



Technische Hochschule
Ingolstadt

Fakultät für Elektrotechnik
und Informatik

*Zukunft in
Bewegung*

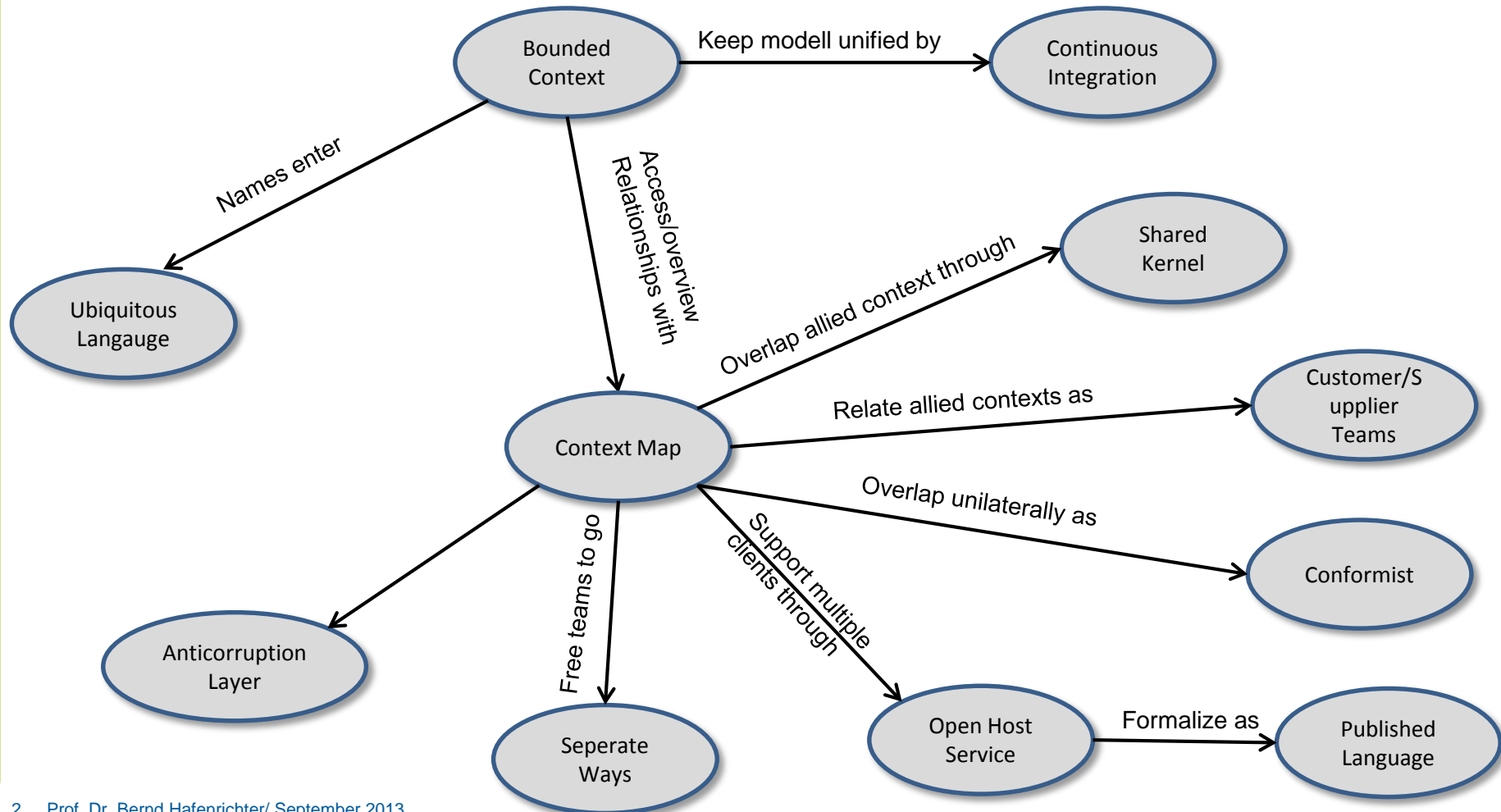
Domain-Driven Design

*Architektur- und Entwurfsmuster
der Softwaretechnik*


Prof. Dr. Bernd Hafenrichter 06.03.2015



Maintaining the Modell Integrity



Bounded Context



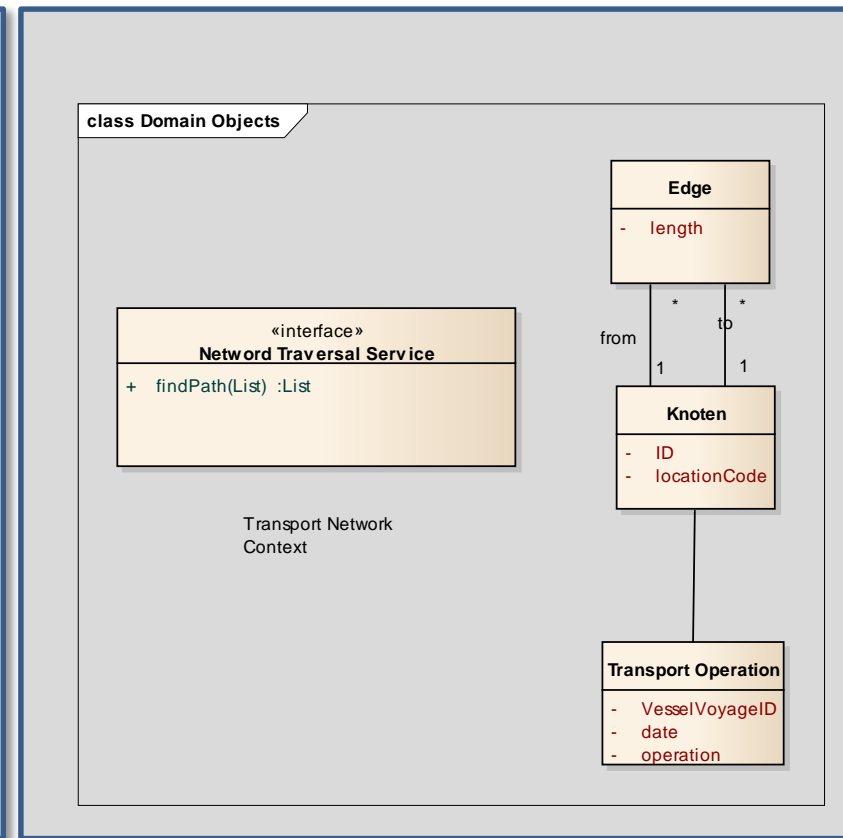
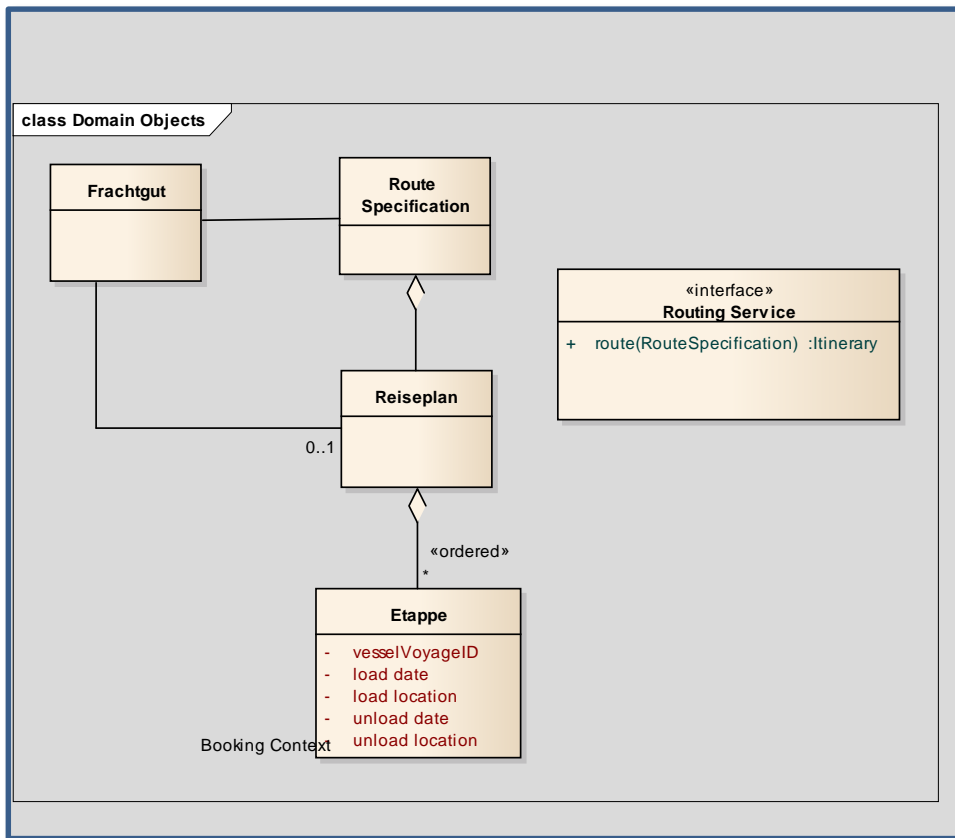
Bounded
Context

- In großen Projekte werden verschiedene Modelle verwendet
- Wird der Code (basierend auf verschiedenen Modellen) kombiniert wird die Software fehleranfällig, unzuverlässig und schwer verständlich
- Es ist nicht klar in welchem Kontext ein Modell nicht verwendet werden soll

Bounded Context

- Definieren Sie explizit den Kontext in welchem ein Modell Anwendung findet
- Weisen Sie explizite Grenzen in Form von z.B. Teamorganisation, Spezielle Anwendungsbereiche der Applikation, Codebases, Database schemas
- Das Modell muss konsistent innerhalb dieser Grenzen sein
- Lassen Sie sich nicht von Problemen ausserhalb beeinflussen

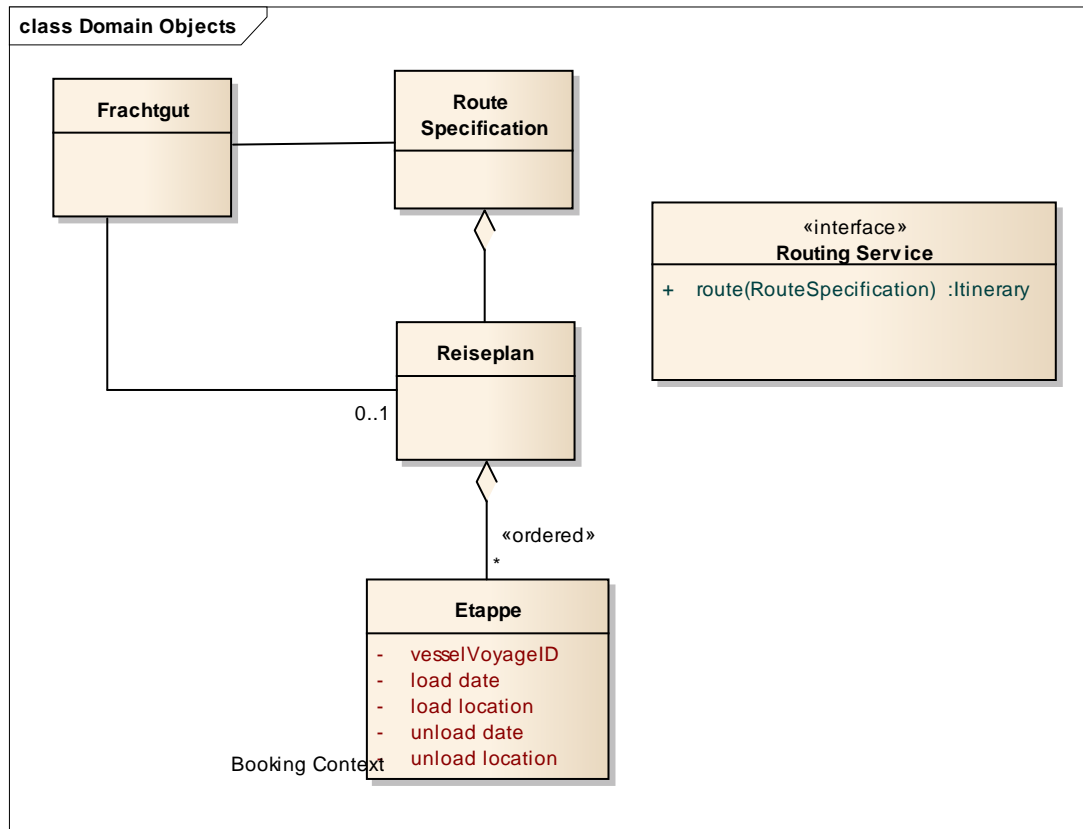
Bounded Context



Team 1: Reise- und Tourplanung

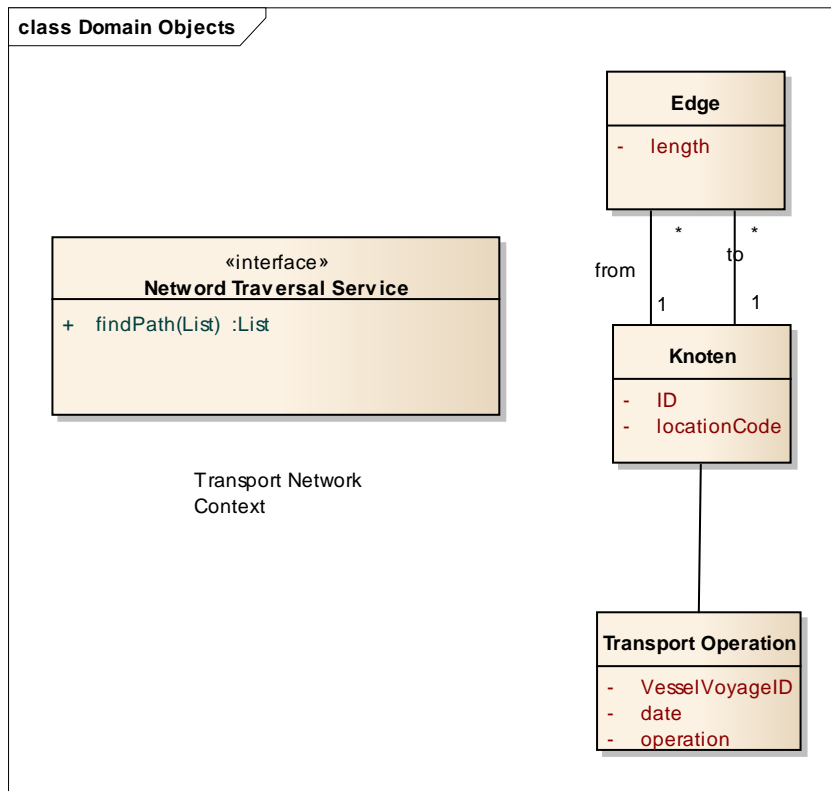
Team 2: Wegberechnung

Context Map



- Buchungssapplikation für Frachtgut
- Bei der Buchung soll online die beste Route berechnet werden
- Der RoutingService hat die Aufgabe die optimale Route zu berechnen

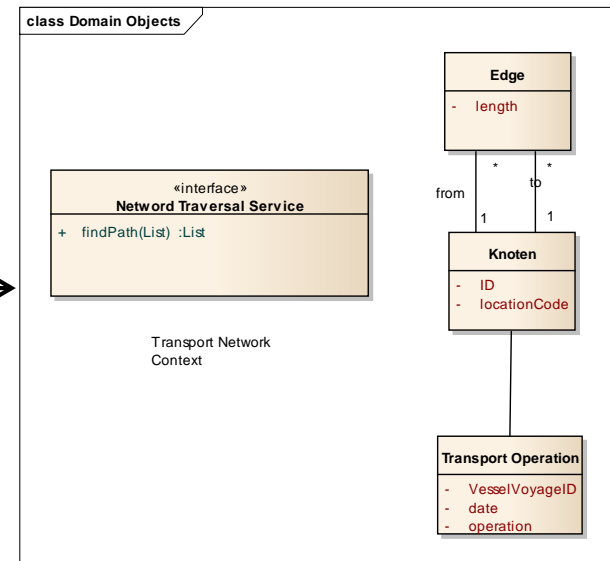
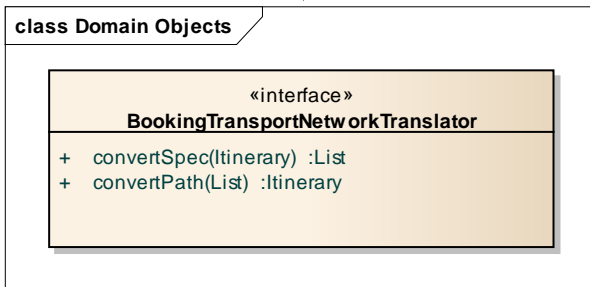
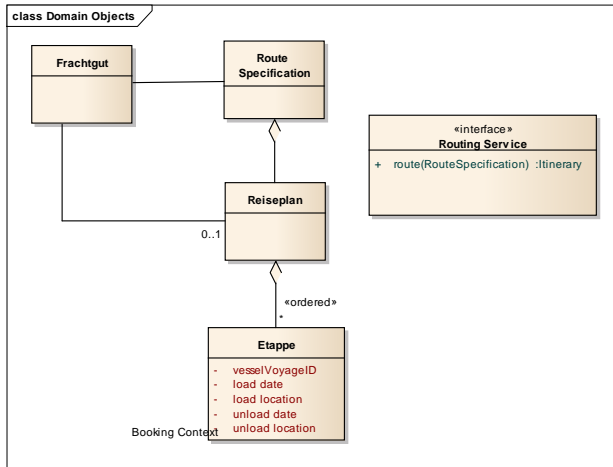
Context Map



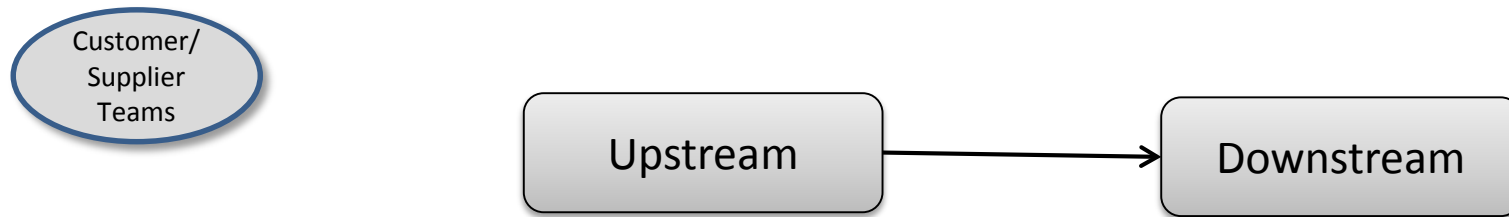
- Optimale Routenberechnung verwendet z.B. den Algorithmus von Dijkstra
- Die notwendigen Daten müssen in Form von Matrizen abgelegt werden um optimale Performance zu gewährleisten
- Lösungsansatz:
 - Definiere zwei Bounded Context
 - Definiere eine Abbildung der relevanten Objekte aus beiden Contexten

Context Map

- Beschreibe die Berührungspunkte zwischen den Modellen
- Bilde die relevanten Bereiche der Modelle aufeinander ab
- Definieren einen Transformationsservice



Customer/Supplier Teams



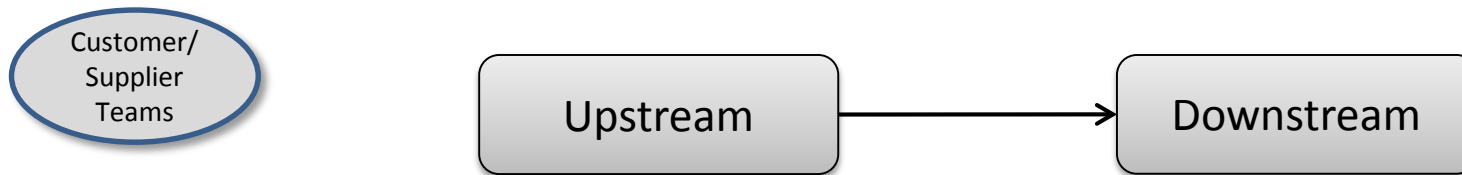
Ausgangssituation

- Mehrere Teams konkurrieren um die Änderung von gemeinsamen Ressourcen/Modellen

Problem

- Änderungen sind schwierig da immer mehrere Parteien betroffen sind

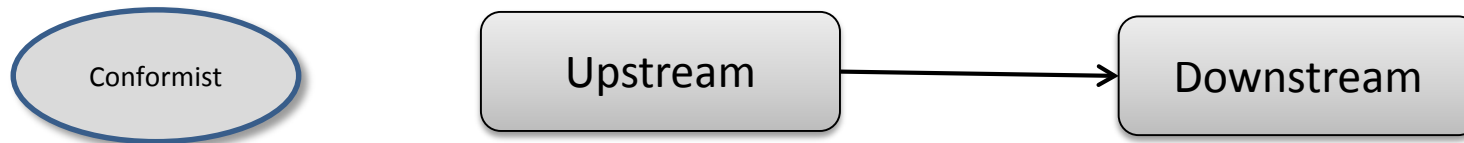
Customer/Supplier Teams



Lösung

- Definieren sie eine klare Customer/Supplier-Beziehung zwischen den Teams
- Das „Downstream“ Team ist der Kunde, das „Upstream“ Team der Lieferant
- Verhandeln und definieren Sie Kosten für Requirements des Downstreams
- Definieren Sie gemeinsame automatisierte Akzeptanztests.
- Fügen Sie diese Tests dem Upstream Team hinzu. (=Vorteil: Das Team kann entwickeln ohne das Risiko den Downstream negative zu beeinflussen)

Conformist



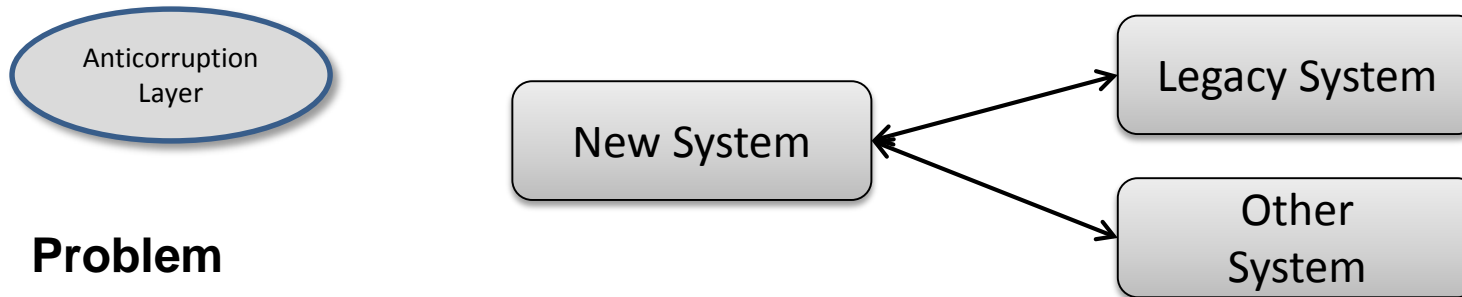
Problem

- Das Arbeiten in einer Customer/Supplier-Beziehung ist problematisch wenn der Supplier nicht willig ist seine Rolle anzunehmen
- Der Kunde ist hilflos

Lösung

- Der Kunde muss nehmen was er bekommt
- Downstream und Upstream teilen sich Fragmente des Domänenmodells
- Der Kunde verwendet das Domänenmodell des Upstream-Teams
- Reduktion der Komplexität bei der Übersetzung der Modelle
- Aber das Design des Downstreams wird vom Upstream beeinflusst

Anticorruption Layer



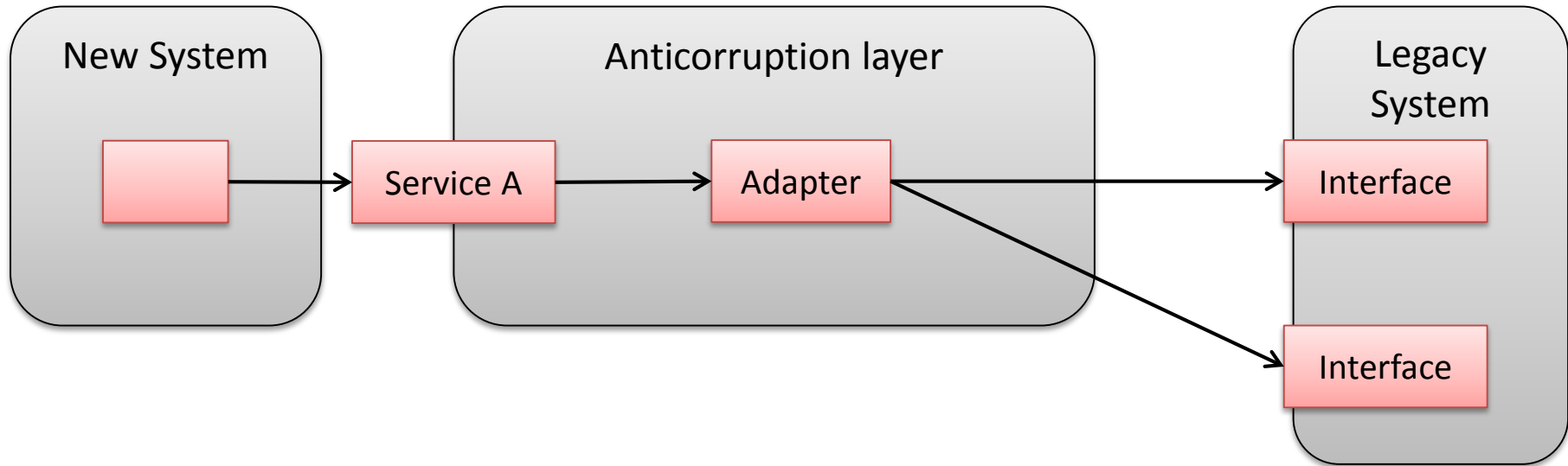
Problem

- Ein System muss immer mit Altsystemen (Legacy) bzw. existierenden Systemen integriert werden
- Die Modelle von Altsystemen sind oftmals widersprüchlich und labil
- Findet eine Vermischung statt, ist auch das neue System davon betroffen.

Lösung

- Definieren Sie eine „Isolation Layer“ welche dem Client externe Funktionalität innerhalb des eigenen Domänenmodells zur Verfügung stellt
- Die Layer kommuniziert mit den externen Systemen durch die existierenden Schnittstellen ohne bzw. mit geringer Modifikation.
- Intern ist die Isolation Layer ein Übersetzer in beide Richtungen


Anticorruption Layer



Patterns

- Adapter: Ein Wrapper welche dem Client ein anderes Protokoll zur Verfügung stellt als durch das Subsystem gesprochen wurde

Open Host Service



Open Host Service

- Ein Subsystem muss oftmals mit vielen verschiedenen anderen Systemen interagieren

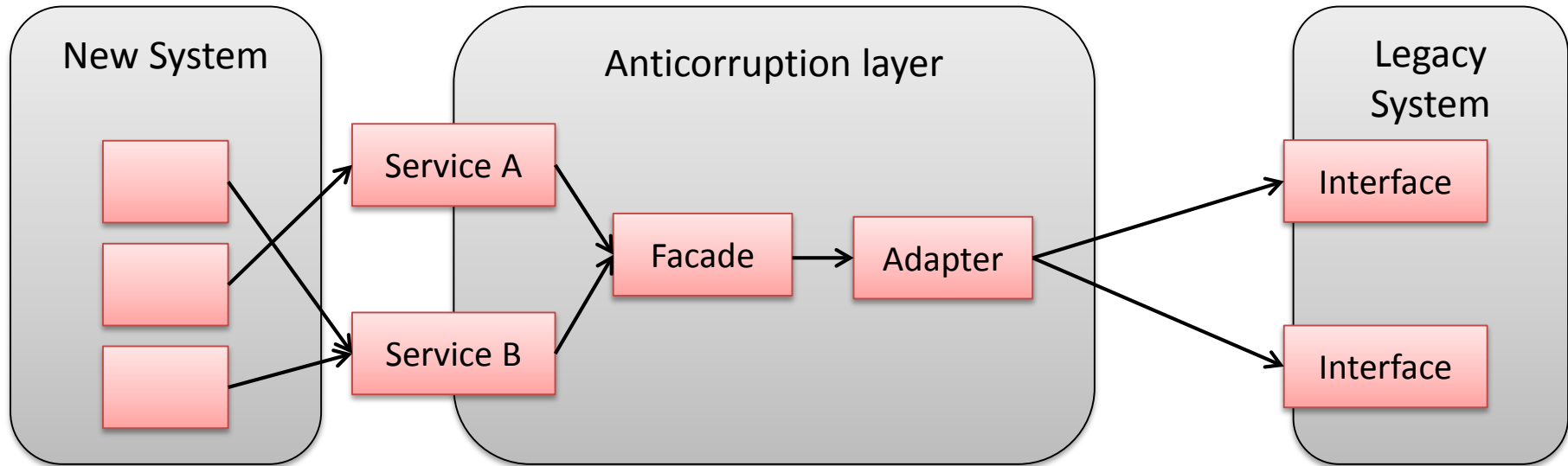
Problem

- Ein Translator pro Subsystem kann hinderlich und aufwendig sein
- Der Wartungsaufwand wird erhöht
- Die Entwicklungsteams werden gebremst

Lösung

- Definieren Sie ein Protokoll welches Zugriff auf das Subsystem mit einer Reihe von Services bietet
- Gestalten Sie das Protokoll so offen dass alle Nutzer es verwenden können
- Erweitern Sie das Protokoll falls notwendig


Open Host Service



Patterns

- Facade: Fasst die Implementierung mehrere Schnittstellen zusammen. Stellt die lose Kopplung sicher
- Adapter: Ein Wrapper welche dem Client ein anderes Protokoll zur Verfügung stellt als durch das Subsystem gesprochen wurde

Published Language



Published
Language


Problem

- Die direkte Übersetzung zwischen Modellen verschiedener Domänen kann aufwendig sein und stellt eine suboptimale Lösung dar
- Wird ein Modell als Austauschformat verwendet ist dieses Modell fixiert und kann nur schwer weiterentwickelt werden

Lösung

- Verwenden Sie eine öffentliche Sprache welche die Konzepte beider Modelle ausdrücken kann
- Beispiel: Standards zum Austausch von B2B-Informationen (z.B. ebXML)

Seperate Ways



Seperate
Ways

Problem

- Integration von Subsystemen ist aufwendig und teuer
- Der erzielte Nutzen ist teilweise sehr gering

Lösung

- Definieren Sie einen Bounded Context der keine Beziehungen zu allen anderen hat
- Dies erlaubt den Entwicklern einfache, spezialisierte Lösungen innerhalb eines schmalen Bereichs zu finden



Continuous Integration

Continuous
Integration

- Arbeiten viele Personen innerhalb des gleichen Context kann das Modell fragmentieren
- Es fehlt ein gemeinsames Verständnis
- Je größer das Team um so größer das Problem
- Unbewusste Änderungen an einer Modellklasse können Problem in anderen Bereichen der Software hervorrufen

Continuous Integration

- Entwicklungsartefakte werden regelmäßig zusammengeführt (=merge) so dass Fehler frühzeitig erkannt und behoben werden können
- Automated test suites
- Regeln die festlegen wann eine Änderung spätestens integriert werden muss

Maintaining Modell Integrity

