# Objektorientierte Komponenten-Architekturen

## Übungsblatt Nr. 2

Hochschule Bonn-Rhein-Sieg
Tim Wahrburg

Unter Prof. Dr. Sascha Alda

Date: May 4, 2024

# Informationen

Anforderungen 1 bis 9 wurden vollständig bearbeitet und die RTE liegt als ausführbare .jar-Datei vor, welche über das Commandline Interface gesteuert werden kann. Load-Balancing, Meta-File und ein separates Lib-Verzeichnis wurden ausgelassen. Der JDBC-Treiber wurde dem Classpath der RTE direkt hinzugefügt. Es wurde mit der in Aufgabe 1 erstellten Komponente und einer weiteren 'Greeting' Komponente (welche einen Nutzer freundlich grüßt) gearbeitet, welche beide bereitgestellt sind. Beide Komponenten printen bei Start, Stop, und in 10 Sekunden Updates zur Commandline. Aufgabenteil 3 ist in Section  und Aufgabenteil 5 in Section  zu finden. Aufgabenteil 4 wurde ebenfalls bearbeitet.

Die gesamte Aufgabe ist auch auf Git zu finden: `https://github.com/FavouredFool/JVM-RTE-Components.git`

# Zustandsmodell

Abbildung 1 zeigt eine Übersicht darüber, wie die Laufzeitumgebung auf die von dem CLI kommenden Befehle reagiert. Hierbei ist der Ablauf bedingt durch den `ComponentState` einer Komponente.

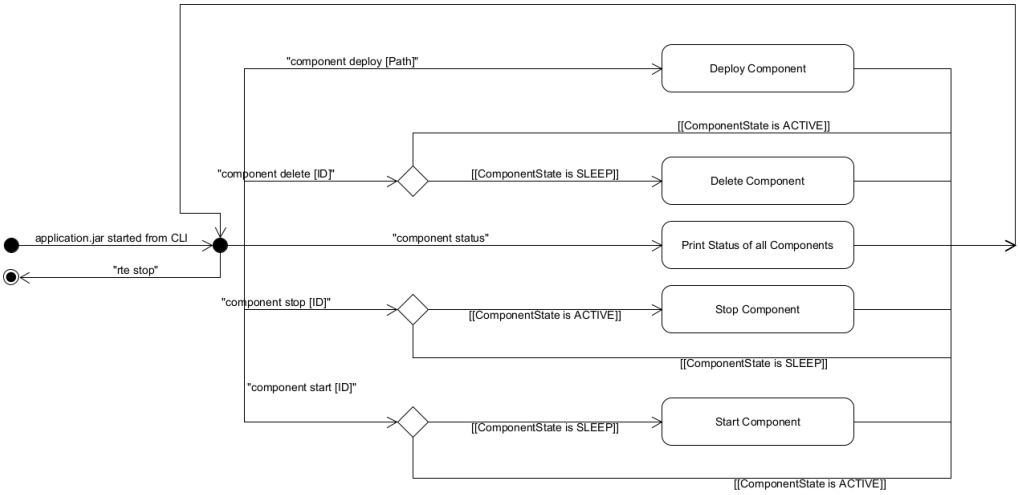


Figure 1: Übersicht über die Aktionen der Laufzeitumgebung

Abbildung 2 zeigt einen detailiierteren Einblick in die Abläufe von `Start Component` und `Stop Component` und wie die Zustände gesetzt werden. Die Zustände einer Komponente sind aus Zeitgründen sehr simpel ausgefallen. Eine Komponente hat ausschließlich den SLEEP und den ACTIVE Zustand. Der Zustand wird von

der RTE, nicht der Komponente selbst, manipuliert. Dies verhindert die Implementierung von Übergangszuständen wie "START", "STOP", oder "DISPOSE", welche für eine ausführlichere Komponentenimplementierung notwendig wären. Das Fehlen dieser Zustände zeigt in dem Zustandsdiagramm ein Problem auf: um die Komponente zu stoppen und dabei aus den Update-Loop zu `breaken`, muss eine separate `Stop-Flag` verwendet werden. Wenn die Komponente ihren State selbst steuern und betrachten würde, wäre eine solche Flag nicht von Nöten.

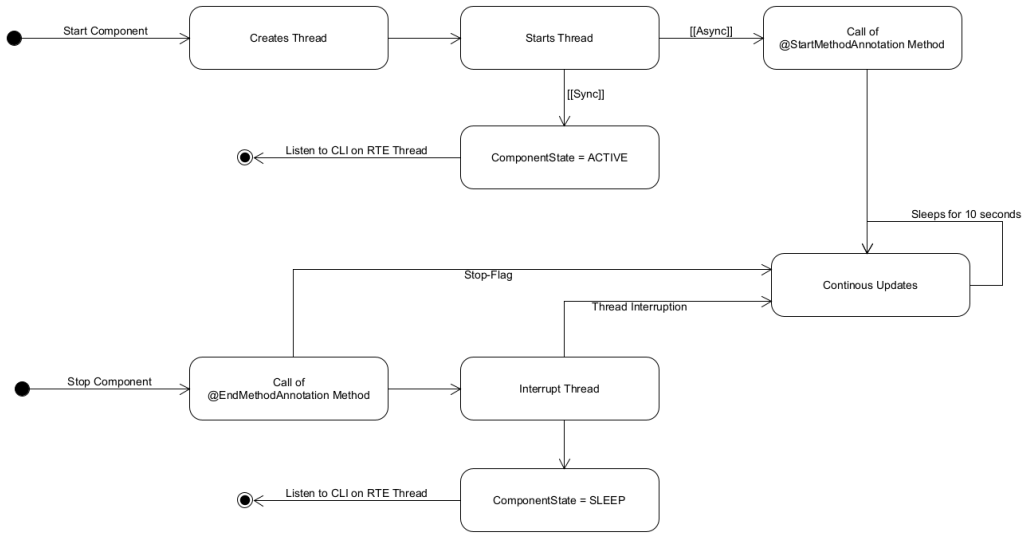


Figure 2: Komponentenzustände und Threading im Detail

Trotz dieser Schwächen bieten die simplen Zustände genug Informationen um sicherzustellen, dass eine bereits gestartete Komponente nicht erneut gestartet werden kann und eine noch laufende Komponente nicht gelöscht werden kann. Viel Potential für Verbesserungen ist jedoch vorhanden.

## Komponentenmodell nach Crnkovic

Im Folgenden wird dargestellt, welche der Lifecycle und Construction Kriterien nach dem Classification Framework von Iciva Crnkovic die implementierten Komponenten erfüllen.

### Lifecycle

Lifecycle beschriebt die Schritte die im Lebenszyklus einer Komponente durchlaufen werden. All diese Schritte durchlaufen auch die von mir erstellten Komponenten.

Initial wurden die Komponenten nach Spezifikationen (z.B. dass es eine Start- und eine Endmethode geben muss) modelliert [L.1], danach implementiert [L.2], dann zu einer .jar verpackt [L.3], um final in der Runtime-Environment in einem separaten Thread deployed und entlang der initial spezifizierten Methoden ausgeführt zu werden [L.4].

### Construction

Die Construction Kriterien beschreibt Mechanismen, die für das Zusammenspiel zwischen Komponente und dem restlichen System (Runtime Environment, andere Komponenten) von Relevanz sind.

Die von mir implementierten Komponenten bieten keine Interfaces an [C.1], weshalb kein Informationsaustausch zwischen Komponenten möglich ist. Da keine Interfaces angeboten werden, kann auch kein Binding [C.2] zwischen Komponenten bestehen und keine Interaction Styles [C.3] definiert werden.

### Erkenntnis

Die Komponenten können für sich allein stehend in eine Runtime-Environent deployed werden, da sie alle Lifecycle-Anforderungen erfüllen. Es ist jedoch deutlich, dass die Komponenten nicht in der Lage sind zu einem komplexeren System zusammen zu kommen, da keine der Construction-Anforderungen erfüllt sind. Um die Komponenten nutzbar zu machen, muss ein geregelter Informationsaustausch zwischen ihnen gewährleistet sein.