mit teilweise hohem Einführungs- und Administrationsaufwand.

Asynchrone und nachrichtenbasierte Programmierung ist signifikant aufwendiger als einfacher call-and-return- Stil. Fehlversuche in asynchronen Systemen kann aufwendig sein.

[QUELLE hab ich zu den punkten -> folgt! (Max)]

### Java Messaging Service (JMS)

JMS ist eine API, die als Bestandteil von Java EE Interfaces definiert, welche die Interaktion von Java-Anwendungen mit einer Message Oriented Middleware (MOM) ermöglichen.

#### JMS-Destinations

JMS bietet zur Übermittlung von Nachrichten zwei JMS-Destinations an, Queue und Topic. Die Queue dient der asynchronen Point-to-Point Kommunikation. Nachrichten werden i.d.R. nach dem FIFO-Prinzip vom Sender in der Queue abgelegt und vom Empfänger dort abgeholt. Topics werden hingegen eingesetzt, wenn die Nachricht im Rahmen eines Publish-Subscribe-Verfahren an mehrere Empfänger versendet werden soll.

#### JMS-Provider

Für den Einsatz von JMS wird ein JMS-Provider benötigt, der die genannten JMS-Destinations verwaltet. Als Beispiel für einen JMS-Provider kann HornetQ (ehemals JBoss Messaging) angeführt werden.

# Konzeption der verteilten Chatanwendung

Ausgehend von der zu Beginn bereitgestellten Chatanwendung, gilt es eine, für ein verteiltes System optimierte Anwendung zu erstellen. Durch diese Änderung der Anforderungen, ist die bisherige monolithische Struktur nicht passend. Es galt daher, die Architektur neu zu überdenken. Im nachfolgenden ist daher ein technisches Konzept für die veränderte Umgebung und die grundlegende Architektur für die beiliegende Chatanwendung beschrieben.

## Struktur des Anwendungsszenarios

## Komponentensicht

## Datenmodell

## Fehlertolerante Transaktionsverarbeitung für REST-Clients

# Implementierung

## Umsetzung der Basisarchitektur

### Ablösung des TCP-Websocket durch JMS

(„The number of MDB threads is equal to 16 (this is the default value)) Es wird automatisch ein Threadpool mit 16 Threads erzeugt, die durch EJB bzw. den Message Driven Beans verwaltet werden [4]. Dies bezüglich sind keine weiteren Anforderungen an die Chatanwendung gegeben, weshalb keine weiteren Konfigurationsarbeiten an dem Pool vorgenommen wurden.

### Umsetzung der REST-APIs

* JAX-RS / Jersey

### Anbindung der Datenbankinstanzen

## Umsetzung einer gemäß XA verteilten Transaktion

X/OpenXA (XA) ist ein Standard für die Verarbeitung von verteilten Transaktionen. Wesentliches Element dieses Standards ist das unter A. beschriebene Zwei-Phasen-Commit Protokoll.

* EJB

## Umsetzung eines Admin-Clients

### Angular2 als Entwurfsentscheidung

# Benchmarking

## Testaufbau und Rahmenbedingung

### Verwendete Infrastrukturkomponenten

### Verwendete Testmetriken

### Testspezifikation

## Dokumentation der Messergebnisse

## Evaluation der Messergebnisse

# FAZIT

An excellent style manual and source of information for science writers is [9]. A general IEEE style guide, *Information for Authors,* is available at <http://www.ieee.org/organizations/pubs/transactions/information.htm>

Appendix

Appendixes, if needed, appear before the acknowledgment.

Acknowledgment

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” **Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page**. [5]

References

[1] A. Schill und T. Springer, *Verteilte Systeme*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[2] H. Balzert, *Lehrbuch der Softwaretechnik, Entwurf, Implementierung, Installation und Betrieb*, 3. Aufl. Spektrum, Akademischer Verlag.

[3] U. Hammerschall, *Verteilte Systeme und Anwendungen*. Pearson Education.

[4] „Message-Driven EJBs“. [Online]. Verfügbar unter: http://docs.oracle.com/cd/E11035\_01/wls100/ejb/message\_beans.html. [Zugegriffen: 17-Nov-2016].

[5] „Messaging configuration - WildFly 8 - Project Documentation Editor“. [Online]. Verfügbar unter: https://docs.jboss.org/author/display/WFLY8/Messaging+configuration. [Zugegriffen: 13-Okt-2016].

**First A. Author** (M’76–SM’81–F’87) and the other authors may include biographies at the end of regular papers. Biographies are often not included in conference-related papers. This author became a Member (M) of IEEE in 1976, a Senior Member (SM) in 1981, and a Fellow (F) in 1987. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author’s educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state or country, and year degree was earned. The author’s major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author’s last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (city, state: publisher name, year) similar to a reference. Current and previous research interests ends the paragraph.

The third paragraph begins with the author’s title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, the biography will be indented around it. The photograph is placed at the top left of the biography. Personal hobbies will be deleted from the biography.

1. [↑](#footnote-ref-1)