HTWK

IM Fakultät Informatik und Medien

Enterprise Architektur-Muster

Modul “Software Engineering” (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

9. Januar 2025
Julian Bruder, Abdellah Filali, Luca Franke

Fakultät Informatik und Medien
HTWK Leipzig

Anwendungsfall E-Commerce I

- Ziel: Backend für internationales E-Commerce-System
- MVP: Bestellungen, Bezahlung und Versand
- Zukünftig viele Nutzer und hoher Traffic erwartet
- Geringes Kapital für Infrastruktur
- Rechtliche Regularien teilweise unklar, weil international
- Hohe Sicherheitsanforderungen
- Agiles Team von acht fähigen Entwicklern
- Geldgeber wollen erste Auslieferung in zwei Wochen

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien

Anwendungsfall E-Commerce II

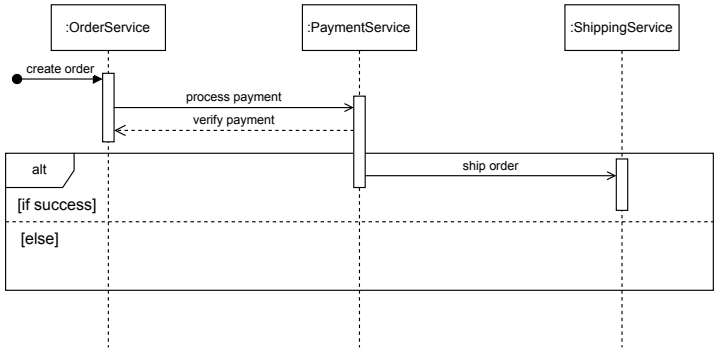

Abbildung 1: Sequenzdiagram zum Aufgeben einer Bestellung

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Grundlagen und Anforderungen

- Architectura lateinisch für: *Wissenschaft der Baukunst*
- Software-Architektur: Grundlegende Struktur und Beziehungen von Teilen einer Software [3]
- Enterprise-Architektur (EA): Grundlegende Struktur und Beziehungen von Komponenten eines Systems
- EA beschreibt Prozess und Ergebnis (vgl. Bedeutung latein. *architectura*)
- Ziel: Ausrichtung von Business und IT
- Umsetzung durch EA-Muster: Spezifische Strategien zur Ausrichtung

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Klassische Enterprise-Architektur

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Service-oriented Architecture

- Bisher: Eng gekoppelte Funktionalitäten
- Jetzt: Lose Kopplung von Funktionalitäten durch Kapselung als Dienst
- Ziel Service-oriented Architecture (SOA): Wiederwendung von Funktionalitäten

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Service-oriented Architecture: Komponenten

- Service Provider: Stellt spezifischen Dienst bereit
- Service Consumer: Nutzt bereitgestellten Dienst
- Broker: Vermittler, der Kommunikation zwischen Consumer und Provider regelt
- Service Registry: Sammlung von Metadaten zu Services und deren Provider

Service-oriented Architecture: Struktur

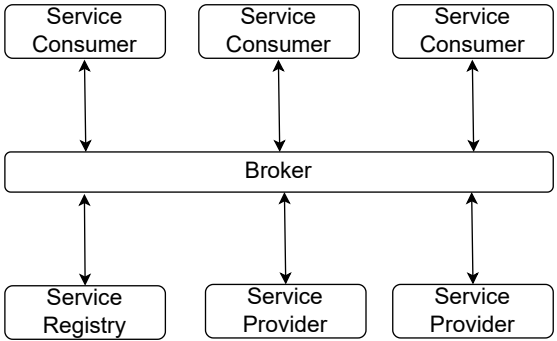

Abbildung 2: Aufbau der Service-oriented Architecture

Service-oriented Architecture: Beispiel E-Commerce I

- `OrderService`: Dienst für Verwaltung von Bestellungen
- `PaymentService`: Dienst für die Abwicklung von Zahlungen (Service von PayPal, …)
- `ShipmentService`: Dienst für den Versand (Service von DHL, …)

Service-oriented Architecture: Beispiel E-Commerce II

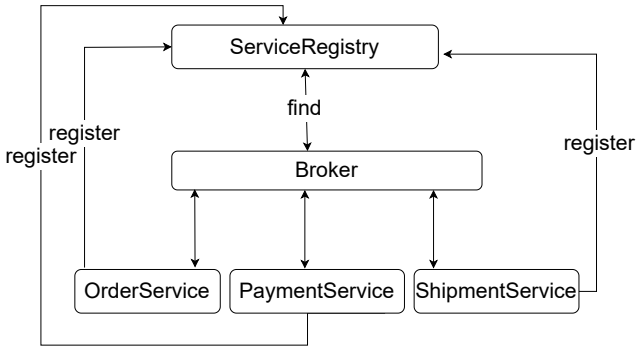

Abbildung 3: E-Commerce-Beispiel mit Service-oriented Architecture

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Service-oriented Architecture: Agilität

- Wie bei modularem Monolithen: Lose Kopplung ⇒ kleine autonome Teams
- Aber zusätzlich: Deployment in Teams ⇒ kürzere Iterationen & häufigere Auslieferung
- Dadurch: Flexibler gegenüber wechselnden Anforderungen
- Zeit- und Kosteneinsparungen durch Wiederverwendung von Diensten
- Eigenständige Dienste ermöglichen horizontale Skalierung
- Aber: Langfristig Abhängigkeiten zwischen Diensten, besonders für grob-granulare Dienste

Moderne Enterprise-Architektur

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Microkernel Architecture

- Aufteilung der Anwendung in zwei Komponenten [5]
- Kern:
  - Minimale Funktionalität
  - Bietet Schnittstelle für Erweiterungen/Plugins
  - Enthält Plugin-Registry
- Module:
  - Erweitern Kern um Funktionalitäten
  - Kommunikation über definierte Schnittstellen
  - Lose gekoppelt, unabhängig und isoliert voneinander
  - Verbindung über verschiedene Wege möglich (REST, Messaging, Objekt Instanziierung, …)

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul “Software Engineering” (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Microkernel Architecture: Struktur

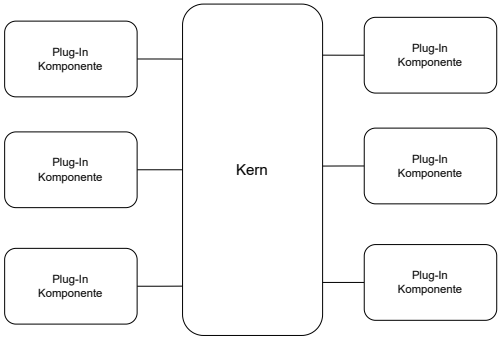

Abbildung 4: Aufbau einer Microkernel Architecture

Microkernel Architecture: Beispiel E-Commerce I

- Kern: Großteil der Funktionalität
- Module: Auslagerung von Business-Logik zum Bezahlen und Versenden
- Einfache Erweiterbarkeit durch neue Dienstleister
- Komplexer Kern, hohe Kopplung der übrigen Funktionalitäten
- Als einziges Architekturmuster eher nicht geeignet, jedoch in Kombination mit anderen sinnvoll

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Microkernel Architecture: Beispiel E-Commerce II

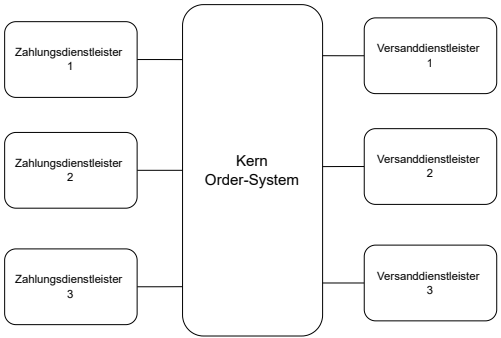

Abbildung 5: E-Commerce-Beispiel mit Microkernel Architecture

Microkernel Architecture: Agilität

- Lose Kopplung der Module ⇒ schnelle Reaktionsfähigkeit auf Änderungen
- Module können einfach ausgetauscht werden ⇒ geringe Downtime
- Einfach testbar, da Module unabhängig und isoliert voneinander
- Kurze Iterationen und Auslieferungszeiten bei Erweiterung der Anwendung
- Aber: Hoher initialer Aufwand durch teuren Kern, eher hohe Time-to-Market
- Aber: Hoher Aufwand, wenn Kern später Anpassungen benötigt
- Fazit: In ausgewählten Anwendungsfällen sinnvoll, aber nicht universell in agilen Umgebungen einsetzbar

Event-Driven Architecture

- Bisher: Expliziter Aufruf von Funktionalitäten
- Jetzt: Impliziter Aufruf durch Reaktion auf Ereignisse [2]
- System reagiert asynchron auf Zustandsänderung (Ereignis in System)
- Alte Idee: David Garlan und Mary Shaw, 1994, *An Introduction to Software Architecture*

Event-Driven Architecture: Komponenten

- Event: Kapselt Information einer Zustandsänderung eines Systems [4]
- Produzent: Erzeugt Event
- Publisher: Publiziert erzeugtes Event
- Konsument: Reagiert auf Event
- Mediator: Vermittler zwischen Produzenten und Konsumenten
- Event-Broker: Infrastruktur für Gesamtheit der Vermittler

Event-Driven Architecture: Struktur

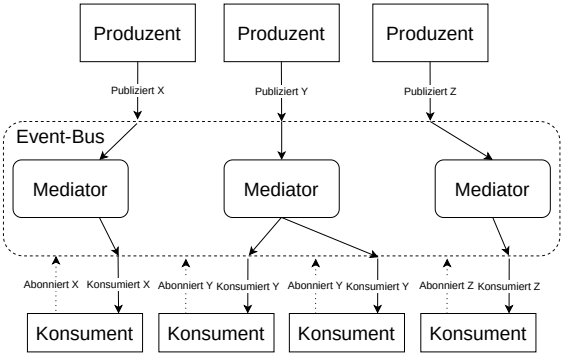

Abbildung 6: Vertrag zwischen Produzenten und Konsumenten am Event-Bus

Event-Driven Architecture: Beispiel E-Commerce I

- `OrderCreated`: Genau dann, wenn Bestellung aufgegeben wird
- `PaymentProcessed`: Genau dann, wenn Bezahlvorgang abgeschlossen wird
- `ShipmentInitiated`: Genau dann, wenn Bestellung versandt wird
- Event-Kette: `OrderCreated` $\rightarrow$ `PaymentProcessed` $\rightarrow$ `ShipmentInitiated`
- Implementierung in Diensten: OrderService, PaymentService, ShipmentService

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Event-Driven Architecture: Beispiel E-Commerce II

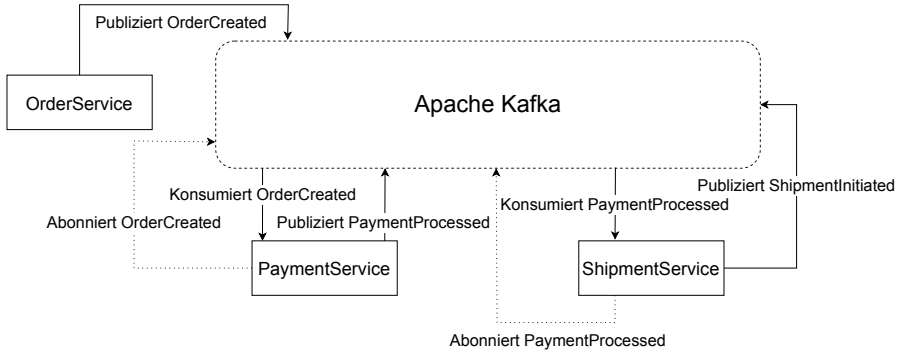

Abbildung 7: E-Commerce-Beispiel mit Event-Driven Architecture

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul "Software Engineering" (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Event-Driven Architecture: Agilität

- Event ist Vertrag zwischen Produzent und Konsument am Event-Broker
  $\Rightarrow$ Hohe Kohäsion $\Rightarrow$ Lose Kopplung
- Feature: Menge von Events, deren Produzenten und Konsumenten
  $\Rightarrow$ Klare Abgrenzung $\Rightarrow$ einfach definierbare Iterationen
- Events sind sehr realitätsnah - domain-driven
- Sehr hohe Flexibilität & maximale Skalierung durch lose Kopplung
- Schnelle Auslieferung, kurze Intervalle
- Exzellente Kombination mit Microservices & Cloud-Integration
- Aber: Erhöhte Komplexität $\Rightarrow$ Hohe Anforderungen an Entwickler

Julian Bruder, Abdellah Filali, Luca Franke, Fakultät Informatik und Medien
Modul “Software Engineering” (Prof. Dr. Andreas Both, Wi-Se 2024/2025) an der HTWK Leipzig

Cloud-Native Architecture

- Bisher: Vorab-Allokation von Ressourcen
- Jetzt: Allokation genau dann, wenn notwendig
- Cloud-Native: Explizit für die Cloud entwickelte Applikationen [1]
- Annahme: Infrastruktur ist in ständigem Wandel
- Folgerung: Infrastruktur auslagern - an Cloud-Vendor
  - Globale Nutzung durch Geo-Redundanz: Starke Verteilung und hohe Verfügbarkeit
  - Auto-Scaling: Dynamische Skalierung basierend auf Nachfrage
  - Pay-as-you-go, Scale-to-zero: Nur verwendete Ressource wird bezahlt
  - Zero Downtime

Cloud-Native Architecture: Technologie

- Containerisierung: Jede Komponente eines Systems ist Container
- Dynamische Orchestrierung: Aktives Container-Management zur Ressourcenoptimierung
- Microservice-Architektur ist Fundament - ergänzt durch Cloud-Dienste
- Fully Managed Cloud-Services: Business-Logik statt Infrastruktur

Cloud-Native Architecture: Beispiel E-Commerce

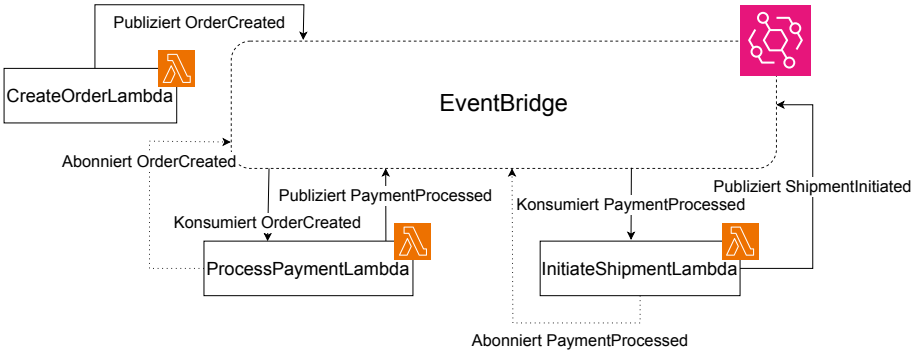

Abbildung 8: E-Commerce-Beispiel mit Cloud-Native Architecture in AWS

Cloud-Native Architecture: Agilität

- Alle agilen Vorteile von Microservice- und Event-Driven Architecture
- Fokus auf Business-Logik & einfaches Deployment $\Rightarrow$ Kurze Iterationen
- Maximale Flexibilität für Nachfrage durch Auto-Scaling
- Finanzielle Agilität: Pay-as-you-go
- Aber: Kostenrisiken durch Auto-Scaling und Pay-as-you-go
- Achtung: Vendor-Lock-In durch proprietäre Fully Managed Cloud-Services

Zusammenfassung

- EA ist dynamische Strategie (Vision), um IT auf Business auszurichten
- Wahl EA beeinflusst Agilität in Entwicklung substanziell
- (Modularer) Monolith: Eine (aus Funktionalitäten bestehende) Komponente
- SOA: Wiederverwendbarkeit von Funktionalitäten durch (meist grobe) Dienste
- Micro-Kernel: Kernfunktionalität wird um Zusatzfunktionalitäten erweitert
- Microservices: Isolierung von Funktionalitäten durch feine Dienste
- EDA: Asynchrone Reaktion auf Ereignisse
- Cloud-Nativ: Auslagerung Verwaltung Infrastruktur $\Rightarrow$ Fokus auf Businesslogik
- Es gibt keine beste Architektur - Wahl ist anwendungsspezifisch und meist Kombination mehrerer EA-Muster

Literatur I

[1] Dennis Gannon, Roger Barga und Neel Sundaresan. “Cloud-Native Applications”. In: *IEEE Cloud Computing* 4.5 (2017), S. 16–21. DOI: 10.1109/MCC.2017.4250939.

[2] David Garlan und Mary Shaw. *An Introduction to Software Architecture*. Techn. Ber. CMU/SEI-94-TR-021. Accessed: 2025-Jan-2. Jan. 1994. URL: https://insights.sei.cmu.edu/library/an-introduction-to-software-architecture/.

[3] Danny Greefhorst und Erik Proper. “The Role of Enterprise Architecture”. In: *Architecture Principles: The Cornerstones of Enterprise Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 7–29. ISBN: 978-3-642-20279-7. DOI: 10.1007/978-3-642-20279-7_2. URL: https://doi.org/10.1007/978-3-642-20279-7_2.

Literatur II

[4] Ramakrishna Manchana. “Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries”. In: *International Journal of Science and Research (IJSR)* 10 (Jan. 2021), S. 1706–1716. DOI: 10.21275/SR24820051042.

[5] Mark Richards. *Software Architecture Patterns*. O’Reilly Media, Inc., 2015, value. ISBN: 9781491925409.