Service Engineering: SOA im Überblick

Version 1.1

© J. Heinzelreiter

Literatur

- Bücher
  - Krafzig et al: *Enterprise SOA. Service Oriented Architecture Best Practices*
  - Erl*: Service-Oriented Architecture. Concepts, Technology, and Design*
  - Margolis: *SOA for the Business Developer: Concepts, BPEL, and SCA.*
  - Juval Löwy: *Programming WCF Services*

Geschichtlicher Überblick (1)

- ~ 1800: Mechanische „Computer“: Jacquard-Webstuhl
  - Daten in Form von Lochkarten, Realisierung des „Programms“ mit Hardware.
  - Kopplung an Hardware.
- 1920/30: Elektromechanische „Computer“: Enigma
  - Neuer Verschlüsselungsalgorithmus → Austausch der Hardware
  - Kopplung an Hardware.
- 1940/50: Röhren-/Transistor-basierte Universalrechner
  - Frei programmierbar, Sprache war allerdings maschinenspezifisch.
  - Kopplung an Hardware, die Maschinensprache unterstützt.
- Anfang 1960: Assemblersprachen
  - Kopplung an Maschinenarchitektur

Geschichtlicher Überblick (2)

- 60er-Jahre: Hochsprachen wie COBOL und FORTRAN
  - Compiler gewährleistet Unabhängigkeit von der Hardwarearchitektur.
  - Nicht strukturierte Programmierung, geringe Wiederverwendbarkeit.
- 70er-Jahre: Strukturierte Programmierung (Pascal, C)
  - Datenstrukturen, Funktionen und Prozeduren.
  - Funktionen sind an Repräsentierung der Daten gekoppelt.
- 80er-Jahre: Objektorientierte Programmierung (Smalltalk, C++)
  - Erhöhung der Wiederverwendbarkeit durch Zusammenführen von Daten und Funktionen bzw. Vererbung.
  - Monolithische Anwendungen, keine „binäre Wiederverwendbarkeit“.
  - Viele Implementierungsdetails sind in C/C++-Headerdateien enthalten.
  - Vererbung stellt enge Bindung zwischen Klassen her und ist nur innerhalb derselben Sprache möglich.
  - Anwendung ist an Programmiersprache gebunden.

Geschichtlicher Überblick (3)

- 90er-Jahre: Komponentenorientierte SW-Entwicklung (COM, JavaBeans)
  - Binäre Komponenten: Code + Schnittstellenbeschreibung.
  - Standardisierung der Binärlayouts -> Verwendbarkeit in verschiedenen Sprachen.
  - Implementierung erfordert viel Overhead-Code.
- 2002: .NET
  - Sprachen/Laufzeitumgebung unterstützen die Entwicklung/Verwendung von Komponenten.
  - Reichhaltiges, sprachübergreifendes Typsystem.
  - Gute Unterstützung von Metadaten.
  - Durchdachtes Versionierungskonzept.
  - Entwicklung von Komponenten wurde radikal vereinfacht.

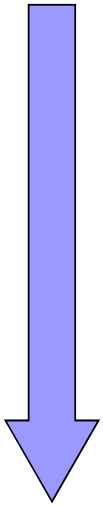
Schwächen „einfacher“ Plattformen (Java SE/.NET)

- Plattformabhängigkeit (.NET)
- Nebenläufigkeit: Komponenten sind selbst für Synchronisation zuständig.
- Transaktionen: Anwendung ist für Transaktionsmanagement zuständig.
- Kopplung an Remoting- und Kommunikations-APIs
- Kopplung an Kommunikationsmuster: synchron vs. asynchron
- Sicherheit: Komponente muss Authentifizierung und Autorisierung selbst vornehmen.

  → **Enge Kopplung zwischen Technologiecode und Geschäftslogik**

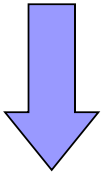
Kopplung

Maschinencode



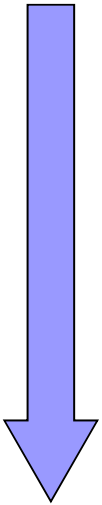
SOA

- Enge Kopplung an Hardware-architektur oder Plattform.
- Hoher Anteil von Overhead-Code („plumbing“)



- Unabhängigkeit von der Plattform und Programmiersprache.
- Trennung von Technologiecode und Geschäftslogik.
- Geschäftslogik steht im Vordergrund.

Starke Kopplung



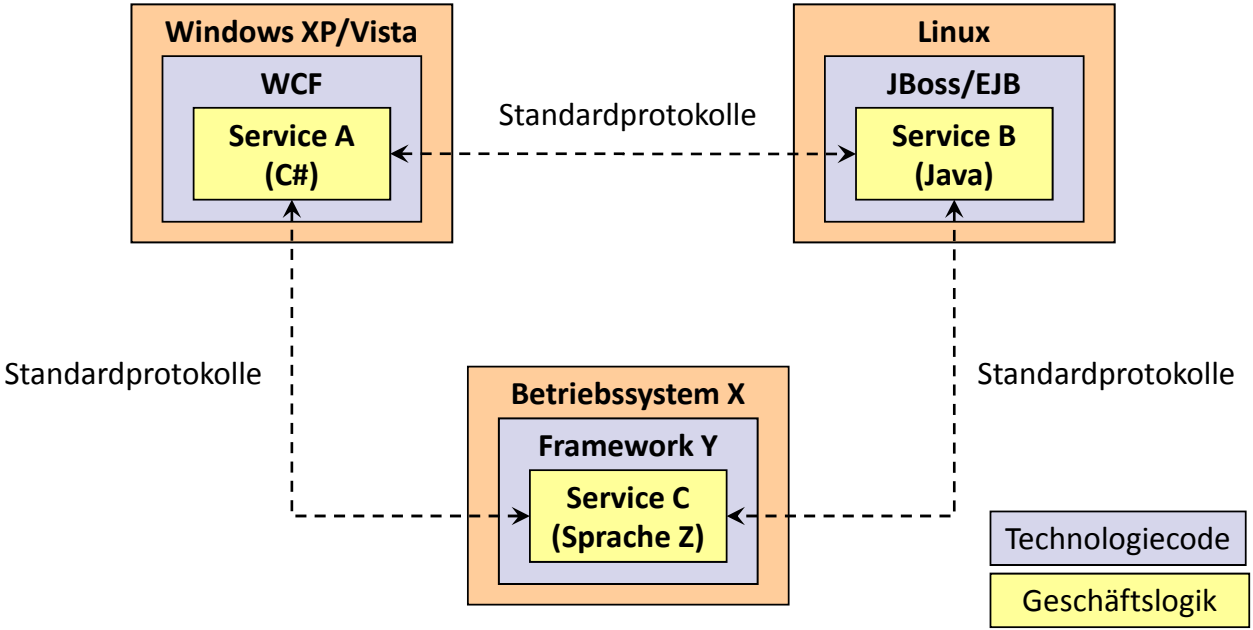
Schwache Kopplung

Definition Service-orientierte Architektur (SOA)

"SOA is an application architecture in which all functions, or services, are defined using a *description language* and have invokable *interfaces* that are called to perform business processes. *Each interaction is independent of* each and every *other interaction and the interconnect protocols* of the communicating devices (i.e., the infrastructure components that determine the communication system do not affect the interfaces). Because interfaces are *platform-independent*, a client from any device using any operating system in any language can use the service."

www.webopedia.com

Serviceorientierte Anwendungen



© J. Heinzelreiter

Komponenten vs. Services

- Services und Komponenten haben ähnliche Eigenschaften.
- Bei Komponenten stehen die Anforderungen des Komponentenentwicklers im Vordergrund:
  - Komponente soll in möglichst vielen Anwendungsszenarien wiederverwendet werden können.
  - Komponente soll einfach austauschbar sein.
- Bei Services treten Anforderungen des Service-Konsumenten in den Vordergrund:
  - Service soll verwendet werden können, ohne die Implementierungsdetails bzw. Kommunikationsprotokolle zu kennen.
  - Einzelnes Services sollen austauschbar sein, ohne dass das Gesamtsystem davon betroffen ist.
- Services sind Grundbausteine vieler verteilter Architekturen.
- Verteilungsaspekt ist aber keine unabdingbare Anforderung an Service-Architekturen (→ OSGi).

Konzeptionelle Eigenschaften von Services (D. Box)

- Explizite Servicegrenzen
  - Service gibt explizit (mit Kontrakten) seine exportierte Funktionalität bekannt. Nachrichtenaustausch anstatt entfernter Methodenaufrufe.
  - Alles andere (Technologie, Speicherort, …) ist hinter Servicegrenzen verborgen.
- Services sind autonom
  - Service ist vollkommen unabhängig vom Client: Versionierung, Security.
  - Services sind fehlerresistent.
- Services kommunizieren über Operations- und Datenkontrakte
  - Kontrakte (auch die Datentypen) sind technologieunabhängig.
  - Services sind zustandslos (keine verteilten Objekte).
- Interoperabilität zwischen Services basiert auf Policies
  - Anforderungen an Clients (z. B. unterstützte Protokolle) werden in Form von Policies veröffentlicht.

Praktische Auswirkungen auf SOA-Anwendungen

- *Sicherheit:* Services kommunizieren über sichere Kommunikationskanäle auf Basis von Standardprotokollen.
- *Konsistenz:* Unabhängig vom Ausgang einer Operation bleibt das Service in einem konsistenten Zustand.
- *Thread-Sicherheit:* Services können gleichzeitig von mehreren Clients angesprochen werden.
- *Zuverlässigkeit:* Nachrichten des Clients kommen sicher und in der vorgegebenen Reihenfolge beim Service an.
- *Robustheit:* Stabilität des Services wird nicht durch Fehler beeinträchtigt.
- *Interoperabilität*
- *Skalierbarkeit*
- *Verfügbarkeit*