

Konzepte und Standards zur domänenübergreifenden Integration von komplexen Webanwendungen

Markus Tacker · 24. Januar 2012



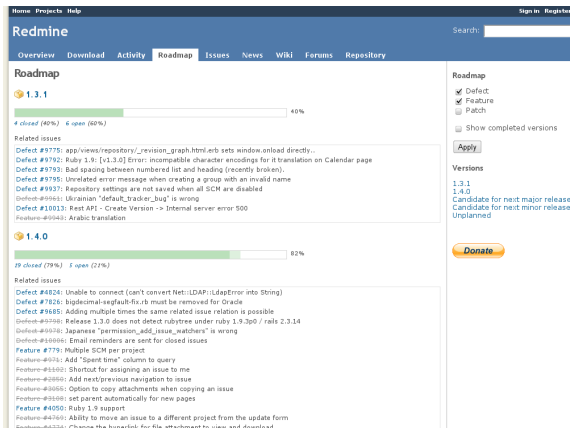
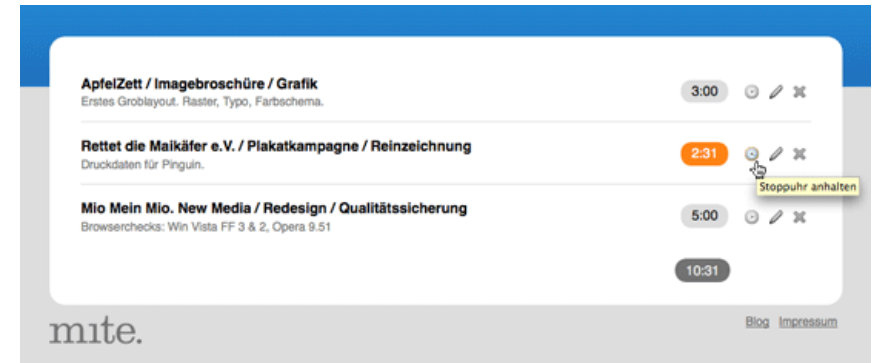
Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Fragestellung

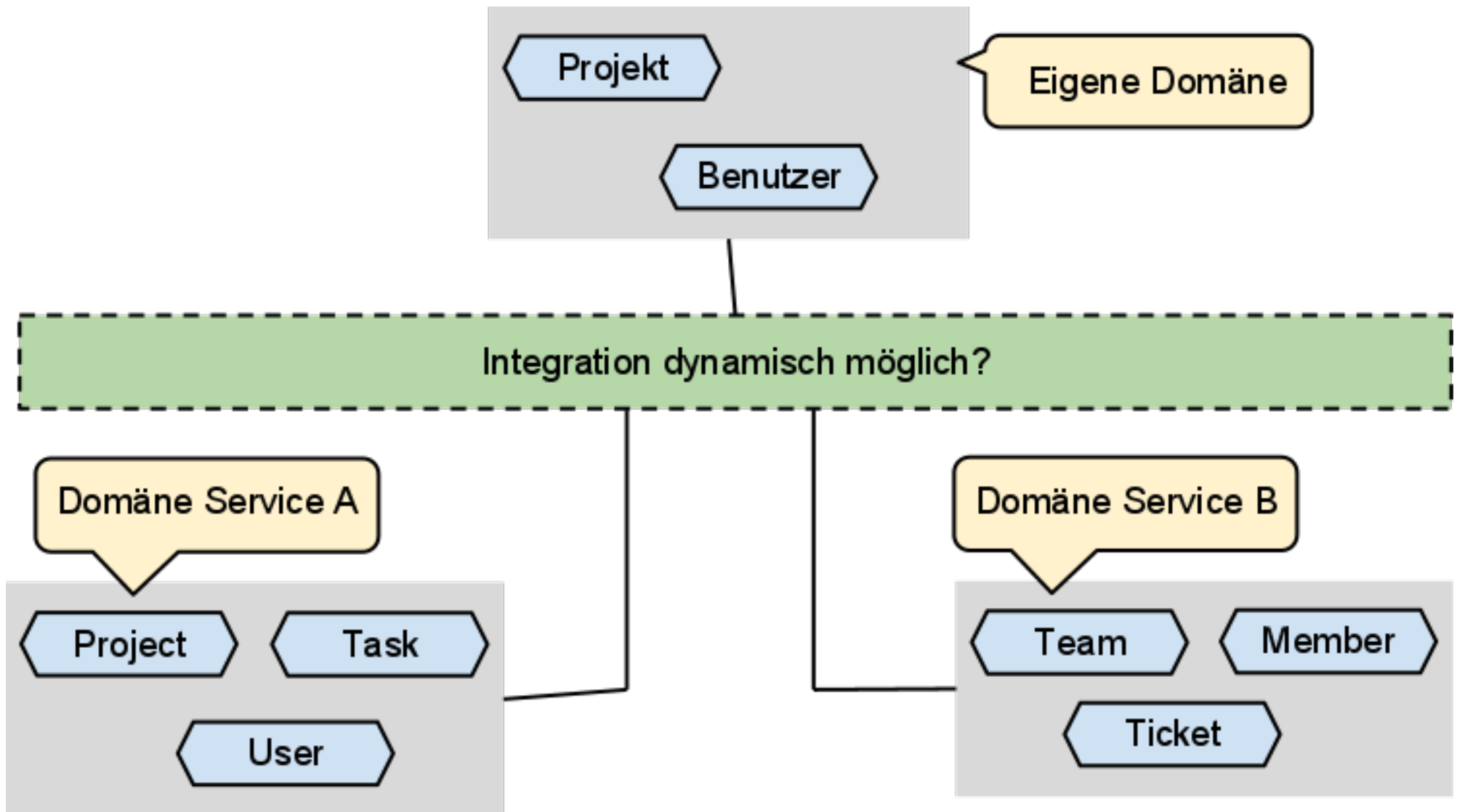
Wie kann man *komplexe Webanwendungen* so miteinander verbinden, dass diese Verbindung *nicht fest definiert* ist und damit *dynamisch austauschbar* ist?

Fragestellung

Beispiel: Integration einer Zeiterfassung in ein Projektverwaltungstool.



Fragestellung



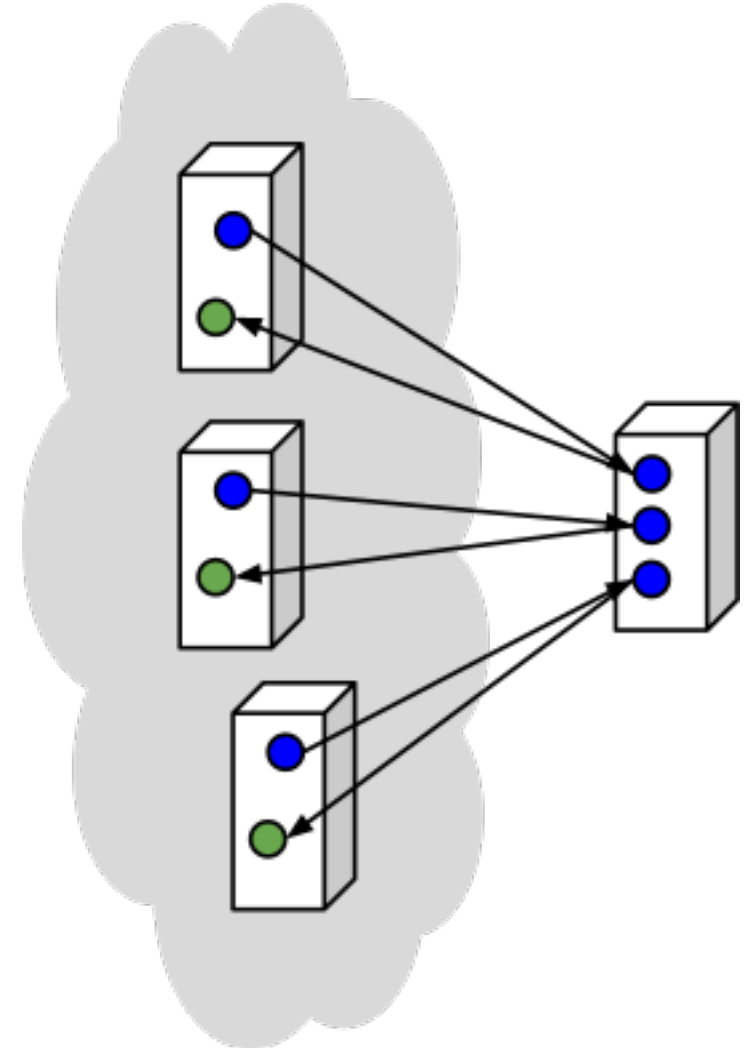
Inhalt

- Was sind komplexe Webservices?
- Dynamische Bindung / Lose Kopplung
- Semantische Beschreibung von Webservices
- Architektur mit lose gekoppelten Webservices

Was sind *unkomplexe* Webservices?

Beispiel: Webservice für Wetterdaten

- Anfrage:
GET /ig/api?weather=D-65195
- Antwort:
<temp_c data="9"/>



Was sind *unkomplexe* Webservices?

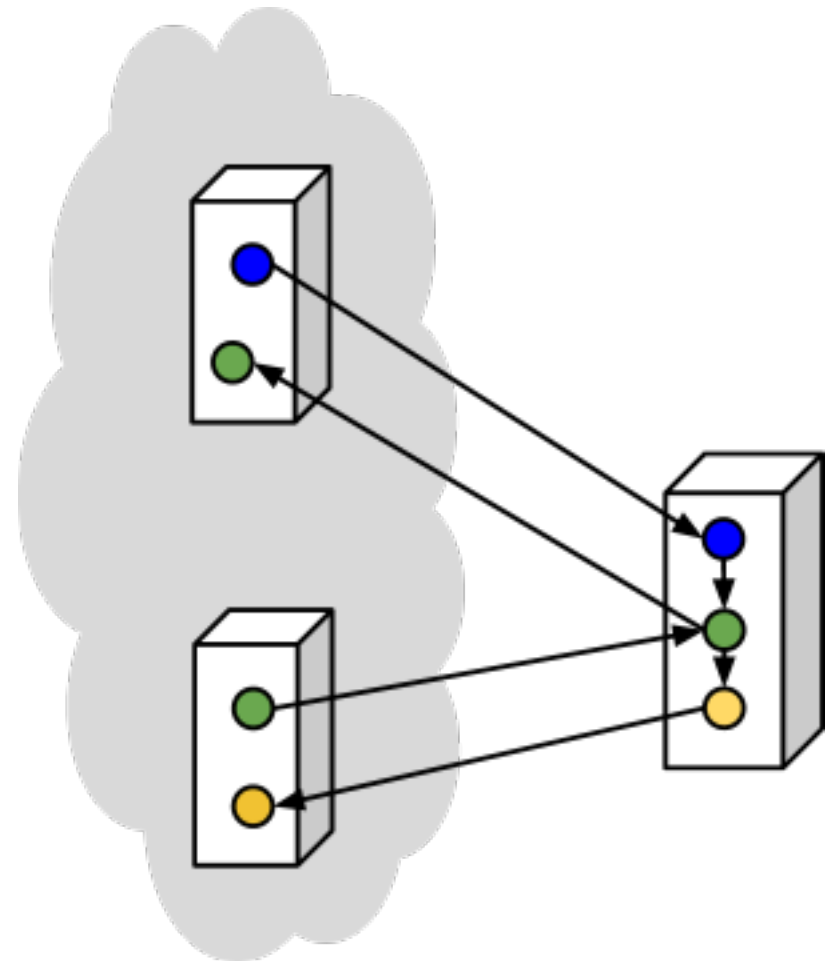
Herkömmlicher Webservices sind „Black-Boxes“

- sie sind **zustandslos**
- jede Anfrage wird ohne Berücksichtigung einer vorhergegangenen Anfrage verarbeitet
- die verarbeiteten Daten werden atomar betrachtet

Was sind *komplexe* Webservices?

Beispiel: ebay

- Anfrage 1:
POST /item/123456790/bid
amount=**10**
- Antwort:
OK: bid_ok
- Anfrage 2:
POST /item/123456790/bid
amount=**10**
- Antwort:
ERROR: bid_to_low



Was sind *komplexe* Webservices?

Komplexe Webservices sind „Smart-Boxes“

- sie sind **zustandsbehaftet**
- bilden Arbeitsabläufe ab
- Anfragen verändern Zustand auf Seite des Service-Anbieter
- nachfolgende Anfragen werden davon beeinflusst

Anbindung von Webservices

Anbindung an eigene Systeme erfolgt *in der Regel statisch*:
„Nimm den XML-Wetter-Service von Google um die aktuelle Temperatur an einem Ort zu erhalten“

1. Suche Schnittstellenbeschreibung (z.B. WSDL)

2. Schreibe (oder generiere) Einbindung:

```
def getTemperature(location):  
    url = "http://www.google.com/ig/api?weather="  
    resp = urlopen(url + location)  
    tree = etree.fromstring(resp.read())  
    temp_node = tree.find('weather/current_conditions/temp_c')  
    return temp_node.attrib['data']  
print(getTemperature("D-65195"))
```

Anbindung von Webservices

Nachteile der statischen Bindung:

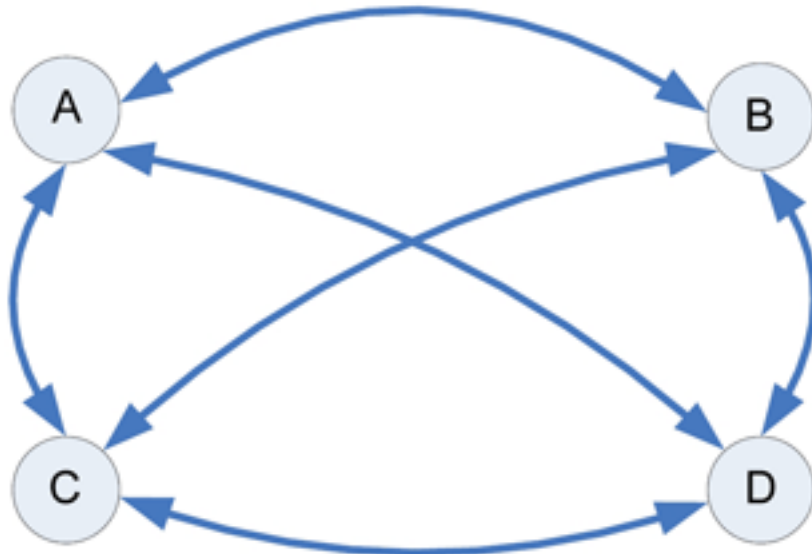
- der verwendete Dienst ist alternativlos
- seine Verwendung wird allen anderen Teilen und Benutzern des Systems aufgezwungen
- Austausch nicht ohne Aufwand möglich
- System ist vom verwendeten Dienst abhängig

Die ideale Anbindung von Webservices

Dienste sollten dynamisch gebunden werden um eine **lose Kopplung** mit dem Dienst zu erreichen.

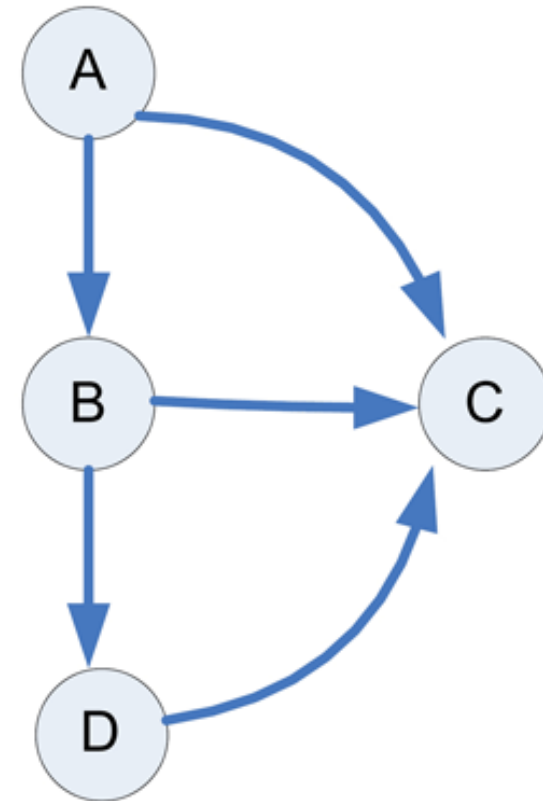
Lose Kopplung: Grundlagen

Strong Coupling



Wenn A geändert wird, sind B, C und D betroffen.

Loose Coupling



Wenn A geändert wird, sind keine anderen Komponenten betroffen.

Lose Kopplung: Grundlagen

Eigenschaften einer lose gekoppelten Architektur:

- besteht aus einzelnen, abgeschlossenen Komponenten
- ist einfach anpassbar und erweiterbar
- Komponenten können in der jeweils besten Umgebung betrieben werden
- Fehler in einer Komponente betrifft nicht zwangsläufig das gesamte System

Lose Kopplung von Webservices

Ziel: passende Webservices zu einer Aufgabe zur Laufzeit finden und verwenden.

„Verwende irgend einen Service, der mir zu einer Ortsangabe die aktuelle Umgebungstemperatur liefert.“

Lose Kopplung von Webservices

Beschreibung von Webservices mit WSDL ist nicht ausreichend:

- Beschreibt nur das „Wie“
 - *„Wie muss ich meine Anfrage formulieren, um eine Antwort zu erhalten?“*
- nicht das „Was“
 - *„Was für Daten verarbeitet der Webservice: Orstangaben, Personen, ...?“*

Den passenden Dienst anhand der WSDL auszuwählen ist nicht möglich. Es fehlt die **semantische Beschreibung** des Dienstes.

Missing Link: Semantik

Semantik

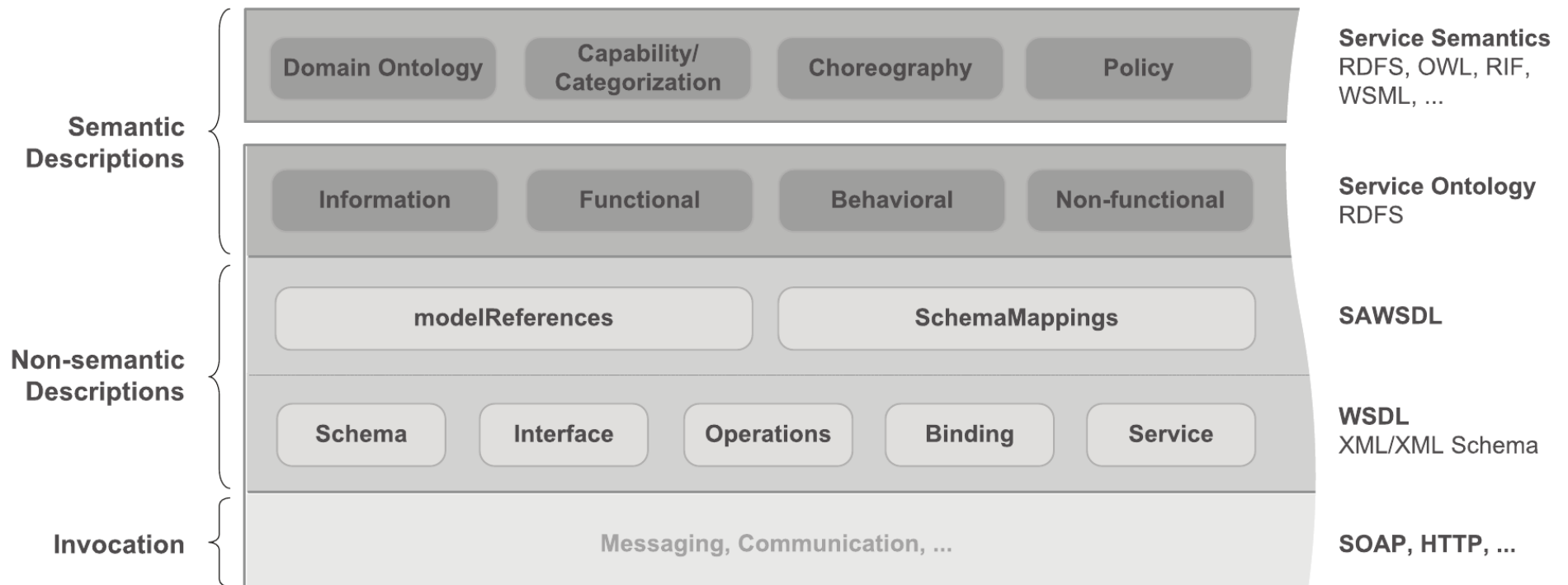
- beschreibt das Wesen von Dingen und ermöglicht die Interpretation und Übertragung von Konzepten auf konkrete Begebenheiten.
- wird in der Informatik durch **Ontologien** beschrieben.

Ontologien

- sind maschinenlesbar
- beschreiben aus der Sicht des Dienstanbieters die Zusammenhänge in dessen „Welt“

SAWSDL

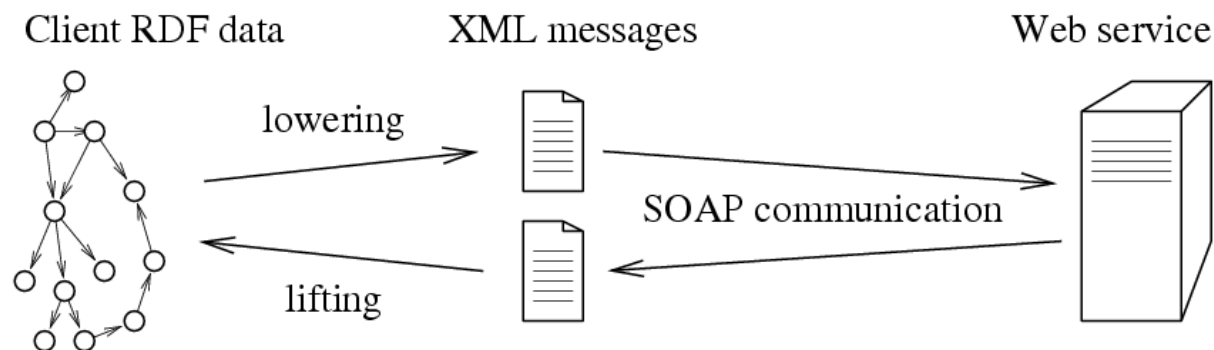
W3C-Empfehlung für einen Standard zur semantischen Beschreibung von Webservices



SAWSDL

SAWSDL definiert drei neue Attribute in WSDL

- **modelReference**: Komponenten in der WSDL können einem Objekt im semantischen Modell zugeordnet werden
- **liftingSchemaMapping** und **loweringSchemaMapping**: gibt an, wie Nachrichten-Daten (z.B. XML) in semantische Daten (z.B. RDF) übertragen werden können und umgekehrt



Quelle: [http://www.w3.org/2007/Talks/www2007-sawSDL/2007-05-sawSDL.html#\(1\)](http://www.w3.org/2007/Talks/www2007-sawSDL/2007-05-sawSDL.html#(1))

Semantische Webservices verwenden

Mit SAWSDL sind Webservices nur so beschrieben, dass sie dynamisch verwendet werden können.

Problem: Die Beschreibung

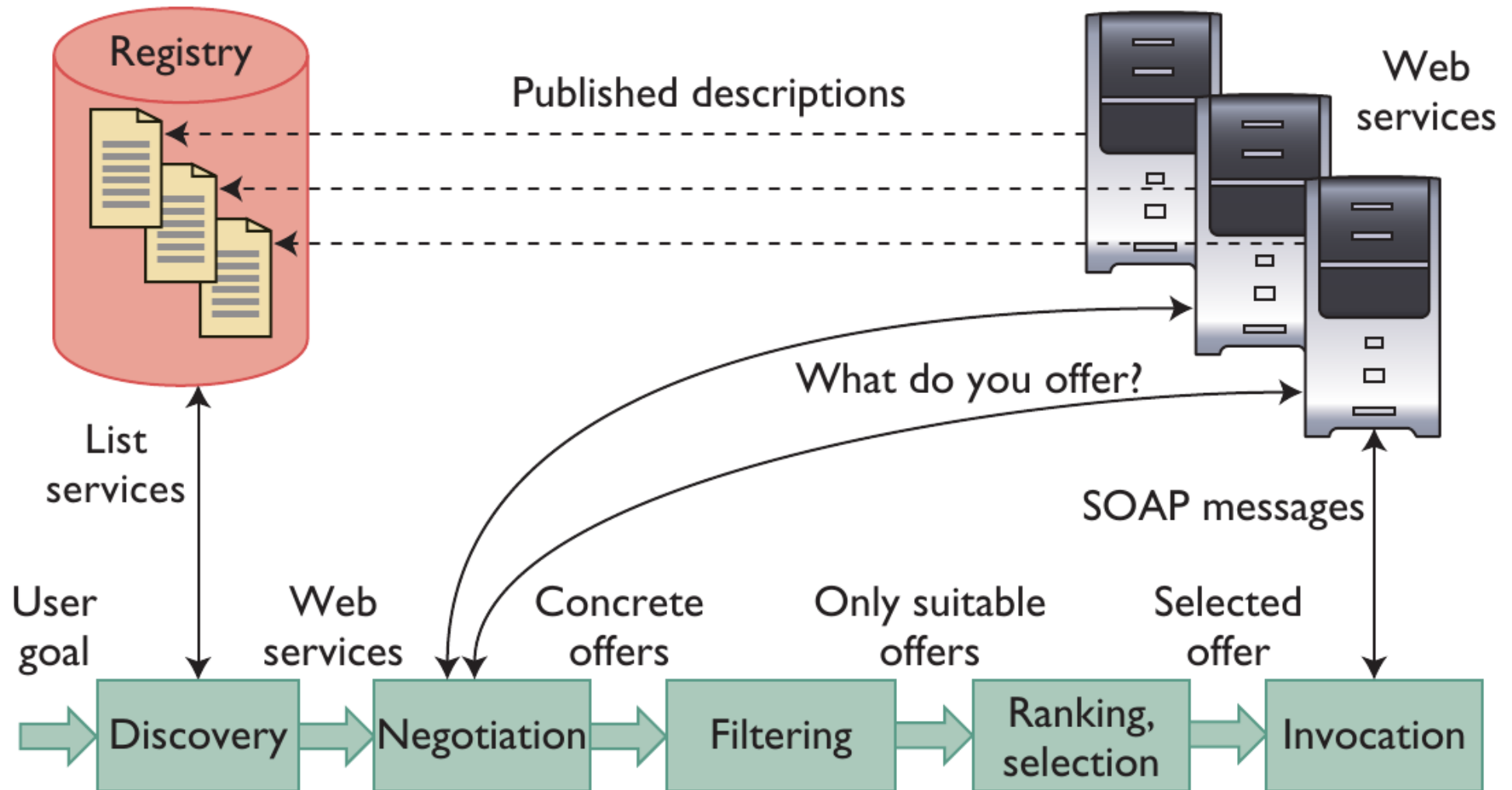
- muss durch den Dienstbetreiber geliefert werden
- an Domäne der Architektur des Verwenders angepasst sein
- aber: globale Ansätze existieren
<http://semanticweb.org/wiki/Ontology>

Semantische Webservices verwenden

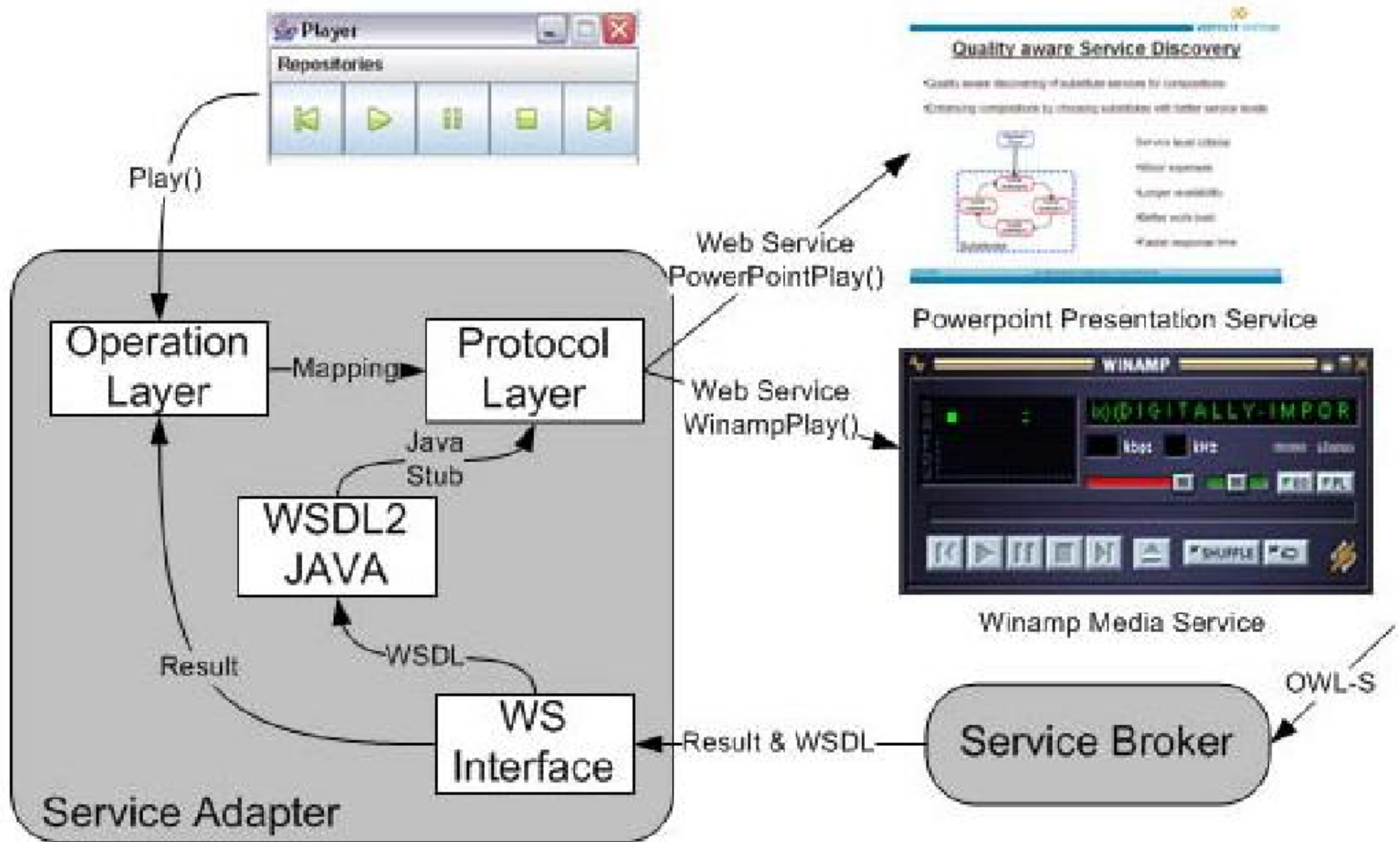
Eine mögliche Architektur mit lose gekoppelten Services muss folgende Aufgaben abbilden:

- *publication*: Die Beschreibungen der Dienst müssen veröffentlicht werden
- *discovery*: Die Dienste müssen gefunden werden
- *composition*: Die gefunden Dienste müssen passend zur Anfrage zusammengestellt werden
- *invocation*: Die zusammengestellten Dienste müssen aufgerufen werden

Semantische Webservices verwenden



Beispiel einer semantischen Architektur



Zusammenfassung

- Die dynamische Bindung von Webservices ist möglich.
- Voraussetzung dafür ist die semantische Beschreibung mit Hilfe von Ontologien, z.B. durch SAWSDL.
- Architektur dazu ist komplex aber realisierbar.

Literatur

- Artikelserie in Java-Spektrum 2004 zu semantischen Webservices

<http://bit.ly/AtRQWS>

- SAWSDL: Semantic Annotations for WSDL and XML Schema, IEEE, 2007

<http://bit.ly/yxP8sl>

- Flexible automatic service brokering for SOAS, IEEE, 2007

<http://bit.ly/Amx2fQ>

