ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Seminararbeit

Service orientierte Architekturen

vorgelegt von

Fatoumata Diarrassouba

Meris Krupic

Jens Leiner

betreut und begutachtet von

Prof. Dr. Markus Esch

Saarbrücken, 29. 03. 2020

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 29. 03. 2020

Fatoumata Diarrassouba

Meris Krupic

Jens Leiner

Zusammenfassung

Service Oriented Architecture (SOA) bietet die Möglichkeit, eine monolithische Software in einzelne leichter wartbare und erweiterbare Teile aufzuteilen.

In dieser Arbeit wird versucht eine möglichst allgemeine Definition der Bestandteile und Modelierung zu finden und diese in einer Fallstudie anzuwenden. Außerdem wird auch noch darauf eingegangen, wie sich SOA von einem anderen ähnlichen Ansatz(Microservices) unterscheidet.

Inhaltsverzeichnis

1 Definitionen

1.1 SOA

Service-orientierte Architektur ist ein Architekturstil für Geschäftsanwendungen, der in den 90er Jahren entstanden ist. 1996 wurden die ersten Reports über SOA von Roy W. Schulte und Yefim V. Natis -zwei Analytikern der Firma Garner inc- veröffentlicht. Doch schon zwei Jahre zuvor hatte Alexandre Pasik den Begriff SOA geprägt und seine Grundprinzipien entwickelt.[7, S. 8]
Trotz ihrer Existenz und Verwendung seit mehreren Jahren, bleibt SOA ein abstraktes Konzept mit keiner exakten Definition. Es gibt mehreren Definitionen, die von einem Autor zum anderen variieren. Das lässt sich dadurch erklären, dass SOA keine konkrete Architektur ist, sondern ein Ansatz, Software-Architekturen zu entwerfen. Es gibt keine harten Kriterien, um SOA exakt zu beschreiben. [7, S. 15] Laut Josuttis ist SOA kein konkretes Framework, sondern ein Paradigma, ein Denkmuster, eine Philosophie, ein Wertesystem, das dazu führt, dass bestimmte konkrete Entscheidungen beim Entwurf vom Aufbau einer Anwendung getroffen werden.
Daraus folgt, dass SOA kein kaufbares Werkzeug ist, das als universelle Lösung für jede Software eingesetzt werden kann. Bei jedem System muss eine Analyse der Geschäftsprozesse durchgeführt werden, um den geeigneten SOA-Ansatz zu definieren.
Die Open Group in ihrem Referenzmodell definiert SOA folgendermaßen: „Service oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development".[21] Dabei ist zu verstehen, dass SOA ein auf Service-Orientierung basierender Architekturstil ist. Service-Orientierung ist der Ansatz, die Funktionalitäten eines Systems, einer Anwendung als Services zu implementieren. In Abschnitt 1.2 wird der Begriff „Service" explizit beschrieben.
Trotz fehlender einheitlicher Definition gibt es Basis-Eigenschaften, die das SOA-Konzept charakterisieren. Diese werden später in dieser Arbeit erörtert. Darüber hinaus gibt es Referenzmodelle, wie das Model von der oben genannten Open Group oder das von der Organization for the Advancement of Structured Information Standards (OASIS), deren Ziel ist, das Konzept zu standardisieren.

1.2 Service

Services stellen die Grundlage einer SOA. Der Begriff „Service" ist das englische Wort für „Dienstleistung", die Durchführung einer Aufgabe durch jemanden für einen anderen. In SOA wird diese Definition durch weitere Konzepte ergänzt. In einer SOA stellt ein Service eine fachliche Funktionalität, eine Funktion des Systems dar. Beispielsweise im Verwaltungssystem von Kundenbestellungen eines Händlers können Abläufe, wie die Erfassung der Bestellung, die Verfügbarkeitsprüfung von Artikeln oder der Zahlungsprozess als Services implementiert werden. Zusammen kombiniert bilden Services die Geschäftsprozesse.
Im SOA-Umfeld stehen Services zueinander in einer Producer-Konsument Beziehung. Sie interagieren miteinander, um die Ergebnisse von Prozessen zu erstellen. Die Interaktion

erfolgt durch definierte Schnittstellen. Allerdings ist ein Service eine Black-Box für die anderen Services. Seine interne Struktur bleibt den Konsumenten unbekannt. Damit die Interaktion überhaupt erfolgen kann, müssen die Services sichtbar und auffindbar sein. Im kommenden Unterkapitel wird auf dieses Thema eingegangen.
Darüber hinaus sollte ein Service eine in sich abgeschlossene Komponente sein. Ein Service soll möglichst unabhängig seine Aufgabe erledigen können soll. Um seine Aufgabe durchzuführen, fordert ein Service Daten in einem definierten Format und liefert das Ergebnis in einem bestimmten Datenformat. Der Zahlungsprozess im vorherigen Beispiel könnte als Input-Datenformat Bankverbindungen, wie z.B. IBAN oder Kreditkartenummer und PIN erwarten und als Output eine Rechnung erstellen.
Eine weitere Eigenschaft von Services ist die Wiederverwendbarkeit. Im SOA-Konzept -wie im Normalfall in der Softwareentwicklung- wird versucht, Redundanzen zu vermeiden. Ein Service, der einmal implementiert wurde, soll mehrmals verwendet werden können. Dieses Ziel ist in der Praxis wegen Performanz-Problemen nicht immer erreichbar. Beispielerweise kann die Antwortzeit eines Services lang sein, weil viele Daten verarbeitet werden müssen. Braucht der Aufrufer nur einen kleinen Teil dieser Daten, lohnt sich diese Zeit nicht.[7, S. 42, 202]
Zuletzt kommt die Zustandslosigkeit. Services werden in zwei Gruppen kategorisiert, „statefull“ und „stateless“ Services, ins Deutschen übersetzt zustandsbehafte und zustandslose Services. Ein „Statefull“ Service hat Zustände, in die er zur Laufzeit eintritt. Bei einem stateless Service werden zwischen verschiedenen Service-Aufrufen Zustände aufrecht gehalten werden.[7, S. 238] Ein konkretes Beispiel von zustandsbehafteten Service ist eine herkömmliche Warenkorbanwendung bei einem Webshop. Beim Auflegen der ersten Artikel in den Warenkorb, gerät dieser in einen ersten Zustand. Dieser Zustand wird beibehalten, bis ein zweiter Artikel hinzugefügt wird. Und so weiter wird sich der Zustand des Warenkorbs ändern, bis der Kunde mit der Bestellung fertig ist. Somit ist das gelieferte Ergebnis von allen Zwischenzuständen abhängig. [7, S. 238]
Ein zustandsloser Service hingegen ist von keinem Zustand abhängig. Er hält zwischen den konsekutiven Aufrufen keine Zustände aufrecht. Nach dem Aufruf so eines Services werden die für die Durchführung der Aufgabe temporär erzeugten Variablen und Objekte sofort gelöscht. Gelöscht werden nur die Daten, die als Hilfsmittel intern von dem Service selbst verwendet werden. Das Aufräumen der Daten betrifft weder den Aufrufer noch die Frontend-Applikation oder die Datenbank. Somit wird für jeden Service-Aufruf eine neue Service-Instanz verwendet. Das kann ein Thread oder ein Prozess sein. Hier kann als Beispiel ein Service zur Geldeinzahlung auf Bankkonten vollzogen werden. Bei zwei hintereinander Einzahlungen können zwei Threads initiiert werden. Die Daten im Backend werden somit mit jedem Service-Aufruf geändert.[7, S. 235]
SOA bevorzugt zustandslose Services, da sie bezüglich Load-Balancing und Ausfallsicherheit einfacher zu verwalten sind. Das System wird so aufgebaut, dass bei jedem Service-Aufruf auf die bestmögliche Service-Instanz zugegriffen werden kann. Dabei kann ein gleicher Thread für die erste Durchführung eines Aufrufs und dann für die zweite genutzt werden. Da der Service von keinem Zustand abhängig ist, kann er sofort verwendet werden. Die einzelnen Instanzen des Services sind voneinander unabhängig. Beim Ausfallen einer Service-Instanz, springt eine andere ein, und übernimmt die Bearbeitung des Aufrufs, ohne dass der Aufrufer es bemerkt. Stateless Services haben auch den Vorteil, dass die Anzahl der bereitgestellten Instanzen einfach erhöht werden kann.[7, S. 241] Zustandsbehaftete Services können nicht alle oben genannte Vorteile bieten, denn sie sind meistens an Sessions(Benutzersitzungen) gebunden.

1.3 Bestandteile

Neben Services kann eine Service-orientierte Architektur aus folgenden weiteren Bausteinen bestehen: Anwendungs-Frontend, Service Registry bzw. Repository, Service Bus.[8, S. 76]

1.3.1 Anwendungs-Frontend

Die Benutzeroberfläche eines auf SOA basierenden Softwaresystems unterscheidet sich nicht von der einer beliebigen Anwendung. In SOA können User-Interfaces als Webanwendung oder als Rich-Client implementiert werden.[8, S. 78] Jedoch werden bei vielen SOA-Herstellern Portale als User-Interfaces vorgesehen. Gesprochen wird von SOA-Enterprise-Portal. In einer Veröffentlichung der Firma Oracle heißt es: „Portal ist the face of SOA".[20, S. 14]
Unternehmensportale sind heute integrale Bestandteile von Business Prozess Plattformen (BPP). Portale unterstützen die Integration von Applikationen in einer gemeinsamen Oberfläche. Über das Portal sind alle Systeme integriert und Prozesse abgebildet. Dadurch bekommt der Benutzer eine zentrale Sicht über Anwendungen und Prozesse. Portale ermöglichen die Verknüpfung und den Datenaustausch zwischen heterogenen Anwendungen, was in einer SOA sehr wichtig ist.
Das Anwendungs-Frontend muss aber nicht zwingend ein grafisches User-Interface sein. Falls die Anwendung nicht mit Endbenutzern interagieren können muss, kann das User-Interface auch als Batch-Programm realisiert werden.[8, S. 78]

1.3.2 Enterprise-Service Bus

Ein Enterprise-Service-Bus (ESB) ist eine Integrationsmiddleware, deren Funktion ist, den Aufruf von Services durch Benutzer möglichst schnell zu ermöglichen. Dazu verbindet der ESB die SOA-Teilnehmer miteinander, wie in Abbildung 1 dargestellt. Es werden Services mit Anwendungs-Frontend sowie Services mit Services verbunden. Um dieses Ziel zu erreichen, erfüllt ein ESB unterschiedliche Aufgaben. Dazu gehören Datentransformation, Routing und Umgang mit Sicherheitspaketen.[7, S. 63]
Benötigt ein Service Informationen, welche von einem anderen Service bereitgestellt werden, so ruft er diesen über den ESB auf. Der Anruf wird von dem ESB weitergeleitet und die Konnektivität erstellt.
Der ESB unterstützt über Konnektoren vielfältige Transportprotokolle. Er ermöglicht den Datenaustausch zwischen Services, die oft mit unterschiedlichen Protokollen kommunizieren. Beispielerweise kann ein ESB einen Service, der mit http kommuniziert mit einem anderen verbinden, der mit Message-Queuing kommuniziert.[5] Bei so einem Datenaustausch muss der ESB nicht nur die Verbindung herstellen und Daten weiterleiten, sondern auch umwandeln.

1.3.3 Registry bzw. Repository

Service Repository und Registry sind Verzeichnisse zur Speicherung von Metainformationen zu Services. In einem SOA-Umfeld erfüllen sie die Aufgaben von Zentralregister und Dienstvermittler.[7, S. 287] Beide Begriffe werden häufig als gleichbedeutend angesehen, doch besteht zwischen ihnen ein feiner Unterschied.
Service-Repositories dienen zur fachlichen Verwaltung der Services. Sie stellen alle Informationen bereit, die zur Lokalisierung, Erkennung und Verwendung von Services er-

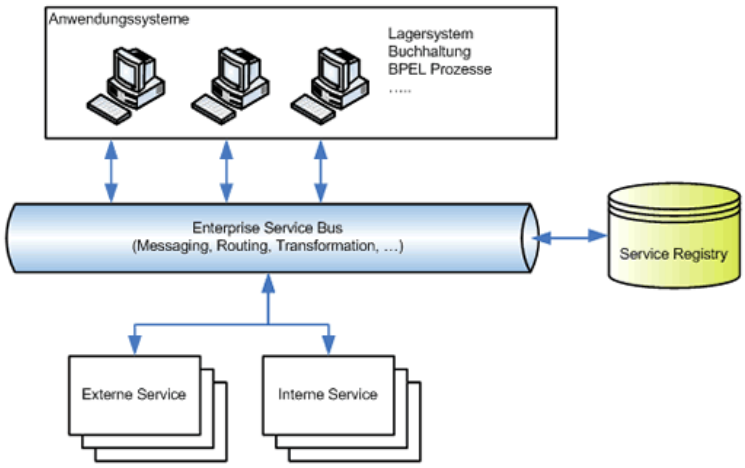

Abbildung 1.1: Darstellung von der Rolle des ESBs [6]

forderlich sind. Diese sind unter anderem Service-Beschreibung, Anbieter, Schnittstellen, Zugriffsrechte, Abhängigkeiten und Informationen über die vorgesehene Leistung des Services, die in SLA-Vorlagen (Service Level Agreement) enthalten sind. Zusammengefasst werden in Repositories alle Daten gespeichert, die aus fachlicher Sicht relevant sind. Während ein Repository die fachliche Organisation der Services übernimmt, repräsentiert die Registry die technische Sicht. Da werden die zum Betrieb des Services erforderliche Informationen abgelegt. Es handelt sich um Laufzeitinformationen, wie IP-Adressen sowie Ports. Eine Service-Registry kann eine weitere Aufgabe erfüllen. Sie kann zur Weiterleitung von Service-Aufrufen verwendet werden. Somit wird die Registry zum Teil des ESB. [7, S. 287]
Zwar sind Repositories, Registries und ESBs sehr praktische Komponenten in einer SOA-Umgebung, sie sind aber keine unabdingbaren Elemente. SOA-Systeme können auch ohne sie aufgebaut werden. Es handelt sich um proprietäre Software, die große Kosten mit sich bringen. Daher sollte beim Entwurf der Architektur überlegt werden, ob ihr Einsatz nötig ist. In großen SOA-Landschaften mit mehreren Services sind Repositories und Registries von Nutzen; oft werden die gleichen Services in verschiedenen Projekten oder Prozessen verwendet werden. In dem Sinn, ist der Einsatz eines zentralen Orts, an dem potenzielle Dienstnutzer nach Services suchen können, fast unverzichtbar.
Dasselbe gilt für den ESB. Es wird in großen SOAs empfohlen, die Services nicht direkt miteinander zu verknüpfen, sondern über einen ESB, um die Verwaltung zu vereinfachen.[17, S. 74]

2 Eigenschaften

Zusammen mit dem Business driven Development(BDD), der Vendor Independence und der hohen Interoperabilität bildet die lose Kopplung die Basiskonzepte von SOA.

2.1 Hohe Interoperabilität

In einer mittels SOA realisierten Anwendung, werden die Systemfunktionalitäten um logische Funktionen gruppiert und als Services implementiert. Somit wird das System in Services aufgeteilt, die als Komponenten getrennt voneinander entwickelt werden. Sie müssen miteinander sprachfähig sein und zusammen arbeiten können. Eine Kommunikation benötigt die Verbindung der Partner. Diese sollte möglichst schnell und einfach hergestellt werden. Die Hohe Interoperabilität bedeutet, dass die SOA-Teilnehmer schnell und einfach Informationen austauschen können. Genau dazu tragen ESBs bei. Sie kümmern sich um die Konnektivität und die Interoperabilität.

2.2 Lose Kopplung

Lose Kopplung ist ein Konzept zur Erreichung von Flexibilität, Fehlertoleranz und Skalierbarkeit in einem System. Das Konzept besteht darin, innerhalb einer SOA Abhängigkeiten zwischen den Komponenten zu minimieren.
Bestehen in einem System wenige Abhängigkeiten zwischen seinen Komponenten, haben Modifizierungen innerhalb von einzelnen Komponenten wenige Auswirkungen auf die anderen. Ein solches System wird als fehlertolerant bezeichnet. Je fehlertoleranter ein System ist, desto flexibler ist es.[7, S. 22] Mit der Flexibilität kann ein System leicht angepasst werden, was das System skalierbar macht, wenn es wächst. Es können neue Funktionen hinzugefügt werden, ohne die vorhandenen Funktionalitäten zu beeinträchtigen, weil nicht alles zentralisiert ist.
Wie SOA ist lose Kopplung wieder kein Werkzeug, sondern ein Prinzip, das subjektiv ist. Es gibt kein vordefiniertes Maß an loser Kopplung, das überall gültig ist. Bei dem Entwurf eines Systems auf Basis von SOA, liegt es an den Entwicklern, zu überlegen, wie lose Kopplung eingeführt wird. In seinen Ausführungen nennt Josuttis typische Bereiche für die Einsetzung von loser Kopplung in einer SOA. Einige davon sind in der Tabelle 1 dargestellt. s

2.3 BDD

Wie schon in der Definition erwähnt, kann SOA nicht als fertig einsatzbares System gekauft werden. Bei jeder Anwendung ist eine eigene SOA-Denkweise erforderlich, denn bei SOA wird die zu entwickelnde Software an den Geschäftsprozessen des Unternehmens ausgerichtet. [3, S. 12]
Services werden definiert, nachdem Geschäftsprozesse gestaltet wurden. Services werden auf Aktivitäten der Geschäftsprozesse abgebildet.

Bereich	Enge Kopplung	Lose Kopplung
Physikalische Verbindung	Punkt zu Punkt	über Vermittler
Kommunikationsstyl	synchron	asynchron
Plattform	starke Abhängigkeit	plattformunabhängig
Deployment	gleichzeitig	zu verschieden Zeitpunkten
Versionierung	explizite Upgrade	implizite Upgrade

Tabelle 2.1: Tabelle 1: Formen von Loser Kopplung

2.4 Vendor Independance

Das SOA-Architekturmodell ist nicht von einer Anbieterplattform abhängig. SOA bietet die Möglichkeit Systeme mit unterschiedlichen Anbietertechnologien zusammen zu bringen. Im OASIS-SOA- Referenzmodell heißt es, dass Systeme in einer SOA von unterschiedlichen Eigentümern kontrolliert werden können. [7, S. 24] Auf dieses Thema wird in Kapitel 5 tiefer eingegangen.

3 Einsatzbereiche

Einsatzbereiche für SOA sind beispielsweise verteilte Systeme, Webservices und Anwendungsintegration im Rahmen der Enterprise Application Integration (EAI).

3.1 Verteilte Systeme

Nach der SOA-Definition des OASIS-SOA-Referenzmodells ist SOA ein Konzept, das für Verteilte Systeme gedacht ist.
„SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations“[11]
SOA ermöglicht Entitäten, die sich in verschiedenen Systemen befinden, sich untereinander zu erkennen, miteinander kommunizieren und arbeiten zu können. Diese Systeme können von unterschiedlichen Eigentümern mit unterschiedlichen Technologien verwaltet werden. SOA bietet den Vorteil mit Heterogenität umgehen zu können, da Aufrufschnittstellen von der internen Implementierung getrennt sind. Weitere SOA-Eigenschaften, die für verteilte Systeme sehr wichtig sind, sind Flexibilität und Skalierbarkeit. Systeme wachsen immer wieder und werden immer komplexer, wenn das Geschäft wächst.

3.2 Web-Services

Für viele gelten Web-Services und SOA als äquivalent, was aber nicht der Fall ist. Web-Services sind ein Weg, die SOA-Konzepte umzusetzen. Es ist die Möglichkeit, die die meisten Hersteller und Autoren zur Realisierung von SOA vorsehen.[7, S. 259]
Web-Services haben das Konzept von Service-Orientierung übernommen. Web-Services sind eigenständige, modulare und verteilte Anwendungen, die über ein Netzwerk veröffentlicht und aufgerufen werden. Einer der Vorteile von Web-Services ist die Interoperabilität, was ein sehr wichtiges Prinzip von SOA ist. Diese Interoperabilität wird durch eine Reihe von Standards, wie Web Service Definition Language (WSDL), Simple Object Access Protocol (SOAP) oder Representational State Transfer (REST) und Universal Description, Discovery, and Integration (UDDI) erreicht. Diese Standards bieten einen gemeinsamen Ansatz für die Definition, Veröffentlichung und Verwendung von Webdiensten.

3.3 EAI

Bei EAI geht es darum, Geschäftsanwendungen im Unternehmen entlang der Wertschöpfungskette zu integrieren. Da SOA ein Konzept ist, das sich an Geschäftsprozessen orientiert, stellt es eine gute Lösung dar. Zudem gehört Heterogenität zu den SOA-Prinzipien, was für EAI sehr wichtig ist, denn Geschäftsanwendungen können auf unterschiedlichen Plattformen verteilt sein. Interoperabilität spielt auch dabei eine große Rolle.

4 Modellierung

Wie wir in den vorherigen Kapiteln festgestellt haben, gibt es bei SOA keine feste Definition, der die Architektur zu folgen hat. Dementsprechend gibt es auch für die Modelierung keine einheitlichen Vorgaben.

Im Folgenden wird allerdings eine Möglichkeit beschrieben, welche in Kombination mit Web-Services verwendet wird. Wie in Kapitel 3.2 beschrieben ist, sind Web-Services ein großer, in manchen Augen sogar äquivalenter, Teil von SOA.

Um diese Darstellungsart beschreiben zu können, müssen allerdings erst ein paar Begriffe geklärt werden:

Web Service Definition Language (WSDL) ist eine auf XML basierende Sprache, um die Schnittstelle eines Web-Services zu beschreiben. WSDL beschreibt hierbei ausschließlich, auf welche Funktionen man bei dem bestimmten Service zugreifen kann und wie. Es wird nicht beschrieben wie der Service intern funktioniert. [16, S. 14]

Web Service Business Process Execution Language (BPEL) kann verwendet werden, um Web-services zu erstellen, Web-Services miteinander zu verbinden und auch um große Geschäftsprozesse zu beschreiben, die in sich andere Web-Services verwenden. BPEL wird auch oft verwendet, um Business-to-Business Transaktionen zu beschreiben. Hierbei stellt ein Unternehmen den Web-Service bereit und das andere Unternehmen verwendet den Service.

BPEL basiert wie auch WSDL auf XML. Die Logik eines Programms wird hierbei in BPEL beschrieben, während Dateitypen in XML Schema Definitions (XSD) und input/output mit WSDL beschrieben werden.

BPEL wird oft auch Orchestrierungssprache genannt, da sie zentral ein komplexes und großes System an Services verwalten kann. Hierbei ist zentral der wichtige Punkt, um nicht von Choreographie zu sprechen sondern von Orchestrierung. Für BPEL gibt es keine eigene Darstellungsart und daher haben viele Firmen ihre eigenen entwickelt. [16, S. 20]

Business Process Modeling Notation (BPMN) ist eine Darstellungsart für Geschäftsprozesse und ist daher eine allgemeine Möglichkeit, um mit BPEL erstellte Programme darzustellen. BPMN ist ähnlich zu UML ein Aktivitätsdiagramm und besteht aus folgenden Elementen:

Start Event Jeder BPEL Prozess muss mit einem Start Event beginnen.

End Event Jeder BPEL Prozess muss auch ein End Event besitzen.

Gateways In BPMN gibt es vier Mögliche Gateways Exclusive(XOR), Inclusive(OR), Complex und Parallel(AND). Complex Gateways kommen allerdings in BPEL nicht vor und sollten daher nicht verwendet werden.

Pools Pools sind externe Komponenten, von welchen nur Input und Output bekannt sind(Blackboxen).

Activities Activitäten können entweder einen Sub-Prozess(Eigener Ablauf von BPMN Elementen) oder einen Task(Eine atomare Tätigkeit) darstellen.

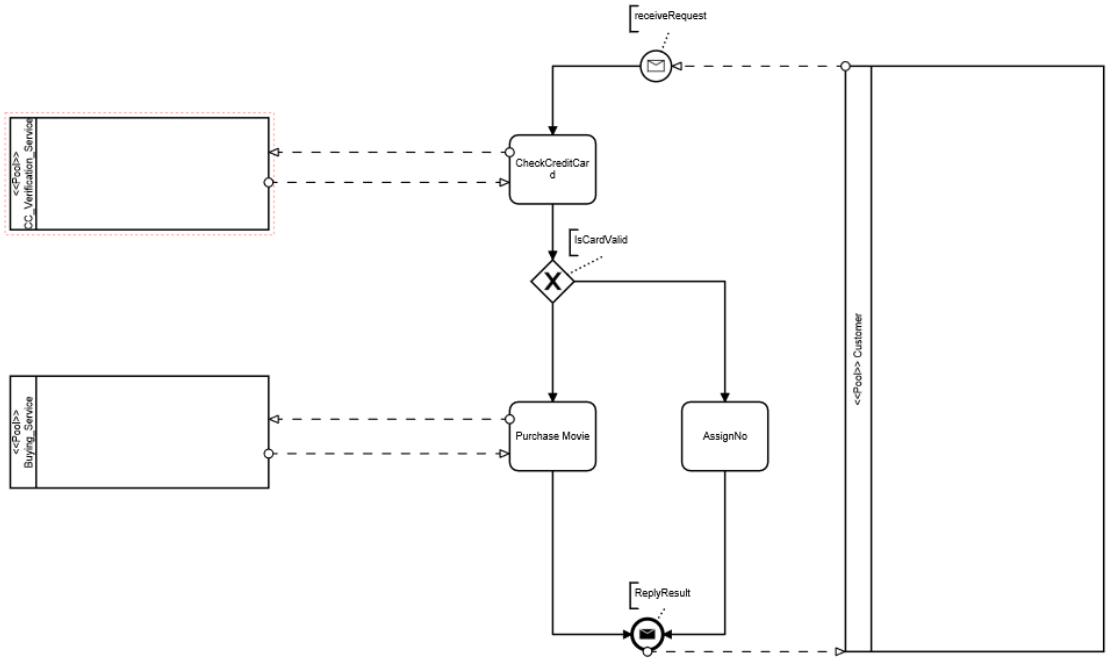

Abbildung 4.1: Darstellung, der Verknüpfung von zwei Services mit BPEL, mit BPMN

Wie man sich mit den oben beschriebenen Begriffen vielleicht schon denken kann, ist BPMN der zentrale Teil zur Modellierung von einer SOA. Wie in den vorangegangenen Kapiteln auch schon beschrieben, handelt es sich bei den Services von SOA um Geschäftsprozesse.

4.1 Verwendung von BPMN

Wie sieht so ein mit BPMN dargestellter Geschäftsprozess nun aus? Die Abbildung 4.1 beschreibt hierbei einen Geschäftsprozess, der auf dem im folgenden Kapitel 5 beschriebenen Fallbeispiel basiert, dort an sich allerdings nicht vorkommt.

In dem dargestellten Geschäftsprozess werden zwei Services verwendet und diese mit Hilfe von Logik verknüpft. Diese Verknüpfung und Logik kann mit BPEL umgesetzt werden. Hierbei gibt es auch Tools, die diese Aufgabe, anhand der Modellierung, übernehmen können(siehe Kapitel 6).

Inhaltlich beschreibt Abbildung 4.1 die Kaufanfrage eines Kunden mit einer Kreditkarte. Wenn wir nun den abgebildeten Ablauf verfolgen, kommt natürlich zuerst die Anfrage. Diese Anfrage wird zuerst an den CC-Verification-Service weitergereicht, welcher überprüft, ob die angegebene Kreditkarte gültig ist. Dieser Service könnte gegebenenfalls auch von einem anderen Unternehmen kommen. Wenn dieser Service nun antwortet, wird die Antwort verarbeitet und entweder der Kauf abgelehnt oder die Informationen an den zweiten Service den Buying-Service weitergereicht. Dieser Service ist nun für die Auslieferung der Ware zuständig.

Im Anschluss wird die erstellte Nachricht, entweder „Kauf abgelehnt“ oder „Kauf erfolgreich“ an den Kunden weitergegeben.

Wenn dieser dargestellte Geschäftsprozess nun als ein Service dargestellt würde, würde es sich hierbei um einen übergeordneten Service handeln, da er aus mehreren untergeordneten Services zusammengesetzt ist.

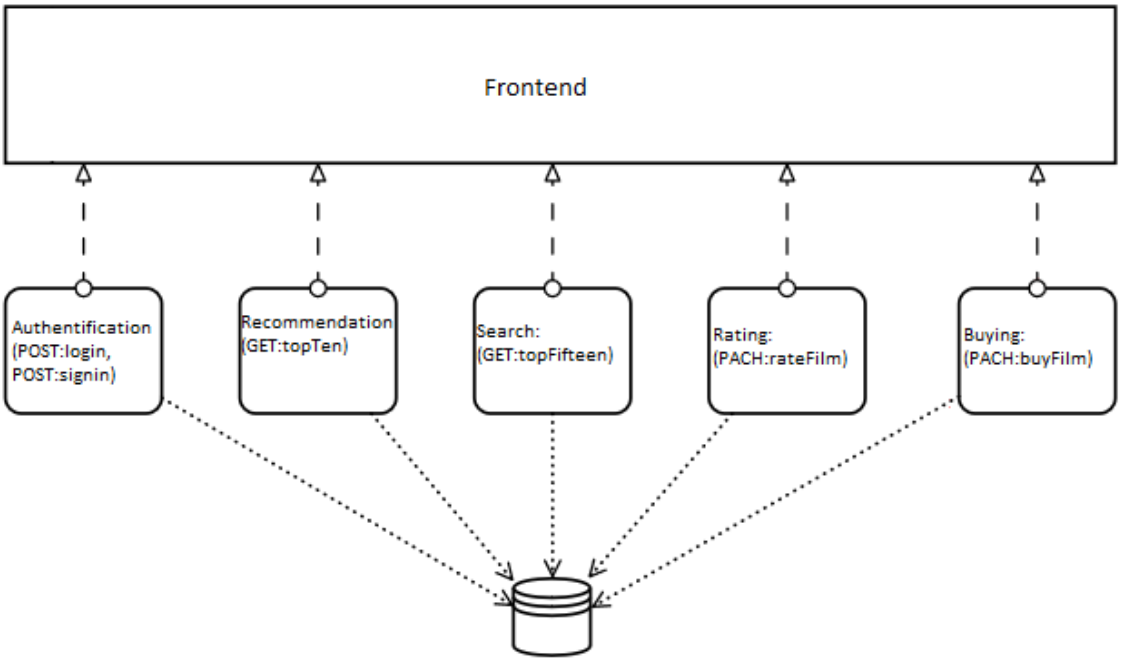

Abbildung 4.2: Der Aufbau des Fallbeispiels

4.2 Aufbau einer SOA grafisch dargestellt

Stark vereinfacht betrachtet besteht eine SOA aus einem Frontend, dieses kann auch ein Terminal sein, mehreren Services und der Datenbank, hierbei ist wichtig, dass SOA, im Gegensatz zu Microservices(siehe Kapitel 8) nicht definiert, wie diese Datenbank aussieht. Das bedeutet, es kann sein, dass jeder Service eine eigene Datenbank hat, mehrere Services sich eine teilen oder alle auf die gleiche Datenbank zugreifen.

In Abbildung 4.2 wird der Fall dargestellt, in dem sich alle Services eine Datenbank teilen.

5 Fallbeispiel

Für das Fallbeispiel, das in diesem Kapitel erklärt wird, wurde ein Recommendation-System gewählt. Es ist ein Recommendation-System welche Filme für einen bestimmten Benutzer vorschlägt, abhängig von seinem Such- und Bewertungsverhalten. Das Recommendation-System wird als eine SOA (Service Oriented Architecture) umgesetzt. Bei der Umsetzung wird nur die grobe Service-Struktur aufgebaut, also ohne Business-Logik und ohne Verwendung einer Datenbank. Außerdem wird für Demonstrationszwecke anstatt einem User Interface ein Demonstration-Script umgesetzt, das einen Benutzer simuliert.

5.1 Struktur der Services

Die umzusetzende Struktur besteht zum einen aus einem Web-basierendem Frontend, welches in diesem Fallbeispiel nicht weiter verfolgt wird. Das Backend besteht aus folgenden Services, die noch genauer erklärt werden:

- **Authentification Service:** welcher zwei REST-Endpoints zu Verfügung stellt, um einen Benutzer zu registrieren und um sich mit einem Benutzer einloggen zu können.

- **Recommendation Service:** welcher einen REST-Endpoint bereitstellt, um eine Liste der wichtigsten Empfehlungen bekommen zu können.

- **Search Service:** welcher einen REST-Endpoint bereitstellt, für die Suche nach einem bestimmten Film.

- **Rating Service:** welcher einen REST-Endpoint bereitstellt, um Filme bewerten zu können.

- **Buying Service:** welcher einen REST-Endpoint bereitstellt, um sich gewünschte Filme kaufen zu können.

Das Backend, bestehend aus den erwähnten Services, nutzt gemeinsam eine Datenbank zur Speicherung der verschiedenen Daten, wie beispielsweise Benutzerdaten oder Filmempfehlungen.

5.2 Aufbau eines Services

Jeder der oben erwähnten Services stellen REST-Schnittstellen zur Verfügung, um diese im Frontend (bzw. in unserem Demonstration-Script) nutzen zu können. Die Frage, warum hier REST gewählt wurde, wird noch im nachfolgenden Kapitel erläutert. Um den Aufbau eines Services näher zu erklären, wird hier als Beispiel der Authentification-Service näher betrachtet. Die Services wurden in Node.js entwickelt unter der Benutzung des Frameworks Express.js. Wie man in der Abbildung 6.1 sehen kann, werden beim Authentification-Service zwei Schnittstellen zur Verfügung gestellt. Zu einem ist es der „signin“, bei dem man sich mit einem POST-Request registrieren kann. Die zweite Schnittstelle bietet die Möglichkeit, sich mit Hilfe des Service einzuloggen.

```
16  app.use(express.json())
17
18  app.post('/signin', (req, res) => {
19     res.writeHead(200, {'Content-Type': 'application/json'});
20     // ... signin ...
21     const response = {
22        "response" : `The User: (
23           ${req.body.name},
24           ${req.body.surname},
25           ${req.body.email}
26        ) has signed in.`,
27     }
28     res.end(JSON.stringify(response));
29  })
30
31  app.post('/login', (req, res) => {
32     res.writeHead(200, {'Content-Type': 'application/json'});
33     // ... login ...
34     const response = {
35        "response" : `The User with the E-Mail (
36           ${req.body.email}
37        ) has loged in.`,
38     }
39     res.end(JSON.stringify(response));
40  })
```

Abbildung 5.1: Authentification Service (signin and login)

5.3 Warum REST?

Warum und wann entscheidet man sich für REST-basierende Services, bei einer SOA? In der Regel kommen bei einer SOA Protokolle zum Einsatz wie JMS, AMQP, MSMQ, aber am meisten SOAP. Trotz allem wurde bei diesem Beispiel REST gewählt. Da es sich bei erwähntem Beispiel nicht um sensible Daten handelt, das Thema Sicherheit keine besondere Rolle spielt und kein ESB (enterprise service bus) zum Einsatz kommt, so sind die Vorteile von SOAP an der Stelle nicht zu gebrauchen. Des Weiteren braucht die erwähnte Anwendung nicht mehrere Arten von Protokollen zu benutzen, was auch ein Vorteil von SOAP wäre. Nachdem man sich wichtige Vorteile von SOAP angeschaut hat, sieht man, dass diese beim erwähnten Beispiel nicht zutreffend sind, doch dies heißt noch nicht, dass REST eine bessere Wahl ist. REST ist an der Stelle eine passende Wahl, da Berechnungen eines Recommendation-Systems hohe Datenmengen mit vielen Berechnungen erfordern. Des Weiteren ist bei dem Recommendation-System ein User Interface (UI) geplant, somit wäre man mit REST nicht nur auf XML beschränkt, man würde also JSON benutzen, das schneller und einfacher zu benutzen ist. Ein weiterer Punkt, der zu Gunsten von REST spricht ist, dass REST einen besseren Support für Browser-Clients bietet und somit die bessere Wahl für die Benutzung einer Web-basierenden UI ist. [15, S. 33] [15, S. 34]

5.4 Demonstration Script

Das Demonstration Script dient hier, wie es der Name schon sagt, um zu demonstrieren, wie die einzelnen Services über ein Netzwerk verteilt sein können und dabei in dem Demo-Script benutzt werden können. Das Demo-Script ruft die einzelnen Services

nach einander auf, wie es auch ein Benutzer über ein User Interface machen könnte. Wie man in der Abbildung 6.2 sehen kann, wird zuerst der Authentification-Service aufgerufen, um sich zu registrieren und nach dem Registrieren sich dann auch einzuloggen. Danach könnte der Benutzer, durch das Ansprechen des Such-Services, nach einem Film suchen. Nachdem ein Film geöffnet wurde, könnte man ihn, mit Hilfe des Rating- oder des Buying-Services, bewerten oder auch kaufen. Wie man beispielsweise den Authentifi-

```
14  main()
15
16  async function main(){
17      // User signing in
18      await signin()
19
20      // User loging in
21      await login()
22
23      // User searching ...
24      await search()
25
26      // User rating a film
27      await rating()
28
29      // User buying a film
30      await buying()
31
32      // User gets recommendations
33      await recommendation()
34  }
```

Abbildung 5.2: Schritte des Demonstration Scripts

cation Services einsetzten kann, sieht man in der Abbildung 6.3. Sowohl beim Registrieren, als auch beim Einloggen wird ein POST-Request an den Authentification Service gesendet. Dabei setzt sich das Request aus dem Server-Namen, dem Port, der Funktion (wie zum Beispiel /signin") und dem Request-Body. Der Request-Body besteht aus den nötigen JOSN-Daten, die die Schnittstelle benötigt. Im Anschluss kann man abhängig von Response-Daten sehen, ob das Vorgehen funktioniert hat und somit den weiteren Prozess bestimmen.

```
36  // - Authentification: (POST:login, POST:signin)
37  function signin(){
38      return new Promise( async (resolve, reject) => {
39          logger.warn('User signing in')
40          const body = {
41              name: 'Meris',
42              surname: 'Krupic',
43              pw: '1!m"K2',
44              email: 'test@hotmail.com'
45          }
46          const response = await axios.post(`
47                  ${config.AuthenticationService.server}:
48                  ${config.AuthenticationService.port}/signin
49              `, body)
50          console.log(response.data)
51          console.log(' ')
52          await delay(3000)
53          resolve()
54      })
55  }
56  function login(){
57      return new Promise( async (resolve, reject) => {
58          logger.warn('User loging in')
59          const body = {
60              email: 'test@hotmail.com',
61              pw: '1!m"K2'
62          }
63          const response = await axios.post(`
64                  ${config.AuthenticationService.server}:
65                  ${config.AuthenticationService.port}/login
66              `, body)
67          console.log(response.data)
68          console.log(' ')
69          await delay(3000)
70          resolve()
71      })
72  }
```

Abbildung 5.3: Ansprechen des Authentification Services

6 Tools

Wenn wir nun über Tools für SOA reden, müssen wir erst einmal festlegen, welche Arten von Tools es gibt:

Erstellung von SOA Hierbei handelt es sich um Tools, wie auch schon in Kapitel 4.1 erwähnt, die die Möglichkeit der Modellierung von Geschäftsprozessen bieten und bei der Generierung von Code helfen können.

Ein Beispiel dazu wäre der:

Enterprise Architect von SparxSystems Software GmbH [19] Ist kein SOA spezifisches Tool, bietet aber unter anderem die Möglichkeit, anhand von grafisch erstellten Modellen, in unserem Fall mit BPMN, den dazugehörigen Code in BPEL, automatisch zu generieren.

Wartung von SOA [2] Hierbei handelt es sich um Tools, welche bei einer bestehenden SOA die Wartung und Kontrolle vereinfachen. In diesem Fall muss das Tool auf die jeweiligen Bedürfnisse des Unternehmens eingehen und muss daher eine gewisse Größe mitbringen oder auf das Unternehmen zugeschnitten sein. Auch lohnen sich diese Tools erst, wenn es sich um recht große und komplexe Architekturen handelt.

Beispiele hierzu sind:

CA's Wily SOA Solution[1] kommt mit drei Kernfunktionen: Es ist in der Lage automatisch die Komponenten in einer SOA zu finden, die Abhängigkeiten und Transaktionen zwischen diesen Komponenten darzustellen und den Benutzer zu warnen, wenn es Einbrüche in der Performance gibt. Besonders der zweite Punkt kann äußerst hilfreich sein, da in vielen Fällen die zu Beginn erstellten Modelle nicht unbedingt die tatsächlichen technisch umgesetzten Abhängigkeiten und Komponenten darstellen.

CA's Wily SOA Solution wird unter anderem von Amdocs, Bear Stearns, FedEx, Geico, Home Depot, Intel und Pacific Gas & Electric verwendet.

AmberPoint's SOA Management System[12] Bietet ebenso eine automatische Erkennung der Komponenten einer SOA und einer der besonderen Punkte dieses Pakets ist die übersichtliche grafische Darstellung dieser Komponenten. Ein weiterer Punkt, durch den sich dieses Tool auszeichnet, ist die erhöhte Beachtung von Sicherheit, dies wird umgesetzt durch die Verfolgung von Transaktionen in Echtzeit und der daraus folgenden direkten Erkennung von Problemen. Außerdem können den erkannten Problemen direkt die betroffenen Kunden zugewiesen werden, wodurch beschleunigt reagiert werden kann.

2010 wurde AmberPoint von Oracle gekauft[13]. Seitdem bietet AmberPoint auch die oben beschriebenen Vorteile bei der Erstellung der SOA.

AmberPoint's SOA Management System wird unter amderem von der BT Group, H&R Block, Motorola und dem U.S. Department of Defense verwendet.

Wie auch schon innerhalb dieses Kapitels beschrieben sind diese Tools meist recht groß und haben dementsprechend im Fallbeispiel(siehe 5) keine Verwendung gefunden. Dies

ist allerdings kein Problem, da dieses Fallbeispiel in der Praxis nicht im Rahmen einer SOA vorkommen würde. Da eine SOA normalerweise erst ab einer bestimmten Größe überhaupt eingesetzt wird und genau dafür sind die oben genannten Tools gedacht.

7 Vor-und Nachteile

7.1 Vorteile

Es gibt viele Vorteile, die für die Einführung von SOA sprechen. Einige davon wurden in den vorherigen Kapiteln genannt; diese werden in diesem Abschnitt detaillierter erläutert.
Durch lose Kopplung ist ein SOA-System fehlertolerant und flexibel. Änderungen und Modifikationen in einzelnen Komponenten haben wenige Auswirkungen auf das gesamte System, im bestens Fall gar keine. Daraus ergibt sich eine bessere Austauschbarkeit und Erweiterbarkeit des Systems. Es können Komponenten einfach geändert und hinzugefügt werden. Geschäftsprozesse werden somit leicht anpassbar.
Ein weiterer Vorteil von SOA ist die Skalierbarkeit. Diese wird einerseits mittels loser Kopplung -wie in Abschnitt 2.2 erläutert erreicht- und andererseits durch den Einsatz von stateless Services. Bei zustandslosen Services gibt es keine Benutzersitzungen, also sind die Daten nicht an Ressourcen gebunden. Es können Anfragen auf verschiedene Systeme einfach verteilt werden, um die Leistungsfähigkeit des Systems zu erhöhen und Grenzen zurückzusetzen.
SOA akzeptiert Heterogenität. Da Services Black-Boxes sind, können unterschiedliche Technologien für die Implementierung der Services eingesetzt werden, wie z.B. Programmiersprachen und Kommunikationsprotokolle. Es besteht eine klare Trennung zwischen der internen Struktur und der API oder dem Service-Container.
Wenn Services wiederverwendet werden können, entsteht dadurch eine Kosten- und Aufwandsreduzierung.[5] Es werden schon existierende Services eingesetzt, anstatt neue entwickeln zu müssen, was den Entwicklern Arbeit abnimmt.
Darüber hinaus bietet SOA gut strukturierte Geschäftsprozesse. In SOA werden Services mit ihrer Beschreibung versetzt. Dies erhört das Verständnis und die Transparenz.[5]

7.2 Nachteile

Natürlich hat SOA nicht nur Vorteile. Es gibt auch Nachteile und Risiken. SOA ermöglicht zwar Kosteneinsparungen, aber diese sind langfristig. Am Anfang entstehen deutlich mehr Kosten und Aufwand. SOA kann nicht als fertige Lösung gekauft werden, es ist eine Strategie, deren Einführung im Unternehmen viel eigene Initiative erfordert, Zeit und Geld kostet.[7, S. 323] Es müssen Anforderungen abgestimmt, Geschäftsprozesse modelliert, Services definiert und die ganze Architektur designiert werden. Dabei müssen wichtige Entscheidungen getroffen werden, wie z.B. wo und wie lose Kopplung im System Platz finden soll, welche Technologien oder Protokolle eingesetzt werden sollen. Zugleich muss der Betrieb weiter laufen. Es werden zusätzliche Arbeitskräfte dafür eingestellt und Ressourcen beschafft.
Die Einführung einer SOA benötigt auch bestimmte organisatorische Rahmenbedingungen. Es müssen zum Unternehmen angemessene Arbeitsprozesse, und Einführungsstrategie abgestimmt werden. Wie jede neue Änderung im Unternehmen, benötigt SOA Akzeptanz und Motivation. All diese Aspekte zeigen, dass eine SOA nicht auf einmal auf die Beine gestellt werden kann, sondern Schritt für Schritt implementiert werden muss. Eine

SOA-Einführung kann 3 bis 5 Jahre lang dauern. Für ein System mit 50 bis 100 Services wird mit ca. 4 Jahre gerechnet.[4, S. 92]
Weiterhin kann auch die mangelnde technische Standardisierung ein Hindernis sein. Wie schon am Anfang erwähnt, ist SOA immer noch ein abstraktes Konzept. Es gibt zwar einige Standards, die aber noch nicht abgeschlossen sind.

8 Abgrenzung zu Microservices

8.1 Einführung

Wenn es darum geht, große monolithische Anwendungen in kleinere, wiederverwendbare und besser wartbare Komponenten (Services) aufzuteilen, entsteht die Frage, für welche Architektur man sich entscheiden möchte, SOA oder Microservices. [10, S. 28] Da Microservices oft als eine verbesserte SOA (Service Oriented Architecture) gesehen werden, entsteht ein Missverständnis, der einzelnen Architekturen und ein Missverständnis, wann man sich für Microservices, oder doch eher für SOA entscheiden sollte. Man kann die Microservice-Architektur als eine Weiterentwicklung von SOA ansehen, die für einen spezielleren Anwendungsbereich eingesetzt werden kann. [18, S. 13] [10, S. 28] Dass Microservices nicht jede SOA Anwendung ersetzten sollten, ist klar, jedoch ist schon wichtig zu wissen, bei welchen Anwendungsfällen es eine bessere oder auch eine schlechtere Wahl ist. Um diese Problematik besser angehen zu können, wird in diesem Kapitel erklärt, was die wichtigen Unterschiede von SOA zu Microservices sind, aber auch wann man sich für welche der beiden Architekturen entscheiden sollte. [22, S. 2][15, S. 7]

8.2 Hintergrund

SOA (Service Oriented Architecture) und Microservices sind weitverbreitete und oft vorkommende Architekturen bei verteilten Systemen. Die Wiederverwendbarkeit von (Teil-) Prozessen und die Verteilbarkeit der einzelnen Services, auch Netzwerk übergreifend, stehen dabei im Mittelpunkt. Architekturen verteilter Anwendungssysteme bieten große Vorteile gegenüber Monolithen-Architekturen, oder Schichten basierenden Architekturen, wie beispielsweise in den Punkten Skalierbarkeit, Wiederverwendbarkeit und die bessere Kontrolle von der Entwicklung bis hin zum Deployment. Neben den vielen Vorteilen von Service-basierten Architekturen gibt es auch Nachteile, die nicht zu vernachlässigen sind, wie zum Beispiel die steigende Komplexität solcher Systeme. Die SOA (Service Oriented Architecture) wird weniger in Betracht gezogen, als Microservice Architekturen, aufgrund des Hypes von Microservice Architekturen, die in der aktuellen Zeit stattfinden. Da die erwähnten Architekturen, einige Gemeinsamkeiten haben, werden sie oft so eingesetzt, dass sie weder den Richtlinien der einen, noch der anderen Architektur entsprechen. Um besser abwiegen zu können, wann man Microservices vor SOA vorziehen sollte, werden im nachfolgendem Abschnitt die Vor- und Nachteile der erwähnten Architekturen verglichen und somit wird eine klarere Trennung der einzelnen Architekturen geschaffen. [22, S. 3] [15, S. 11]

8.3 Abgrenzung

Im Folgendem wird die Abgrenzung von SOA (Service Oriented Architecture) zu Microservices erklärt. Um die verschiedenen Aspekte der einzelnen Architekturen vergleichen zu können, wird zuerst die Microservice Architektur erläutert, um im Anschluss die Unterschiede besser zu verstehen, verglichen und somit wird eine klarere Trennung der einzelnen Architekturen geschaffen.

8.3.1 Microservices

Die Microservices-Architektur ist ein Architekturmuster der Anwendungsentwicklung, welche sich von einer monolithischen Anwendung so unterscheidet, dass die Business-Logik und die Datenhaltung nicht zentral gehalten wird, sondern in mehrere kompakte Komponenten (Services) aufgeteilt wird. Der einzelne Service kann unabhängig entwickelt und implementiert werden, aber auch unabhängig von anderen Services funktionieren. Ziel ist es, dass ein entwickelter Service nur eine Aufgabe erledigt, nicht von anderen Services beeinflusst werden kann und somit einfacher als ein Teil, einer modular aufgebauten Anwendung, wiederverwendet werden kann. [22, S. 4] Beim Einsatz von Microservices geht es nicht nur um die reine Entwicklung, sondern auch um die Umstrukturierung der Entwicklerteams, die Kommunikation in- und außerhalb der Teams, aber auch das Management von großer Anzahl an Services.

8.3.2 Unterschiede

In folgenden Punkten werden wichtige Unterschiede von SOA zu Microservices verglichen, die bei einer Entscheidungsfindung, zwischen den zwei Architekturen, helfen können. Die einzelnen Unterschiede stellen nicht unbedingt Vor- oder Nachteile, der einzelnen Architektur dar.

Aufgabe eines Services: Ein SOA-Service erfüllt in der Regel mehrere Aufgaben und ein Microservice soll nur eine einzige Aufgabe erfüllen, somit können Microservices flexibler miteinander kombiniert und zusammengestellt werden. [9, S. 6]

Dependency: Die einzelnen Business-Komponenten sind bei einer SOA-Architektur abhängig voneinander im Gegensatz zu einer Microservices-Architektur. [9, S. 6] [22, S. 8]

Size: Eine SOA-Anwendung ist in der Regel größer als eine Microservices-Anwendung, wobei diese Betrachtung eher eine grobe Richtlinie ist und nicht in jedem Falle zutreffend ist. [9, S. 6]

Entwicklungsteams: Bei der Verwendung von Microservices sind Teams so aufgebaut, dass ein Team einen oder mehrere Services vollständig betreut, von der Datenbank bis zum "User Interface", also ein Team, dass nach der Silo-Architektur aufgebaut ist. Bei der Verwendung von SOA, sind die Teams eher nach den hierarchischen Ebenen aufgebaut, somit gibt es Teams, die sich um die Services kümmern, Teams für die "User Interfaces" und eventuell Teams die für das Daten-Management verantwortlich sind. [22, S. 8]

Deployment: SOA wird oft als ein Monolith deployt, im Gegensatz zu Microservices, wo der Service eher einzeln deployt wird. Somit ist das Deployment von einzelnen Services zeiteffizienter und es muss nicht für jede kleine Anpassung ein großer Deployment-Prozess durchgeführt werden. Erwähnenswert ist es auch zu sagen, dass nicht jede SOA-Anwendung als ein Monolith deployet wird, die einzelnen SOA-Deployment-Prozesse können teils sehr unterschiedlich sein. [9, S. 6]

Scalability: Mit Microservices lässt sich im Gegensatz zu SOA eine höhere Skalierbarkeit erreichen, da eine Microservices-Anwendung granularer aufgeteilt ist, als eine SOA-Anwendung und somit auch wiederverwendbarer als eine SOA-Anwendung. [9, S. 6] [22, S. 8]

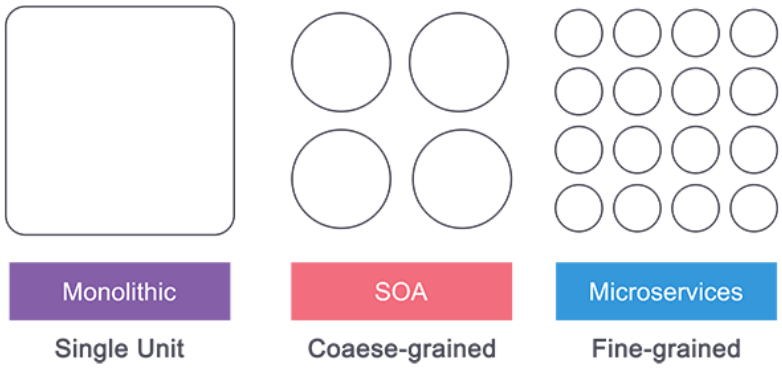

Abbildung 8.1: Monolithic vs. SOA vs. Microservices[14]

User Interface: Bei Anwendungen, die nach SOA entwickelt wurden, gibt es in der Regel für alle Services der Anwendung ein gemeinsames "User Interface", wohingegen bei Microservices in manchen Fällen versucht wird das "User Interface" auch in kleinere und besser verwaltbare Komponenten aufzuteilen. An der Stelle ist es wichtig zu erwähnen, dass so eine Zerlegung einer UI als Micro-Frontend-Architektur genannt wird, das wiederum als eine Microservice Nachahmung gesehen wird. [22, S. 8]

Daten-Management: Ziel der Datenhaltung bei Microservices ist es, dass jeder Service seine eigenen Daten (eigene Datenbank) hat, im Gegensatz zu SOA, wo mehrere Services eine oder auch mehrere Datenbanken gemeinsam nutzen. [22, S. 8]

Service-Größe: SOA Services sind in der Regel größer als ein Microservice, deren Aufgabe ist umfangreicher und bildet einen größeren Teilprozess in einer Anwendung ab. Wohingegen ein Microservice nur eine und ganz kleine Aufgabe erledigt, oft ist die Größe eines Services der Größe einer kompakten Klasse sehr ähnlich.

8.4 Fazit

Nachdem die Vorteile, Nachteile und Unterschiede erklärt wurden, geht es um die wichtige Frage, wann man sich für SOA und wann für Microservices entscheiden soll. Obwohl die Architekturen viele Eigenschaften gemeinsam haben, verfolgen die Architekturen oft verschiedene Ziele, oder haben zumindest unterschiedliche Schwerpunkte. Zusammenfassend gesagt, sieht der Trend in den letzteren Jahren eher zu Gunsten von Microservices aus. Obwohl Microservices oft als der Gewinner, der beiden Architekturen, dasteht, so kommen in der letzten Zeit immer öfters Meinungen, die eher für SOA sprechen, eventuell geht der Trend in der Zukunft einen Schritt zurück zu SOA, unter Verwendung von moderneren Technologien, wie sie auch bei Microservices verwendet werden. [18, S. 14] [22, S. 9] [15, S. 33]

Literatur

[1] Enterprise Management Associates. *Wily SOA Manager*. Website. Online erhältlich unter https://de.slideshare.net/Zubin67/wily-soa-manager; abgerufen am 14.03.20. 2007.

[2] Denise Dubie. *10 tools to manage SOA*. Website. Online erhältlich unter https://de.slideshare.net/Zubin67/wily-soa-manager; abgerufen am 14.03.20. 2007.

[3] T. Erl, C. Gee, J. Kress, B. Maier, H. Normann, P. Raj, L. Shuster, L. Trops, P. Wik und T. Winterberg. *Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation*. Upper Saddle River, USA: Prentice Hall, 2014.

[4] Roger Heutschi. *Serviceorientierte Architektur: Architekturprinzipien und Umsetzung in die Praxis*. Heidelberg, Deutschland: Springer-Verlag, 2007.

[5] Torsten Horn. *SOA (Service Oriented Architecture)*. Website. Online erhältlich unter https://www.torsten-horn.de/techdocs/soa.htm; abgerufen am 25.01.20.

[6] Jaxenter. *Enterprise Service Bus*. Website. Online erhältlich unter https://www.jaxenter.de/enterprise-service-bus-4-9187; abgerufen am 15.03.20. 2008.

[7] Nicolai Josuttis. *SOA in der Praxis:System-Design für verteilte Geschäftsprozesse*. Heidelberg, Deutschland: dpunkt.verlag, 2008.

[8] Dirk Krafzig, Karl Banke und Dirk Slama. *Enterprise SOA: Best Pratices für Serviceorientierte Architekuren, Einführung, Umsetzung, Praxis*. Heidelberg, Deutschland: Hüthig Jehle Rehm, 2007.

[9] *Microservice Over SOA*. https://www.tutorialspoint.com/microservice_architecture/microservice_architecture_tutorial.pdf. Tutorials Point, 2020.

[10] Sam Newman. *Building Microservices*. O'Reilly Media, 2015.

[11] OASIS. *OASIS Reference Model for Service Oriented Architecture*. Website. Online erhältlich unter https://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED; abgerufen am 31.01.20.

[12] Oracle. *Oracle Buys Amberpoint*. Website. Online erhältlich unter http://www.oracle.com/us/assets/amberpoint-general-presentation-072335.pdf; abgerufen am 14.03.20. 2010.

[13] Oracle. *Oracle buys AmberPoint*. Website. Online erhältlich unter http://www.oracle.com/us/corporate/press/048842; abgerufen am 14.03.20. 2010.

[14] Mohammed Ramees. *Understanding Microservices Architecture*. https://www.dotnettricks.com/learn/microservhttps://www.overleaf.com/project/5e6e628da08d4300015a8672ices/architecture-example-advantages. 2020.

[15] Mark Richards. *Microservices vs. Service Oriented Architecture*. O'Reilly Media, 2015.

[16] Doug Rosenberg. *Modeling Service-Oriented Architectures*. https://sparxsystems.com/downloads/ebooks/Modeling%20Service-Oriented%20Architectures.pdf. Sparx Systems Pty und ICONIX, 2010.

[17] *Service-orientierte Architekturen:Leitfaden und Nachschlagewerk*. https://www.bitkom.org/sites/default/files/file/import/bitkom-soa-leitfaden.pdf. Bitkom, 2009.

[18] Kameswara Eati Carlos M Ferreira Dejan Glozic Vasfi Gucer Manav Gupta Sunil Joshi Valerie Lampkin Marcelo Martins Shishir Narain Ramratan Vennam Shahir Daya Nguyen Van Duy. *Microservices from Theory to Practice*. https://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf. Redbooks, 2015.

[19] SparxSystems. *Enterprise Architect 14 im Überblick*. Website. Online erhältlich unter https://www.sparxsystems.de/fileadmin/user_upload/pdfs/EAReviewersGuide_EA-141-DE.pdf; abgerufen am 10.03.20. 2019.

[20] *The Evolving Role of Portals in an SOA World*. https://www.oracle.com/technetwork/middleware/ias/oow06-s281758-soa-portals-133833.pdf. Oracle, 2020.

[21] *The SOA Source Book*. Website. Online erhältlich unter https://www.opengroup.org/soa/source-book/soa/p1.htm; abgerufen am 31.01.20.

[22] Jiri Pechanec Tomas Cerny Michael J. Donahoo. *Disambiguation and Comparisonof SOA, Microservices and Self-Contained Systems*. https://www.researchgate.net/publication/320765439_Disambiguation_and_Comparison_of_SOA_Microservices_and_Self-Contained_Systems. ResearchGate, 2017.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

SOA Service Oriented Architecture
UI User Interface
REST Representational State Transfer
SOAP Simple Object Access Protocol
ESB Enterprise Service Bus
JSON JavaScript Object Notation
XML Extensible Markup Language
JMS Java Message Service
AMQP Advanced Message Queuing Protocol
MSMQ Microsoft Message Queuing
ESB Entreprise Sevice Bus
BDD Business Driven Development
EAI Enterprise Application Integration
BPP Business Prozess Plattform
SLA Service Level Aggreement
OASIS Organization for the Advancement of Structured Information Standards
WSDL Web Service Definition Language
UDDI Universal Description, Discovery, and Integration
BPEL Web Service Business Process Execution Language
BPMN Business Process Modeling Notation

Anhang

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt