Live Service Composition

AUSARBEITUNG SoftwareArchitektur

Philipp Ewen

Jan Zipfler

2014

Inhalt

[Abstract 2](#_Toc382687072)

[Grundlagen 3](#_Toc382687073)

[Serviceorientierte Architektur 3](#_Toc382687074)

[Service 3](#_Toc382687075)

[Registry 3](#_Toc382687076)

[Service Composition 3](#_Toc382687077)

[Choreografie 3](#_Toc382687078)

[Orchestrierung 3](#_Toc382687079)

[Execution Engine 4](#_Toc382687080)

[Implementierung 5](#_Toc382687081)

[Programmiersprache 5](#_Toc382687082)

[Registry 5](#_Toc382687083)

[Random Servcie 5](#_Toc382687084)

[IsPrime Servcie 5](#_Toc382687085)

[Execution Engine/Concatenate Service 6](#_Toc382687086)

[Menü 6](#_Toc382687087)

# Abstract

Bei Service Composition handelt es sich um Verfahren in serviceorientierten Architekturen, bei dem neue Services aus den bestehenden Services erstellt werden. Dies kann zum Einen über Choreografie oder zum Anderen über Orchestrierung erfolgen. In dieser Arbeit werden wir anhand einer Beispielimplementierung zeigen, wie es mittels Orchestrierung möglich ist aus vorhandenen Services einen neuen Service zu erzeugen. Das Ganze geschieht zur Laufzeit, das heißt es ist möglich während das System läuft einen zusammengesetzten Service zu erzeugen. Um die Services zu verknüpfen wird dabei eine sogenannte Execution Engine eingesetzt. Diese werden wir beispielhaft für die Verknüpfungsart „verketten“ implementieren. Dabei handelt es sich um ein Verketten im Mathematischen Sinne f(g()). Zur Implementierung wird die Programmiersprache Go verwendet, diese eignet sich besonders zur Entwicklung von nebenläufigen und verteilten Anwendungen.

# Grundlagen

Dieser Abschnitt behandelt alle, für das Verständnis, notwendigen Grundlagen und Begriffserklärungen. Dabei werden wir insbesondere die Begriffe Service und Service Composition genau definieren. Außerdem wollen wir hier Techniken erläutern, die später für das Implementieren der Software notwendig sind. Dies sind im Wesentlichen Registry und Execution Engine.

## Serviceorientierte Architektur

Bei einem SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird. (https://de.wikipedia.org/wiki/Serviceorientierte\_Architektur)

## Service

Bei einem Service handelt es sich um eine autarke Einheit, die eine bestimmte Funktionalität zur Verfügung stellt. Diese Funktionalität wird dann im Netzwerk bereitgestellt. Damit ein Service benutzt werden kann, stellt dieser eine Schnittstelle nach außen bereit, welche im Service Contract definiert ist. Dieser Contract umfasst insbesondere: Name und Beschreibung des Servcies, Rückgabewert und Parameter mit Typ und Beschreibung.

## Registry

In einer serviceorientierten Architektur wird eine Registry wird zum Lokalisieren von Services verwendet. Services können sich entweder bei der Registry anmelden, um ihren Standort (z.B.: IP) mitzuteilen oder den Standort eines Servcies erfragen.

## Service Composition

Service Composition beschreibt das Erzeugen eines neuen Services aus bereits im Netzwerk vorhandenen Services. Um dies zu bewerkstelligen, gibt es zum einen den Ansatz der Choreografie und zum anderen den Ansatz der Orchestrierung. Der Vollständigkeit halber werden im Folgenden beide Varianten definiert. Wir werden in dieser Arbeit jedoch nur Orchestrierung behandeln.

## Choreografie

Hierbei wird kein neuer Service erzeugt, sondern es werden Regeln für die Zusammenarbeit bereits vorhandener Services definiert. Dabei trägt jeder Service einen bestimmten Teil zum gesamten Prozess bei. Außerdem kann jeder Service nur seinen eigenen Kontext sehen und nicht den gesamten Prozess. Ein Beispiel hierzu bilden nachrichtenbasierte Systeme.

## Orchestrierung

Orchestrierung ist das Zusammensetzen von mehreren Servcies zu einem neuen Servcie, welcher die zentrale Steuerung es des neuen Prozesses übernimmt.

## Execution Engine

Die Execution Engine dient dem Ausführen eines zusammengesetzten Servcies (Servcie Composition). Dazu benötigt die Execution Eingine eine Definition der Service Composition. Hierfür verwenden wir in dieser Arbeit die Namen der Servcies, welche in der Form *f(g())* verkettet werden sollen.

# Implementierung

In diesem Kapitel werden wir erläutern, welche Methoden und Werkzeuge wir zum Implementieren unserer Live Service Composition verwendet haben. Dazu zählen vor allem detaillierte Informationen über die Kommunikation zwischen den einzelnen Servcies und der Registry.

## Programmiersprache

Als Programmiersprache haben wir Go verwendet. Go wurde von Google entwickelt und entstand 2007 im Rahmen eines 20% Projekts. Zu den stärken Go zählen unter anderem Netzwerk- und nebenläufige Programmierung. Außerdem ist Go Plattformübergreifend nutzbar, d.h. in Programme können u.a. unter Windows und Linux kompiliert werden.

## Registry

Zu den Aufgaben der Registry in unserem Projekt gehören:

* Das zuordnen vom Namen eines Servcies zu dessen IP-Adresse und Servcie Contract.
* Verwalten einer Liste mit allen vorhandenen Servcies.
* Eigene IP-Adresse über Multicast bekannt geben, damit die Registry ohne Konfiguration von anderen Netzwerkteilnehmern gefunden werden kann (Multicast discovery).

Aus der Verwendung von Multicast folgt insbesondere, dass für die Multicast discovery das UDP Protokoll zum Einsatz kommt. Da mit UDP keine gesicherte Verbindung möglich ist kommt sonst immer TCP zum Einsatz. Außerdem verwenden wir für das Übertragen von Daten in allen Fällen das JSON (JavaScript Object Notation), welches von Go (ohne zusätzliche Bibliotheken) unterstützt wird.

## Random Servcie

Dieser Service generiert eine Zufallszahl im Bereich von 0 bis 2^31-1.

Service Contract:

|  |  |
| --- | --- |
| Service Name | random |
| Beschreibung | Generates a random integer. |
| Parameter | Keine |
| Returns | int |

## IsPrime Servcie

Dieser Service testet, ob die übergebene natürliche Zahl eine Primzahl ist. Hierzu wird ein 16-Facher Miller-Rabin-Test durchgerührt. Dabei handelt es ich hierbei um einen probabilistischen Test, d.h. das Ergebnis gilt mit einer bestimmten Wahrscheinlichkeit (in diesem Fall: 1 - 1/4^16 = 99.999999977%).

Service Contract:

|  |  |
| --- | --- |
| Service Name | isprime |
| Beschreibung | Test if x is prime. (Ausgabe als Text) |
| Parameter | x, int, number to test |
| Returns | string |

## Execution Engine/Concatenate Service

Die Execution Engine führt zwei Services hintereinander aus. Dieses Hintereinander Schalten entspricht der mathematischen Operation verketten, und zwar im Sinne von ***newservice := service1(service2()).***

Service Contract:

|  |  |
| --- | --- |
| Service Name | concatenate |
| Beschreibung | Passes output of service2 to service1 (service1(service2()) and creates a new service. |
| Parameter 1 | service1, string, first service name |
| Parameter 2 | service2, string, second service name |
| Parameter 3 | newservice, string, new service name |
| Returns | string |

## Menü

Das Menü stellt die Schnittstelle zwischen Mensch und unsrer Implementierung einer serviceorientierten Architektur dar. Dies haben wir durch eine einfache Kommandozeilenschnittstelle realisiert. Die folgenden Operationen stehen dem Benutzer zur Auswahl.

|  |  |
| --- | --- |
| Liste anzeigen | Zeigt eine Liste aller Services an (jeweils mit IP und Service Contract). |
| Service Info anzeigen | Zeigt Informationen für den angegebenen Service an (IP und Service Contract) |
| Service aufrufen | Führt den angegebenen Service aus. Die Parameter, welche vom Benutzer eingegeben werden müssen, werden dynamisch aus dem Service Contract generiert. |

**TODO: Sequence Charts für alle möglichen Aufrufe.**

**TODO: Overviw Chart für das System**

**TODO: Screenshots von dem laufenden System (eventuell mit erklärung)**