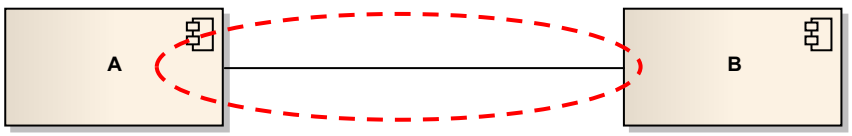
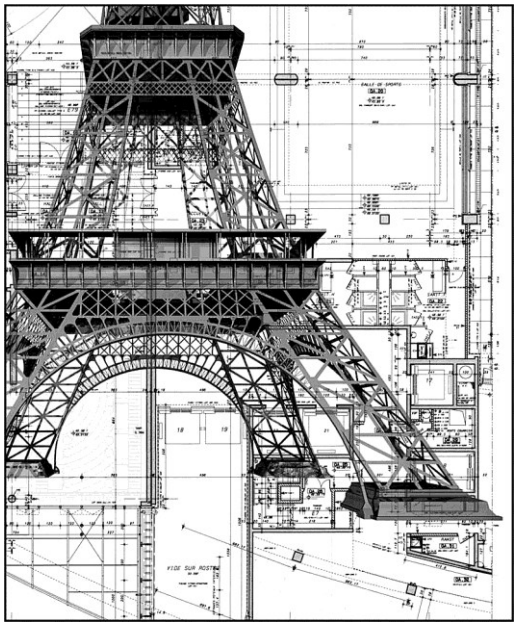
Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften Hamburg
Fachbereich Informatik

Prof. Dr. Stefan Sarstedt
(stefan.sarstedt@haw-hamburg.de)

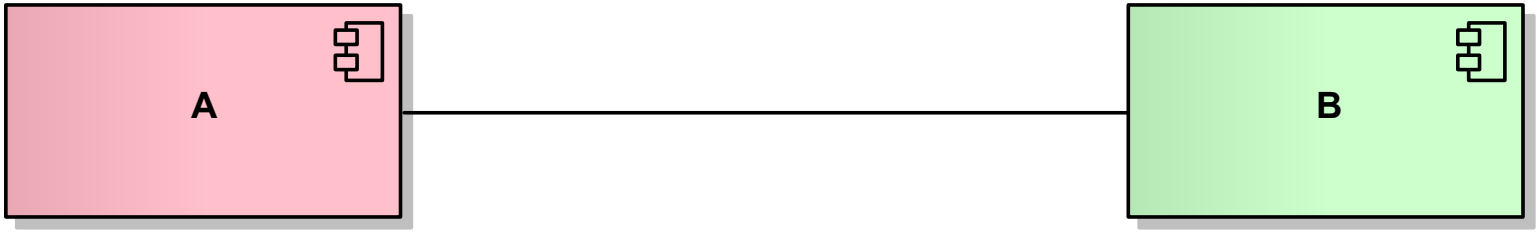
Konnektoren





Rollen und Herausforderungen von Software-Konnektoren

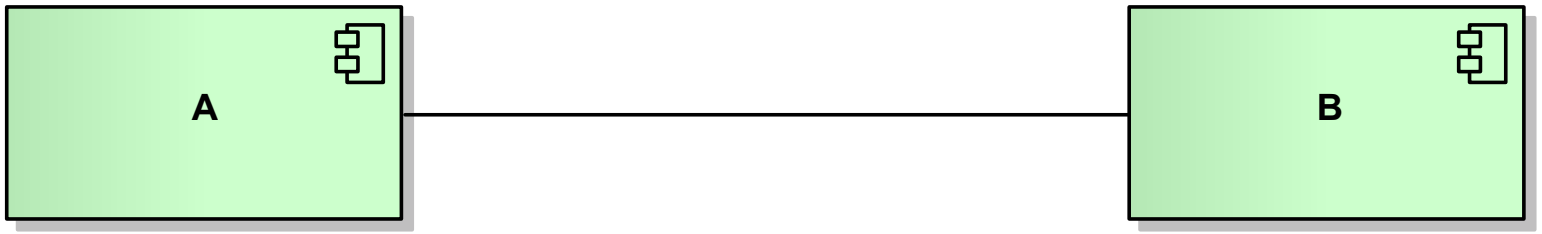
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



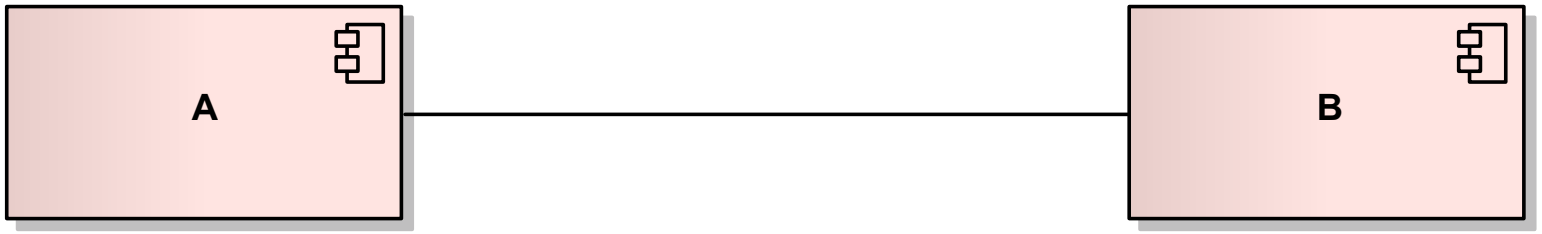
Rollen und Herausforderungen von Software-Konnektoren

Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?

A an Bs Gestalt anpassen?

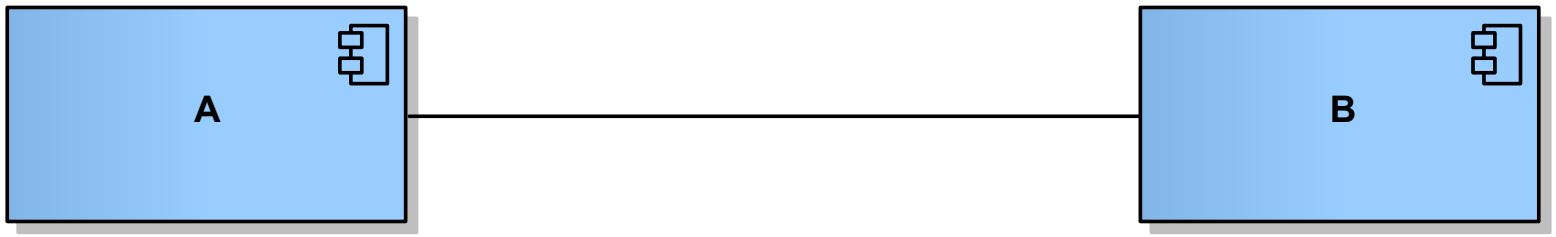


oder B an As Gestalt anpassen?



Rollen und Herausforderungen von Software-Konnektoren

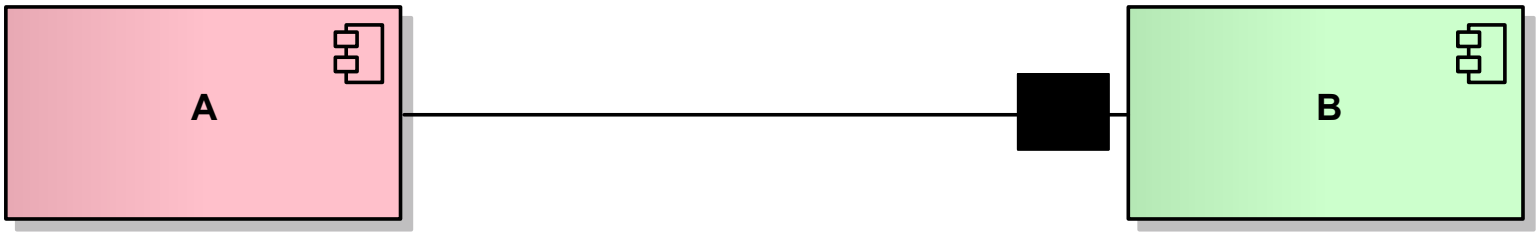
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



Verhandeln, um eine gemeinsame Form zu finden?

Rollen und Herausforderungen von Software-Konnektoren

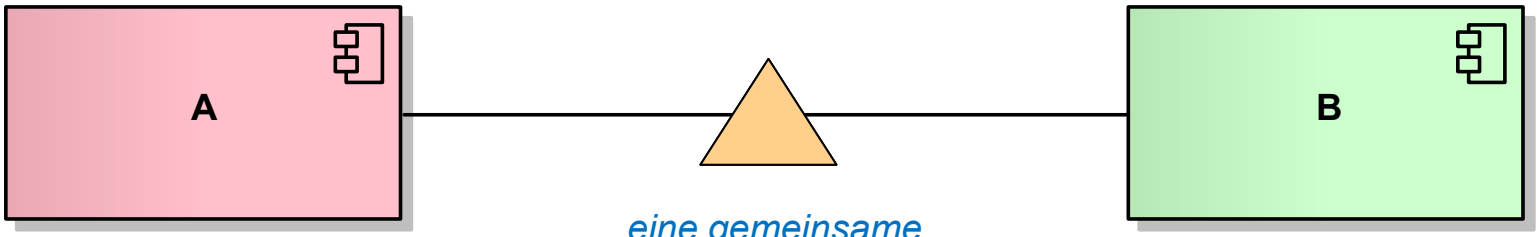
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



einen Import/Export-Konverter für B einführen?

Rollen und Herausforderungen von Software-Konnektoren

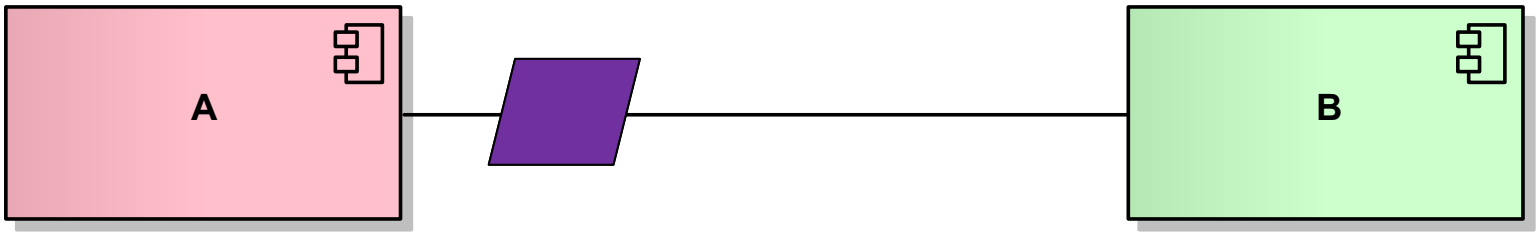
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



eine gemeinsame Zwischenrepräsentation erstellen?

Rollen und Herausforderungen von Software-Konnektoren

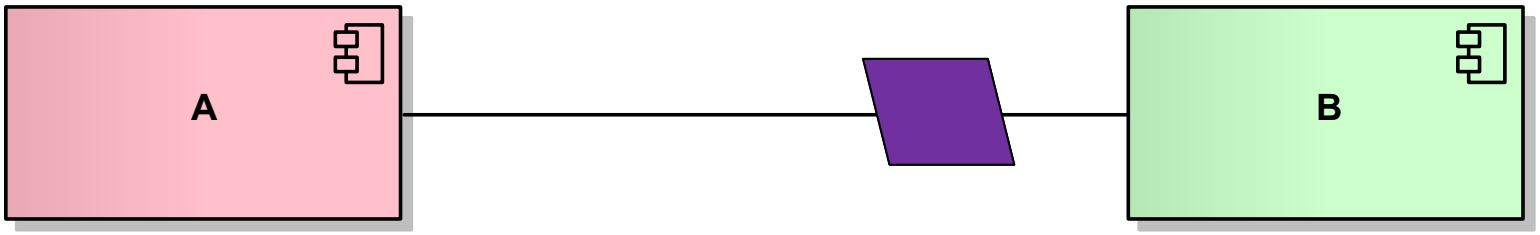
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



B auf A adaptieren?

Rollen und Herausforderungen von Software-Konnektoren

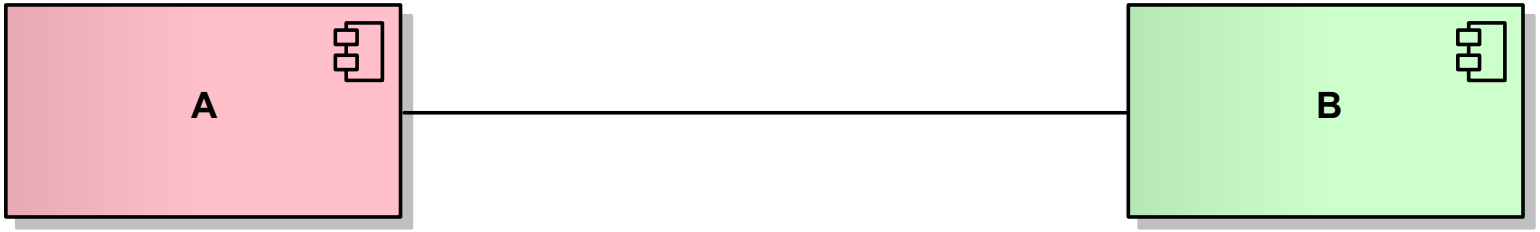
Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



oder A auf B adaptieren?

Rollen und Herausforderungen von Software-Konnektoren

Wie können wir die Interaktion zwischen zwei Komponenten A und B ermöglichen?



u.v.m.

Welches ist die richtige Antwort?

Was ist ein Software-Konnektor?

- **Architekturelemente**, die Interaktionen und Interaktionsregeln zwischen Komponenten modellieren
- Einfache Interaktionen
  - Prozeduraufruf
  - Gemeinsamer Variablenzugriff
- Komplexe, semantikreiche Interaktionen
  - Client-Server-Protokolle
  - Datenbankzugriffsprotokolle
  - Asynchroner Multicast von Ereignissen
- Jeder Konnektor stellt zur Verfügung:
  - Interaktionskanal
  - Transfer von Kontrolle und/oder Daten

Implementierte vs. konzeptionelle Konnektoren

- Implementierte Konnektoren
  - besitzen oftmals keinen dedizierten Code
  - besitzen oftmals keine Identität
  - korrespondieren typischerweise nicht mit einer (einzelnen) Übersetzungseinheit
  - sind oftmals verteilt implementiert:
    - über mehrere Module
    - über verschiedene Interaktionsmechanismen
- Konzeptionelle Konnektoren in Software-Architekturen
  - sind Einheiten/Elemente “erster Klasse”
  - besitzen Identität
  - beschreiben alle Systeminteraktionen

Abgrenzung Konnektoren von Komponenten

- Konnektor ≠ Komponente
  - Komponenten stellen applikationsspezifische Funktionalität zur Verfügung
  - Konnektoren stellen applikationsunabhängige Interaktionsmechanismen zur Verfügung
- Konnektoren ermöglichen die Spezifikation komplexer Interaktionen
  - binär / n-är
  - asymmetrisch / symmetrisch
  - Protokolle
  - Interaktionsdefinition separat
  - können Komponenteninteraktion flexibel gestalten
  - …

Vorteile von expliziten Konnektoren

- Separierung von Berechnung und Interaktion
- Komponentenabhängigkeiten werden minimiert
- Unterstützung von Software-Evolution
  - auf der Ebene von Komponenten, Konnektoren und des Systems
- Vereinfacht
  - Heterogenität
  - Verteilung
  - Analyse und Test der einzelnen Systembestandteile

Rollen von Konnektoren

- Jeder Konnektor bietet Dienste an, die zu mindestens einer der folgenden Rollen zuzuordnen sind:
  - **Kommunikation**
  - **Koordination**
  - **Konvertierung**
  - **Moderation**

  → im Folgenden beschrieben

Rolle: Kommunikation

- Hauptrolle, die mit Konnektoren assoziiert werden
- =„Daten schaufeln“
- verschiedene Kommunikationsmechanismen
  - z. B. Prozeduraufruf, RPC, Gemeinsamer Speicher, Nachrichtenaustausch
- Einschränkungen der Kommunikationsstruktur/-richtung
  - z. B. Pipes
- Einschränkung der Quality-of-Service (QoS)
  - z. B. durch Persistenz
- Kann nichtfunktionale Eigenschaften beeinflussen
  - z. B. Performanz, Skalierbarkeit, Sicherheit

Rolle: Koordination

- Bestimmen Kontrolle der **Ausführung** („thread of execution“)
- Bestimmen Kontrolle der **Auslieferung von Daten**

- Ein „Prozedurkonnektor“ hat Rolle Kommunikation und Rolle Koordination

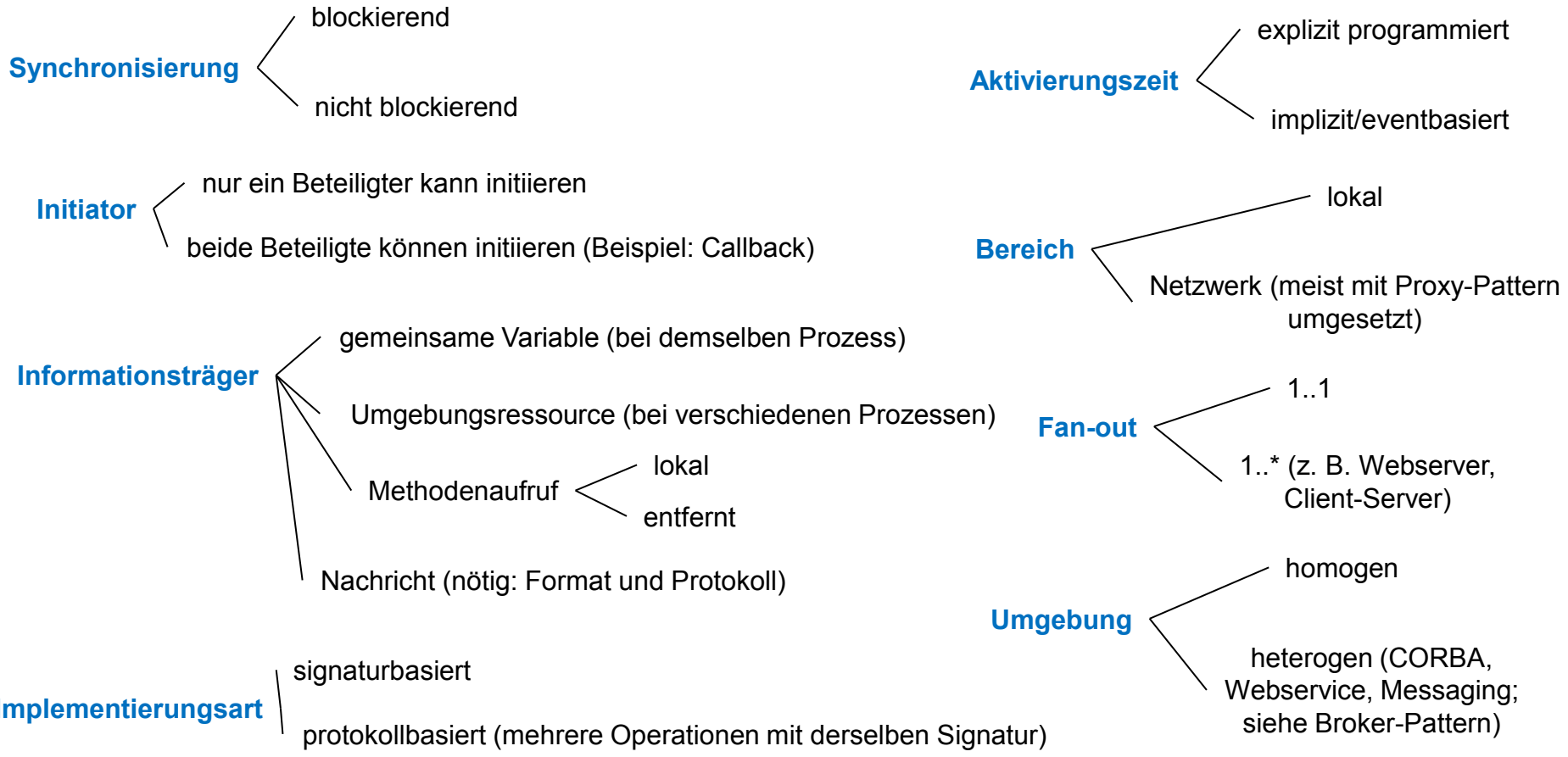
Rolle: Konvertierung

- ermöglichen Interaktion von unabhängig entwickelten Komponenten
- Diskrepanz besteht normalerweise zwischen Komponenten bzgl.
  - Typen, Formaten
  - Interaktionsreihenfolgen
  - Frequenz der Interaktion
- Beispiele von Konvertern
  - Adapter
  - Wrapper

Rolle: Moderation

- Vermitteln und “glätten” die Interaktion
- Stellen Performanzprofile sicher, z. B. durch Load-Balancing
- Stellen Synchronisation sicher
  - Kritische Abschnitte
  - Monitore

Konnektorklassifikation nach [Qian2010]



Arten von Konnektoren (nach [Taylor2010])

- Prozeduraufruf
- Ereignis (Event)
- Datenzugriff
- Kopplung (Linkage)
- Stream
- Vermittler (Arbitrator)
- Adapter
- Distributor

Prozeduraufruf-Konnektor

- Modellieren Kontrollfluss zwischen Komponenten
  → also Rolle „Koordination“
- Transferieren Daten zwischen Komponenten
  → also auch Rolle „Kommunikation“
- Ausprägungen (Beispiele)
  - Methodenaufruf in Objektorientierten Sprachen
  - „fork“ in Unix (http://de.wikipedia.org/wiki/Fork_(Unix))
  - Call-back in eventbasierten Systemen/Frameworks
  - Betriebssystemaufrufe

Ereignis (Event)-Konnektor

- Ereignis/Event: „*Alles was passiert, oder als passiert angesehen wird*“ [Event Processing Glossary]
- Falls ein Ereignis-Konnektor ein Ereignis wahrnimmt, wird eine Ereignisnachricht („event message“) durch ihn ausgelöst
  - an alle Interessengruppen verschickt
  - und dadurch der Kontrollfluss an diese delegiert
- Unterscheidung „einfache Ereignisse“ vs. „komplexe Ereignisse“ (Ereignismuster, siehe Complex-Event-Processing)
- Änderung der Interessensgruppen kann dynamisch erfolgen
  - im Ggs. zum Prozedurkonnektor

Datenzugriff-Konnektor

- Ermöglichen Komponenten den Zugriff auf Daten in einem Datenspeicher
- Übertragungs- und Speicherformate können verschieden sein
  - der Konnektor übernimmt somit auch die Rolle Konvertierung
- Beispiele
  - Persistente Datenzugriffs-Konnektoren
    - Relationale Datenbank, Zugriff mit SQL
    - NoSQL Datenbanken
  - Transiente Datenzugriffs-Konnektoren
    - Heap, Stack
    - Caches

Stream-Konnektor

- Streams werden verwendet, um große Datenmengen zwischen autonomen Prozessen zu transferieren
- Beispiele
  - Unix Pipes
  - TCP/UDP Sockets
  - `java.io.InputStreamReader`

Vermittler (Arbitrator)-Konnektor

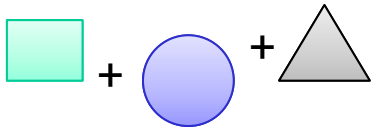
- Lösen Konflikte zwischen Komponenten auf
  - bieten also die Rolle Moderation
- Beispiele
  - Multithreaded-Zugriff auf einen gemeinsamen Speicher durch Sperren
  - Load-Balancing
  - Ausfallsicherheit (Nichtfunktionale Eigenschaft!) durch Backup-Systeme

Adapter-Konnektor

- ermöglichen Interaktion von Komponenten, die nicht mit dem Ziel der Interoperabilität entworfen wurden
- Beispiele
  - Verschiedene Middleware
    - CORBA
    - RPC
    - Java-.NET-Bridges
  - XML meta-data interchange (XMI) zum Austausch von Modellen zwischen Applikationen

Distributor (Verteilungs)-Konnektor

- bieten die Identifikation von Kommunikationspfaden und das Routing über diese Pfade
- immer in Kombination mit anderen Konnektoren (z. B. Prozeduraufrufen, Streams) verwendet
- Beispiele
  - DNS (domain name service), IP-Routing-Protokolle
  - JNS (Java Naming Service), Java RMI (Remote Method Invocation)
  - WCF (Windows Communication Foundation für .NET)
  - Diverse CORBA Services



Kombination von Konnektoren

- In manchen Systemen muss ein Konnektor aus verschiedene Arten zusammengesetzt werden
- Nicht alle Konnektoren können komponiert werden
  - manche sind nicht interoperabel
  - bei allen Kombinationsarten muss eine Abwägung von Vor- und Nachteilen erfolgen
- Beispiel: Peer-to-Peer Konnektoren (z. B. Bittorrent)
  - Vermittler
  - Datenzugriff
  - Stream
  - Verteilung (Distributor)

Auswahl von Konnektoren – Beispiel

Szenario

- Zwei separate Prozesse auf demselben Host
- Kommunikation
  - meist intra-Prozess
  - manchmal inter-Prozess
  - synchron
- Die Komponentenoperationen sind eher rechen- als kommunikationsintensiv

Auswahl von Konnektoren – Beispiel

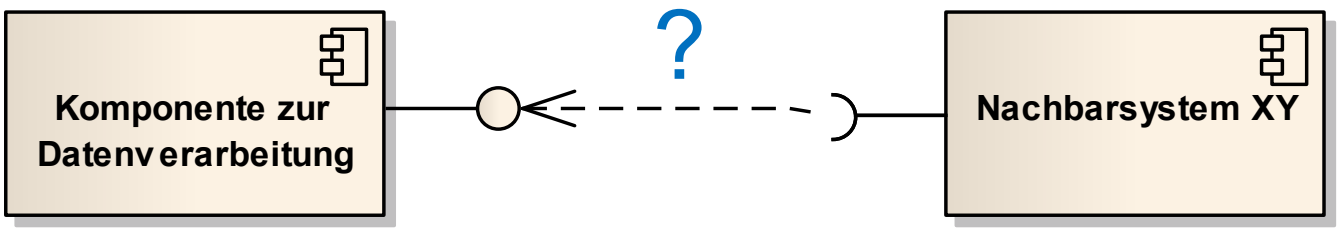
- Wähle den **Prozedurkonnektor** für die Intraprozesskommunikation
- Kombiniere **Prozedur- und Distributor-Konnektor** für die Interprozesskommunikation
  - falls wir alles in Java implementieren nehmen wir Java RMI
  - bei plattformübergreifenden Interaktionen eher Webservices, REST oder CORBA

Auswahl von Konnektoren – Weitere Szenarien

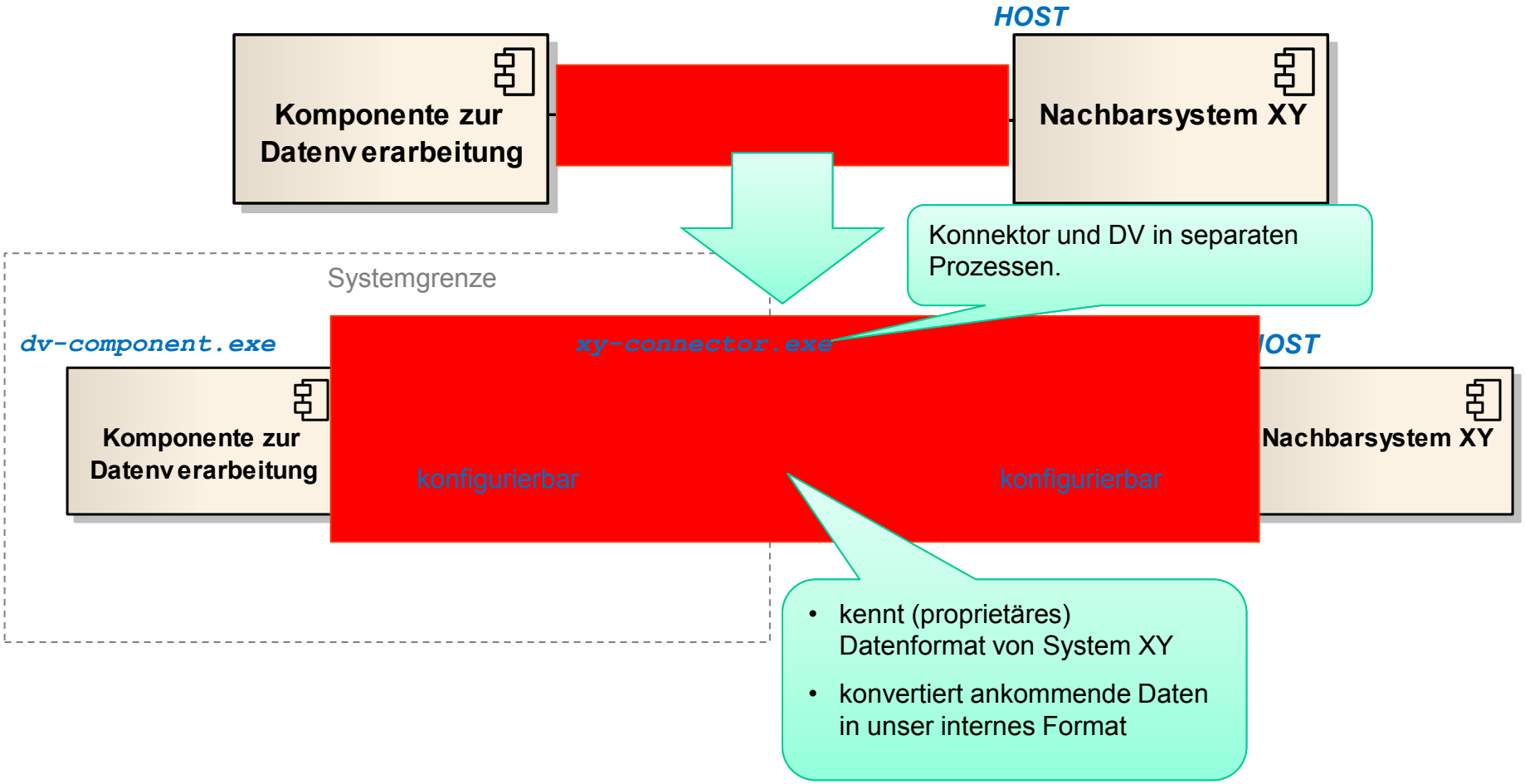
- Bei verschiedenen Prozessen
  - verwende lokale Message-Queue
  - verwende lokale Pipe
  - verwende ein lokales Verzeichnis als gemeinsamen Speicher
- Bei Komponenten auf verschiedenen Rechnern (statt RMI)
  - verwende Webservices
  - verwende ReST
  - verwende entfernte/verteilte Pipes
  - laufe mit einem USB-Stick zwischen den Rechnern hin und her (kein Scherz!)
- …

Beispiel aus einem realen Projekt – Anforderungen

- Eine Komponente unseres Systems erhält Daten eines Nachbarsystems XY und verarbeitet sie
  - Das Nachbarsystem wird durch einen Konnektor entkoppelt
  - Der Kommunikationskanal des Konnektors zum Nachbarsystem muss flexibel bleiben; was darf‘s sein?
    - heute Message-Queue-Schnittstelle?
      - welcher Queuename, welcher Port?
    - oder morgen Dateischnittstelle?
      - welches Verzeichnis?
    - oder …

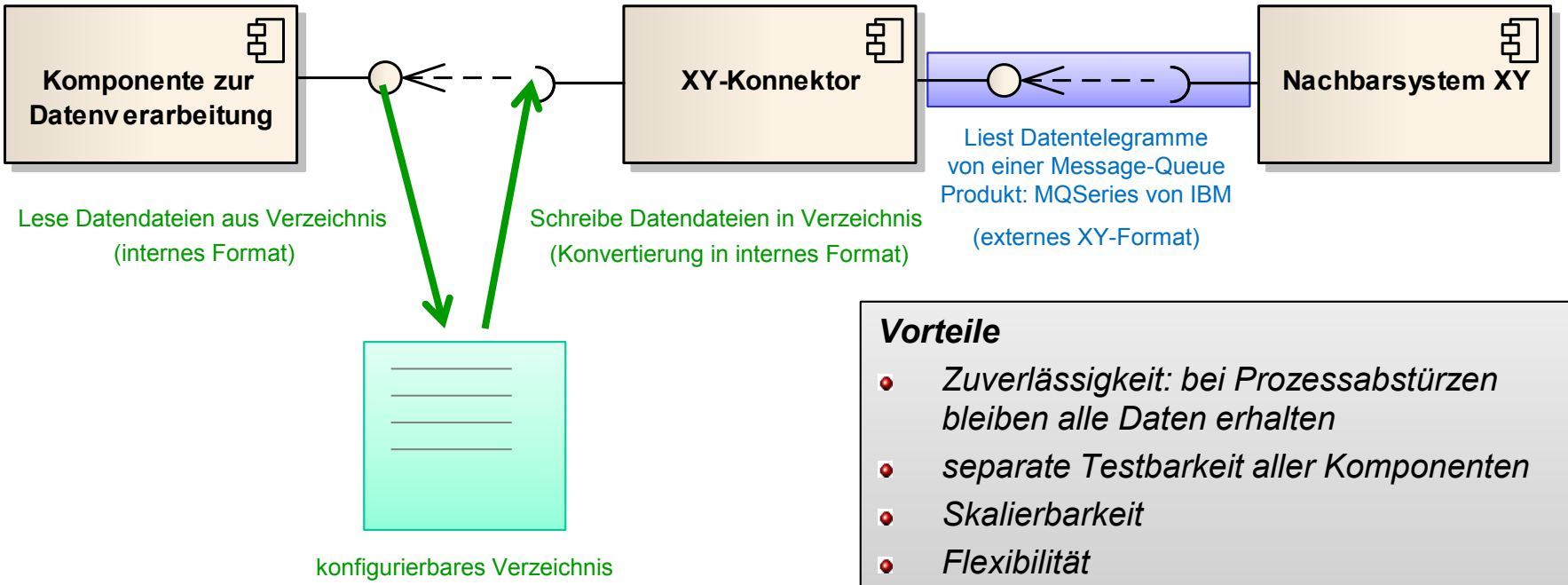


Beispiel aus einem realen Projekt – Technische Lösung



Beispiel aus einem realen Projekt – Konfiguration

- bei Auslieferung des Systems:



Vorteile

- *Zuverlässigkeit: bei Prozessabstürzen bleiben alle Daten erhalten*
- *separate Testbarkeit aller Komponenten*
- *Skalierbarkeit*
- *Flexibilität*

Technische Voraussetzung (!!!)

- *Dateisystem-Schreiboperationen sind atomar!*

Beispiel aus einem realen Projekt – Konfiguration(Code)


```
XYReaderUseCase::XYReaderUseCase(I_Reader_shr xyReader) throw () : fXYReader(xyReader) { … }
std::string& XYReaderUseCase::read() throw (Technica xception) {
  try {
    return convertToInteralFormat(
                fXYReader->readData());
  }
  catch (TechnicalEx tion& ex) { … }
}
```

- Übergabe des Readers „xyReader “ an die Komponente in main() (Dependency Injection!)
- Konfiguration des Readers in einer separaten Datei

- Dem Konnektor ist es egal wovon er liest
- danach kümmert er sich noch um die Datenkonvertierung in das interne Datenformat

Konfigurationsdatei (Skizze)

```
XY-Reader.Implementation = MQSeries
XY-Reader.Config[„Queue“] = „XYQueue123“
```

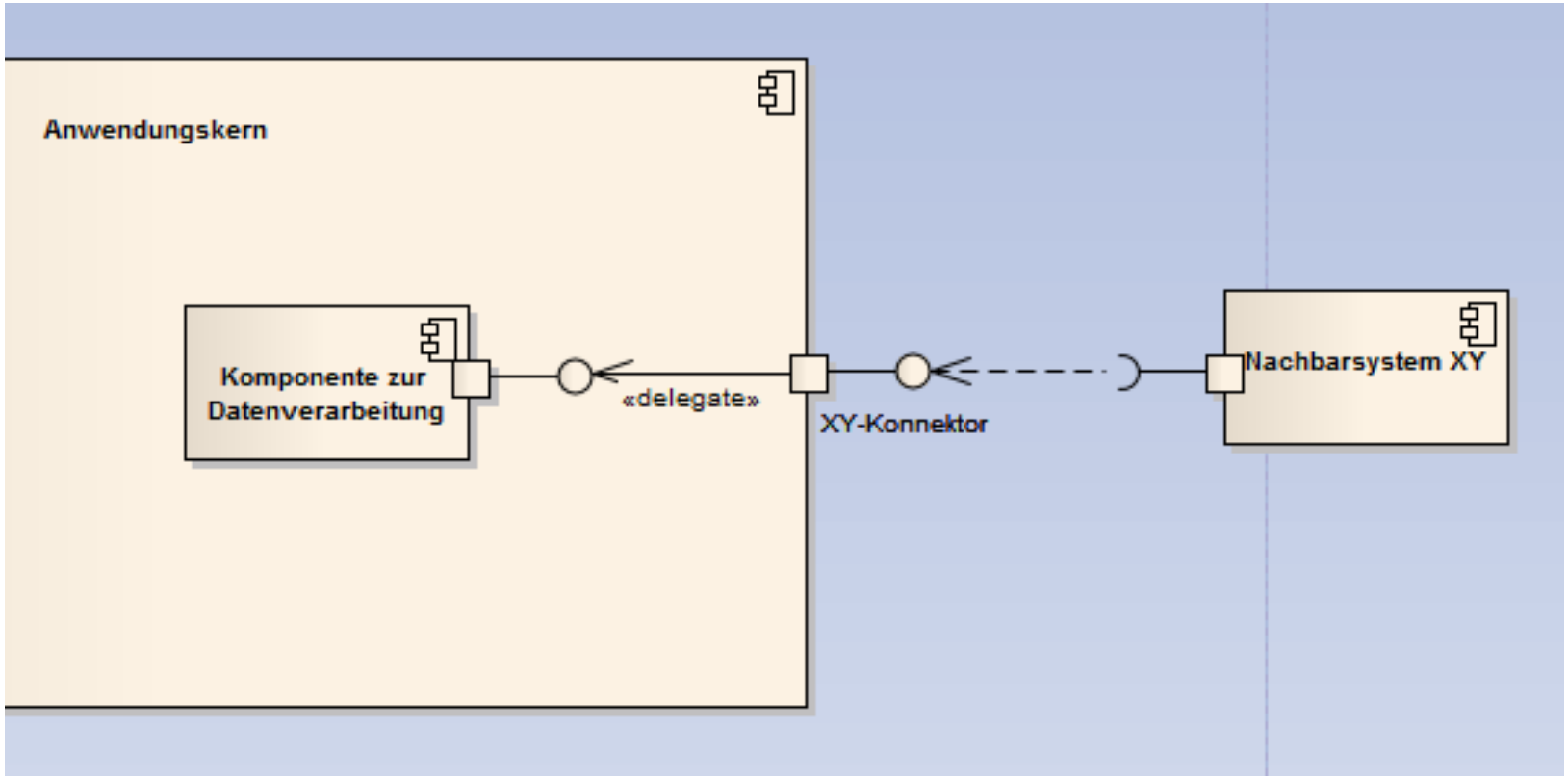
(analog für die Verbindung Konnektor->DV-Komponente)

Beispiel aus einem realen Projekt

Entwurfsdokument Demo

Beispiel aus einem realen Projekt – Modellierung

- Modellierungsmöglichkeit mit Ports



Konnektoren – Fazit

- Konnektoren erlauben die Modellierung beliebig komplexer Interaktionen
- Ihre Flexibilität ermöglicht eine System-Evolution, d. h.
  - Hinzufügen/Entfernen von Komponenten
  - Ersetzung von Komponenten
  - Neuverknüpfung von Komponenten
  - Migration von Komponenten
- Konnektoren-Austauschbarkeit ist erwünscht
  - ein Austausch sollte nicht die Systemfunktionalität beeinflussen
- OTS (“Off-the-Shelf”)-Konnektorimplementierungen ermöglichen die Fokussierung auf applikationsspezifische Funktionalitäten
- Schwierigkeit
  - Zerstreuung eines Konnektors in verschiedene Implementierungsmodule
- Die Schlüsselfrage ist
  - Performanz vs. Flexibilität