Message Oriented Middleware

Programmentwurf

vorgelegt am 03. September 2024

Sommersemester 2024

CAS / DHBW Mosbach

Kurs T3M70306

Fortgeschrittene Konzepte Programmierung

von

Nils Hepp und Florian Böhm

1. Einleitung

[fl028/little\_mom: Assignment project Advanced concepts programming DHBW CAS SoSe24 (github.com)](https://github.com/fl028/little_mom)

In dieser Arbeit wurde eine Message Oriented Middleware (MOM) sowie deren Clients (Publisher und Subscriber) entwickelt. Die Publisher (P) senden dem Message Broker (MB) Nachrichten zu dem jeweiligen Thema. Der MB verteile die Nachrichten an die Subscriber (S), welche sich zuvor bei dem MB zu deren Wunschthemen anmelden müssen. Der Systemaufbau ist in Abbildung 1 dargestellt. Alle Programmteile laufen auf einem System, jedoch auf unterschiedlichen Threads. Hierfür mussten Synchronisation-Mechanismen implementiert werden. Der Programmentwurf wurde in der Programmiersprache Java implementiert.

A diagram of a diagram

Description automatically generated

Abbildung 1: Struktureller Aufbau des Systems

**Angewendete Paradigmen**

In der Arbeit wurden folgende Programmierparadigmen verwendet:

* Konsistente Formatierung des Codes
* Sinnvolle Bezeichner. Strukturiertheit, Klarheit, Ausführbarkeit Lesbarkeit
* Keep it simple
* Verwendung der kennenglernte Konstrukte aus der Vorlesung
* Englisch

**Umfang**

Ziel und Umfang der Arbeit

eine ZIP-Datei pro Team mit folgenden Elementen

* Kurzdokumentation mit einer kurzen (max. 5-6 Seiten)
* ausführbare JAR-Datei.
* mit JavaDoc kommentierter Quellcode

1. Hauptteil

In dem Kapitel wird der Aufbau des Programmentwurfs kurz beschrieben.

**Aufbau**

Der strukturelle Aufbau des Programmes ist in Abbildung 2 dargestellt. Die auszuführende Datei ist „MOMSimulation.java“ (bzw. „MOMSimulation.jar“). Diese Klasse enthält die Main-Methode. In ihr werden die Objekte der weiteren Klassen sowie deren Threads erzeugt und verwendet. Zudem können hier über die konstanten „SIMULATION\_DURATION\_MS“ und „MAX\_MESSAGES“ variiert werden, um das Verhalten (Grund des Terminierens) zu ändern.

Der MB sowie die P und S besitzen jeweils eigene Klassen. Zudem gibt es ein Interface „Topic.java“, welches die Struktur der Themenklassen wie z.B. „FinanceTopic.java“ vorgibt. Alle Topic-Klassen implementieren dieses Interface. Zudem gibt es noch eine Klasse „Message.java“, diese wird versendet[[1]](#footnote-1). Sie enthält ein Topic-Objekt und ein String „content“, welcher die Nachricht enthält.

Ein Bild, das Text, Visitenkarte, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 2: Strukturelle Aufbau des Programmes

In Tabelle 1 wird eine kurze Übersicht über die Klassen MessageBroker, Publisher und Subscriber gegeben. Die Klassen implementieren die nötigen Funktionalitäten, um eine Kommunikationsstruktur nach Abbildung 1 zu erhalten. Es wird das „Push – Message Broker – Push“ Prinzip verwendet. Die Publisher ergreifen die initiative und pushen ihre Nachrichten zu dem Message Broker Objekt. Das Message Broker Objekt pusht die Nachrichten folgend an die entsprechenden Subscriber.

|  |  |  |
| --- | --- | --- |
| **MessageBroker** | **Publisher**  (implements Runnable) | **Subscriber**  (implements Runnable) |
| **Variablen**:   * Map<Topic, List<Subscribers>> subscribers * List<Message> messageQueue * Map<Topic, Integer> messageCountByTopic * boolean running * int maxMessages * int message | **Variablen**:   * int speed * MessageBroker broker * Topic topic | **Variablen:**   * String name * Map<Topic, Integer> messageCountByTopic |
| **Methoden:**   * synchronized void publish(Message) * synchronized void subscribe(Topic[], Subscribers) * void dispachMessage() * synchronized void stop() * synchronized int getMessageCount() * Map<Topic, Integer> getMessageCountByTopic() * List<Subscriber> getSubscribers() | **Methoden:**   * Static void SetSpeed(int) * void run() | **Methoden:**   * synchronized void receive (Message) * synchronized Map <Topic, Integer> getMessageCountByTopic * void run() |

Tabelle 1: Übersicht der Klassen

Die beiden Kommunikationspartner (Klasse Publisher und Subscriber) implementieren das Interface Runnable und überschreiben deren run Methode.

**Beschreibung der Main Methode in MOMSimulation.java**

MOMSimulation.java

* Variablen
* Initialisierung
* Starten der Threads
* Warten auf Abbruch (Zeit oder Anzahl)
* Stoppen der Threads
* Summary

**Beschreibung der Models**

Beschreibung des Topic Interfaces und der spezifischen Topics (Wetter,Sport,Finanz)

Beschreibung Message (Topic und Content)

A screenshot of a computer

Description automatically generated

**Beschreibung der Clients**

A screenshot of a computer

Description automatically generated

* Synchronized Methoden

**Beschreibung des Brokers**

MessageBroker.java

* Variablen (Fokus auf Map und List)
* Synchronized Methoden
* Wichtigste Funktion: dispatchMessages

**Beschreibung des Outputs**

<BILD/TEXT des Konsolen Outputs> + Beschreibung

1. Schluss

Beschreibung von Besonderheiten und Design Entscheidungen

* Entkoppelt Publisher und Subscriber (Separation of Concerns)
* einfache Implementierung von asynchroner Kommunikation (Thread Safety) -> synchronized
* Saubere Beendigung des Programms durch Start, Stoppen und Joinen der Threads
* Unterstützt Skalierbarkeit, da neue Publisher und Subscriber
* Fokus auf simpelste Datentypen (Nutzung von Collections Framework)
  + List<Message> messageQueue nicht LinkedBlockingQueue
  + Map<Topic, Integer> nicht ConcurrentHashMap

1. Es wird eine Klasse, welche das Interface Message implementier, anstatt beispielsweise ein String-Array versendet, um die Nachricht in einem „Container“ zu halten. [↑](#footnote-ref-1)