Spezifizieren und Suchen von Services

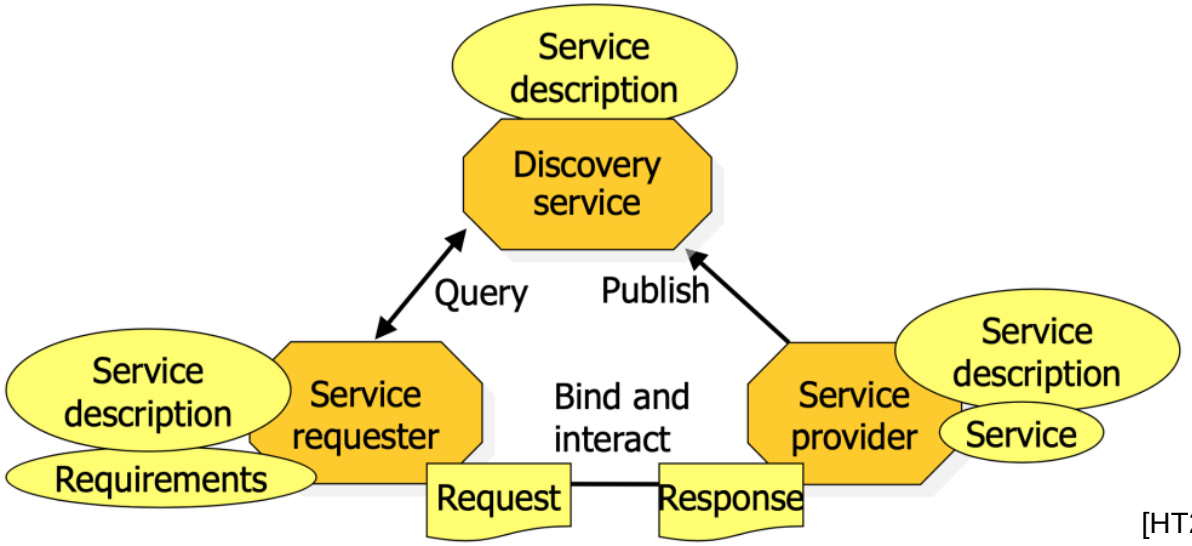
Jens Kosiol

27. Mai und 3. Juni 2024

Überblick

- Web-Services sind Softwarekomponenten, die über das Internet angeboten und gesucht werden.
- Während die Service-Syntax leicht spezifiziert (und gesucht) werden kann, ist die Semantik von Services meist nur informell beschrieben.
- Wie kann die Semantik von angebotenen und gesuchten Services so spezifiziert werden, dass sie leicht verständlich ist und zu Service-Gesuchen passende Services gefunden werden?

Serviceorientierte Softwarearchitektur



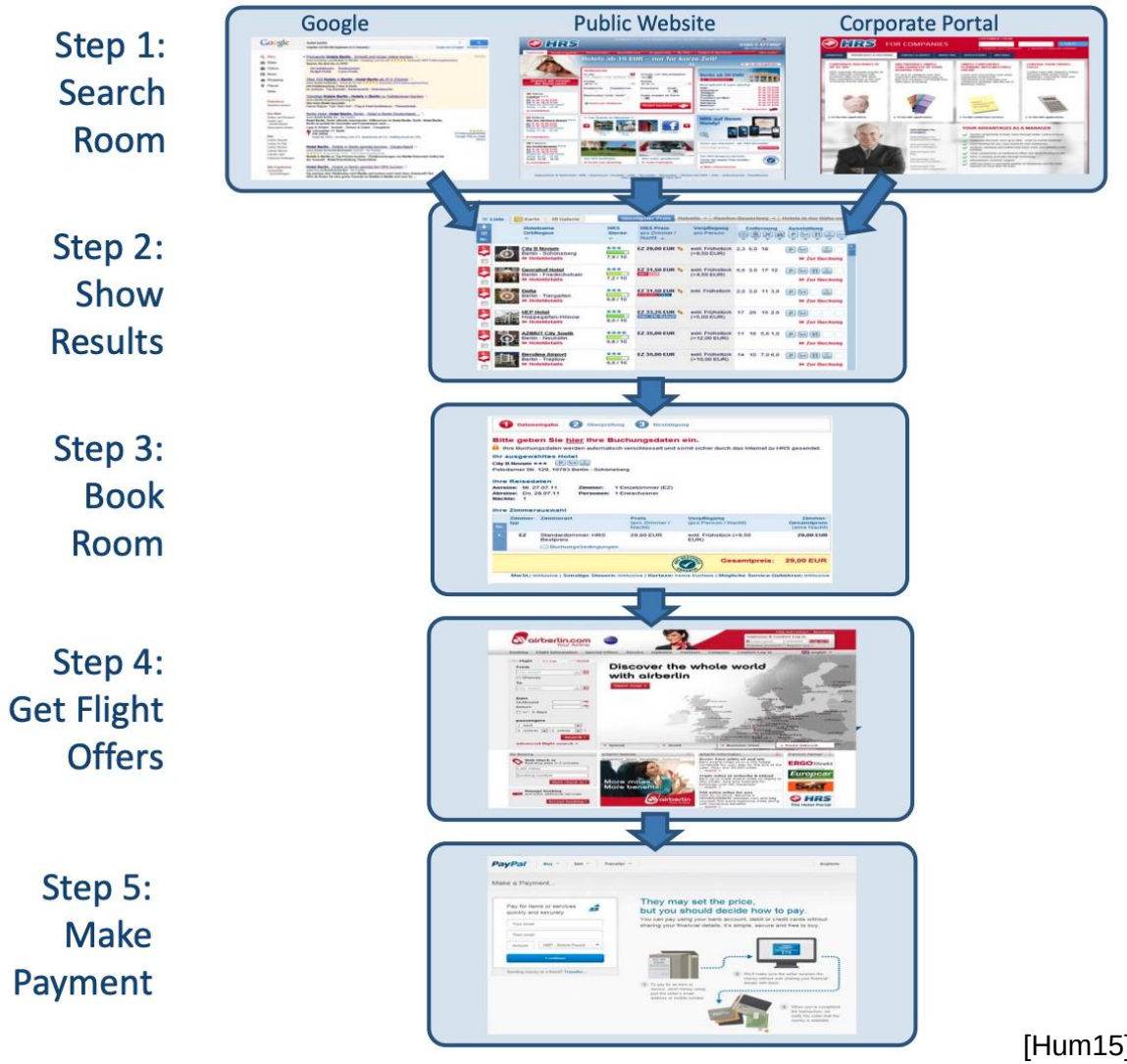
[HT20]

- Anwendungsentwickler fragt Services an (Service Requester)
- Service-Entwicklerin bietet Services an (Service Provider)
- Service-Vermittlung durch Discovery Service

Beschreibung von Services

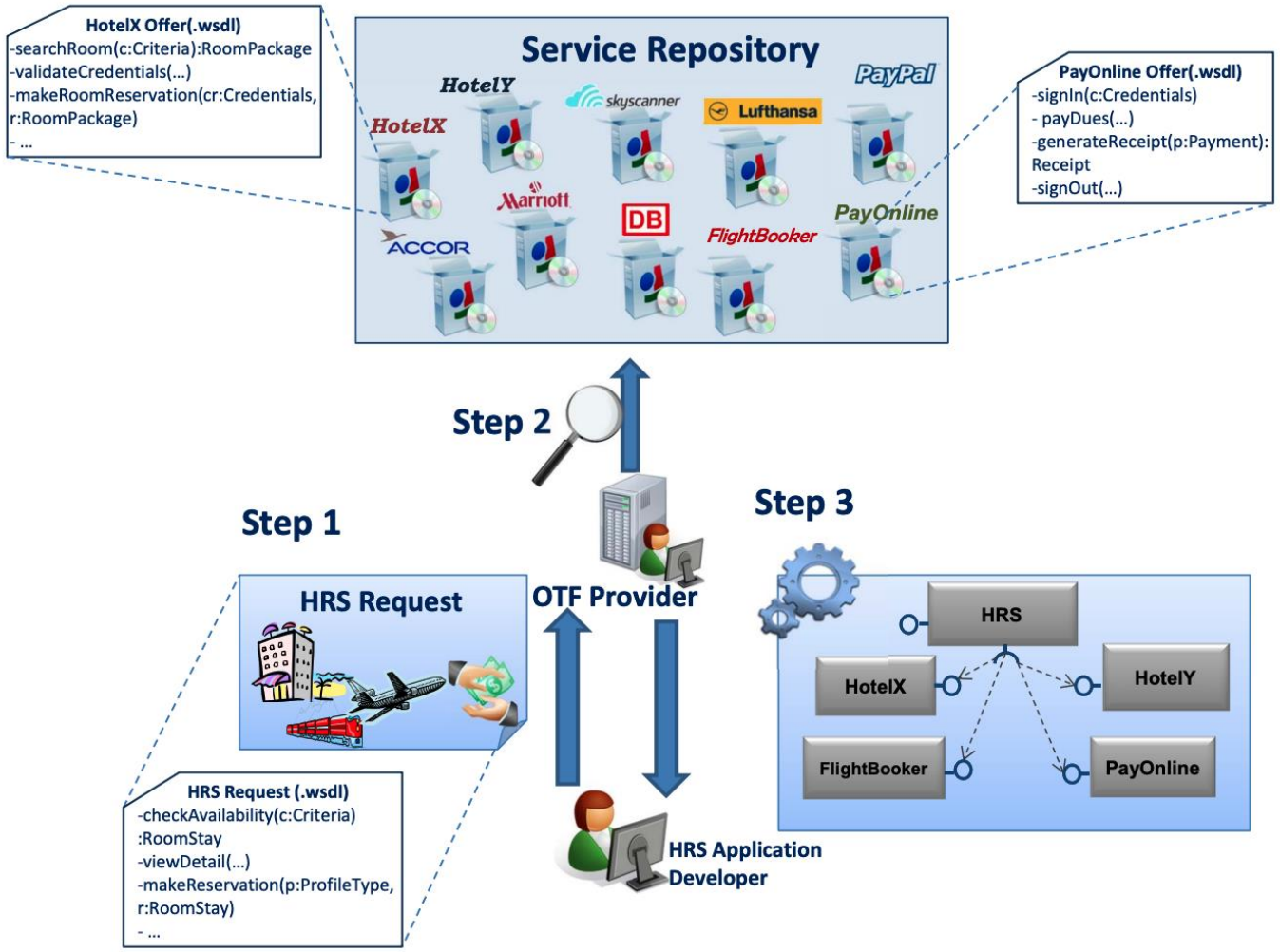
- Syntaktisch:
  - *Service-Signatur: Name und Parameterliste*
- Semantisch:
  - *Informelle Beschreibung (natürlichsprachlicher Text)*
  - *Durch fachspezifische Ontologien*
  - *Durch Vor- und Nachbedingungen (Design-by-Contract)*

Beispiel: Hotelreservierung



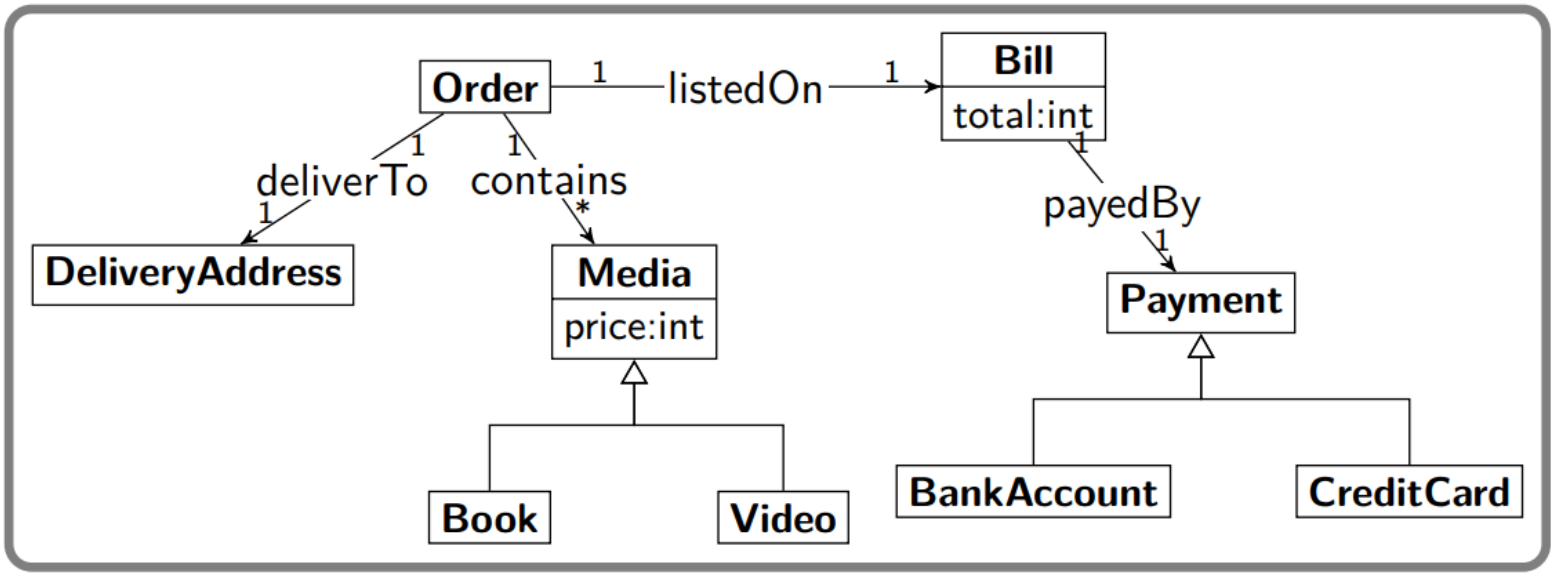
[Hum15]

Beispiel: Auffinden von Services



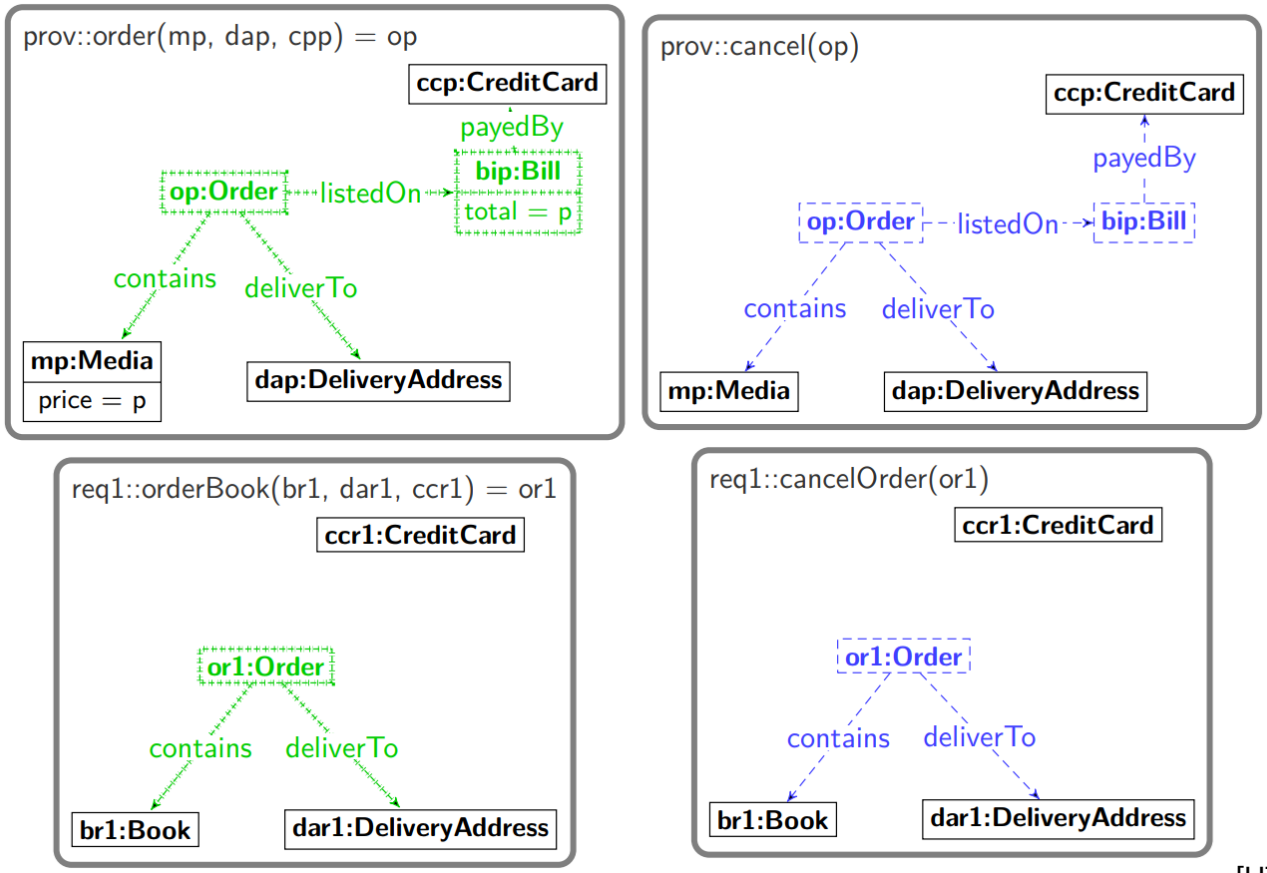
[Hum15]

Beispiel: Einfache Ontologie für den Verkauf von Büchern und Videos



[HT20]

Beispiel: Service-Spezifikation und Suche



[HT20]

- Media Provider: Service-Angebot für Medienkauf
- Media Requester: Service-Suche für Medienkauf

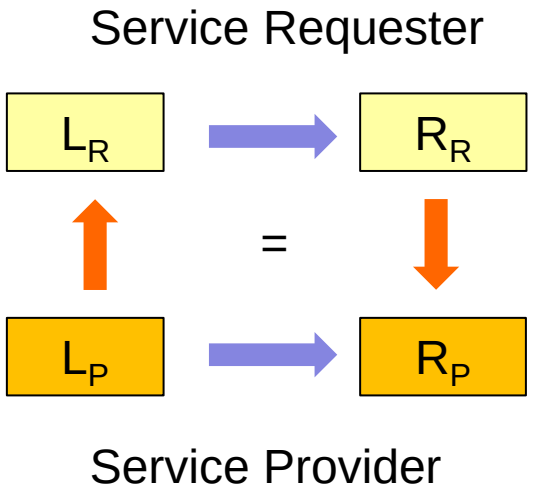
Services als Visual Contracts

- Graphtransformationsregeln $L \Rightarrow R$ (Visual Contracts) werden benutzt, um die gewünschte bzw. die angebotene Funktionalität zu beschreiben
- Service-Provider:
  - L_P: *Vorbedingung für den eigenen internen Zustand und die Daten des Requesters*
  - R_P: *Nachbedingung, falls die Ausführung erfolgreich war*
- Service-Requester:
  - L_R: *Annahme über den Provider-Zustand und Daten, die der Requester für die Benutzung des Services bereitstellen möchte*
  - R_R: *Erwartung über den resultierenden Zustand*
- Intuition: Der Contract kann geschlossen werden wenn:
  - *Der Provider fordert nicht mehr als der Requester anbietet.*
  - *Der Provider führt die Aktionen aus, die der Requester ausführt.*

Service Matching

Service-Matching (Überblick):

- L_P lässt sich auf L_R so abbilden, dass
  - *alle zu erhaltenden Elemente aus L_P auf zu erhaltende Elemente aus L_R abgebildet werden und*
  - *alle zu löschenden Elemente aus L_R im Bild sind.*
  - *Abstrakte Typen können auf konkrete abgebildet werden.*
- $R_R \setminus L_R$ lässt sich auf $R_P \setminus L_P$ abbilden.



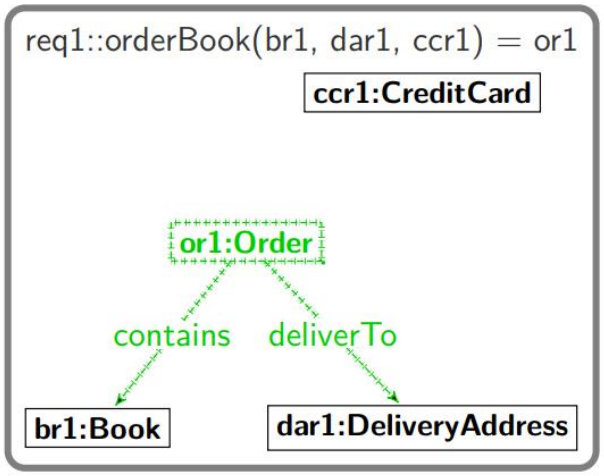
- Zu erhaltende Elemente werden so abgebildet, dass die Service-Abbildungen kommutieren.

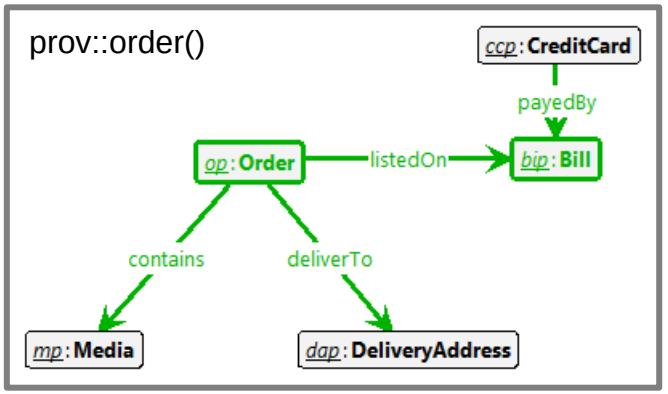
Definition: Service Matching

Gegeben zwei Visual Contracts $L_P \Rightarrow R_P$ (Provider) und $L_R \Rightarrow R_R$ (Requester), dann besteht ein Service Match zwischen diesen, wenn:

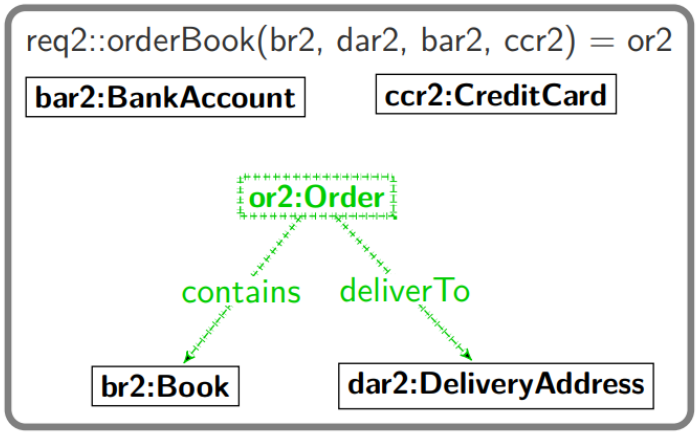
1. Ein injektiver Graphmorphismus $f: L_P \to L_R$ existiert (f darf heruntertypen), s.d.
   - $f(L_P \cap R_P) \subseteq L_R \cap R_R$
   - $f(L_P \setminus R_P) = L_R \setminus R_R$
   - $f(L_P) \cup R_R \setminus L_R$ *ist ein Graph*
2. Eine strukturkonforme injektive Abbildung $g: f(L_P \cap R_P) \cup R_R \setminus L_R \to R_P$ existiert (g darf sogar "herauftypen"), s.d.
   - $g(R_R \setminus L_R) \subseteq R_P \setminus L_P$
3. Für jedes $x \in L_P \cap R_P$ gilt: $g\big(f(x)\big) = x$

Beispiel: Service-Matching





[HT20]



Matching req1::orderBook() → prov::order():

Abbildung f:

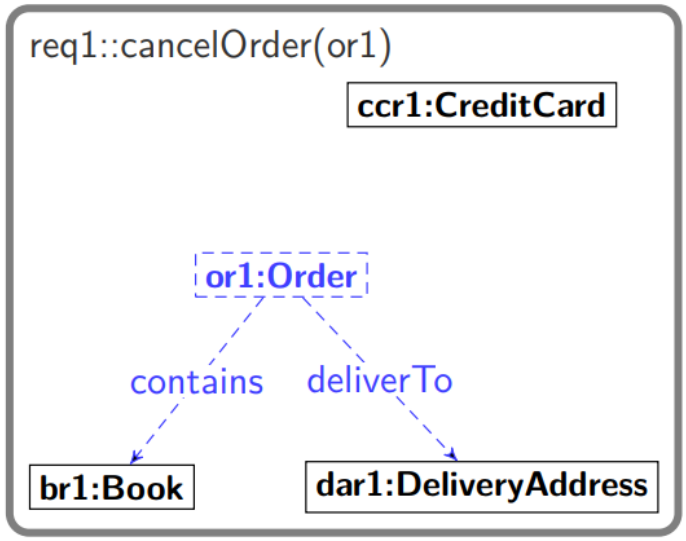
- ccp ↦ ccr1
- mp ↦ br1
- dap ↦ dar1

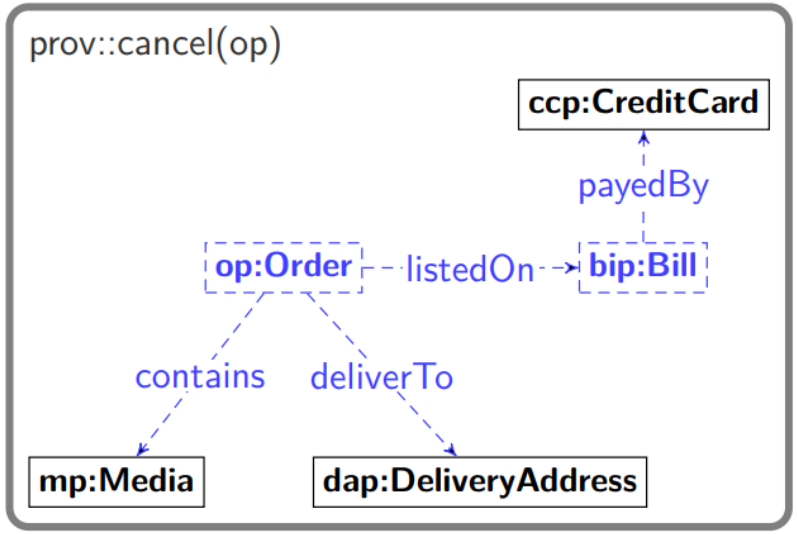
Und Abbildung g:

- or1 ↦ op (und angrenzende Kanten entsprechend)

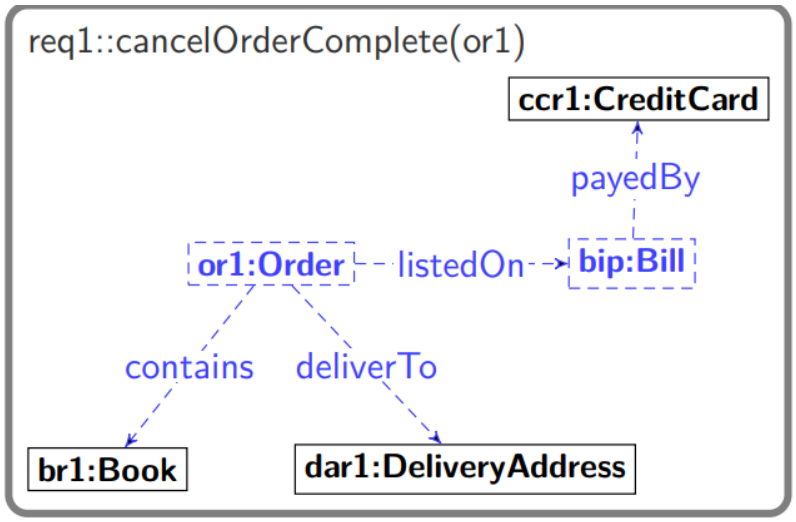
Lässt sich req2::orderBook() auch auf order() matchen?

Beispiel: Kein Service-Matching





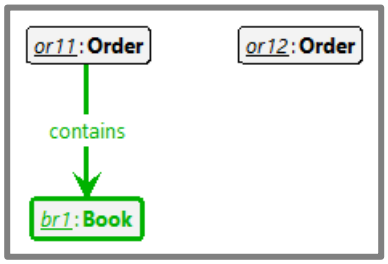
Kein Matching von cancelOrder() auf cancel(), da $L_P \subseteq L_R$ nicht gilt.



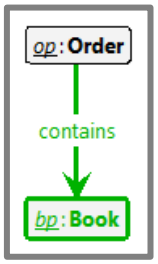
[HT20]

Beispiel: Service-Matching

req1::addBookToOrder



prov::addBookToOrder



Für einen Service-Match muss die Abbildung f von der linken Seite von *prov::addBookToOrder* die Order op auf or11 abbilden.

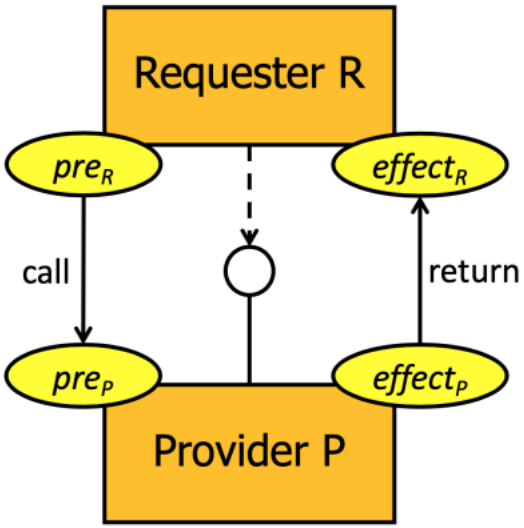
- Nur so bildet $f(L_P) \cup R_R \setminus L_R$ einen Graph.
- Nur so kann die Abbildung g anschließend passend definiert werden.

Intuition: Das Buch muss nicht nur geschaffen werden können, sondern auch an der Position, an der vom Requester gefordert.

Service-Aufruf: Operationale Interpretation des Matchings

Service P wird von R aufgerufen:

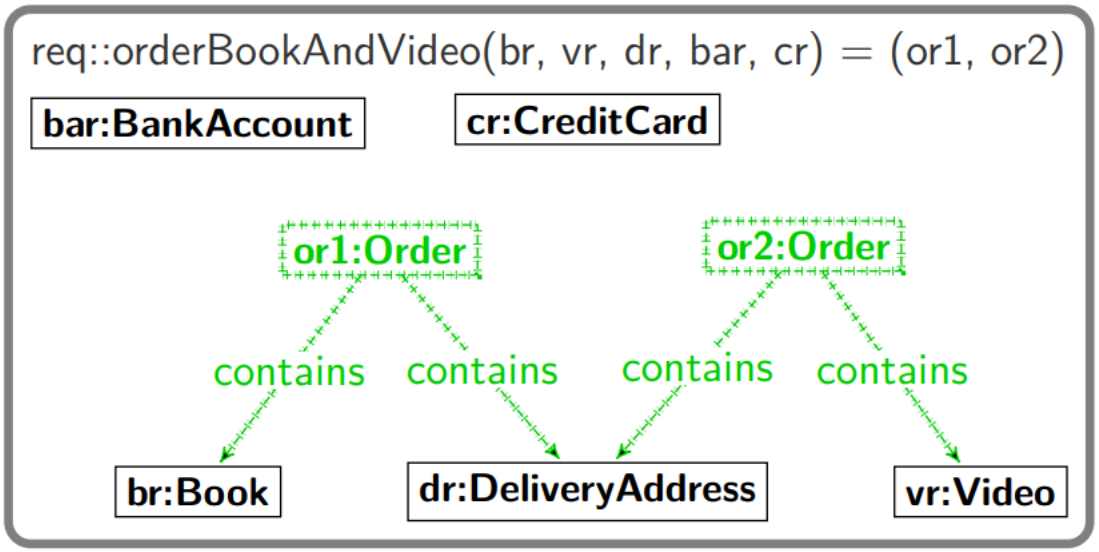
- Vorbedingung pre_R ist erfüllt.
- Vorbedingung pre_P ist erfüllt, da sie von pre_R umfasst wird.
- Ausführung von P ergibt $effect_P$.
- R kann annehmen, dass $effect_R$ erfüllt ist, da er von $effect_P$ umfasst wird.

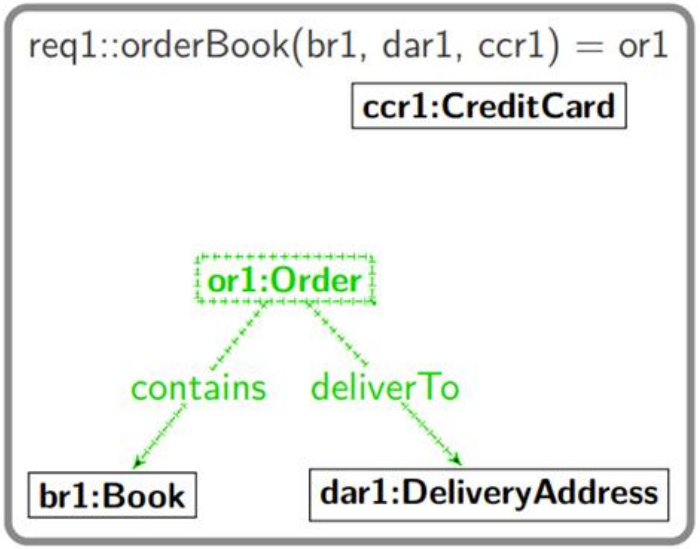


[HT20]

Inkrementelle Servicekomposition

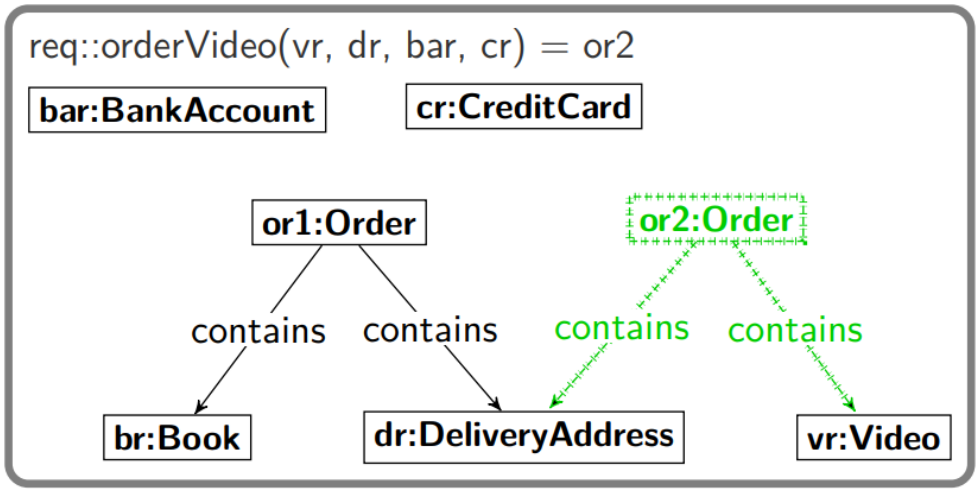
[HT20]





- Wenn eine Service-Suche nicht durch einen Service erfüllt werden kann, werden inkrementell passende Services gesucht.
- Falls eine Suche nur partiell erfüllt werden kann, muss der Requester einen angebotenen Service auswählen. Es bleibt ein unerfüllter Teil der Suche.

Inkrementelle Servicekomposition



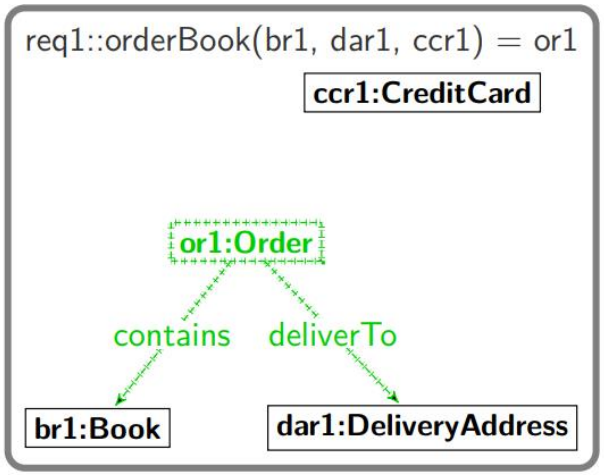
[HT20]

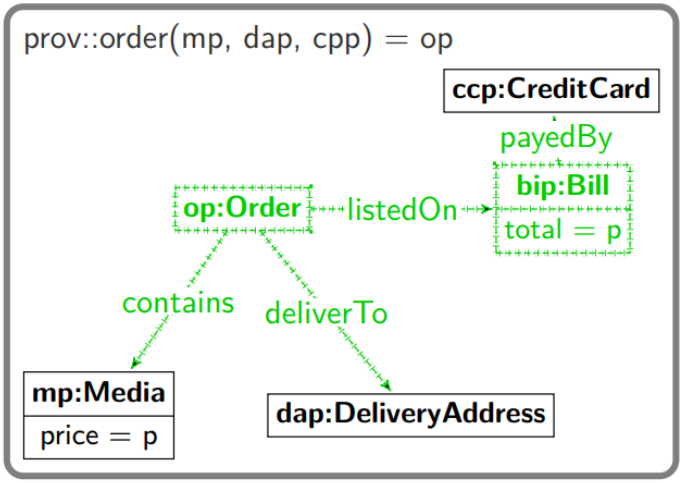
- Die Suche wird auf den unerfüllten Teil aktualisiert. Formal:
  - *Die Abbildung f darf auf Elementen aus $L_P \setminus R_P$ auch undefiniert sein.*
  - *Die Abbildung g darf auf Elementen aus $R_R \setminus L_R$ auch undefiniert sein.*
  - $L'_R = L_R \setminus f(L_P \setminus R_P) \cup g(R_R \setminus L_R), R'_R = R_R \cup g(R_R \setminus L_R)$
- Für das Ergebnis werden weitere erfüllende Services gesucht.
  - *Die Iteration stoppt, wenn für das erreichte Zwischenergebnis ein vollständiges Service Matching erreicht wird.*
- Die inkrementelle Suche ist vollständig, entscheidbar und terminierend.

Erweiterung um Attribute

- Das Verfahren kann ganz analog auf attributierten Graphen eingesetzt werden (da Attributierung durch Kanten geschieht).
- Bei der inkrementellen Servicekomposition mit Attributierung entstehen häufig transiente Elemente:
  - *Die Requester-Regel (attributiert über einer Termalgebra) schreibt eine komplexere Berechnung vor.*
  - *Ein erster partieller Service Match beginnt die Berechnung; das Attribut ist nun mit einem Zwischenergebnis belegt, das in der Ausgangsregel nicht auftaucht.*
  - *Weitere partielle Service Matchs setzen die Berechnung fort und verschieben dabei die Attributierungskanten (Transienz).*
  - *Am Ende ist der durch die Requester-Regel geforderte Zustand erreicht.*

Beispiel: Kein Service-Matching





[HT20]

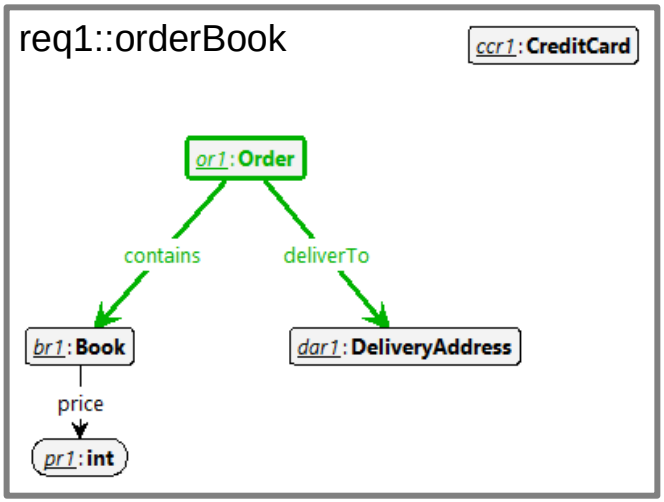
Kein Matching req1::orderBook() → prov::order():

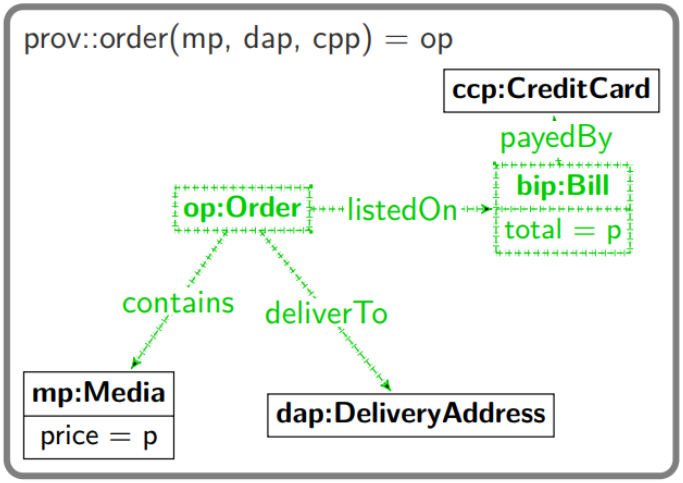
Abbildung f:

- ccp ↦ ccr1
- mp ↦ br1
- dap ↦ dar1

kann nicht vollständig definiert werden, da die Kante „price“ nicht abgebildet werden kann.

Beispiel: Service-Matching





[HT20]

Service Matching req1::orderBook() → prov::order():

Abbildung f:

- ccp ↦ ccr1
- mp ↦ br1
- dap ↦ dar1 und
- price ↦ price und p ↦ pr1

Zusätzliche Erweiterungen

In [HGE15, Hum15] wird das in einem noch allgemeineren Kontext betrachtet:

- Die Betreiber vom Service Repository pflegen globale Ontologie und bieten semi-automatische Übersetzung von lokalen Ontologien in diese an.
- Service Requester erfragt Visual Contracts + Sequenzdiagram (Aufrufreihenfolge)
- Service Provider bietet Visual Contracts + Statechart (erlaubte Aufrufreihenfolgen)
- (n,m)-Matching zwischen angefragten und angebotenen Services, das die erforderten und angebotenen Aufrufreihenfolgen berücksichtigt

Zusammenfassung

- Visual Contracts können Vor- und Nachbedingungen von Services spezifizieren. Sie können zur Spezifikation von vorhandenen Services und zur Service-Suche eingesetzt werden.
  - *Visual Contracts basieren auf Ontologien.*
- Das Service-Matching kann eine Service-Suche (semi-)automatisch auf einen oder mehrere angebotene Services abbilden.
- Das vorgestellte Konzept lässt sich auf Micro-Services, Bibliotheken, Plugins, Apps, etc. übertragen.

Literatur

- [Bal00] H. Balzert: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Aufl., Spektrum Akademischer Verlag 2000
- [HGE15] Z. Huma, C. Gerth, G. Engels: On-The-Fly computing: automatic service discovery and composition in heterogeneous domains, in: Comput. Sci. Res. Dev. 30(3-4): 333–361 (2015)
- [HT20] Reiko Heckel, Gabriele Taentzer: Graph Transformation for Software Engineers, Springer, 2020 (Kapitel 6)
- [Hum15] Zille Huma: Automatic Service Discovery and Composition for Heterogenous Service Partners, Doktorarbeit, Universität Paderborn, 2015