Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Event-gesteuerte Architektur im RESTful-API Kontext

1. Projektarbeit

Im Rahmen der Prüfung zum Bachelor of Science (B. Sc.)

4. September 2023

VerfasserIn:	Jona Rumberg
Kurs:	WWI22B5
Dualer Partner:	SAP SE, Walldorf
Betreuer der Ausbildungsfirma:	Steven Rösinger
Wissenschaftlicher BetreuerIn:	Prof. Dr. Thomas Freytag
Abgabedatum:	4. September 2023

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende 1. Projektarbeitmit dem Thema:

Event-gesteuerte Architektur im RESTful-API Kontext

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 31. August 2020, _____

Jona Rumberg

Todo list

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABAP Advanced Business Application Programming

API Application Programming Interface

BO Business Object

BTP Business Technology Platform

CEP Complex Event Processing

EDA Event Driven Architecture

ERP Enterprise-Resource-Planning

IT Informationstechnologie

RAP Advanced Business Application Programming (ABAP) RESTful Application Programming Model

REST Representational State Transfer

WWW World Wide Web

HTTP Hypertext Transfer Protocol

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Die wohl wichtigste Entwicklung in der betrieblichen Informationstechnik der letzten Jahre ist wohl die in Richtung Cloud.Immer mehr Unternehmen setzten auf Cloud und profitieren in dem Zuge von kürzeren Entwicklungs- und Auslieferungszyklen, von geringeren Risiken bei der Anschaffung und schnelleren Amortisierungzeiten.Im Zuge dieser Entwicklung ist es für den Softwarearchitekten von heute immer relevanter geworden, die Software von Grund auf als verteiltes System und nicht monolithisch zu konzipieren. Schnittstellen und Lösungen zur Modularisierung sind also relevanter denn je.
Ein Ansatz in der Systemarchitektur, der seit einigen Jahren an Relevanz gewinnt, ist hierbei die Event Driven Architecture (EDA). Sie verspricht, durch den Fokus auf Ereignisse bei der Systemarchitektur eine Reihe von Vorteilen. In der Prozessmodellierung lassen sich Geschäftsvorfälle einfacher modellieren, in der Implementierung wird von Beginn an eine modulare Struktur geschaffen, die Ausfallsicherheit, Integrationsmöglichkeiten und eine bessere Lesbarkeit des Programmcodes bietet.

Cite!

Cite!

Cite!

1.2 Zielsetung

Das Ziel der Arbeit soll es sein, die Vorteile dieses Ansatzes näher zu untersuchen. Im Umfeld der Personalwirtschaft soll im Rahmen eines Migrationsprojektes eine Applikation auf eine Cloud-Infrastruktur umgezogen werden. In diesem Kontext bietet es sich an, das Potenzial einer EDA näher zu untersuchen. Im Verlauf dieser Arbeit soll daher ein Prototyp im genannten Kontext erstellt werden, der eine EDA implementiert. So soll exemplarisch geprüft werden, welche Hindernisse bei einem solchen Vorhaben auftreten können und daraus folgend ein allgemeines Urteil über den Architekturansatz gefällt werden.

1.3 Betrieblicher Kontext

Die SAP SE ist ein deutsches Softwareunternehmen, das seit 1972 Unternehmenssoftware entwickelt. Heute beschäftigt es rund 105000 Mitarbeiter und hat Standorte weltweit. Die Enterprise-Resource-Planning (ERP) Systeme der Firma haben in der Geschäftswelt entscheidenden, branchenübergreifenden Einfluss. SAP bietet hierbei Möglichkeit, durch umfassende Funktionen und eine einheitliche, integrierte Datenbasis, Geschäftsprozesse zu überblicken, dieses digital abzuwickeln und zu automatisieren.[1]
Das HCM ist die Personallösung des SAP ERP und kommt bis heute in einer großen Anzahl Firmen zum Einsatz. Besonders relevant sind hierbei sogenannte Self-Services über die Mitarbeiter Daten pflegen können und HR-relevante Prozesse anstoßen können. Dementsprechend stehen diesen Anwendungen besonders hohe Ansprüche an Leistungsfähigkeit und Verfügbarkeit entgegen.

1.4 Abgrenzung

1.5 Vorgehensweise und Aufbau der Arbeit

[1] Vgl. SAoJ.

2 Theoretischer Hintergrund

2.1 Event-Driven Architecture

Ereignisse

Zu Beginn der Betrachtung sollte der grundlegende Begriff des Ereignisses geklärt werden. Die heute gängige Definition eines Ereignisses im Kontext von EDA ist, dass ein Ereignis eine 'signifikante Änderung des Zustands' ist.[2] Diese relativ breite Definition hat zur Folge, dass eine große Menge an Geschäftsvorfällen als Ereignis begriffen werden kann. Als Standardbeispiel kann hier die Stornierung eines Fluges genannt werden, aber auch ein Eingang eines Auftrages, die Einstellung eines Mitarbeiters oder so etwas Reguläres wie der Beginn eines Arbeitstages, gemessen durch eine Stechuhr, konstituieren ein Ereignis. Schulte misst diesen Ereignissen in Studien für Gartner eine übergreifende Signifikanz zu. Im Grunde sei die Welt an sich ereignisgesteuert und Ereignisse als Grundlage von Architekturüberlegungen bilden diesen Umstand am besten ab.[3] Mit dieser Annahme als Grundlage der Betrachtung wird noch einmal die Relevanz der Überlegungen klar. Die Modellierung von Geschäftsvorfällen, oder besser Geschäftsereignissen, in einer ereignisgesteuerten Weise bildet tatsächliche Verhältnisse akkurater und intuitiver und somit besser ab, als herkömmliche Architekturansätze.[4] Diese Erkenntnis wird besonders relevant, wenn man betrachtet, dass mit einer wachsenden Gesamtmenge an Ereignissen die strukturierte Abarbeitung dieser immer wichtiger wird. Durch die immer ausgeprägteren Informationstechnologie (IT) Landschaften von Unternehmen, die immer filigraner in der Lage sind Geschäftsvorfälle zu erfassen, oder Entwicklungen wie das Internet of Things, wächst die Datenmenge, die verarbeitet werden kann rapide. Die Interpretation dieser Daten als Ereignisse und deren strukturierte Verarbeitung stellt eine Möglichkeit dar, diese Daten sinnvoll zu nutzen und aus dieser Entwicklung Wert zu schöpfen.[5] Eine im Kontext der Unternehmenssoftware relevante Unterscheidung ist dabei die zwischen technischen und Anwendungsereignissen. Anwendungsereignisse sind

[2] Vgl. Ch06, S. 4.
[3] Vgl. Sc03, S. 2.
[4] Vgl. Br10, S. 13.
[5] Vgl. Br10, S. 16.

Ereignisse, die fachliche Bedeutung haben, wie beispielsweise ’ein Kundenauftrag geht ein’, ’eine Zahlung wurde getätigt’ oder ’Ein Mitarbeiter betritt das Gebäude’. Ein technisches Ereignis hingegen ist ein Ereignis, dass ausschließlich systemintern relevant ist und zu Kommunikation und Koordination von Systemkomponenten genutzt wird. Beispiele wären ’Ein Datenbankeintrag wurde erstellt’ oder ’Eine Datei wurde hochgeladen’. Es ist üblich, dass Anwendungsereignisse einen höheren semantischen Stellenwert einnehmen, also dass in der Verarbeitung eines Anwendungsereignisses viele technische Ereignisse ausgelöst werden.[6]

Ereignisorientierung als Architekturansatz

Zuerst einmal handelt es sich bei EDA[7] um ein Konzept der Prozessmodellierung. Im Gegensatz zur gewöhnlichen Ablauf-orientierten Modellierung werden die Prozesse nicht als aufeinanderfolgende Schritte, sondern als Reaktionen auf Zustände konzeptioniert. Daraus resultiert, dass nicht mehr die prozedurale Abhandlung von Arbeitsschritten die zentrale Aufgabe in der Anwendungssystem-Entwicklung darstellt, sondern die Reaktion auf Ereignisse. Im Mittelpunkt von Architekturentscheidungen steht die Frage: ’Was passiert, wenn dieses Ereignis eintritt?’ und nicht mehr: ’Welche Schritte müssen zur Erfüllung dieser Anforderung gegangen werden?’. Was daraus resultiert, ist eine Architektur, die schon mit Beginn der Konzeption wesentlich agiler und robuster ist, da von Anfang an mit der Annahme gearbeitet wird, dass prinzipiell zu jedem Zeitpunkt jedes Ereignis eintreten kann.[8] Ein Definitionsversuch für EDA könnte also wie folgt lauten: Event-Driven Archtitecture bezeichnet einen Modellierungsansatz für ein verteiltes, asynchrones System, das verschiedene Komponenten durch eine zentrale Verarbeitung von Events verbindet.[9]

[6] Vgl. Go21, S. 245f.

[7] Da sich bis jetzt keine allgemeingültige deutsche Übersetzung der Fachterminologie durchgesetzt hat, sollen in dieser Arbeit die englischen Begrifflichkeiten verwendet werden.

[8] Vgl. Br10, S.30.

[9] Vgl. Go21, S. 248.

Technische Grundkonzepte der EDA

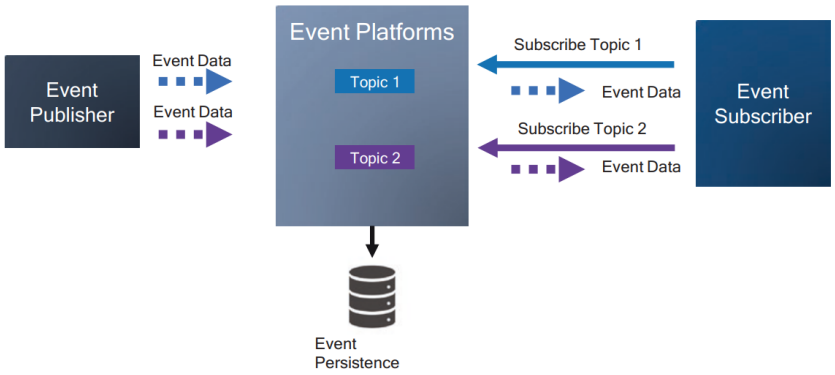

Abbildung 1: Komponenten der EDA[10]

Die wichtigste Komponente eines durch EDA modellierten Systems ist die zentrale Plattform zur Verarbeitung der Ereignisse in der Mitte der Architektur. Sie stellt die Infrastruktur bereit, um Events anzunehmen und diese weiterzugeben. Um einen Mehrwert aus dem System zu ziehen, muss sie darüber hinaus in der Lage sein, einen Kontext um Events herzustellen, d.h. sie in Verbindung mit anderen Ereignissen zu setzten, Ereignisse auf höheren Abstraktionsebenen zu erstellen und Ereignisse gegebenenfalls zu konsolidieren. Man spricht bei diesem Prozess von Complex Event Processing (CEP).

Weitere Komponenten des EDA sind Publisher und Subscriber.[11] Sie sind explizit von außen an das System herangeschaltet, d.h. sie haben keine Kenntnis voneinander und können auch auf völlig unterschiedlichen Plattformen basieren. Das bringt den Vorteil, dass ein durch EDA modelliertes System inhärent modular aufgebaut ist und so zum einen weniger anfällig für Totalausfälle ist, da die Komponenten unabhängig sind, und zum anderen prädestiniert für Integrationsvorhaben ist. Zu diesen grundlegenden Komponenten können im Zuge des CEP noch einige weitere Konzepte hinzukommen. Die Abbildung zeigt beispielsweise eine Datenbank auf der Ereignisse persistent abgelegt werden können und die Einteilung von Ereignissen in Klassen, sogenannte Topics, die die Handhabung von verschiedenen Ereignisarten über ein System ermöglichen.

[10] [Go21, S. 249]

[11] Für diese Komponenten finden sich in der Fachliteratur verschiedene Bezeichnungen. Außer Publisher und Subscriber findet man noch Producer und Receiver oder Producer und Listener.

Hieraus ergeben sich ein paar grundlegende Fragen. Die Spezifikation der Event Platform entscheidet, wie genau Events aufgebaut sein müssen und wie Publisher und Subscriber angebunden werden. Auch weitere Überlegungen bezüglich Analyse des Ereignisflusses, Abstraktion von Ereignissen und Kompatibilität zu anderen Plattformen fallen auf Ebene der Event Platform an. Die Event Platform ist also der zentrale Baustein für die technische Umsetzung einer EDA.[12]

Technische Umsetzung von EDA

Aus den bis hierher besprochenen Grundlagen ergeben sich einige technische Überlegungen, die bei dem Aufbau einer EDA gefasst werden müssen.

Abbildung 2: Beispiel für ein Ereignismodell nach dem offen Standard 'CloudEvents' [13]

Ereignisse als grundlegendes Konzept der EDA müssen in einem Ereignismodel beschrieben werden. Dieses Modell muss alle relevanten Informationen über ein Ereignis vermitteln, es sollten aber auch weitere Anforderungen an das Modell beachtet werden. Das Ereignis muss technologisch kompatibel zu allen Publishern und Subscribern sein, die an das System angeschlossen werden sollen. Es sollte zu den Nutzdaten Metadaten enthalten, die eine analytische Betrachtung des Ereignisstroms zulassen. Weiterhin sollte immer betrachtet werden, auf welcher Abstraktionsebene das Ereignis agiert, hier kann

[12] Vgl. Go21, S. 244.
[13] Eigene Darstellung nach einem Beispiel der CloudEvents Dokumentation [CloJ]

beispielsweise die technische Unterscheidung von technischen und Anwendungsereignissen sinnvoll sein. Was jedoch nie im Ereignismodell enthalten ist, ist die Verarbeitungslogik, diese liegt allein bei den Subscribern.[14] In der Anwendung werden Ereignisse häufig als strukturierter Datentyp dargestellt, JSON oder XML als Datenformat sind üblich. Ein Beispiel findet sich in Abbildung 2.

Die Architektur der Event Platform hat, wie schon angerissen, besondere Relevanz. Im Grunde unterscheiden sich zwei gängige Topologien, die hier angewandt werden können: Die 'Mediator-Topology' und die 'Broker-Topology'.

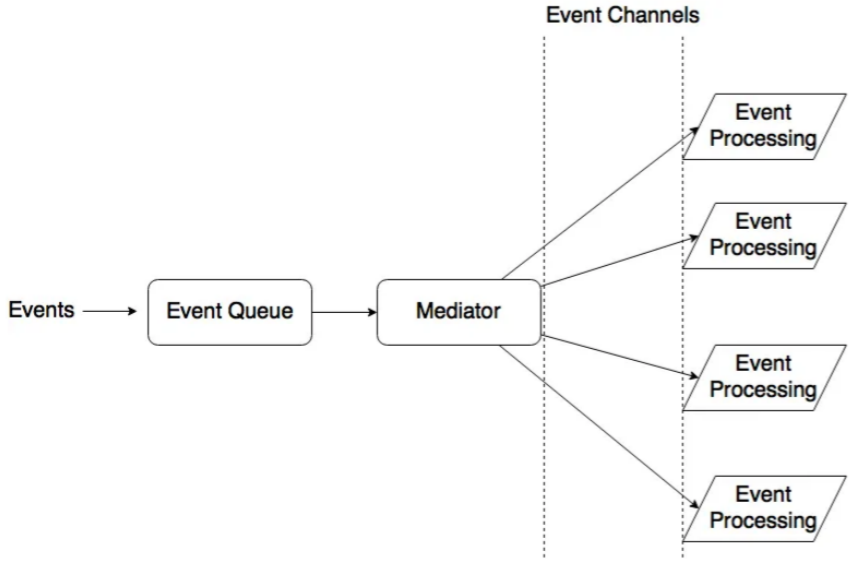

Abbildung 3: Mediator-Topology [15]

Die 'Mediator-Topology' sieht eine zentrale Event-Queue vor, in die Ereignisse eingespeist werden. Diese werden dann an die verschiedenen Subscriber verteilt, die anhand verschiedener Topics[16] gruppiert werden. Im Grunde werden in dieser Topologie also die ursprünglichen Ereignisse konsumiert und daraus folgend Ereignisse für jeden Channel, an den es gesendet werden soll, erzeugt und weitergegeben. Die Idee hinter diesem Prinzip ist es, auch Ereignisse, deren Verarbeitung mehrere Schritte benötigt orchestrieren zu können.[17]

[14] Vgl. Br10, S. 95.
[15] [Wi17]
[16] In der Abbildung 3 als Channels bezeichnet
[17] Vgl. Wi17.

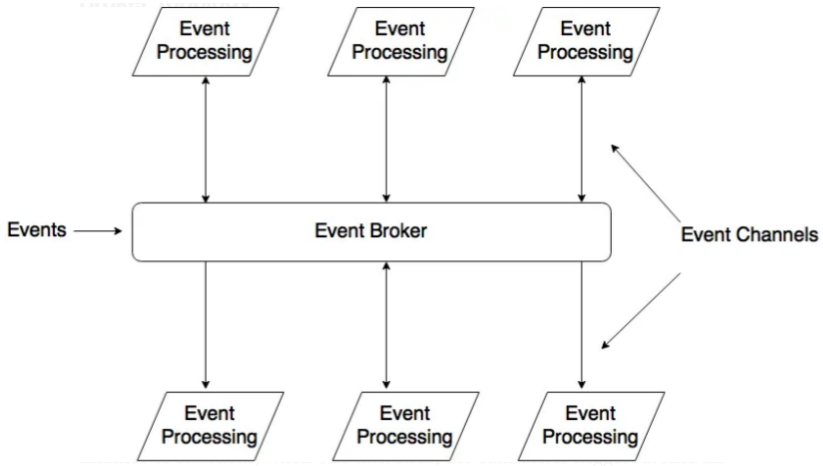

Abbildung 4: Broker-Topology [18]

Die 'Broker-Topology' sieht keine zentrale Event-Queue vor. Hier werden die Ereignisse stattdessen zentral an alle relevanten Channels verteilt, ohne sie weiterzuverarbeiten. Daraus folgt, dass wenn Ereignisse mehrschrittig abgearbeitet werden müssen eine Lösung mit Callback-Ereignissen gefunden werden muss. In Abbildung 4 ist dies an Doppelpfeilen zu erkennen, die zwischen manchen Subscribern und dem Broker laufen.[19]

2.2 RESTful API

API

Martin Reddy definiert in der Einleitung seines Buch "API Design for C++"den Begriff des Application Programming Interface (API) als Abstraktion eines Problems und die zugehörige Spezifikation mit der ein Anwender mit einem Software-Komponenten interagieren sollte, welcher eine Lösung für dieses Problem implementiert.[20] Eine API ist also vordergründig eine Spezifikation, welche es ermöglicht mit Software zu interagieren. Das bedeutet sowohl, dass eine Maschine zu Maschine möglich wird, wenn die Formalisierung der Spezifikation stark genug ist, als auch, dass man nach dieser Definition eine gewöhnliche Anwendungssoftware als API, die für eine Anwender-Maschine gedacht ist, begreifen

[18] [Wi17]
[19] Vgl. Wi17.
[20] Vgl. Re11, S. 1.

kann. Im Fachjargon der Softwareentwicklung verwendet man den Begriff der API jedoch vordergründig, um ersteren Fall zu beschreiben. Eine praktischere Definition des Begriffes ist es also, die API als Verbindungsstück nach außen, also als Schnittstelle, einer Software-Komponente zu verstehen.

Eine für den Zweck der Arbeit besonders relevante Klasse von APIs sind sogenannte Web-APIs. Web-APIs sind Software-Schnittstellen, die sich das World Wide Web (WWW) zur Nutze machen und über Protokolle des Internets angesprochen werden können. Für solche APIs haben sich über die Entwicklung des Internets eine Reihe verschiedener Standards und Architekturmuster enwtickelt, das heute eingängigste ist jedoch wahrscheinlich der REST-Standard.

Cite!

Cite!

REST

Den Begriff Representational State Transfer (REST) prägte erstmals Roy Fielding in seiner Dissertation im Jahr 2000. Er beschreibt damit ein Architekturprinzip für verteilte Systeme, das auf dem WWW aufbaut.[21] APIs, die dem REST Standard folgen, sogenannte RESTful APIs, sind mittlerweile ein häufig anzutreffendes Design Muster in der Softwarearchitektur. Dabei beschreibt Fielding mit REST an sich eigentlich kein solches Muster, sondern legt viel mehr eine Reihe von Anforderungen fest, die erfüllt sein müssen, damit eine API RESTful genannt werden kann.[22] Die sechs Anforderungen, die Fielding definiert, sollen im folgenden kurz beschrieben werden:

- **Einheitliche Schnittstelle:** Die API muss eine einheitliche Schnittstelle für alle Clients bereitstellen. Das bedeutet, dass alle Clients die gleichen Methoden verwenden, um mit der API zu interagieren. Im Kontext des WWW sind das beispielsweise die Methoden des Hypertext Transfer Protocol (HTTP) Protokolls.

- **Client-Server Architektur:** Die API unterscheidet zwischen Client und Server, wobei diese Komponenten minimal gekoppelt sind, das heißt, im wesentlichen unabhängig in ihrer internen Funktionsweise. Client und Server kommunizieren über ein gemeinsames Protokoll, das ihre gesamte Abhängigkeit kapselt.

[21] Vgl. Fi00, S. 76.
[22] Vgl. RR07, S. XV.

- **Zustandslosigkeit:** Die Kommunikation zwischen Client und Server ist zustandslos. Das bedeutet, dass der Server keine Informationen über den Zustand des Clients speichert und der Client bei jeder Anfrage alle Informationen, die der Server benötigt, um die Anfrage zu bearbeiten, mitgibt.

- **Caching:** Die API muss die Möglichkeit bieten, Antworten auf Anfragen zu cachen. Zur Optimierung des Diensts kann der Server also Daten mitgeben, die eine Gültigkeitsdauer haben und vom Client für diese Zeit zwischengespeichert werden können. Dadurch ist ein schnellerer Zugriff auf diese Daten möglich. Das Verfahren nennt man Caching.

- **Schichtensystem:** Die API muss ein Schichtensystem unterstützen. Das bedeutet, dass der Client nicht direkt mit dem Server kommuniziert, sondern die Anfrage über eine Reihe von Zwischenstationen an den Server weitergeleitet werden kann. Diese Zwischenstationen können beispielsweise Firewalls oder Load Balancer sein.

- **Code on Demand:** Die API muss die Möglichkeit bieten, Code an den Client zu senden, der dort ausgeführt wird. Das bedeutet, dass der Server nicht nur Daten an den Client sendet, sondern auch ausführbaren Code. Dieser Code kann dann beispielsweise die Darstellung der Daten übernehmen. Hierbei handelt es sich um die einzige optionale Anforderung, die Fielding definiert.[23]

RESTful APIs in der Praxis des WWW

Seit Fielding diese Anforderung im Jahr 2000 definiert hat, ist der daraus erwachsene Design-Ansatz für APIs zu einem der populärsten Muster im Entwurf von Web-APIs geworden und die rapide Entwicklung in der IT-Branche hat natürlich auf vor diesem Thema keinen Halt gemacht. In der modernen API-Entwicklung für REST spielen sogenannte Ressourcen eine entscheide Rolle, nach denen die Struktur des Dienstes modelliert wird. Eine Ressource ist dabei ein inhaltliches Element, das Gegenstand der API ist. Eine Ressource könnte beispielsweise eine Produktinformation zu einem Produkt, der Bestand eines

[23] Vgl. Re20.

Lagers oder die nächste Lieferung, die im Lager eintreffen soll sein.[24] Wird ein Dienst so modelliert, dass sich Anfragen an diesen immer auf solche Ressourcen beziehen, so lassen sich einfach die von Fielding definierten Anforderungen einhalten.

Man könnte sich als Beispiel eine Web API vorstellen, deren einzige Aufgabe es ist, die Temperaturdaten für eine spezifische Region zurückzugeben. Eine Ressource wäre also die Temperatur an einem bestimmten Ort zum aktuellen Zeitpunkt. Als Web API können inhärent die ersten beiden Punkte der Liste an Anforderungen abgehakt werden. Das Web legt eine Client-Server-Architektur zugrunde und die Kommunikation mit HTTP ist einheitlich. Interessant wird die Betrachtung der Zustandslosigkeit der API. Man könnte annehmen, dass das wechselnde Wetter durchaus einen Zustand darstellt und somit die REST Spezifikationen nicht mehr eingehalten werden. Die Definition gibt aber vor, dass Zustandslosigkeit nicht bedeutet, dass die API deterministisch sein muss, viel mehr wird nur verlangt, dass der Server keine Informationen über den Zustand des Clients speichert. Es darf also keine persistente Session geben, oder in anderen Worten, der Client muss mit jeder Anfrage sämtliche Informationen mitliefern, die der Server zur Verarbeitung derselben benötigt. Das ist hier durchaus der Fall. Der Client gibt mit jeder Anfrage an, für welchen Ort er die Temperatur zurückgegeben haben möchte. Das reicht dem Server aus, um die gefragte Information zurückzuliefern und er muss keine weiteren Informationen über den Zustand des Clients speichern. Weiterhin ist es in diesem Beispiel möglich die Temperaturdaten mit einem Gültigkeitszeitraum zu versehen, sodass Caching möglich wäre. Code on Demand wäre in diesem Beispiel nicht sinnvoll, ist aber auch nur optional.

2.3 Technologie im Anwendungsbeispiel

Cloud Events

Cloud Events sind ein Standard für ein Ereignismodell[25], um Ereignisse allgemeingültig beschreiben zu können. Von der Cloud Native Computing Foundation entwickelt, beschreibt er, wie Ereignisse aufgebaut sein müssen, um diese allgemeingültig verarbeiten und so eine Unabhängigkeit zwischen Publisher und Subscriber gewährleisten zu können. Der

[24] Vgl. RR07, S. 81.
[25] Siehe Abschnitt 2.1

Standard hat mittlerweile weite Anwendung in der Industrie gefunden und wird unter anderen von Firmen wie Google, IBM und SAP in EDA-Lösungen verwendet. Cite! Der Standard unterstützt dabei unterschiedliche Protokolle und gibt hauptsächlich vor, welche Metadaten über das Ereignis angegeben werden müssen. So trifft er keine Aussage über die Struktur der Nutzdaten im Ereignis, sondern spezifiziert viel mehr, dass Beispielweise eine Information über den Publisher vorhanden sein muss, die Zeit und das Datum angegeben sein muss, zu der das Ereignis gesendet wurde oder eine Versionierung des Ereignisses erkennbar ist. Die Zielsetzung des Standards ist es somit nicht, inhaltliche Kompatibilität herzustellen, sondern schlicht die korrekte Verarbeitung und Weiterleitung der Ereignisse auf Seite der Event-Platform zu gewährleisten. Diese kann, da die Ereignisse, die sie verarbeitet, einem Standard folgen, Ereignisse von verschiedensten Publishern annehmen. Zudem kann, da Cloud Events für verschiedene Protokolle definiert ist, mit verschiedenen Protokollen an sie angeschlossen werden und noch mehr Unabhängigkeit geboten werden. Cloud Events selbst definiert Ziel so, ”die Interoperabilität von Ereignissystemen zu definieren, die es Diensten ermöglichen, Ereignisse zu produzieren oder zu konsumieren, wobei Produzenten und Konsumenten unabhängig voneinander entwickelt und eingesetzt werden können.” Cite!

RAP und Business Objects

Das ABAP RESTful Application Programming Model (RAP) ist ein Programmiermodel, das eine Reihe von Konzepten, Sprachen und Frameworks einschließt, die zusammen die Möglichkeit bieten im SAP Umfeld Applikationen unter Verwendung der Paradigmen einer RESTful API zu entwickeln. Cite!

Den Kern der Entwicklung nach diesem Modell bildet die Arbeit mit sogenannten Business Object (BO)s. Es handelt sich bei diesen um das Konzept von hierarchisch aufgebauten Objekten, die den Zugriff auf Daten und Aktionen, sogenannte Behaviors zu ermöglichen. An diese BOs lassen sich zudem weitere Dienste anknüpfen, die beispielsweise die Datenstruktur in ein User-Interface umsetzen, aus ihr eine API generieren oder Ähnliches. Ein BO kann dabei ein beliebiges Objekt aus der echten Welt modellieren, so könnte es beispielsweise ein Produkt, eine Reise oder einen Verkaufsabschluss mit

den zugehörigen Daten repräsentieren. Unter einem Hauptknoten eines solchen Objektes hängen dann weitere zugehörige Daten, aber auch Aktionen. Bei diesen kann es sich zum Beispiel um gewöhnliche transaktionale Operationen wie erstellen, löschen oder ändern handeln, aber auch dem Anwendungsfall spezifische Operationen, wie die Weiterverarbeitung eines Produktes oder die Genehmigung einer Reise sind denkbar. Cite!

Business Events

Mit Business Events können Konzepte von EDA im SAP Umfeld umgesetzt werden. Als Business Event wird ein Ereignis bezeichnet, das durch ein Business Object im Zuge eines Behaviors erzeugt wird. Wie in Abschnitt 2.3 bereits erwähnt, folgen solche Ereignisse im SAP Umfeld dem Standard Cloud Events. Dem Ereignis werden also einige Metadaten mitgegeben, anhand derer es weiter verarbeitet wird. Der Producer eines Business Events ist also ein Business Object.

Event Consumption Model

Der Subscriber im SAP Umfeld ist das sogenannte Event Consumption Model. Ähnlich wie ein Business Object handelt es sich hierbei um eine Reihe von hierarchisch miteinander verknüpften Artefakten. Bereitgestellt wird beispielsweise eine ABAP-Klasse, die die Verarbeitung des Ereignisses übernimmt, aber auch ein Service, mit dem sich der Subscriber an die Event Platform anbinden lässt. Cite!

Event Mesh

Die Event Platform im SAP Umfeld setzt sich aus mehreren Teilen zusammen. Der Kern ist das sogenannte Event Mesh. Dieses ist ein Dienst der Business Technology Platform (BTP), der SAP eigenen Cloud Platform. Er ist die zentrale Instanz, die die Ereignisse annimmt und weiterleitet. Er ist in der Lage, die Ereignisse zu konsolidieren, zu filtern und zu transformieren. Weiterhin sind Event Consumption Model als Subscriber und das BO als Publisher über verschiedene Komponenten des Enterprise Event Enablement Frameworks an den Event Mesh angebunden. Cite! Cite!

2.4 Forschungsmethodik

Protoyping

In ihrem Buch ”Wirtschaftsinformatik - Einführung und Grundlegung”definieren Heinrich u.A. einen Prototypen wie folgt: ”Ein Prototyp ist ein mit geringem Aufwand hergestelltes und einfach zu änderndes, ausführbares Modell des geplanten, im Entwicklungsprozess befindlichen Systems, das erprobt und beurteilt werden kann”[26] Als Methode zum Erkenntnisgewinn kann die Erstellung eines Prototypes also insofern eingesetzt werden, als das durch den explorativen Prozess des Erstellens des Prototypes sowohl über den Gegenstand des Prototypes, als auch über die Methode des Erstellens Erkenntnisse gewonnen werden können.[27] In dieser Arbeit soll die Erstellung es solchen Prototyps dazu verwendet werden die bis jetzt theoretisch erarbeiteten Konzepte in einem praktischen Beispiel anzuwenden und so die Erkenntnisse aus der Theorie zu vertiefen.

2.5 Zusammenfassung des theoretischen Teils

[26] HHR07, S. 114.
[27] HHR07, S. 119.

3 Anwendung in der Praxis

3.1 Implementierung eines Prototyps

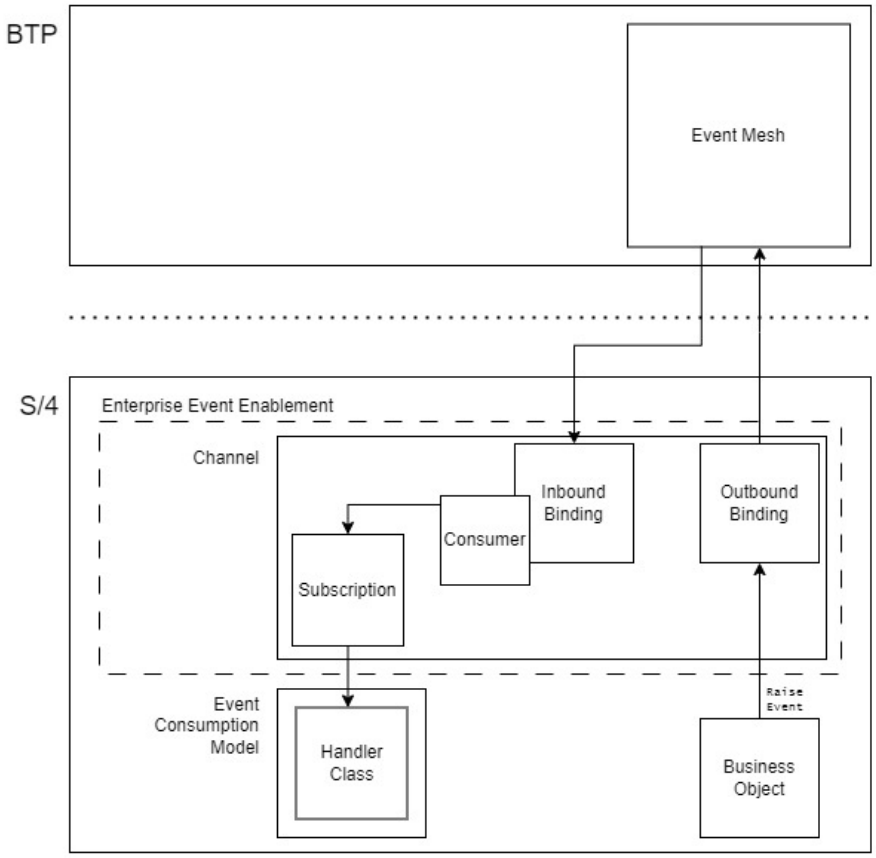

Abbildung 5: Eventing in SAP S/4 unter Verwendung des BTP Event Mesh [28]

[28] Eigene Darstellung

4 Diskussion der Ergebnisse

4.1 Bewertung des Prototyps

4.2 Beurteilung von EDA und REST

4.3 Chancen der Technologie im betriebswirtschaftlichen Kontext

5 Resümee

5.1 Zusammenfassung der wichtigsten Ergebnisse

5.2 Handlungsempfehlung

5.3 Kritische Reflexion der Arbeit und Ausblick

Quellenverzeichnis

Bücher

[Br10] Bruns, R.: Event-Driven Architecture : Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ISBN: 9783642024399.

[Go21] Goniwada, S. R.: CLOUD NATIVE ARCHITECTURE AND DESIGN : a handbook for modern day architecture and design with... enterprise-grade examples. Apress, 2021.

[HHR07] Heinrich, L.; Heinzl, A.; Roithmayr, F.: Wirtschaftsinformatik-Einführung und Grundlegung (3. Ausg.) 2007.

[Re11] Reddy, M.: API Design for C++. Elsevier, 2011.

[RR07] Richardson, L.; Ruby, S.: Web-services mit REST. O'Reilly Germany, 2007.

Artikel

[Sc03] Schulte, R. W.: The growing role of events in enterprise applications. Gartner Research 7/, 2003.

Internetquellen

[CloJ] Cloud Events: JSON Event Format for CloudEvents - Version 1.0.2, o.J. URL: https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/formats/json-format.md#1-introduction, Stand: 27. 06. 2023.

[Re20] Red Hat: Was ist eine REST-API?, www.redhat.com, Mai 2020, URL: https://www.redhat.com/de/topics/api/what-is-a-rest-api, Stand: 17. 07. 2023.

[SAoJ] SAP SE: Was ist SAP? | Definition & Bedeutung | SAP Abkürzung, o.J. URL: https://www.sap.com/germany/about/company/what-is-sap.html, Stand: 14. 06. 2023.

[Wi17] Wickramarachchi, A.: Event Driven Architecture Pattern, Medium, Sep. 2017, URL: https://towardsdatascience.com/event-driven-architecture-pattern-b54fc50276cd, Stand: 27. 06. 2023.

Anhang

1. Digitale Version der Arbeit
2. Interviews
    2.1. Expertmann 2018