Live Service Composition

AUSARBEITUNG SoftwareArchitektur

Philipp Ewen

Jan Zipfler

2014

Inhalt

[Abstract 2](#_Toc382733600)

[Grundlagen 3](#_Toc382733601)

[Serviceorientierte Architektur 3](#_Toc382733602)

[Registry 3](#_Toc382733603)

[Service 3](#_Toc382733604)

[Service Composition 3](#_Toc382733605)

[Choreografie 3](#_Toc382733606)

[Orchestrierung 4](#_Toc382733607)

[Execution Engine 4](#_Toc382733608)

[Implementierung 5](#_Toc382733609)

[Programmiersprache 5](#_Toc382733610)

[Registry 5](#_Toc382733611)

[Random Servcie 6](#_Toc382733612)

[IsPrime Servcie 6](#_Toc382733613)

[Execution Engine/Concatenate Service 7](#_Toc382733614)

[Menü 7](#_Toc382733615)

[Programmablauf mit Bildschirmfotos 9](#_Toc382733616)

# Abstract

Service Composition beschreibt ein Verfahren in serviceorientierten Architekturen (SOA), bei dem neue Services aus bereits bestehenden Services erzeugt werden. Dies kann zum einen über Choreografie und 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 erstellen. Dieser Erzeugungsprozess findet zur Laufzeit statt, das heißt es ist möglich, während das System läuft einen neuen, zusammengesetzten, Service zu anzulegen. Um mehrere Services miteinander zu verknüpfen setzen wir eine sogenannte Execution Engine ein, die wir beispielhaft für die Verknüpfungsart „verketten“ implementieren. Bei dieser Verknüpfungsart handelt es sich um ein Verketten von zwei Funktionen im mathematischen Sinne, wie zum Beispiel: . Zur Implementierung benutzen wir die Programmiersprache Go, welche sich besonders gut zur Entwicklung von nebenläufigen und verteilten Anwendungen eignet.

# 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 eine Serviceorientierten Architektur (SOA) handelt es sich laut Wikipedia um folgendes: „SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität, die von unterschiedlichen Besitzern verantwortet wird.“ (Wikipedia, 2014)[[1]](#footnote-1)

## 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

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.

## 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. Die später beschriebene Implementierung, wird sich jedoch nur mit dem Thema Orchestrierung beschäftigen.

### Choreografie

Bei Choreographie handelt es sich laut Ingo Melzer um folgendes:

„Eine Choreographie beschreibt die Aufgaben und das Zusammenspiel mehrerer Prozesse unter dem Aspekt der Zusammenarbeit.“ [[2]](#footnote-2)

Er beschreibt weiterhin, dass Prozesse miteinander kooperieren und diese Kooperation in dazugehörigen Choreographie-Dokumenten beschrieben wird. Dazu wird kein neuer Service erzeugt, sondern die erwähnten Regeln für die Zusammenarbeit angewandt. Jeder Prozess trägt somit 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

Bei Orchestrierung handelt es sich laut Ingo Melzer um folgendes:

„Eine Orchestrierung beschreibt die ausführbaren Aspekte eines Geschäftsprozesses aus der Sicht des Prozesses.“ [[3]](#footnote-3)

Er beschreibt weiterhin:

„Orchestrierung bezeichnet also die klassische Workflow-Sicht, bei der es eine dedizierte Instanz gibt (Prozess), die alle Aktivitäten steuert. Es wird beschrieben, aufgrund welcher Bedingungen (Geschäftslogik) der Prozess den Fluss der einzelnen Web-Services-Aufrufe steuert. Auch wird beschrieben, wie die verschiedenen Web Services mittels Nachrichtenaustausch miteinander kommunizieren […]. Man erhält ein Protokoll, welches festlegt, in welcher Reihenfolge und mit welchen Aufrufparametern die einzelnen Aktivitäten auszuführen sind.

Es ist jedoch nicht immer erwünscht, dass die genauen Interna eines Prozesses von außen sichtbar sind. Beispielsweise kann das Innenleben eines Prozesses das Ergebnis einer intensiven Forschungsarbeit sein. Die Geheimhaltung der Prozessinterna könnte somit einen Wettbewerbsvorteil darstellen. Oder aber man möchte die zur Kommunikation mit einem Partner notwendigen öffentlichen Aktivitäten eines Prozesses von anderen Aktivitäten separieren und in einen öffentlich einsehbaren Prozess abtrennen. Mit dieser Modularisierung von Prozessen wird eine größere Flexibilität bei der Änderung der nichtöffentlichen Prozesse erreicht, denn nun ist es möglich, private Aspekte zu ändern, ohne dass sich dadurch für die Geschäftspartner etwas ändert.

Da der Schwerpunkt in diesem Fall auf der Kommunikation mit anderen liegt, enthalten öffentliche Prozesse keine ausführbare Geschäftslogik. Es wird lediglich die Sequenz der Nachrichten beschrieben, die zwischen den Prozessen ausgetauscht werden. Somit erhält man also kein ausführbares Protokoll wie bei der Orchestrierung, sondern ein beschreibendes Protokoll.“ ³

ORIGINAL:

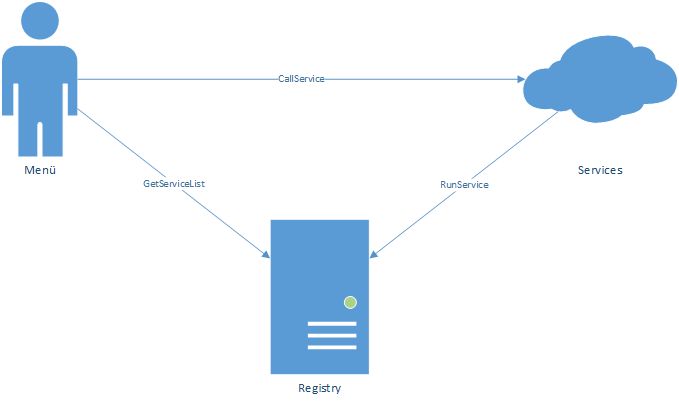
Orchestrierung ist das Zusammensetzen von mehreren Servcies zu einem neuen Servcie, welcher die zentrale Steuerung es des neuen Prozesses übernimmt.

## 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 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. Die folgende Übersicht illustriert die Struktur unserer Anwendung und vermittelt einen Überblick:



## 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. die 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 JSON (JavaScript Object Notation), welches von Go (ohne zusätzliche Bibliotheken) unterstützt wird.

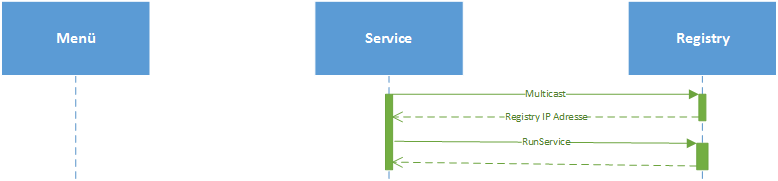
## Random Servcie

Dieser Service generiert eine Zufallszahl im Bereich von 0 bis .

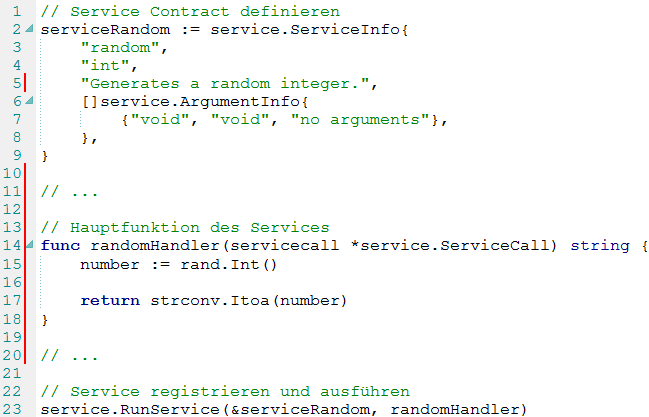
Service Contract:

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

Sequenzdiagramm: Service registrieren und aufführen



Definition des Random Services:



## 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: ).

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:

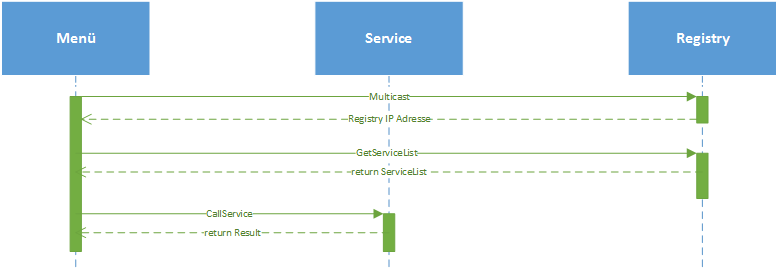
|  |  |
| --- | --- |
| Service Name | concatenate |
| Beschreibung | Passes output of service2 to service1 (service1(service2()) and creates a new service. |
| Parameter 1 | service1, string, first service name |
| Parameter 2 | service2, string, second service name |
| Parameter 3 | newservice, string,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

Sequenzdiagramm: Service lokalisieren und aurfuren



# Programmablauf mit Bildschirmfotos

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

Ich habe mal einige Bilder gemacht. Müssen uns dann später überlegen ob wir wirklich jedes Bild drin lassen oder nur die „wichtigsten“ zeigen und erläutern.

In diesem Abschnitt soll erläutert werden, wie die Implementierung der Service Komposition verwendet werden kann. Dazu werden einige Bildschirmfotos bereitgestellt, die die einzelnen Schritte beschreiben.

Um diese Implementierung verwenden zu können, werden mindestens 3 Programmteile benötigt:

1. Registry Server
2. Service(s)
3. Menü / Benutzeroberfläche

Da es sich um eine Service Komposition handeln soll, werden die drei Services verwendet, die bereits im Abschnitt „Implementierung“ erläutert wurden:

* Random Service
* Primzahltest Service
* Kompositionsservice

Nachdem die Registry, alle Services gestartet wurden, kann das Menü aufgerufen werden und die, im Abschnitt Implementierung erwähnten, Unterpunkte aufgerufen werden.

In Abbildung 1 ist die Ausgabe der Serviceliste zu sehen, die die detaillierten Informationen zu jedem Service anzeigt.

Die durchgeführte Aktion, kann ebenso auf dem Server, auf dem die Registry läuft, nachvollzogen werden. Ein Beispiel für eine solche Protokollierung ist in Abbildung 2 zu sehen.



Abbildung Auflistung aller Service-Contracts

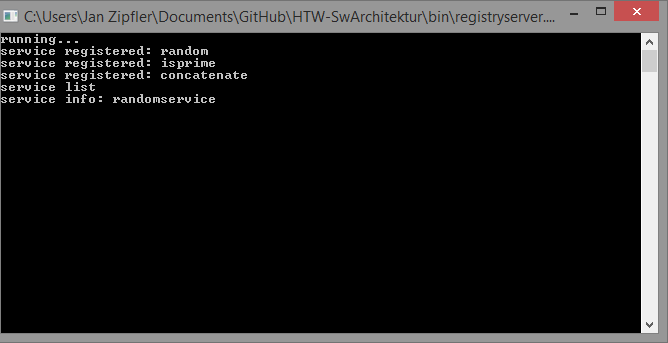


Abbildung Registryserver protokoliert Aktionen

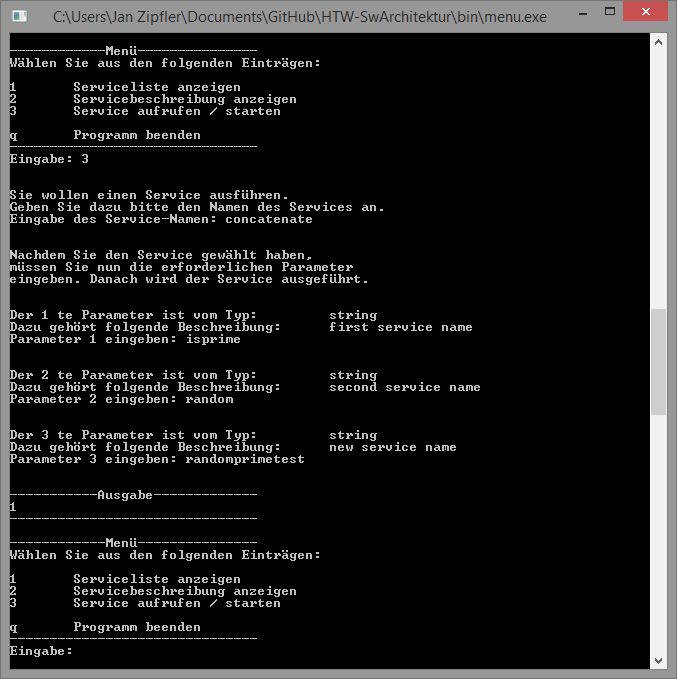


Abbildung Erstellung eines neuen Services

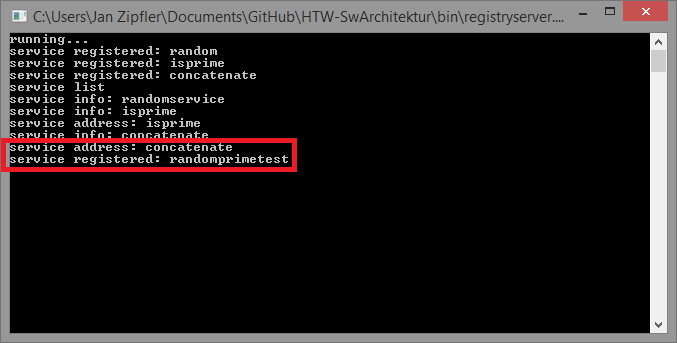


Abbildung Protokollierung, dass neuer Service angelegt wurde

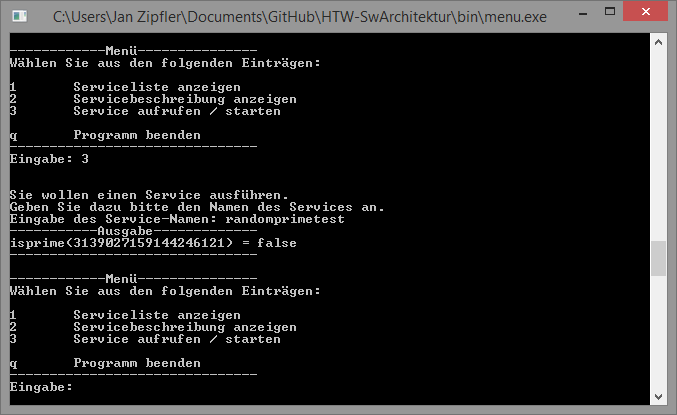


Abbildung Ausführung des zusammengesetzten Service

1. [https://de.wikipedia.org/wiki/Serviceorientierte\_Architektur (16](https://de.wikipedia.org/wiki/Serviceorientierte_Architektur%20(16). März 2014) [↑](#footnote-ref-1)
2. Ingo Melzer – Service-orientierte Architekturen mit Web Services [↑](#footnote-ref-2)
3. Ingo Melzer – Service-orientierte Architekturen mit Web Services [↑](#footnote-ref-3)