SOA Realisierungsstrategien

Code First vs. Contract First
- Ein Streitgespräch -

München, 17.01.2006
Marcel Tilly & Hartmut Wilms

marcel.tilly@innoq.com
hartmut.wilms@innoq.com

OOP 2006/ München

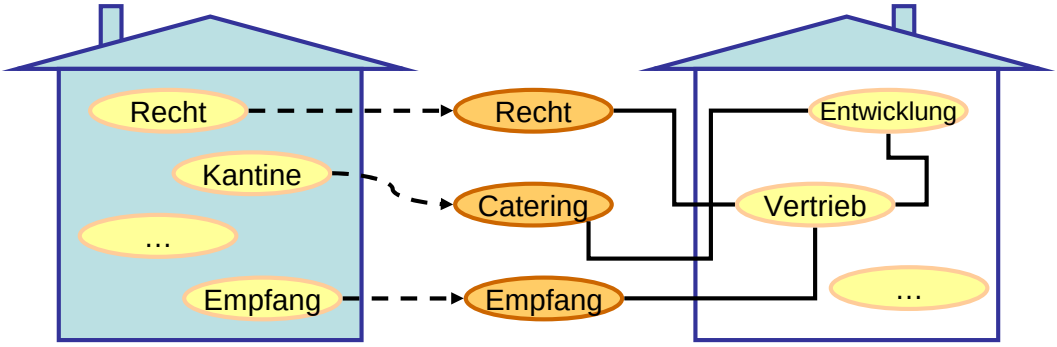
Die handelnden Personen

Der Tragödie I.Teil

Die SOA

Was ist eine SOA?

- *Service-orientierte Architektur*
- Architektur für Unternehmensweiten Einsatz
- Lose Kopplung ist ein Grundprinzip
- Services sind das Basiskonzept einer SOA



Der Service

Was ist überhaupt ein Service?

- ist ein Dienst
- Services repräsentieren Arbeitsvorgänge/Geschäftslogik
- Services sind gekapselt und autonom
- Services definieren eine Schnittstelle (zu ihren Nutzern)
- Ein Service kann aus anderen Services zusammengesetzt werden
- Service (Provider) und Service-Nutzer (Consumer) kommen in einem Vertrag (Contract) über die Inhalte der Interaktion und deren Form überein

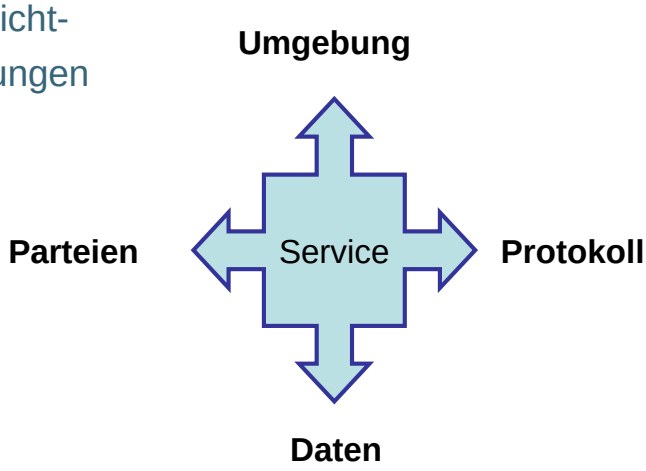
Der Contract

- Was ist ein Contract?
  Schnittstellenbeschreibung
  - Operationen
  - Daten
  - Vertrag (Policy)
    - Nicht-funktionale Anforderungen
    - Zusicherungen (Assertions)
  - Semantik
    - Funktionale Anforderungen
    - “Leistungen” des Service

Lose Kopplung

Lose Kopplung kann es in verschiedenen Ausrichtungen geben

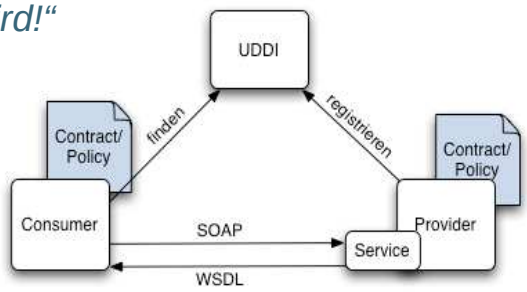
- zwischen Consumer und Provider
- zum verwendeten Protokoll
- zur Ablaufumgebung
- zwischen Daten und nicht-funktionalen Anforderungen



Der *Web* Service

- Eine SOA ist möglich ohne Web Services…
  - … aber auch sinnvoll?
- Web Services sind aber eine mögliche technische Abbildung einer SOA
- Man kann Web Services einsetzen, ohne wirklich ein SOA zu haben

- *„Web Services sind Services, auf die über Standard-Web-Protokolle zugegriffen wird!“*



Die Web Service Description Language (WSDL)

	`<definitions>`

XML-Schema der Datentypen	`<types>`
	`<schema></schema>`
	`</types>`

Parameter der Operationen	`<message>`
	`<part></part>`
	`</message>`

Service-Schnittstellen	`<portType>`
	`<operation></operation>`
	`</portType>`

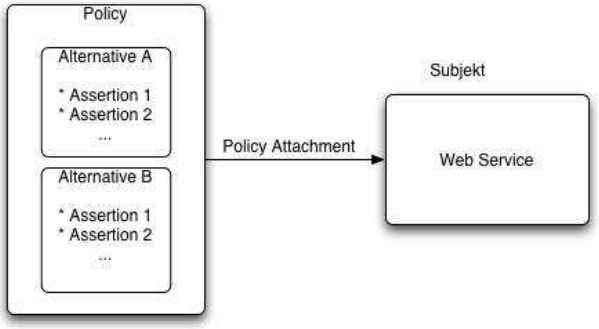
Kommunikations-Protokoll Mapping: SOAP, HTTP, etc.	`<binding>`
	`<operation></operation>`
	`</binding>`

Service-Endpunkte (URL)	`<service>`
	`<port></port>`
	`</service>`

	`</definitions>`

Die Policy

- Teil des Vertrags zwischen Consumer und Provider
- wird vor dem Nachrichten-austausch vereinbart



- Eine Policy wird an einen WS angehangen (Attachment)
- Eine Policy besteht aus Assertions (auch Alternativen)

SOA/Services entwickeln

- Wie entwickle ich ein SOA mit Web Services?
- Was muss ich etwas beachten?
  - Lose Kopplung
  - Interoperabilität
- Wie gehe ich praktisch vor?

Im Dialog/ Streit

Der Macher (aka *Mr. Code*)

- Wichtig ist schnell ein lauffähiges System zu erstellen
- Lieber probieren und testen
- Planen auf ein Minimum reduziert
- Pragmatisches Vorgehen
- keine *EierlegendeWollMilchSau* erschaffen, sondern **das** Problem lösen

Der Planer (aka *Mr. Contract*)

- Möchte mögliche Probleme bzgl. Interoperabilität, Änderungen und Versionen vermeiden
- Das Design legt die Basis für eine erfolgreiche Realisierung

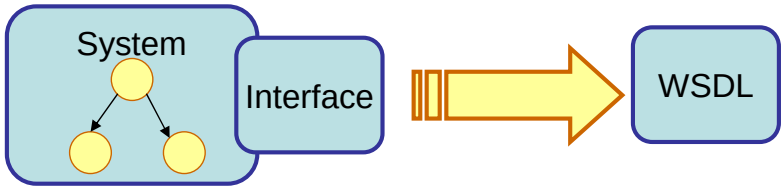
Handlung und Höhepunkt

Der Tragödie II.Teil

Der Macher: Status Quo

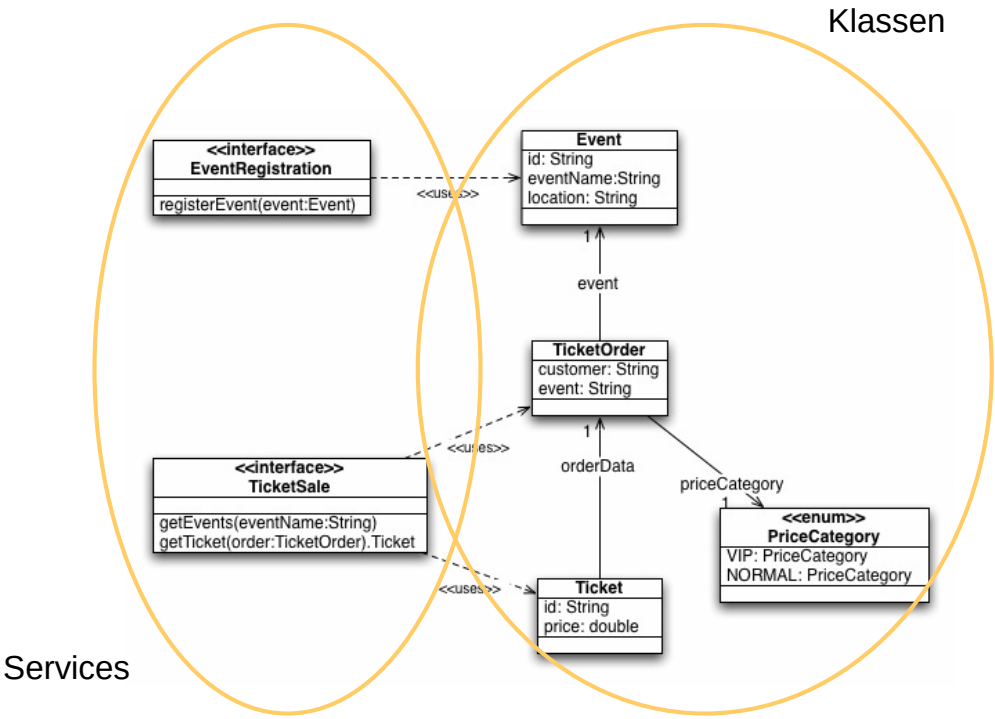
- Es existieren bereits Systeme
  - Wiederverwendung
  - Integration
  - Diese könnten ihre Schnittstellen doch als Services anbieten
    - machen Xyz2WSDL (z.B. Java2WSDL)

- Sieh mal wie einfach das geht!!!



Der Macher: Unser Modell…

Ein kleines Beispiel: Der TicketShop!



Der Macher: Implementation

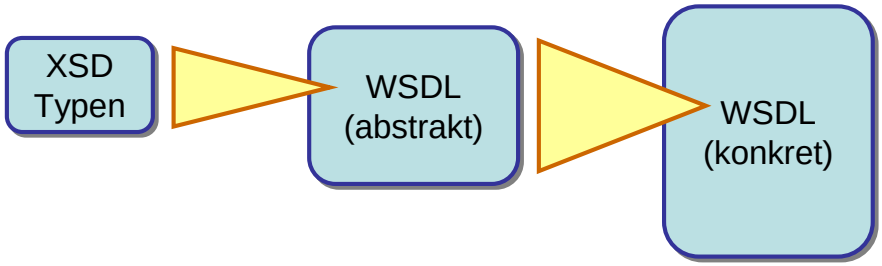
```
public interface TicketShop extends TicketSale, EventRegistration
{

        /**
         * Liefert alle registrierten Events.
         */
        List getEvents(EventData pSearchData)
                throws SecurityException;

        /**
         * Führt eine Bestellung für eine bestimmt Order aus
         * und liefert das Ticket zurück.
         */
        Ticket getTicket(TicketOrder pOrder)
                throws SecurityException, EventNotFoundException;

        /**
         * Dient zum Registrieren eines Events
         */
        void registerEvent(Event pEvent)
                throws SecurityException;
}
```


Der Planer: Planung ...

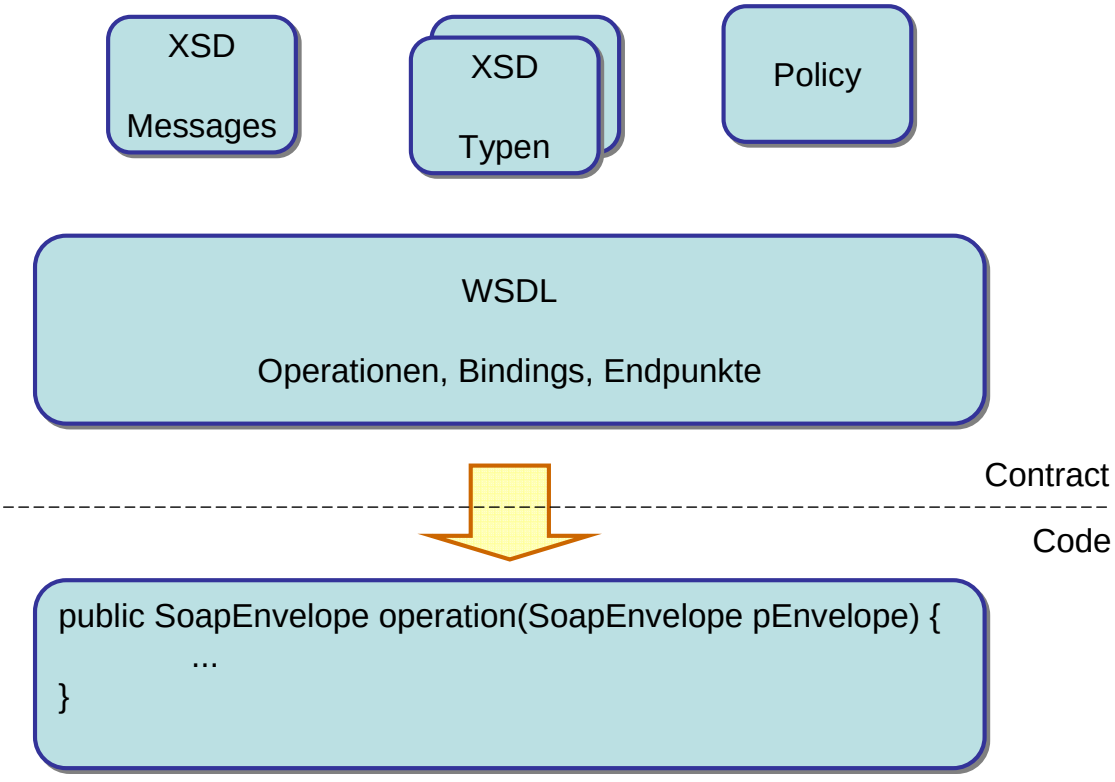
- In stark verteilten (heterogenen) Systemen ist eine vorausblickende Planung unerlässlich
  - Interoperabilität
    - Lingua Franca
    - gemeinsamer Nenner
  - Kollaboration (auch mit dem Feind ✪)
    - Parallele und einheitliche Entwicklung
    - Iterationen und Best Practices
  - Änderungen und Versionen
    - Lose Kopplung mittels solidem Design

Der Planer: ... und Design

- Services sind keine Klassen, Interfaces oder Methoden
- Services interagieren durch Nachrichtenaustausch
- Message-Metapher
- XML, XML Schema (und WSDL)
  - Basis der Kommunikation
  - Kein Implementationsdetail
- Strukturierung von XSD und WSDL

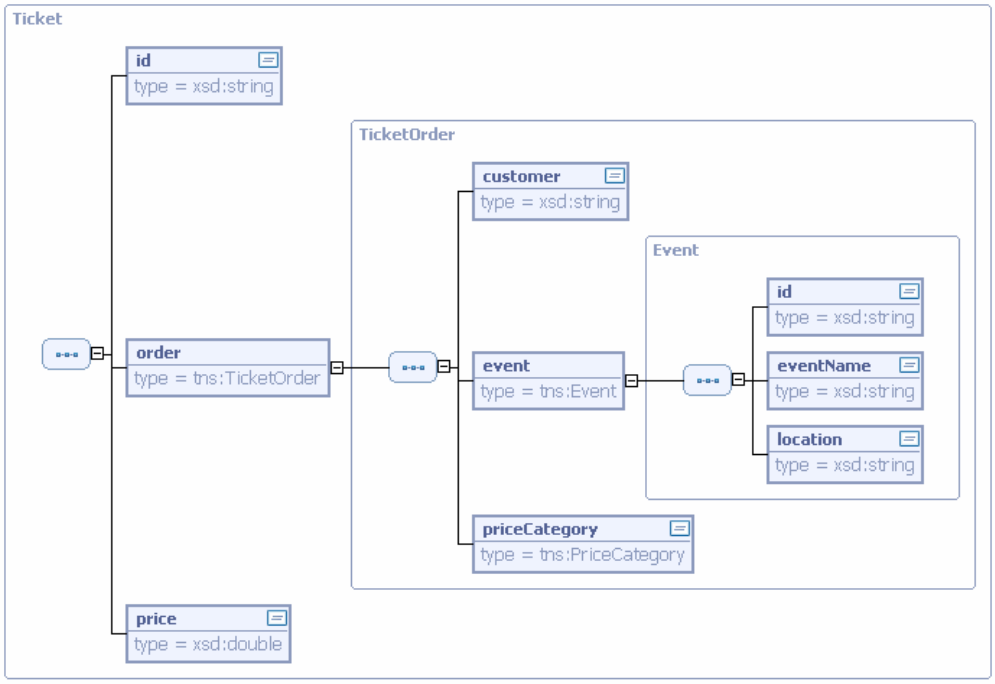


Der Planer: Vom Contract zum Code



Der Planer: Contract-First Schritt 1

Definieren der Typen in der XSD

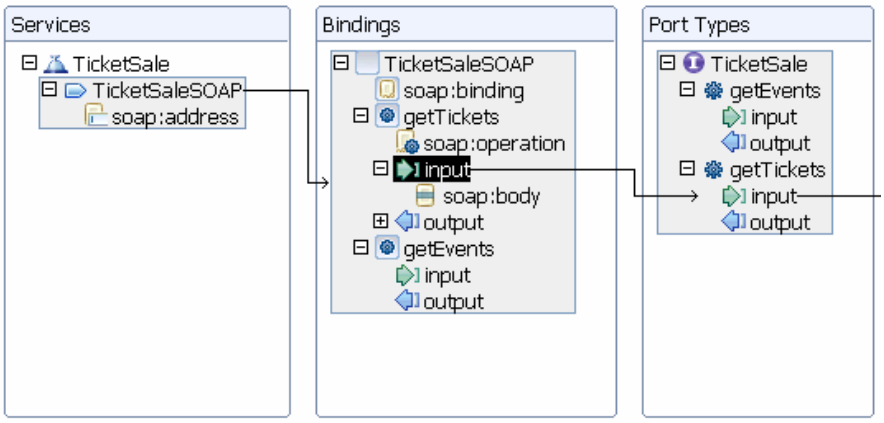


Der Planer: Contract-First Schritt 2

Definieren der Messages, Operationen und Interfaces in der WSDL

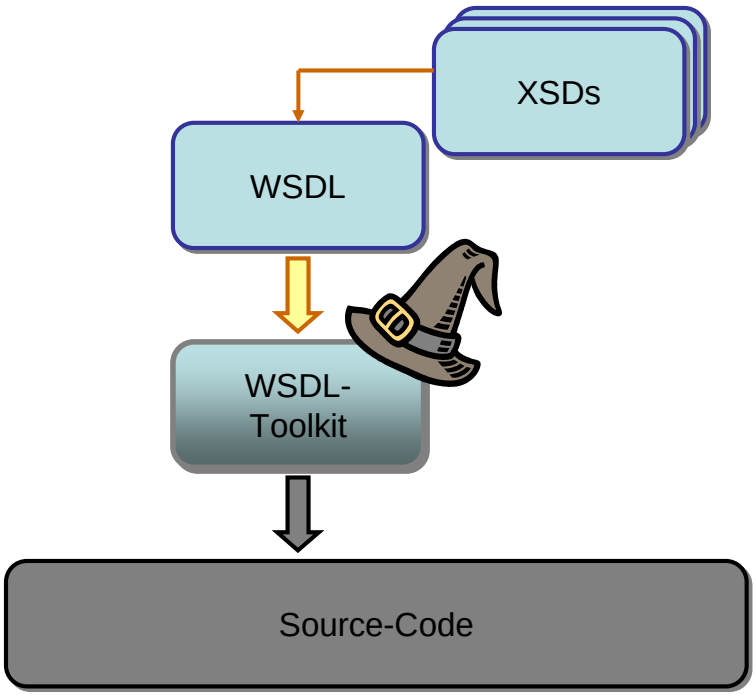
Der Planer: Contract-First Schritt 3

Definieren der Bindings und Services in der WSDL



Der Planer: Contract-First Schritt 4

Generieren der Implementationsrahmen aus der WSDL



Der Planer: Contract-First Schritt 5

Implementieren der Logik in beliebiger Sprache

```
public Ticket getTicket(TicketOrder pOrder) {
       // rpc-style via Proxys
}

       ... oder ...

public SoapEnvelope getTicket(SoapEnvelope pOrder){
       // document-style via Toolkit/XML-API
}
```

! Implementationsdetail !

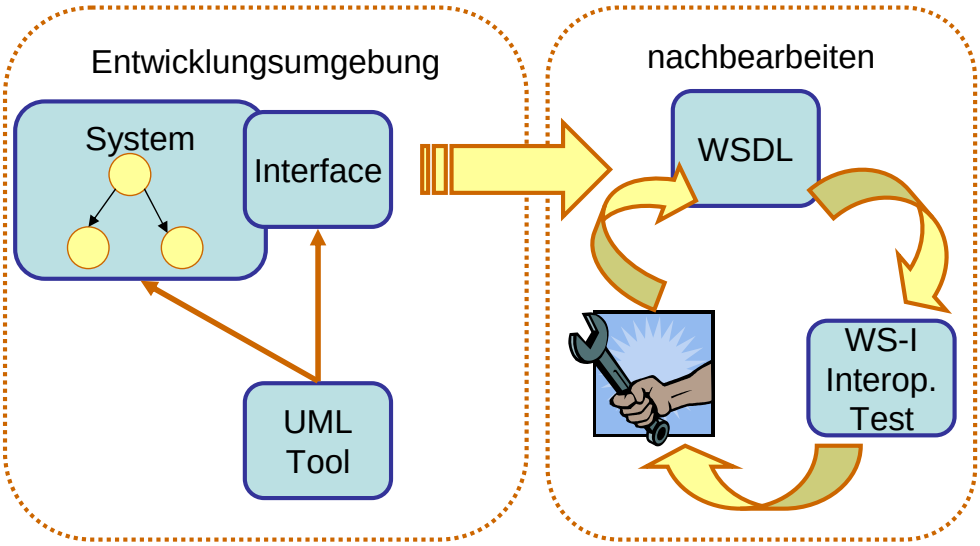
Der Macher: OO-Welt

- Meine Welt ist *OOP*
  - Das Umdenken zu OO war schon teuer und langwierig
  - Meine Investition in das teure OOAD-Tool soll gesichert sein
  - Meine Entwickler sind gewohnt mit der IDE zu arbeiten und damit produktiv zu sein
- UML-Modelle sind verfügbar
  - Wir haben modelliert
  - Wir nutzen Reverse/Forward/Roundtrip-Engineering

- *„Jetzt soll ich wieder viel Geld ausgeben und zunächst die WSDL definieren. Das ist aber aufwendig!“*

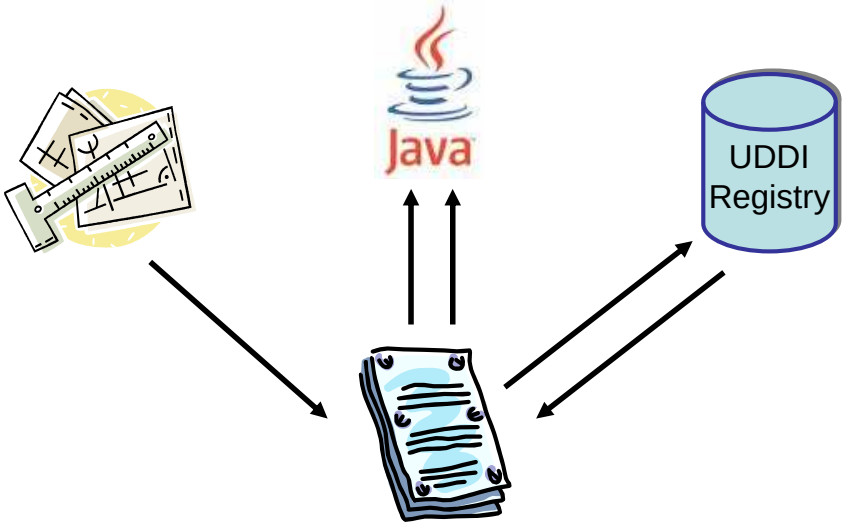
Der Macher: Pragmatisch

- Definition von Klassen und Interfaces
- Generieren der WSDL
- Validieren mit WS-I Tools
- Bei Bedarf anpassen



Der Planer: Contract ist die Basis

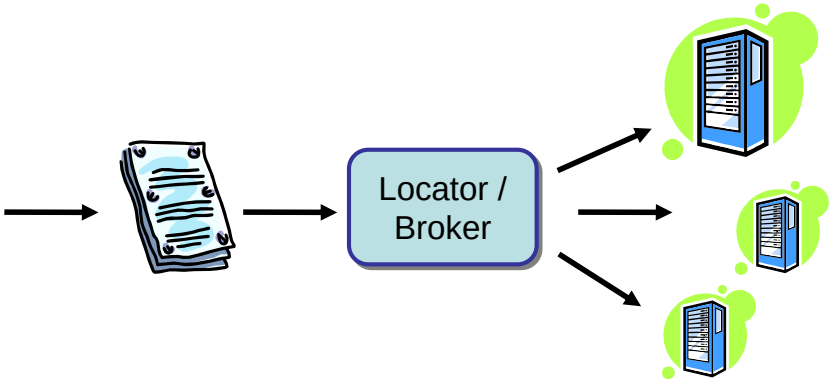
- Entwurfs-, Realisierungs- und ein Laufzeit-Artefakt



- Contracts werden *entworfen* und nicht generiert!
- Mehr als ein “Interface”

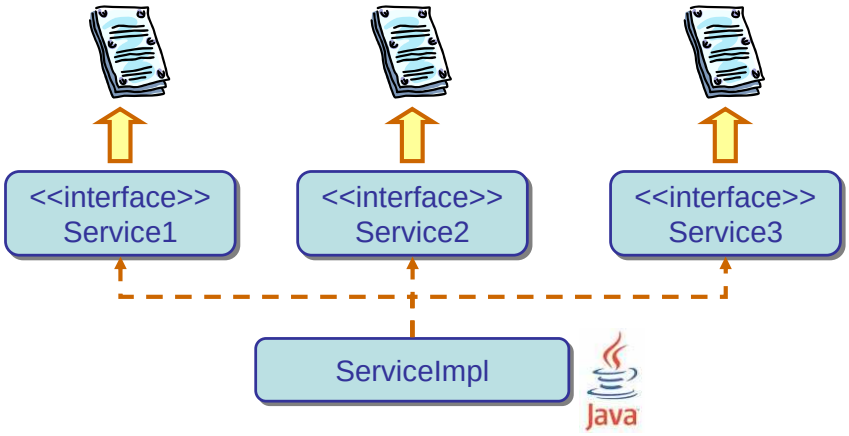
Der Planer: Contract ist mehr ...

- Impedance-Mismatch
  - XML Schema ist mächtiger als die Typenmodelle von C#, Java, ...
  - Optionale Elemente
  - Constraints
- Mehrere Contracts für eine Service-Implementation
- Dynamische Zuordnung von Contract zu Implementation



Der Macher: Verwenden von Metadaten

- Es ist möglich die fehlenden Informationen im Code zu hinterlegen
  - mit Annotations in Java
  - mit Attributen in C#
- Eine Serviceimplementierung kann natürlich auch verschiedene Interfaces umsetzen



Der Planer: Meta-Daten vs. WSDL

„Hilfe, ich sehe den Code vor lauter Meta-Daten nicht.“

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("wsdl", "2.0.50727.42")]
[System.Web.Services.WebServiceBindingAttribute(Name="TicketSaleSOAP",
Namespace="http://www.innoq.com/TicketShop/TicketSaleService/")]
public interface TicketSaleSOAP {

    /// <remarks/>
    [System.Web.Services.WebMethodAttribute()]

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www.inn
oq.com/TicketShop/TicketSaleService/getTickets",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Bare)]
    [return:
System.Xml.Serialization.XmlElementAttribute("getTicketsResponseType",
Namespace="http://www.innoq.com/TicketShop/TicketSaleTypes/")]
    TicketOrder getTickets([System.Xml.Serialization.XmlElementAttribute(
Namespace="http://www.innoq.com/TicketShop/TicketSaleTypes/")]
    TicketOrder getTicketsRequestType);
}
```


Der Macher: Macht auch Design

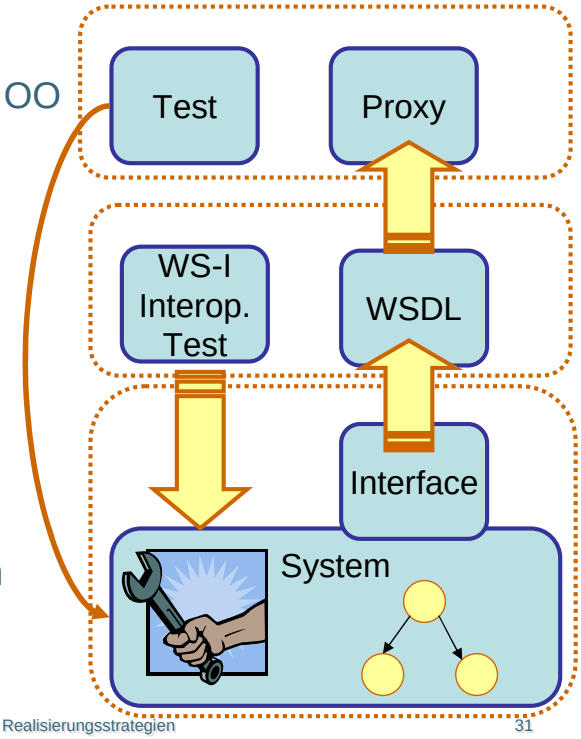
- auch bei Code-First macht man ein Design
  - in der Sprache, die man schon lange benutzt: UML
    - etabliert
    - Prozesse sind klar: Analyse → Design
  - in dem Tool, dass man bereits jahrelang einsetzt
- Prozesse können beibehalten werden
  - Angereichert durch ein Xyz2WSDL

Der Macher: Realisierungsstrategie

- setzt auf bestehende Systeme auf
- man kann in seiner „Gedankenwelt“ bleiben, z. B. OO
- schnell etwas Lauffähiges
  - WSDL
  - WS-I Interop
  - Proxy
  - Tests

  ➔Iterativ-inkrementell

- … die *Java2WSDLs* werden immer besser



... und jetzt?

- Code-First ist zielführend bis zu einem gewissen Grad
  - Sinnvoll bei einfachen Web Service-Anwendungen
    - Consumer und Provider in einer Hand
- Contract-First ist möglicherweise zu überdenken
  - WSDL zu kompliziert
    - Alternativen: SSDL, NSDL, SMEX-D, Resedel…
  - Fehlendes Tool
- Mh, irgendwie kommen wir so nicht weiter ☹
- Ok, wir fassen noch mal zusammen!

Die Lösung (oder Katastrophe?)

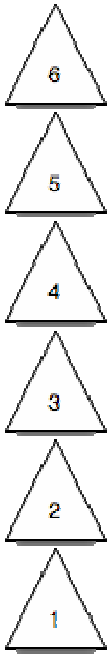
Der Tragödie III.Teil

Realisierungstrategie: Bottom-Up

Code-First

- ☺ Setzt auf bestehenden Entwicklungen und Artefakten auf
- ☺ Implementationsnah, vorhandenes Know-how nutzen
- ☺ Schnelle Resultate
- ☺ xyz2WSDL – Tools werden immer besser

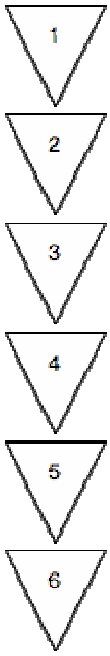
- ☹ Interoperabilitätsprobleme werden oft zu spät erkannt bzw. nicht berücksichtigt
- ☹ Entwicklung im (Kollaborations-)Blindflug
- ☹ Impedance-Mismatch



Realisierungstrategie: Top-Down

Contract-First

- ☺ Macht Probleme im Umfeld von stark verteilten Systemen bewusst
- ☺ Erzwingt eine Service-Sicht-der-Dinge: „Contract (Schnittstelle) ist relevant – Implementation wird zu einem internen Detail“



- ☹ Stellt sehr hohe Anforderungen an Know-how und Geduld
- ☹ Umständlich und teilweise unpraktikabel
- ☹ Und die Tools fehlen doch

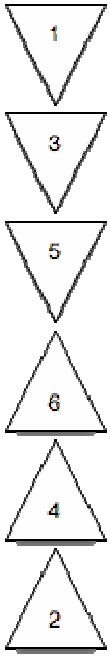
Realisierungstrategie: Meet in the Middle

Beginnen mit Code-First

WSDL generieren („erster Wurf“)

WSDL nachbearbeiten

Code aus bearbeiteter WSDL generieren

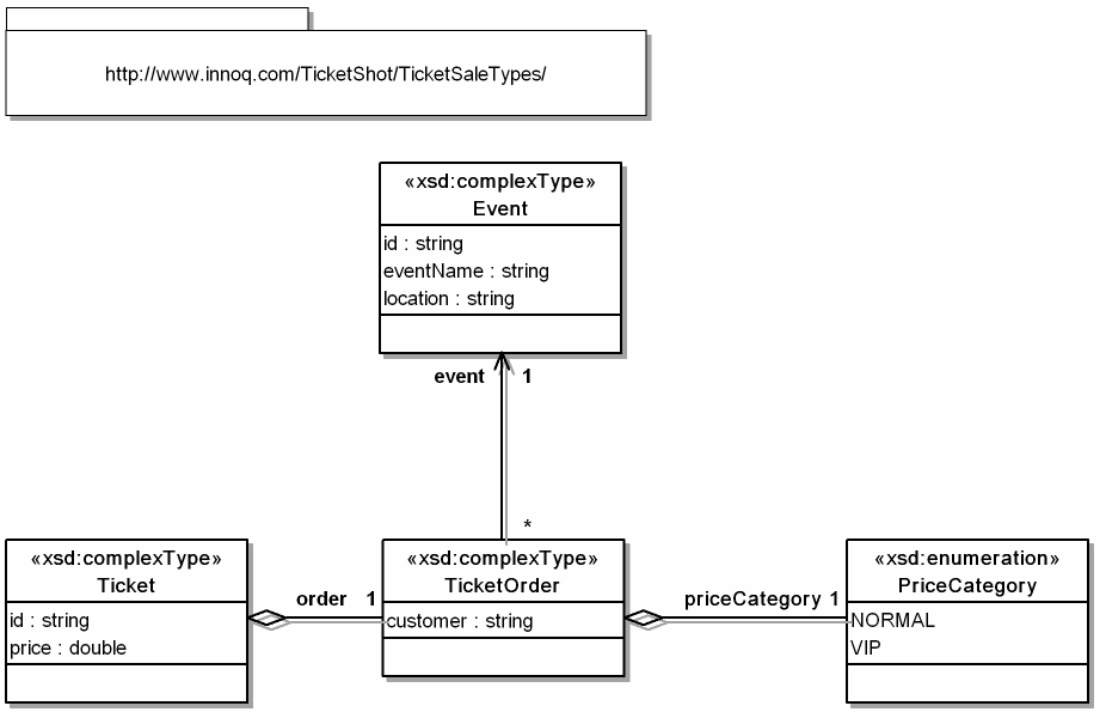


Was passiert bei mehreren Iterationen?

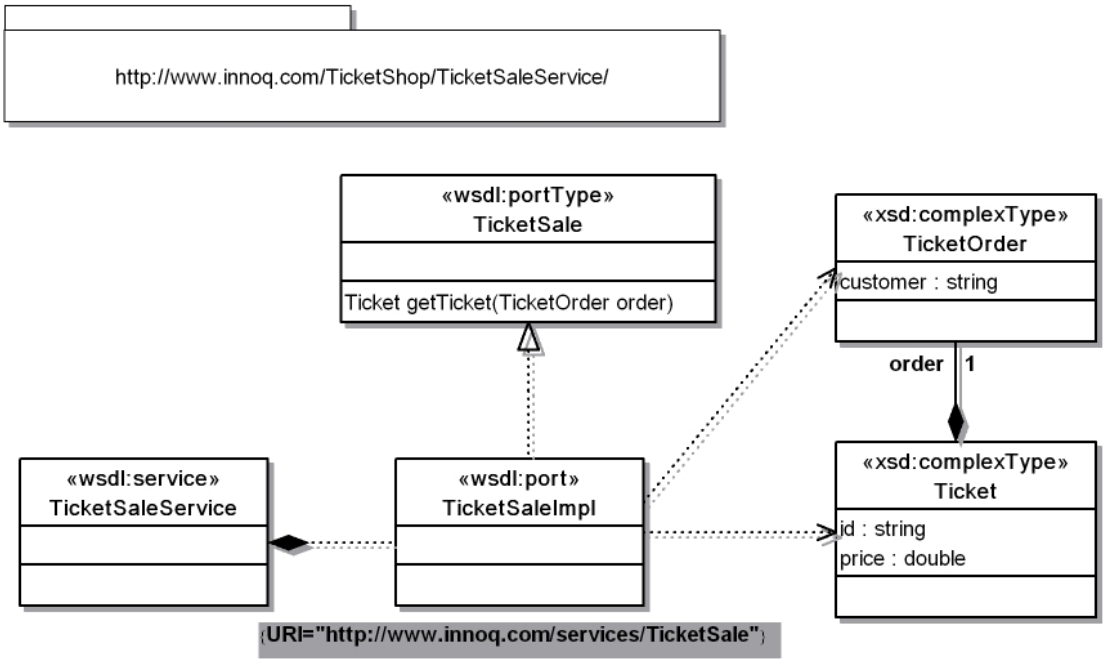
Eine Alternative: MDD & SOA

- **Typen, Operationen und Services als UML-Modell**
  - Welt, die wir kennen
  - WSDL/XSD-UML-Profile (Stereotypes)
- **Generieren der XSD und WSDL aus dem Modell**
  - Template-basierter Code-Generator
  - Transformationslogik steckt in den Templates
  - Erweiterbar durch andere Input-Modelle
    - Policies
- **Generieren der Implementationsrahmen**
  - Interfaces und Klassen in Java, C#, ...
  - Via WSDL-Toolkit oder direkt durch den Generator

Modellieren der Typen



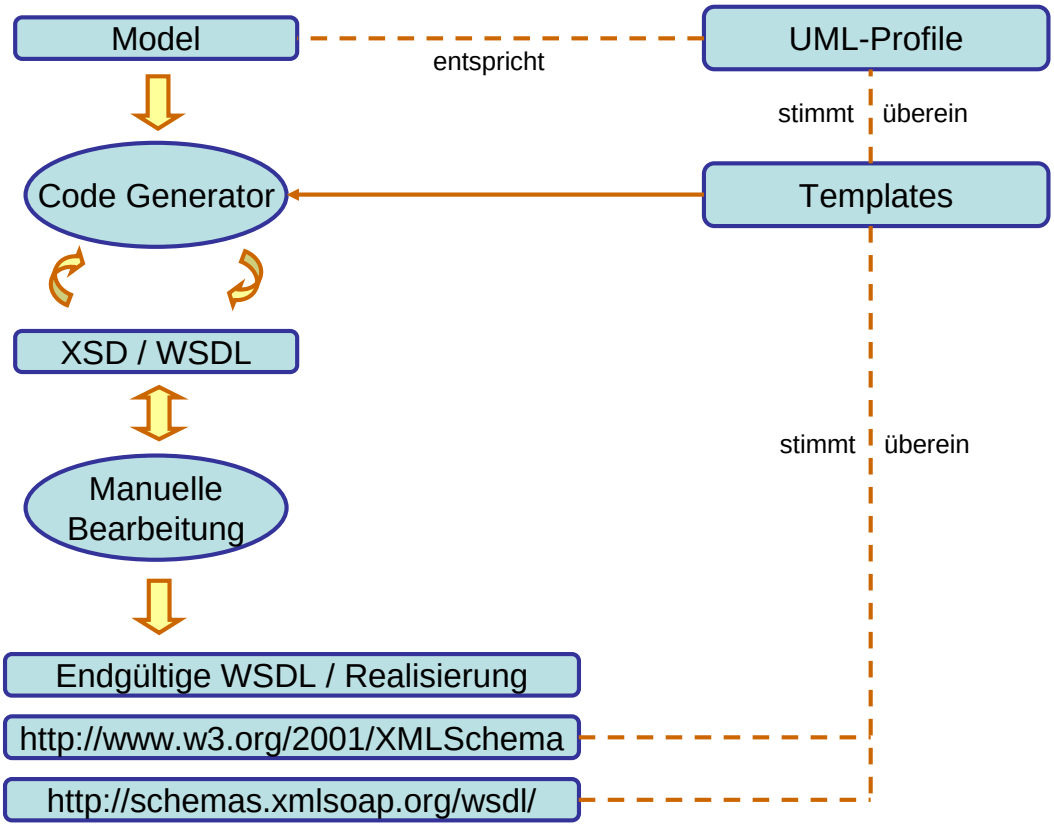
Modellieren der Service-Interfaces



Modellieren der Bindings?

- Bindings sind unabhängig vom Design
  - Bindings sollten leicht zu ändern sein
- Die Bindings werden in der Transformationslogik definiert
  - Templates
    - Template für SOAP-Binding
    - Template für TCP-Binding
    - ...

Template-basierte Generierung



Fazit…

- Es gibt nicht DIE Lösung!
- Jeder muss für sich die passende Strategie finden:
  - Bisher reicht in vielen Fällen Code-First
  - In heterogenen Umgebungen bietet sich die Mixed-Lösung an
  - In komplexen heterogenen und stark verteilten Umgebungen ist es sinnvoll, den Contract-First Ansatz zu wählen

…oder!

- Bei Erfahrungen mit Model-Driven-Development und existierenden UML-Resourcen bietet sich der MDD-Ansatz an
  - UML-Modelle
  - Template-basierter „Code“-Generator
  - WSDL2xyz-Tool

Q&A

Vielen Dank!

Bei Fragen erreichen Sie uns unter:

marcel.tilly@innoq.com
http://www.innoq.com/blog/mt/

hartmut.wilms@innoq.com
http://www.innoq.com/blog/hw/

Artikel & Links

- "UML for Web Services"; Will Provost; August 05, 2003;http://webservices.xml.com/pub/a/ws/2003/08/05/uml.html
- "Java und .NET: Interoperabilität jenseits von Theorie und Spezifikation"; Christian Weyer, Marcel Tilly; dot.net Magazin Sonderheft "Konzepte und Lösungen für moderne Software-Architekturen", 4/2005

Tools, Server & Frameworks

Axis
- http://ws.apache.org/axis/

Axis 2
- http://ws.apache.org/axis2/

Sandesha
- http://ws.apache.org/sandesha/

Systinet Server for Java
- http://www.systinet.com/

Windows Communication Foundation (WCF aka Indigo)
- http://msdn.microsoft.com/webservices/indigo/default.aspx