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

welcher ansatz 2
welche frage 2
Cite! 12
Cite! 13
Cite! 14
Cite! 14
Cite! 14

Inhaltsverzeichnis

Selbstständigkeitserklärung **II**

Inhaltsverzeichnis **IV**

Abkürzungsverzeichnis **VI**

Abbildungsverzeichnis **VII**

1 Einleitung **1**

1.1 Motivation und Problemstellung 1

1.2 Zielsetung 2

1.3 Abgrenzung 2

1.4 Vorgehensweise und Aufbau der Arbeit 2

2 Theoretischer Hintergrund **3**

2.1 Event-Driven Architecture 3

2.2 RESTful API 8

2.3 Technologie im Anwendungsbeispiel 12

2.4 Forschungsmethodik 14

2.5 Zusammenfassung des theoretischen Teils 15

3 Anwendung in der Praxis **16**

3.1 Implementierung aufseiten der BTP 16

3.2 Implementierung aufseiten des SAP S/4 Systems 19

4 Diskussion der Ergebnisse **22**

4.1 Bewertung des Prototyps 22

4.2 Beurteilung von EDA und REST im Kontext des Anwendungsbeispiels . . . 23

4.3 Chancen der Technologie im betriebswirtschaftlichen Kontext 25

5 Resümee **28**

5.1 Zusammenfassung der wichtigsten Ergebnisse 28
5.2 Handlungsempfehlung 28
5.3 Kritische Reflexion der Arbeit und Ausblick 29

Quellenverzeichnis **VIII**

Anhang **X**

Abkürzungsverzeichnis

ABAP Advanced Business Application Programming

ADT ABAP Development Tools

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 ABAP RESTful Application Programming Model

REST Representational State Transfer

WWW World Wide Web

HTTP Hypertext Transfer Protocol

Abbildungsverzeichnis

1 Komponenten der Event-Driven Architecture 5
2 Beispiel für ein Ereignismodell 6
3 Mediator-Topologie 7
4 Broker-Topologie 8
5 BTP Event-Mesh 17
6 Message-Channel Konfiguration 18
7 Screenshot des Prototyps 19
8 Eventing in SAP S/4 20
9 Kosten Nutzen EDA 27

1 Einleitung

1.1 Motivation und Problemstellung

Eine der größten Schwierigkeiten bei der Entwicklung von Softwareprojekten im größeren Maßstab ist es, die Komplexität der Software zu beherrschen. Schnell passiert es, dass über die Verbindung von vielen verschiedenen Softwarekomponenten mit verschiedenen Schnittstellen die Komplexität der Software rapide ansteigt. Dies führt zu einer Reihe von Problemen, die die Entwicklung und den Betrieb der Software erschweren. [Vgl. Pr05, S. 5] Einen Ansatz zur Beherrschung dieser Komplexität bietet die Event Driven Architecture (EDA). Sie verspricht, durch den Fokus auf Ereignisse beim Systementwurf 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. [Vgl. Br10, S. 8f]

Den betrieblichen Kontext dieser Überlegungen stellt die SAP SE. Die SAP SE ist ein deutsches Softwareunternehmen, das seit 1972 Unternehmenssoftware entwickelt. Heute beschäftigt es mehr als 100000 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. [Vgl. SAoJ]

Das HCM ist die Personallösung des SAP ERP und kommt bis heute in einer großen Anzahl von 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, die sie mit einer immensen Größe und Anzahl von Komponenten vereinen müssen.

1.2 Zielsetung

Das Ziel der Arbeit soll es sein diesen Ansatz näher zu untersuchen. Im Umfeld von HCM soll im Rahmen eines Migrationsprojektes eine Applikation von Fiori Freestyle auf Fiori Elements unter Verwendung des SAP eigenen Representational State Transfer (REST) Frameworks, dem ABAP RESTful Application Programming Model (RAP) 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 SAP Kontext erstellt werden, der eine EDA implementiert. So soll exemplarisch geprüft werden, wie der Prozess bei einem solchen Vorhaben aussehen könnte und daraus folgend ein allgemeines Urteil über den Architekturansatz gefällt werden. Der Prototyp dient somit als Grundlage für die Beantwortung der verallgemeinerbaren Überlegung, ob es sich lohnt, die EDA in komplexen Softwareumfeldern einzusetzen.

welcher ansatz

1.3 Abgrenzung

In der Arbeit soll es darum gehen, einen Protoypen zu bauen und zu erforschen, wie sich die Anwendung von EDA in komplexen Systemen auswirkt. Es soll keine theoretische Grundlagenforschung zum Architekturkonzept betrieben werden oder eine umfassende Analyse der Vor- und Nachteile aus theoretischer Perspektive durchgeführt werden. Die Arbeit soll sich auf die praktische Anwendung des Konzepts beschränken und die gewonnenen Erkenntnisse in einen betriebswirtschaftlichen Kontext einordnen.

1.4 Vorgehensweise und Aufbau der Arbeit

Um die genannte Frage zu beantworten, ist es als ersten Schritt der Arbeit notwendig, den aktuellen Forschungsstand näher zu untersuchen und die Grundlagen sowohl der EDA als auch den technischen Kontext einer RESTful API zu erarbeiten. Dafür sollen in einer strukturierten Literaturrecherche die wissenschaftlichen Grundlagen erarbeitet werden. Die Struktur dieses theoretischen Teils ordnet sich nach Thema. Darauffolgend soll die Entwicklung des oben erwähnten Prototypen beschrieben werden, welche auf der wissenschaftlichen Methodik des Prototyping basiert, die unter Abschnitt 2.4 noch etwas

welche frage

ausführlicher beschrieben ist. Hierbei soll zuerst ein abstrahierender Überblick über die Implementierungsschritte gegeben werden und darauffolgend ein Schluss über das Ergebnis gezogen werden. Auf Basis der daraus gewonnen Erkenntnisse und in Verknüpfung mit den vorher dargelegten theoretischen Grundlagen sollen dann weiterführende Schlüsse zur Beantwortung der Fragestellung verdichtet werden.

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. [Vgl. Ch06, S. 4] 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. [Vgl. Sc03, S. 2] 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. [Vgl. Br10, S. 13] 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. [Vgl. Br10, S. 16] Eine im Kontext der Unternehmenssoftware relevante Unterscheidung ist dabei die zwischen technischen und Anwendungsereignissen. Anwendungsereignisse sind 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. [Vgl. Go21, S. 245f]

Ereignisorientierung als Architekturansatz

Zuerst einmal handelt es sich bei EDA[1] 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. [Vgl. Br10, S.30] 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. [Vgl. Go21, S. 248]

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

Technische Grundkonzepte der EDA

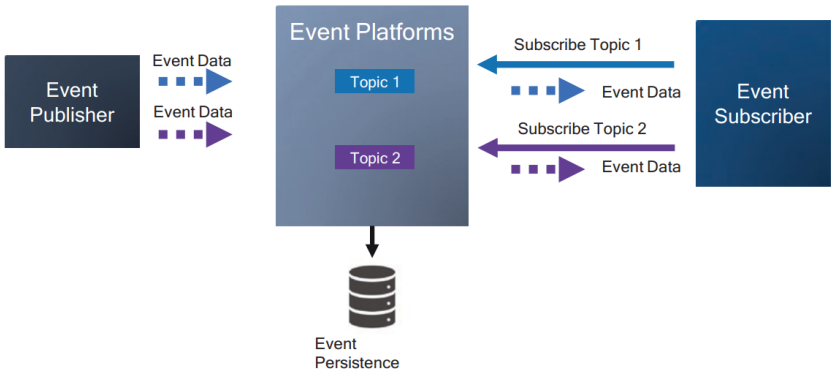


Abbildung 1: Komponenten der EDA[2]

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.[3] 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.

[2] [Go21, S. 249]

[3] 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. [Vgl. Go21, S. 244]

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' [4]

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

[4] Eigene Darstellung nach einem Beispiel der CloudEvents Dokumentation [Cl23c]

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. [Vgl. Br10, S. 95] 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'.

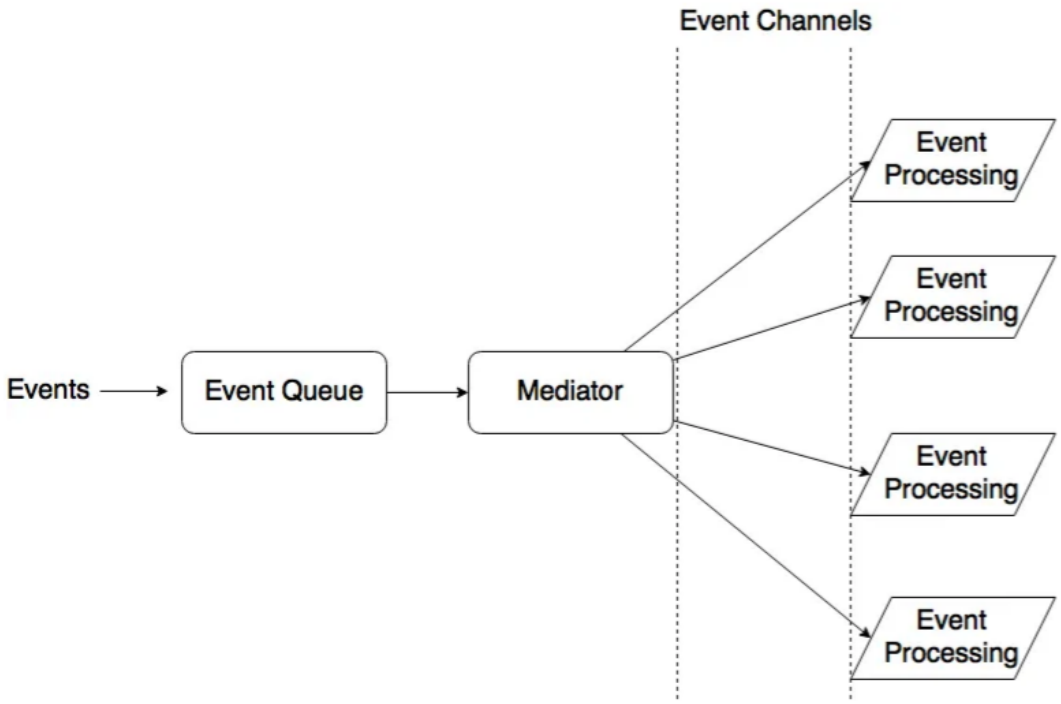


Abbildung 3: Mediator-Topologie [5]

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[6] 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

[5] [Wi17]
[6] In der Abbildung 3 als Channels bezeichnet

können. [Vgl. Wi17]

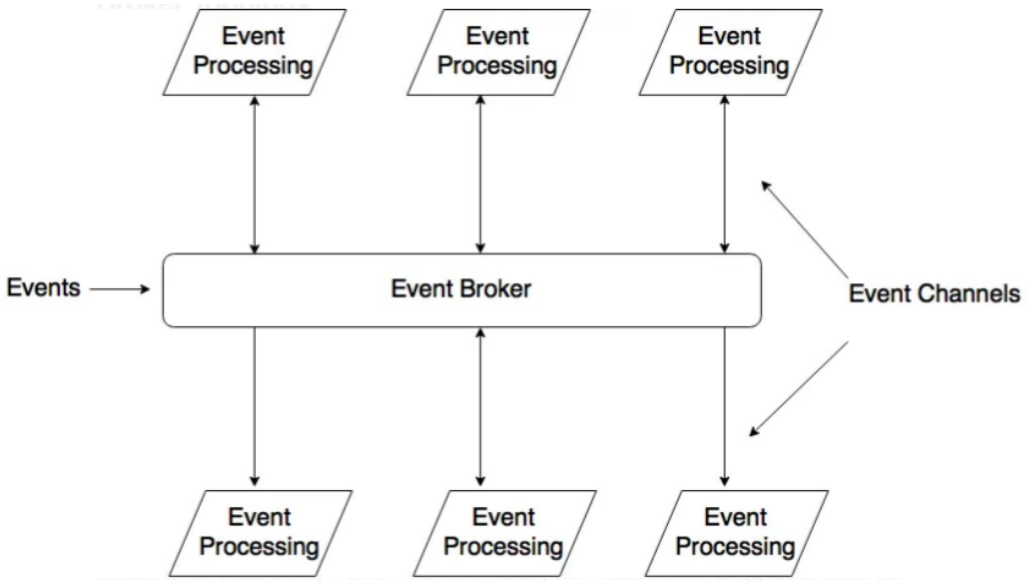


Abbildung 4: Broker-Topologie [7]

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. [Vgl. Wi17]

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. [Vgl. Re11, S. 1] Eine API ist also vordergründig eine Spezifikation, welche es ermöglicht mit Software zu interagieren.

[7] [Wi17]

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 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. [Vgl. RR07, S.5f]

REST

Den Begriff REST prägte erstmals Roy Fielding in seiner Dissertation im Jahr 2000. Er beschreibt damit ein Architekturprinzip für verteilte Systeme, das auf dem WWW aufbaut. [Vgl. Fi00, S. 76] 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. [Vgl. RR07, S. XV] 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 unabhän-

gig in ihrer internen Funktionsweise. Client und Server kommunizieren über ein gemeinsames Protokoll, das ihre gesamte Abhängigkeit kapselt.

- **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. [Vgl. Re20]

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 auch 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 Lagers oder die nächste Lieferung, die im Lager eintreffen soll sein. [Vgl. RR07, S. 81] 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 wirklich sinnvoll, ist aber auch nur optional.

2.3 Technologie im Anwendungsbeispiel

Cloud Events

Cloud Events sind ein Standard für ein Ereignismodell[8], 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 Standard hat mittlerweile weite Anwendung in der Industrie gefunden und wird unter anderen von Firmen wie Google, IBM und SAP in EDA-Lösungen verwendet. [Cl23a, Vgl.] 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.” [Cl23b, Vgl.]

RAP und Business Objects

Das 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!

[8] Siehe Abschnitt 2.1

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. Um ein BO anzulegen, werden einige Artefakte benötigt, die wichtigsten sind dabei vielleicht die Behavior Definition und die Behavior Implementation. In ersterer wird festgelegt, welche Daten und Aktion das BO enthält, zweitere ist eine Advanced Business Application Programming (ABAP)-Klasse, deren Methoden aufgerufen werden, wenn spezifische Aktionen des BO von der Laufzeitumgebung angefragt werden. 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 Cite! Publisher über verschiedene Komponenten des Enterprise Event Enablement Frameworks an den Event Mesh angebunden. 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”[HHR07, S. 114] 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. Hierzu vor allem nicht nur die Erstellung des Prototyps selbst wichtig, sondern auch die strukturierte Bewertung des Prototyps im Nachgang. [HHR07, S. 119] 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

Ereignisse im Kontext von EDA wurden als signifikante Änderungen des Zustands definiert. Diese Ereignisse können sowohl geschäftliche Vorfälle als auch technische Ereignisse umfassen. EDA ist ein Ansatz zur Modellierung von Prozessen, der Ereignisse statt aufeinanderfolgender Schritte in den Vordergrund stellt. In einer EDA stellen Publisher und Subscriber die Hauptkomponenten dar, wobei die zentrale Plattform zur Verarbeitung der Ereignisse die wichtigste Komponente ist. RESTful APIs sind eine populäre Methode für die Gestaltung von Web-APIs. Sie folgen den Prinzipien des REST-Architekturmodells, das von Roy Fielding definiert wurde. Dieses Modell legt eine Reihe von Anforderungen fest, die eine API erfüllen muss, um als RESTful bezeichnet zu werden. Weiterhin wurden verschiedene Technologien und Methoden zur Umsetzung von EDA und RESTful APIs im SAP-Umfeld vorgestellt. Dazu gehören Cloud Events, ein Standard für Ereignismodelle, das RAP und Business Objects zur Entwicklung von Applikationen, Business Events als Konzept für EDA im SAP Umfeld, das Event Consumption Model als Subscriber und das Event Mesh als Event Platform. Abschließend wurde das Konzept des Prototyping als Forschungsmethode vorgestellt, das in dieser Arbeit zur Anwendung und Vertiefung der theoretischen Erkenntnisse genutzt wird. Ein Prototyp wird dabei als ein ausführbares Modell des geplanten Systems definiert, das mit geringem Aufwand erstellt und einfach geändert werden kann.

3 Anwendung in der Praxis

Das Ziel dieses Abschnitts ist es, darzustellen wie die bisher dargelegten Konzepte in der Entwicklung eines Prototyps angewandt wurden und somit ein ’Proof of Concept’ erstellt wurde. Hierzu soll zuerst der Implementierungsprozess aufseiten der Event-Platform, und daran anschließend von Publisher und Subscriber dargestellt werden.

3.1 Implementierung aufseiten der BTP

Der Prototyp soll aus einer Beispielanwendung bestehen, die auf Basis eines BO eine Fiori UI Frontend Anwendung bereitstellt. Die Anwendung soll dabei die grundlegenden Funktionen eines BO implementieren, also das Erstellen, Lesen, Ändern und Löschen von Datensätzen. Die Anwendung soll auf einem SAP S/4 System laufen und die Daten in der HANA Datenbank des Systems speichern. Zudem soll in jeder Speichersequenz des BOs ein Business Event ausgelöst werden. An anderer Stelle im System soll ein Subscriber auf das Ereignis reagieren und es sowohl in einer Log-Datenbank persistieren als auch eine Notification an einen Benutzer senden.
Die inhaltliche Grundlage bildet nach SAP-Kovention eine Anwendung aus dem Bereich des Reisemanagements. Das BO ’Travel’ soll hierbei die Daten einer Reise speichern. Es sollen hierbei Datensätze erstellt, verändert und gelöscht werden können. Relevant ist vor allem die Speicheraktion, denn hier soll die zusätzliche Programmlogik implementiert werden, die das Ereignis generiert. Um zu prüfen, wie mit Nutzdaten eines Ereignisses umgegangen werden muss, sollen zusätzlich ein Code und eine Beschreibung in das Ereignis geschrieben werden.

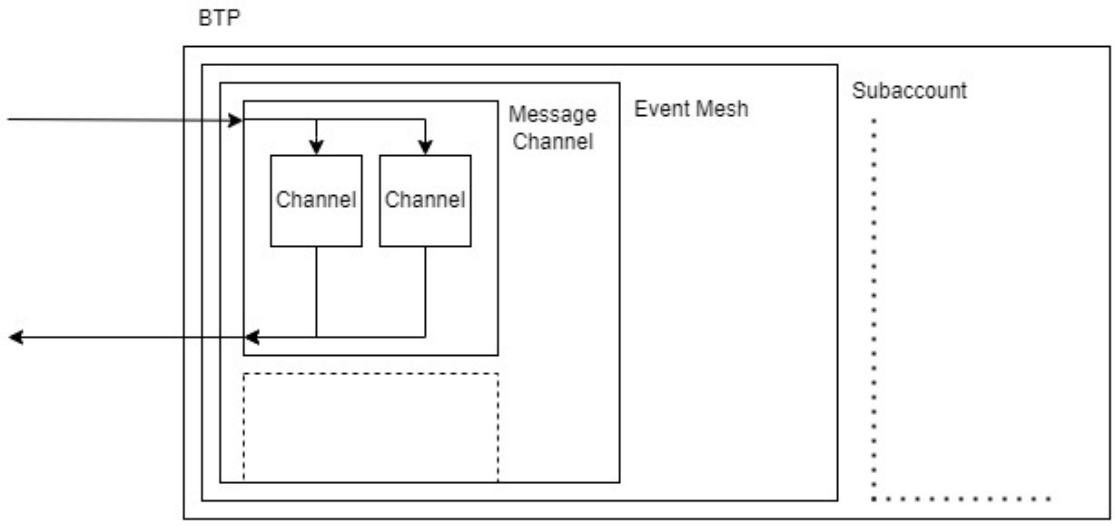


Abbildung 5: Aufbau des BTP Event-Mesh [9]

Wie bereits im theoretischen Teil der Arbeit kurz angerissen und in Abbildung 5 dargestellt soll ein SAP-spezifischer Cloud-Dienst verwendet werden, der als Event-Platform fungieren soll. Die Konfiguration dieses Dienstes beginnt mit der Einrichtung eines Unteraccounts innerhalb einer BTP Instanz. Dieser Unteraccount wird dann mit dem Event-Mesh Dienst verknüpft. Der Dienst kann hierbei über ein User Interface administriert werden. Hier kann dann ein sogenannter Message-Client erstellt werden. Es handelt sich dabei um die aktive Komponente des Event Mesh, die Nachrichten senden und empfangen kann. Auf Ebene des Message-Clients können zudem Regeln konfiguriert werden, die Nachrichten und Ereignisse nach Themen und Absender filtern. Innerhalb des Message-Clients können dann sogenannte Queues erstellt werden. Diese Queues sind die eigentlichen Kommunikationskanäle, über die Nachrichten ausgetauscht werden. Queues horchen auf Ereignisse mit bestimmten Themen und halten diese so lange bis sie von einem Subscriber konsumiert werden.

Im Kontext des Prototypen kann einmal durchgegangen werden, wie ein Ereignis, das durch die Änderung einer Reisebuchung ausgelöst wurde, von dem Event Mesh verarbeitet wird. Um ein Ereignis an ein Message-Channel zu senden, muss ein sogenanntes Service Binding generiert werden. Hierbei handelt es sich um ein Artefakt,

[9] Eigene Darstellung

Abbildung 6: Regeln für die Konfiguration des Message-Channels [11]

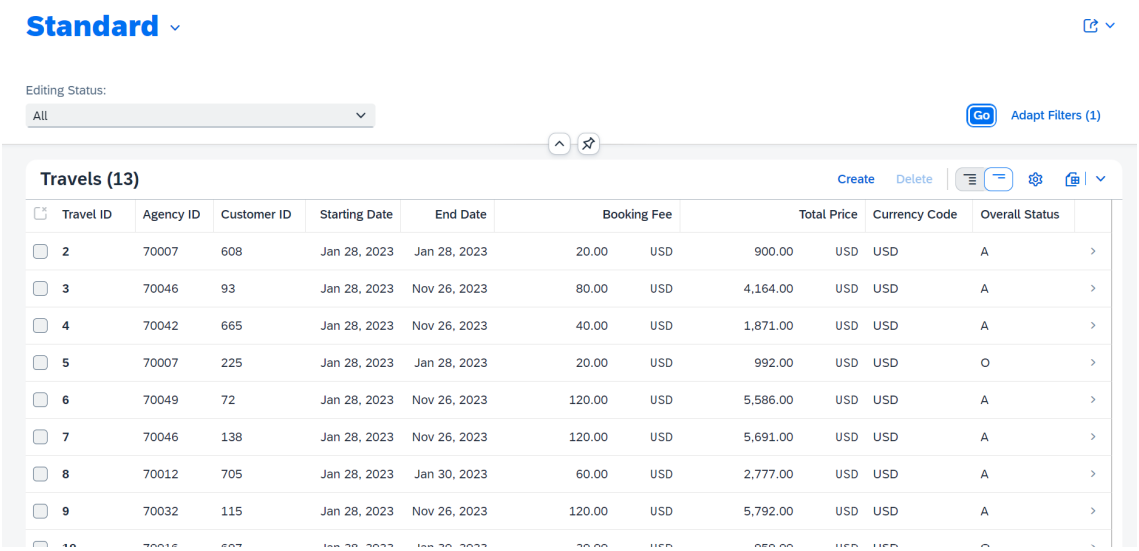
das in einem strukturierten Dateiformat sämtliche Verbindungs- und Authentifizierungsdaten enthält. Der Publisher kann die Informationen aus diesem Dokument, die pro Message-Channel eindeutig sind, nutzen um eine Verbindung zum Event Mesh herzustellen. Der Publisher kann dann eine Nachricht an das Event Mesh senden. Diese Nachricht enthält neben dem eigentlichen Ereignis auch Metadaten, die das Ereignis beschreiben. Hierzu gehören unter anderem das Thema des Ereignisses, der Absender und der Zeitpunkt der Erstellung. Im Kontext des Beispiels könnte das Thema des Ereignisses 'travel.booking.changed' sein, wobei das Thema aber frei wählbar ist. Um das Ereignis zu verarbeiten, müsste nun im Message Channel eine Queue konfiguriert sein, die auf dieses Thema horcht. Damit würde das Ereignis in der Queue landen und könnte von einem Subscriber konsumiert werden. Aufseiten des Message-Channels müsste zudem eine Regel formuliert sein, die zulässt, dass Ereignisse mit dem Thema 'travel.booking.changed' angenommen werden. Die Formulierung dieser Regeln wird bei der Erstellung des Channels im Rahmen einer Konfigurationsdatei, die service descriptor genannt wird, vorgenommen. Der service descriptor ist in Abbildung 6 dargestellt und erlaubt auch die Verwendung von Wildcards. Eine Regel, die im Beispiel eine Annahme des Ereignisses erlauben würde, könnte 'travel/booking/*' oder 'travel/*' sein. Im Rahmen des Prototyps wurde hierbei über die beispielhaft beschriebenen Schritte hinaus keine weitere funktionale Konfigurationsarbeit vorgenommen. Der finale Prototyp enthält eine Queue, die Ereignisse mit dem Thema 'travel.booking.changed' annimmt und eine weitere Queue, die im Grunde genau dasselbe tut, aber ausschließlich zu Testzwecken im Implementierungsprozess verwendet wurde und an die keine Subscriber angebunden wurden.

[11] Eigene Darstellung

Damit ist das grundlegende Gerüst der Event-Plattform fertiggestellt. An das Event Mesh können Publisher und Subscriber auf verschiedene Weisen angebunden werden. Vorgeschrieben ist nur, dass die Ereignisse, die diese erzeugen und konsumieren dem in Abschnitt 2.3 beschriebenen Cloud Event Spezifikationen folgen und somit alle relevanten Metadaten enthalten. Im Folgenden soll zuerst die Implementierung des Publishers beschrieben werden.

3.2 Implementierung aufseiten des SAP S/4 Systems

Eine simple Fiori Anwendung ist mithilfe der ABAP Development Tools (ADT) schnell erstellt. Auf Basis einer Tabelle als Datenquelle können alle Artefakte eines fertigen BO generiert werden. Daraus ergibt sich eine vollständige transaktionale Anwendung, mit der Datensätze erstellt, gelesen, geändert und gelöscht werden können. Der Prototyp verwendet als Datenquelle eine Demo-Tabelle, die Reisedaten enthält. In Abbildung 7 ist ein Auszug aus dem standardmäßig generierten User Interface dargestellt. An späterer Stelle wird noch einmal näher auf die Verwendung des BO als Publisher eingegangen.



Travel ID	Agency ID	Customer ID	Starting Date	End Date	Booking Fee		Total Price		Currency Code	Overall Status
2	70007	608	Jan 28, 2023	Jan 28, 2023	20.00	USD	900.00	USD	USD	A
3	70046	93	Jan 28, 2023	Nov 26, 2023	80.00	USD	4,164.00	USD	USD	A
4	70042	665	Jan 28, 2023	Nov 26, 2023	40.00	USD	1,871.00	USD	USD	A
5	70007	225	Jan 28, 2023	Jan 28, 2023	20.00	USD	992.00	USD	USD	O
6	70049	72	Jan 28, 2023	Nov 26, 2023	120.00	USD	5,586.00	USD	USD	A
7	70046	138	Jan 28, 2023	Nov 26, 2023	120.00	USD	5,691.00	USD	USD	A
8	70012	705	Jan 28, 2023	Jan 30, 2023	60.00	USD	2,777.00	USD	USD	A
9	70032	115	Jan 28, 2023	Nov 26, 2023	120.00	USD	5,792.00	USD	USD	A

Abbildung 7: Fiori Standard User Interface des Prototyps [12]

[12] Eigene Darstellung

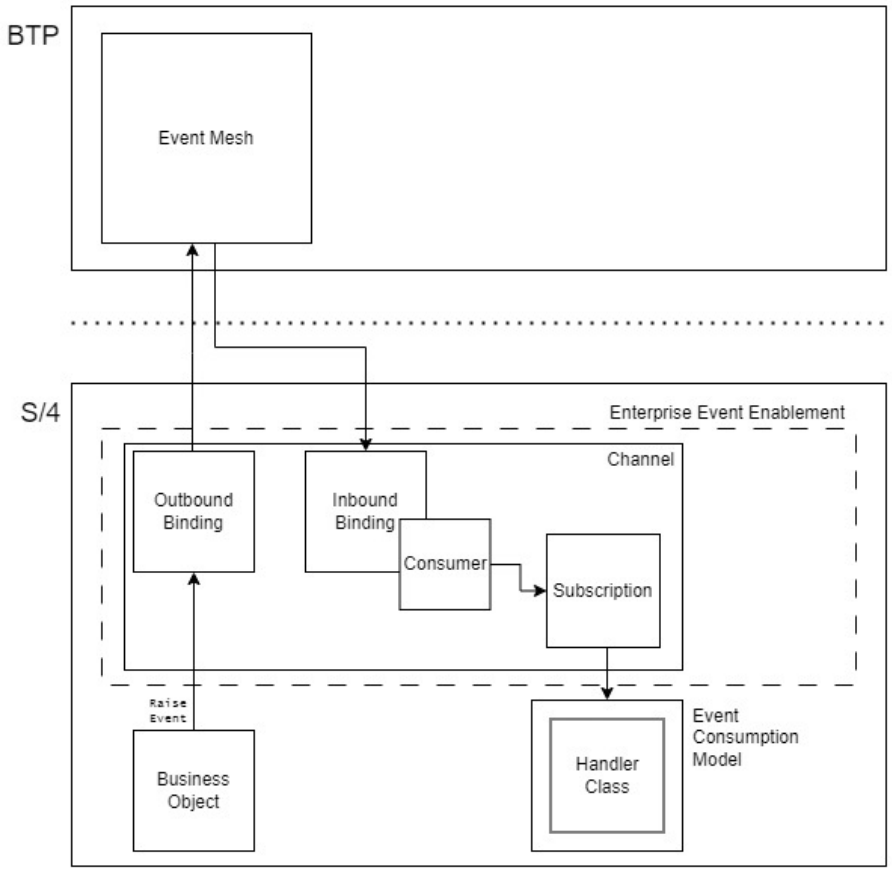


Abbildung 8: Eventing in SAP S/4 unter Verwendung des BTP Event Mesh [13]

Die Artefakte, die benötigt werden, um Ereignisse im S/4 System zu senden und zu empfangen sind in Abbildung 8 dargestellt. Gekapselt werden sie in einem Paket, das Enterprise Event Enablement genannt wird. Für die Kommunikation mit dem Event Mesh in der Cloud muss als grundlegendes Artefakt ein Channel angelegt werden. Dieser kann automatisch generiert werden, indem der Service Key angegeben wird. Wie im vorherigen Abschnitt beschrieben, enthält dieser Service Key alle notwendigen Angaben über Endpunkte, Authentifizierung und Protokolle um sich mit dem Cloud-Service zu verbinden. Somit kann mit dem Channel diese Verbindung abgedeckt werden. Innerhalb des Channels können dann sogenannte Bindings angelegt werden. Ein Binding enthält die Konfiguration der Themen, die durch den Channel an das Event-Mesh weitergeben werden sollen. Unterschieden wird in Outbound- und Inbound-Binding. Tritt im System ein Ereignis auf, dass mit einem Thema im Outbound Binding übereinstimmt, kümmert sich der Channel

[13] Eigene Darstellung

darum, dieses an das Event Mesh weiterzugeben. Auf der anderen Seite ist das Inbound-Binding an sich auch nur eine Spezifikation von Themen, die der Channel vom Event Mesh empfangen kann. Die eigentliche Abholung der Ereignisse wird mit der sogenannten Subscription konfiguriert, die die Basis für das Erstellen eines hintergründig laufenden Jobs bildet, welcher dann die Ereignisse aus der Queue des Message-Clients abholt und an die richtigen stellen im S/4 System weiterleitet. Im Prototyp wurden also Inbound und Outbound Binding mit dem Thema 'travel.booking.changed' angelegt und zudem eine Subscription mit dem entsprechenden Thema konfiguriert.

Schlussendlich muss nur noch die Logik implementiert werden, die die Ereignisse auslöst und empfängt. Soll ein Ereignis durch ein BO erzeugt werden, muss zuerst einmal in der Behavior Definition dieses BOs definiert werden, dass es ein Ereignis senden kann.

```
// dieses BO kann ein Ereignis mit dem Namen TravelCancelled
   ↪ senden, welches Nutzdaten vom Typ ZRU_CanceledReason
   ↪ enthaelt
event TravelCancelled parameter ZRU_CanceledReason;
```

Wenn das getan ist, kann ein Artefakt, das Event-Binding, erstellt werden, das im BO liegt und dazu dient, Ereignisse die Im BO erzeugt werden, an den Event-Channel weiterzugeben.
Schlussendlich muss nur noch im Programmcode der Behavior-Implementation das Ereignis auch tatsälich ausgelößt werden. Hierzu wird die Methode RAISE ENTITY EVENT aufgerufen, die als Parameter den Namen des Ereignisses und optional die Nutzdaten des Ereignisses mit dem Schlüsselwort FROM VALUE erwartet.

```
" löst ein Ereignis mit einem Parameter aus
RAISE ENTITY EVENT zru_r_traveltp~TravelCancelled
FROM VALUE #( ( TravelID = 4 %param ='aTestEvent' ) ).
```

Wird dieser Programmcode ausgeführt, wird das Ereignis ausgelöst und mithilfe der bis jetzt besprochenen Artefakte an das Event Mesh weitergegeben.

Es bleibt zu beleuchten, wie ein Ereignis in S/4 konsumiert wird. Wenn Inbound-Binding sowie Subscription definiert sind, kann mithilfe der ADT ein Event Consumption Model angelegt werden. Wie bereits in Abschnitt 2.3 beleuchtet handelt es sich hierbei um eine Sammlung an Artefakten, deren relevanteste Aufgabe es ist, eine ABAP-Klasse bereitzustellen, deren Methoden immer dann aufgerufen werden, wenn ein Ereignis konsumiert wurde. Zur Generation dieses Event Consumption Models wird eine strukturierte Beschreibung des Ereinisses benötigt, welches verarbeitet werden soll. Im SAP Kontext erfolt diese Beschreibung nach dem offenen Standart von AsyncAPI.[14] Mithilfe dieser Ereignisbeschreibung kann dann ein Event Consumption Model generiert werden. Teil dieses Models ist eine Klasse, in der schlussendlich eine Methode implementiert werden kann, die immer dann aufgerufen wird, wenn ein Ereignis konsumiert wurde.

Mit diesem Prototyp wurde vor allem gezeigt, wie es möglich ist, im gegeben Softwareumfeld eine EDA zu implementieren. Zudem wurde eine Übersicht über die Schritte gewonnen, die nötig sind um das zu erreichen. Einen produktiven Nutzen hat das System nicht weiter, es ist aber denkbar, dass die Nutzung von Nutzerbenachrichtigungen, die über EDA gesteuert werden weiter verfolgt wird.

4 Diskussion der Ergebnisse

4.1 Bewertung des Prototyps

Zur methodischen Erstellung eines Prototyps gehört auch, diesen strukturiert zu bewerten. Ziel der Methode war es, Erkenntnis über den technischen Prozess der Implementierung einer EDA zu erlangen und zu zeigen, dass dies im gegebenen Softwareumfeld überhaupt möglich ist. Gezeigt wurde zuerst einmal, dass es möglich ist, eine EDA zu implementieren. Die Erkenntnis bezüglich des technischen Prozesses ist unter Abschnitt 3 dokumentiert.

[14] Auf diesen Standard soll an dieser Stelle nicht näher eingegangen werden, da das Artefakt, das ein Ereignis beschreibt in den allermeisten Fällen automatisch generiert werden kann.

Hierzu muss aber erwähnt werden, dass die beschriebenen Schritte nur für ein relativ spezifisches System Gültigkeit haben. Voraussetzung ist, dass ein SAP On-Premise System sowohl als Producer als auch als Subscriber dienen soll und als Event-Platform der BTP-Servie Event Mesh verwendet werden soll. Das Beispiel deckt beispielsweise weder die Möglichkeit ab, externe Producer zu registrieren, noch ein SAP S/4 Cloud System als Subscriber zu verwenden. Trotzdem kann die gewonnene Erkenntnis auch in anderen Szenarien teilweise wiederverwendet werden, da die verwendeten Softwarekomponenten im Sinne der EDA prinzipbedingt modular verwendet werden können und somit auch die Implementierung zu in anderen Szenarien in beschriebener Form wiederkehrt. Der praktische Nutzen des Prototyps besteht also in seiner Funktion als Systemdemo von EDA für das System der in der Einleitung beschriebenen SAP HCM Anwendung, könnte aber auch darüber hinaus noch Anwendung finden.

4.2 Beurteilung von EDA und REST im Kontext des Anwendungsbeispiels

Die Fragestellung der Arbeit war, ob es sich lohnt, die EDA in komplexen Softwareumfeldern einzusetzen. In den theoretischen Grundlagen der Arbeit wurden einige Vorteile dieses Architekturansatzes dargestellt, die im Prototypen überprüft werden können. Konkret sollen folgende Vorteile überprüft werden:

- Modularisierung
- Ausfallsicherheit
- Integrationsmöglichkeiten

Modularisierung

Die Modularisierung ist ein zentraler Vorteil der EDA. Durch die lose Kopplung der Komponenten und die Verwendung von Events als Schnittstelle wird die Modularisierung der Softwarearchitektur gefördert. Dieser Vorteil tritt im Prototypen in der Entkopplung der Producer und Subscriber Applikationen zutage. Hierbei wurden das BO, welches das Ereignis produziert vollständig unabhängig von der Subscriber Applikation entwickelt. Es

gibt keine Überschnitte bezüglich der Code-Basis, aber auch zeitlich wurden die beiden Komponenten unabhängig voneinander entwickelt. Es ist also durchaus auch denkbar, dass wenn verschiedenen Teams an der Entwicklung der Anwendungen beteiligt sind, diese ohne größere Abstimmung, also unabhängig entwickeln könnten. Das ist ein großer Vorteil, da es die Entwicklungsgeschwindigkeit erhöht und die Komplexität der Software reduziert.

Ausfallsicherheit

Aus der Modularisierung folt logisch auch eine höhere Ausfallsicherheit der Software. In der Entwicklung trat das hervor, als beide Softwarkomponenten vollständig unabhängig voneinander getestet werden konnten, ohne dass sie aufeinander Einfluss nehmen mussten. Durch die Möglichkeit, in der Event-Platform Ereignisse zu Testzwecken auszulösen, konnten das Verhalten der Applikationen getestet werden ohne das eine Abhängigkeit zu anderen Komponenten bestand. Dieses Beispiel zeigt, dass, selbst wenn eine Softwarekomponente nicht läuft, die andere davon nicht beeinträchtigt wird. Im Beispiel könnte also der vom BO bereitgestellte Service weiterlaufen, auch wenn die vom Subscriber bereitgestellte Benachrichtigung gesendet wird. Es könnten also weiterhin Travel-Objekte angelegt, geändert und gespeichert werden, es würden nur keine Benachrichtigungen mehr gesendet. Das erhöht die Verfügbarkeit des Dienstes, weil nicht jede einzelne Funktionalität zum Totalausfall des Systems führen kann.

Integrationsmöglichkeiten

Die Integrationsmöglichkeiten sind ein weiterer Vorteil der EDA. In der Implementierung des Prototyps ist die Sprache auf den Cloud-Events Standard gekommen, der in der Industrie übergreifend verwendet wird. Nativ bietet das SAP-Event Mesh die Möglichkeit über eine API oder sogenannte Webhooks auch externe Komponenten anzubinden. In der Entwicklung des Prototyps wurde diese Möglichkeit nicht weiter getestet, die Recherche hat aber immer wieder gezeigt, dass die Integrationsmöglichkeiten des Event-Mesh sehr umfangreich sind. Eine Möglichkeit zur Weiterentwicklung des Prototyps ist es also, auch

externe Publisher und Subscriber einzubinden.

4.3 Chancen der Technologie im betriebswirtschaftlichen Kontext

Softwarekomplexität

Als abschließende Überlegung zu dem Thema sollen die bisherigen Erkenntnisse auf einen umfassenderen Rahmen übertragen werden und aus betriebswirtschaftlicher Sicht bewertet werden.
In der Entwicklung von Software in komplexen Softwareumfeldern mit vielen Komponenten und Abhängigkeiten ist EDA ein Architekturansatz, der erheblich zur Wirtschaftlichkeit eines Projekts beitragen kann. Häufig entstehen Kosten in Softwareprojekten durch die Komplexität der Softwarearchitektur. Mit komplexen Abhängigkeiten geht erhebliche Koordinierungsarbeit in den Entwicklungsteams einher und auch das nachträgliche Ändern von Schnittstellen oder nur das Hinzufügen von zusätzlicher Funktionalität zieht häufig einen hohen Aufwand nach sich. Schlussendlich wird durch diese Komplexität die Weiterentwicklung der Software und somit des Produktes behindert.
In der Konzeptionsphase, also im Rahmen von Architekturüberlegungen können solche Probleme antizipiert und reduziert werden. Mit EDA wurde ein Ansatz der Softwarearchitektur vorgestellt, der, wenn er stringent implementiert ist, dazu führt, dass Komponenten lose gekoppelt sind und somit die beschriebenen Abhängigkeiten reduziert werden können.

Agilität

Der zweite große Vorteil der EDA im geschäftlichen Kontext liegt darin, dass er Systeme agiler macht und diese befähigt schneller zu reagieren. Die asynchrone Verarbeitung von Ereignissen führt dazu, dass schneller auf Geschäftsvorfälle reagiert werden kann. Das liegt daran, dass in der Verarbeitung von diesen Ereignissen nicht mehr jede Systemkomponente auf einmal beansprucht wird, sondern immer nur die Komponente die wirklich zur Verarbeitung gebraucht wird, also die Subscriber. So werden parallelisierte Prozesse

möglich, was die allgemeine Verarbeitungsgeschwindigkeit schon auf architektonischer Ebene erhöht. Die Event-Platform, die im Verarbeitungsprozess jedes Ereignisses eine Rolle spielt, ist hierbei meist ein hoch spezialisiertes System, dass sich mit nicht mehr als der korrekten Weiterleitung der Ereignisse befassen muss. Für diese Aufgabe wird keine inhaltliche Verarbeitung der Ereignisse benötigt, es reicht der Blick auf die mitgelieferten Meta-Daten. Somit ist auch dieses System meist sehr einigermaßen performant.
Alles in allem kommt die Einführung einer EDA meist mit dem Versprechen daher, Ereignisse in Echtzeit verarbeiten zu können. In der Praxis hängt die Verarbeitungsgeschwindigkeit natürlich an einer Reihe von Faktoren, aus betriebswirtschaftlicher Sicht liegt in diesem Versprechen trotzdem ein großer Wert. Eine schnelle Verarbeitung von Geschäftsvorfällen bringt einen greifbaren Wettbewerbsvorteil, da Prozesse schneller Ablaufen und auch auf Unvorhergesehenes zeitnaher reagiert werden kann.

Adoptionsrate in der Industrie

In einer 2021 von solace beauftragen Studie zur Nutzung von EDA in der Industrie wurde festgestellt, dass 85% der befragten Unternehmen bereits EDA einsetzen oder planen dies in Zukunft zu tun. Prioritäten, die die Einführung von EDA vorantreiben sind demnach die Verbesserung der Kundenerfahrung, die Verbesserung der Responsivität der Systeme und die Verbesserung der Softwareresilienz. [Vgl. so21] Eine für die betriebswirtschaftliche Perspektive besonders interessante Metrik, die die Studie erhoben hat ist außerdem die Abwägung zwischen Kosten und Nutzen, wie in Abbildung 9 dargestellt. Demnach ist auch aus pragmatischer Kosten-Nutzen Perspektive die Einführung von EDA interessant. Zur Quelle bleibt anzumerken, dass es sich bei solace um ein Unternehmen handelt, welches selbst die Verwendung von EDA vorantreibt und somit ein Interesse an der Verbreitung dieser Technologie hat. Methodisch bildet eine Umfrage mit 840 Teilnehmern die Grundlage der Studie.

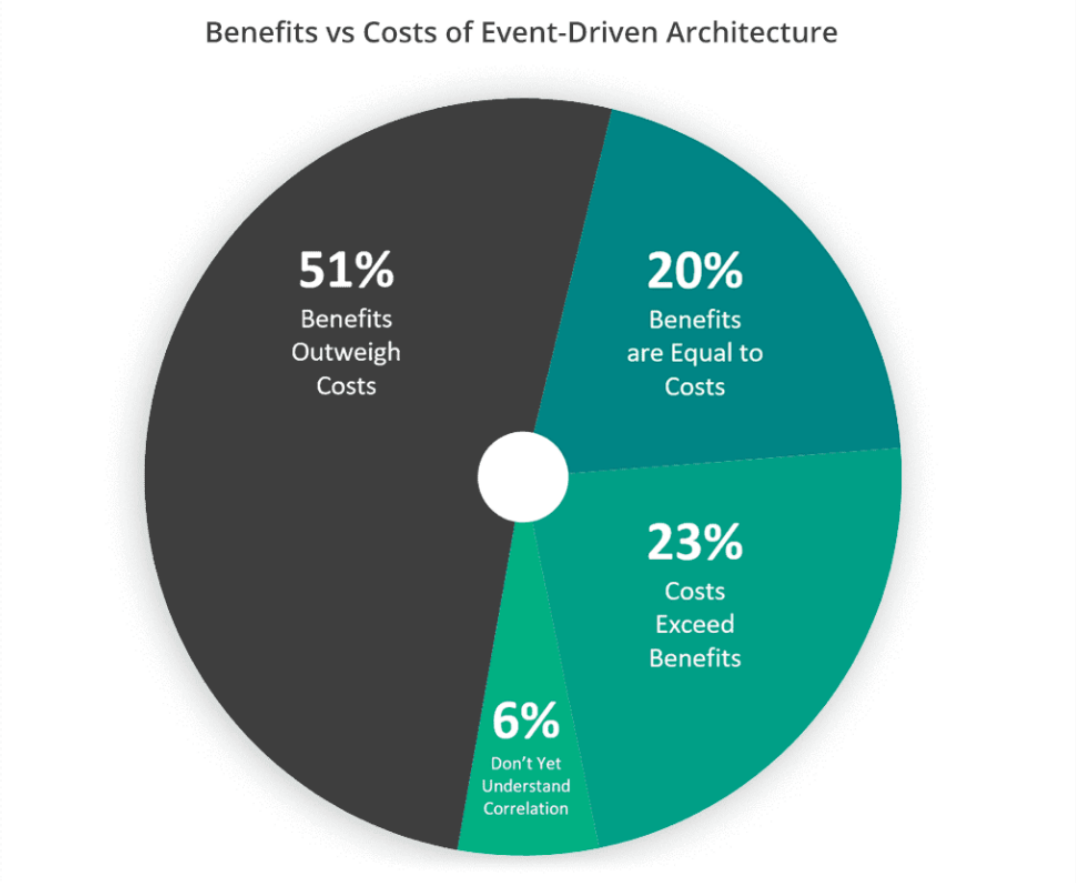


Abbildung 9: Kosten und Nutzen von EDA nach einer Studie von solace [Vgl. so21]

Als schließendes Argument könnte geführt werden, dass eine relativ hohe Adoptionsrate in der Industrie dazu führt, dass die Technologie weiter ausreift, mehr Praxisbeispiele und fertige Software zur Verfügung steht. Gängige Cloud-Anbieter wie AWS [Vgl. aw23], Azure [Vgl. Az21], Google Cloud [Vgl. go21] und eben SAP bieten alle Dienste an, die als Event-Platform fungieren können und somit nativ ins Cloud-Ökosystem integriert werden können. EDA ist also bei weitem keine architektonisch verspielte Nischenlösung mehr, sondern bei Architekturüberlegungen ein ernst zu nehmender Ansatz, der einige Vorteile mit sich bringen kann.

5 Resümee

5.1 Zusammenfassung der wichtigsten Ergebnisse

Im praktischen Teil der Arbeit wurden die theoretischen Ergebnisse im Zuge eines Prototyps angewendet und als 'Proof of Concept' realisiert. Hierbei konnten einige Erkenntnisse sowohl zum Prozess der Implementierung aber auch allgemeiner Natur gewonnen werden. Die Implementierung bezieht sich auf den Kontext einer SAP-OnPremise Anwendung als je als Producer und Subscriber und dem SAP BTP Event-Mesh als Event Platform. Es wurde evaluiert, welche Artefakte erstellt und Konfigurationsschritte vorgenommen werden müssen, um eine EDA in diesem Kontext zu implementieren. Als praktische Erkenntnis konnte aus diesem Prozess gezogen werden, dass es, in dem gegeben Rahmen, möglich ist, eine EDA zu implementieren.
Weiterhin konnten anhand des Prototyps Eigenschaften einer EDA beobachtet werden, die vorher in der Theorie besprochen wurden. Hierbei wurden die Modularisierung, die Ausfallsicherheit und die Integrationsmöglichkeiten der EDA näher betrachtet und im Kontext des Prototyps eingeordnet. Sowohl Modularisierung und Ausfallsicherheit konnten klar beim Prototypen beobachtet werden, Integrationsmöglichkeiten konnten im Ausblick auf die Weiterentwicklung des Prototyps erkannt werden.
Die bis dorthin gesammelten Erkenntnisse wurde abschließend in einen betriebswirtschaftlichen Kontext eingeordnet. Als zentrale Faktoren wurden hier die reduzierte Softwarekomplexität und die gesteigerte Agilität des Geschäfts ausgemacht. Außerdem wurde ein Blick auf eine aktuelle Studie zur Verbreitung von EDA in der Industrie geworfen, die zeigt, dass die Technologie in der Industrie bereits weit verbreitet ist.

5.2 Handlungsempfehlung

Die bisherige Arbeit lief zielstrebig auf eine uneingeschränkte Empfehlung für Architekturfragen hinaus, es soll aber auch ein kritischer Blick auf die Technologie geworfen werden. Beleuchtet wurden vor allem die Vorteile von EDA in komplexen Softwareumfeldern. Da der Ansatz bedingt, dass im kompletten Entwurfsprozess auf dessen Prinzipien geachtet

werden muss, ist er für kleiner Softwarevorhaben meist nicht sinnvoll. Die besprochenen Vorteile wirken sich am stärksten in Systemen mit vielen Schnittstellen und Komponenten aus. In einem weniger komplexen System würde alleine die Einführung einer Event-Platform als zusätzliche Komponente die Komplexität unnötig aufblähen. Würde in der Konzeption von kleinen Softwarelösungen konsequent ereignisgesteuert gearbeitet, könnte das sogar dazu führen, dass eigentlich simple Prozesse in zu viele kleine technische Ereignisse aufgebrochen werden und die Übersichtlichkeit vollständig verloren geht. Zusätzlich muss erwähnt werden, dass der Ansatz am sinnvollsten im Umfeld verteilter Systeme ist. In einer monolithischen Anwendung, die nur aus einer Komponente besteht, verkompliziert eine EDA die Architektur unnötig. Als abschließende Einschränkung muss erwähnt werden, dass eine Migration eines bestehenden Systems auf eine EDA nur bedingt sinnvoll ist. Da die EDA ein auf sehr grundlegender Ebene unterschiedlicher Ansatz zu anderen Architekturansätzen ist, würde eine Migration eines bestehenden Systems einen meist erheblichen Aufwand nach sich ziehen. Vor allem wenn viele unterschiedliche Schnittstellen betroffen wären, ist es sehr aufwändig all diese Schnittstellen auf die neuen Prinzipien umzustellen. Insgesamt ist die EDA ein Architekturansatz, der in komplexen Softwareumfeldern, die aus vielen Komponenten bestehen und viele Schnittstellen haben, sinnvoll eingesetzt werden kann. Speziell bei der Neukonzeption von Systemen ist es deshalb sinnvoll, die EDA als Architekturansatz in Betracht zu ziehen. Migrationen auf eine EDA sind meist nur dann sinnvoll, wenn das bestehende System bereits Merkmale einer EDA aufweist.

5.3 Kritische Reflexion der Arbeit und Ausblick

Die Arbeit hat sich mit der Frage beschäftigt, ob es sinnvoll ist, die EDA in komplexen Softwareumfeldern einzusetzen. Die Fragestellung wurde in der Arbeit beantwortet, es wurde gezeigt, dass die EDA in komplexen Softwareumfeldern sinnvoll eingesetzt werden kann. Der erstellte Prototyp wurde dazu verwendet ein näheres technisches Verständnis der theoretischen Konzepte zu erreichen. Die Erkenntnisse aus der Implementierung wurden in der Arbeit dokumentiert und in einen betriebswirtschaftlichen Kontext eingeordnet.

Möglichkeiten zur weiteren Forschung gibt es, wie schon in der Evaluation des Prototyps erwähnt, auf technischer Seite vor allem bei der Erforschung weiterer technischer Möglichkeiten. Es könnten beispielsweise die Integrationsmöglichkeiten näher untersucht werden oder andere Systemlandschaften in den Fokus genommen werden.

Quellenverzeichnis

Bücher

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

[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.

[Pr05] Pressman, R. S.: Software engineering: a practitioner's approach. Palgrave macmillan, 2005.

[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.

Konferenzen

[Ch06] Chandy, M.: Event-Driven Applications: Costs, Benefits and Design Approaches, Gartner Application Integration und Web Services Summit, 2006.

Internetquellen

[aw23] aws: Was ist ereignisgesteuerte Architektur?, 2023, URL: https://aws.amazon.com/de/event-driven-architecture/, Stand: 24. 07. 2023.

[Az21] Azure: Event Grid, 2021, URL: https://azure.microsoft.com/de-de/products/event-grid, Stand: 24. 07. 2023.

[Cl23a] Cloud Events: Cloud Events, 2023, URL: https : / / cloudevents . io/, Stand: 18. 07. 2023.

[Cl23b] Cloud Events: CloudEvents Primer - Version 1.0.3, 2023, URL: https://github.com/cloudevents/spec/blob/main/cloudevents/primer.md, Stand: 18. 07. 2023.

[Cl23c] Cloud Events: JSON Event Format for CloudEvents - Version 1.0.2, 2023, URL: https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/formats/json-format.md#1-introduction, Stand: 27. 06. 2023.

[go21] google cloud: Eventarc: A unified eventing experience in Google Cloud, 2021, URL: https://cloud.google.com/blog/topics/developers-practitioners/eventarc-unified-eventing-experience-google-cloud?hl=en, Stand: 24. 07. 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.

[so21] solace: Event-Driven Architecture Statistics (2021), 2021, URL: https://solace.com/event-driven-architecture-statistics/, Stand: 24. 07. 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. Quellcode des Prototyps

CL_ZRUTEST

```
CLASS cl_zrutest DEFINITION
  PUBLIC
  INHERITING FROM cl_zrutest_base
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    METHODS if_zrutest_handler~handle_rutravel_cancelled_v1
        REDEFINITION .
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.



CLASS cl_zrutest IMPLEMENTATION.
  METHOD if_zrutest_handler~handle_rutravel_cancelled_v1.
    "create a notification to push to fiori launchpad
    DATA lt_notif TYPE /iwngw/if_notif_provider=>ty_t_notification.
    DATA ls_notif TYPE /iwngw/if_notif_provider=>ty_s_notification.
    DATA lv_provd TYPE /iwngw/if_notif_provider=>ty_s_provider-id.
    DATA lv_system_uuid TYPE REF TO if_system_uuid.
    DATA lt_recipient TYPE
     ↪  /iwngw/if_notif_provider=>ty_t_notification_recipient.
    DATA ls_recipient LIKE LINE OF lt_recipient.
    DATA lv_api_return TYPE string.
    DATA lrx_api       TYPE REF TO /iwngw/cx_notification_api.
```

```
    ls_recipient-id = 'RUMBERGJ'.
    APPEND ls_recipient TO lt_recipient.

    TRY.
        ls_notif-id = lv_system_uuid->create_uuid_x16(  ).
      CATCH cx_uuid_error.
        "handle exception
    ENDTRY.

    ls_notif-type_key = 'demo'.
    ls_notif-type_version = '1.0.0'.
    ls_notif-priority = 'MEDIUM'.
    ls_notif-recipients = lt_recipient.

    APPEND ls_notif TO lt_notif.

    lv_provd = 'ZRU_NOTIF_EVT_DEMO'.

    TRY.
        /iwngw/cl_notification_api=>create_notifications(
          iv_provider_id  = lv_provd
          it_notification = lt_notif

        ).
      CATCH /iwngw/cx_notification_api INTO lrx_api.
        "handle exception
        lv_api_return = lrx_api->get_text( ).
    ENDTRY.
*    CATCH /iwngw/cx_notification_api.
```

```
    " Event Type: zru.rutravel.Cancelled.v1
    DATA ls_business_data TYPE STRUCTURE FOR HIERARCHY
    ↪  rutravel_Cancelled_v1.

    DATA wa TYPE zru_evtlog.

    ls_business_data = io_event->get_business_data( ).

    wa-payload = ls_business_data-Description.
    wa-recorddate = sy-datlo.
    wa-recordtime = sy-timlo.
    wa-notif_flag = lv_api_return.

    INSERT INTO zru_evtlog VALUES wa.
    COMMIT WORK.

  ENDMETHOD.
ENDCLASS.
```

CL_ZRUTEST_BASE

```
    class CL_ZRUTEST_BASE definition
    public
    abstract
    create public .

  public section.

    interfaces /IWXBE/IF_CONSUMER .
```

```
  interfaces IF_ZRUTEST_HANDLER
      all methods abstract .
protected section.
private section.

  constants:
    GENERATED_AT TYPE STRING VALUE `20230710140328` .
  constants:
    GENERATION_VERSION TYPE I VALUE 1 .
ENDCLASS.



CLASS CL_ZRUTEST_BASE IMPLEMENTATION.


METHOD /IWXBE/IF_CONSUMER~HANDLE_EVENT.

  " This is a generated class, which might be overwritten in the
  ↪  future.
  " Go to CL_ZRUTEST to add custom code.

  CASE io_event->get_cloud_event_type( ).
    WHEN 'zru.rutravel.Cancelled.v1'.
      me->IF_ZRUTEST_HANDLER~handle_rutravel_cancelled_v1( NEW
      ↪  LCL_RUTRAVEL_CANCELLED_V1( io_event ) ).
    WHEN OTHERS.
      RAISE EXCEPTION TYPE /iwxbe/cx_exception
        EXPORTING
          textid = /iwxbe/cx_exception=>not_supported.
```

```
    ENDCASE.

  ENDMETHOD.
  ENDCLASS.
```

CL_ZRUTEST_BASE local types

```
    CLASS LCL_RUTRAVEL_CANCELLED_V1 DEFINITION FINAL.
    PUBLIC SECTION.
      INTERFACES:
        IF_RUTRAVEL_CANCELLED_V1.

      METHODS:
        CONSTRUCTOR
          IMPORTING
            IO_EVENT TYPE REF TO /IWXBE/IF_CONSUMER_EVENT.
    PRIVATE SECTION.
      DATA:
        MO_EVENT TYPE REF TO /IWXBE/IF_CONSUMER_EVENT.

  ENDCLASS.

  CLASS LCL_RUTRAVEL_CANCELLED_V1 IMPLEMENTATION.
    METHOD CONSTRUCTOR.
        mo_event = io_event.
    ENDMETHOD.
    METHOD /IWXBE/IF_CONSUMER_EVENT~GET_ARRIVAL_TIMESTAMP.
        rv_timestamp = mo_event->get_arrival_timestamp( ).
    ENDMETHOD.
    METHOD /IWXBE/IF_CONSUMER_EVENT~GET_BUSINESS_DATA.
```

```
    mo_event->get_business_data( IMPORTING es_business_data =
    ↪  es_business_data ).
ENDMETHOD.
METHOD /IWXBE/IF_CONSUMER_EVENT~GET_CLOUD_EVENT_ID.
    rv_id = mo_event->get_cloud_event_id( ).
ENDMETHOD.
METHOD /IWXBE/IF_CONSUMER_EVENT~GET_CLOUD_EVENT_SOURCE.
    rv_source = mo_event->get_cloud_event_source( ).
ENDMETHOD.
METHOD /IWXBE/IF_CONSUMER_EVENT~GET_CLOUD_EVENT_TIMESTAMP.
    rv_timestamp = mo_event->get_cloud_event_timestamp( ).
ENDMETHOD.
METHOD /IWXBE/IF_CONSUMER_EVENT~GET_CLOUD_EVENT_TYPE.
    rv_type = mo_event->get_cloud_event_type( ).
ENDMETHOD.
METHOD /IWXBE/IF_CONSUMER_EVENT~GET_CUSTOM_EXT_ATTR_VALUE.
      mo_event->get_custom_ext_attr_value(
        EXPORTING
          iv_name = iv_name
        IMPORTING
          ev_custom_extension_attr = ev_custom_extension_attr ).
ENDMETHOD.
METHOD IF_RUTRAVEL_CANCELLED_V1~GET_BUSINESS_DATA.
    mo_event->get_business_data( IMPORTING es_business_data =
    ↪  rs_business_data ).
ENDMETHOD.
```

ZRU_R_TRAVELTP

```
managed with additional save implementation in class
↪  ZRU_BP_TravelTP unique;
strict ( 2 );
with draft;

define behavior for ZRU_R_TRAVELTP alias Travel
persistent table zru_atrav
draft table ZRU_DTRAV
//with unmanaged save
etag master LocalLastChangedAt
lock master total etag LastChangedAt
authorization master( global )
early numbering

{

  field ( readonly )
   CreatedAt,
   CreatedBy,
   LastChangedAt,
   LocalLastChangedAt,
   LocalLastChangedBy;

  field ( readonly )
   TravelID;

  create;
  update;
```

```
delete;

action testaction;

event TravelCancelled parameter ZRU_CanceledReason;

draft action Edit;
draft action Activate optimized;
draft action Discard;
draft action Resume;
draft determine action Prepare;

mapping for ZRU_ATRAV
{
  TravelID = travel_id;
  AgencyID = agency_id;
  CustomerID = customer_id;
  BeginDate = begin_date;
  EndDate = end_date;
  BookingFee = booking_fee;
  TotalPrice = total_price;
  CurrencyCode = currency_code;
  Description = description;
  OverallStatus = overall_status;
  Attachment = attachment;
  MimeType = mime_type;
  FileName = file_name;
  CreatedBy = created_by;
  CreatedAt = created_at;
  LocalLastChangedBy = local_last_changed_by;
```

```
    LocalLastChangedAt = local_last_changed_at;
    LastChangedAt = last_changed_at;
  }
}
```

RUTRAVEL_CANCELLED_V1

```
@EndUserText.label: 'rutravel_Cancelled_v1 generated'
define root abstract entity rutravel_Cancelled_v1
{
  @Event.element.externalName: 'Description'
  Description : abap.strg;
  @Event.element.externalName: 'ReasonCode'
  ReasonCode : abap.strg;
  @Event.element.externalName: 'TravelID'
  TravelID : abap.strg;

}
@EndUserText.label: 'rutravel_Cancelled_v1 generated'
define root abstract entity rutravel_Cancelled_v1
{
  @Event.element.externalName: 'Description'
  Description : abap.strg;
  @Event.element.externalName: 'ReasonCode'
  ReasonCode : abap.strg;
  @Event.element.externalName: 'TravelID'
  TravelID : abap.strg;

}
```

ZRU_CANCELEDREASON

```
  @EndUserText.label: 'booking cancelation reason'
  define abstract entity ZRU_CanceledReason
{
  ReasonCode : abap.char(2);
  Description : abap.char(64);
}
```

ZRU_BP_TRAVELTP

```
  CLASS lsc_zru_r_traveltp DEFINITION INHERITING FROM
  ↪ cl_abap_behavior_saver.

  PROTECTED SECTION.

    METHODS save_modified REDEFINITION.

ENDCLASS.

CLASS lsc_zru_r_traveltp IMPLEMENTATION.

  METHOD save_modified.
    RAISE ENTITY EVENT zru_r_traveltp~TravelCancelled
    FROM VALUE #( ( TravelID = 4 %param = VALUE #( ReasonCode =
    ↪ '02' Description = 'a test event' ) ) ).
  ENDMETHOD.

ENDCLASS.
```

```
CLASS lhc_travel DEFINITION INHERITING FROM
↪  cl_abap_behavior_handler.
  PRIVATE SECTION.
    METHODS:
      get_global_authorizations FOR GLOBAL AUTHORIZATION
        IMPORTING
        REQUEST requested_authorizations FOR Travel
        RESULT result,
      earlynumbering_create FOR NUMBERING
        IMPORTING entities FOR CREATE Travel,
      testaction FOR MODIFY
            IMPORTING keys FOR ACTION Travel~testaction.

ENDCLASS.

CLASS lhc_travel IMPLEMENTATION.
  METHOD get_global_authorizations.
  ENDMETHOD.
  METHOD earlynumbering_create.


    DATA:
      entity           TYPE STRUCTURE FOR CREATE zru_r_traveltp,
      travel_id_max    TYPE /dmo/travel_id,
      " change to abap_false if you get the ABAP Runtime error
      ↪  'BEHAVIOR_ILLEGAL_STATEMENT'
      use_number_range TYPE abap_bool VALUE abap_true.

    LOOP AT entities INTO entity WHERE TravelID IS NOT INITIAL.
```

```
  APPEND CORRESPONDING #( entity ) TO mapped-travel.
ENDLOOP.

DATA(entities_wo_travelid) = entities.

DELETE entities_wo_travelid WHERE TravelID IS NOT INITIAL.

IF use_number_range = abap_true.
  "Get Numbers
  TRY.
      cl_numberrange_runtime=>number_get(
      EXPORTING
            nr_range_nr       = '01'
            object            = '/DMO/TRV_M'
            quantity          = CONV #( lines(
            ↪  entities_wo_travelid ) )
          IMPORTING
            number            = DATA(number_range_key)
            returncode        = DATA(number_range_return_code)
            returned_quantity =
            ↪  DATA(number_range_returned_quantity)
       ).
    CATCH cx_number_ranges INTO DATA(lx_number_ranges).
      LOOP AT entities_wo_travelid INTO entity.
        APPEND VALUE #(  %cid      = entity-%cid
                         %key      = entity-%key
                         %is_draft = entity-%is_draft
                         %msg      = lx_number_ranges
                      ) TO reported-travel.
        APPEND VALUE #(  %cid      = entity-%cid
```

```
                          %key      = entity-%key
                          %is_draft = entity-%is_draft
                        ) TO failed-travel.
      ENDLOOP.
      EXIT.
  ENDTRY.

  "determine the first free travel ID from the number range
  travel_id_max = number_range_key -
  ↪ number_range_returned_quantity.
ELSE.
  "determine the first free travel ID without number range
  "Get max travel ID from active table
  SELECT SINGLE FROM zru_atrav FIELDS MAX( travel_id ) AS
  ↪ travelID INTO @travel_id_max.
  "Get max travel ID from draft table
  SELECT SINGLE FROM zru_dtrav FIELDS MAX( travelid ) INTO
  ↪ @DATA(max_travelid_draft).
  IF max_travelid_draft > travel_id_max.
    travel_id_max = max_travelid_draft.
  ENDIF.
ENDIF.

"Set Travel ID for new instances w/o ID
LOOP AT entities_wo_travelid INTO entity.
  travel_id_max += 1.
  entity-TravelID = travel_id_max.

  APPEND VALUE #( %cid      = entity-%cid
                  %key      = entity-%key
```

```
                        %is_draft = entity-%is_draft
                      ) TO mapped-travel.
    ENDLOOP.


  ENDMETHOD.


  METHOD testaction.
    "not working because a event can only be raised in the late
    ↪  save stage
    RAISE ENTITY EVENT zru_r_traveltp~TravelCancelled
    FROM VALUE #( ( TravelID = 4 %param = VALUE #( ReasonCode =
    ↪  '03' Description = 'another test event' ) ) ).
  ENDMETHOD.

ENDCLASS.
```