Ontologie-basierte Web-Service-Komposition

Dzung Phan
Betreuerin: Frau Maßmann
Email: dzungphan200677@yahoo.de

• *[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.*

Inhalt

- Motivation-Beispiel
- OWL-S Ontologie
- KI-Planung
- WSK als Plannungsproblem
- Methodologie
- Situation Calculus, Golog und ConGolog
- ConGolog Ansatz von McIlraith, S., and Son
- Middle Ground ConGolog
- Zusammenfassung

Motivation-Beispiel (1)

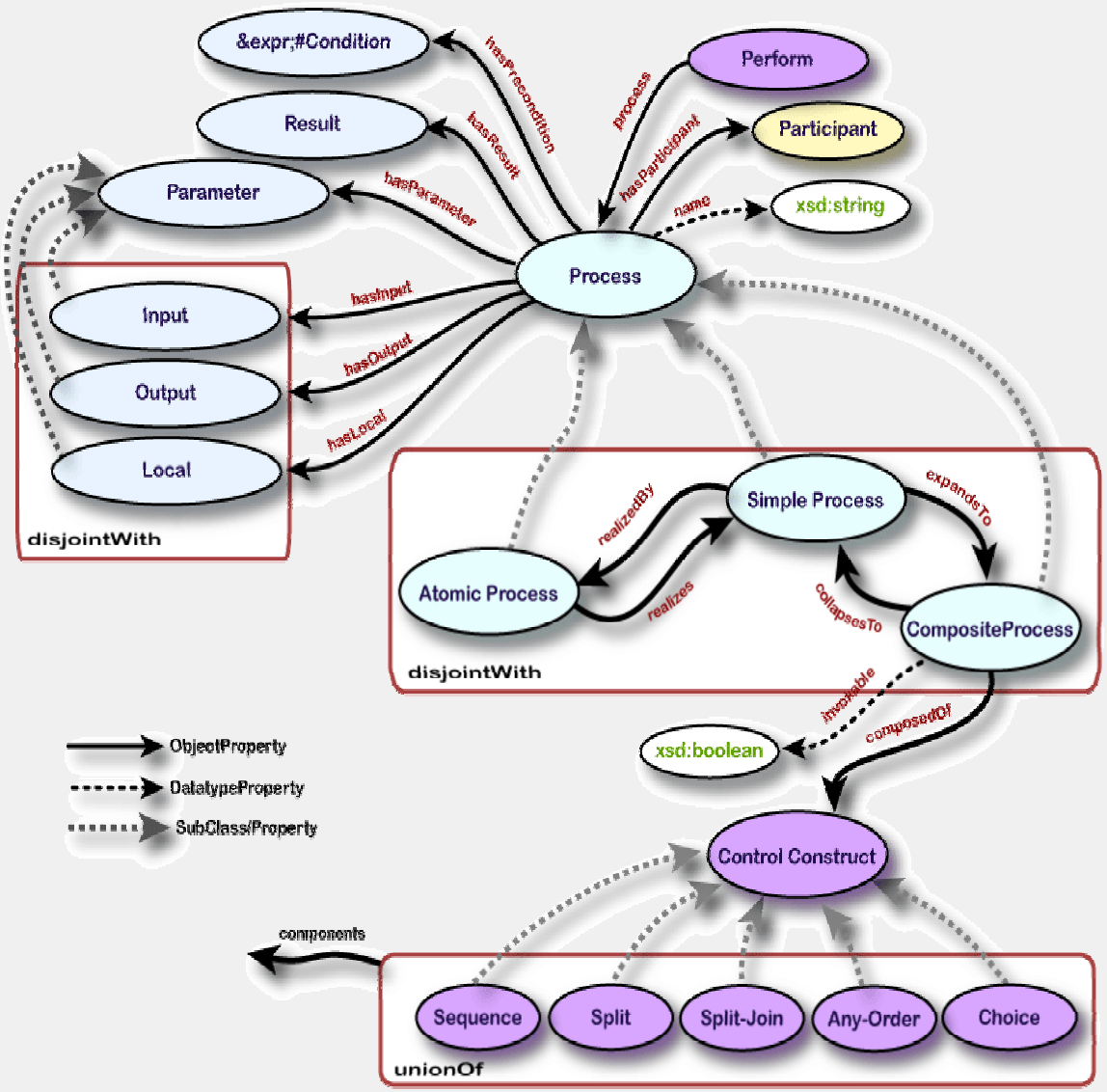
- Eine Studentin plant zu einer Konferenz zu fahren.
- Sie will mit dem Auto fahren, wenn die Fahrzeit nicht zu lang ist (z.B nicht mehr als 3 St.)
- Sonst wählt sie ein anderes Verkehrsmittel (z.B.: Flugzeug).
- Sie will ein Auto vor Ort für die Zeit der Konferenz mieten.
- Sie braucht eine Unterkunft
- Für Zahlung wählt sie die Bank-Überweisung
- Schließlich möchte sie alle Informationen beim nächsten Drucker ausdrucken.

Motivation-Beispiel (2)

Das Problem

- kein einzelner Web-Service kann alle Anforderungen der Studentin entsprechen
- aber eine Kombination von vorhandenen kann es schaffen.
- Dieses Beispiel erfordert mindestens neun Services: „driving-time checking“, „flight searching“, „flight booking“, „car searching“, „car booking“, „hotel searching“, „hotel booking“, „bank transferring“ und ein „printer“ Service.
- ***Problem***: Finde eine Komposition von neun **in OWL-S beschriebenen Web-Services** mit einer Beschreibung des Ziels um die Aufgabe zu lösen

OWL-S (1)



- OWL-S wird verwendet, um Web-Services zu beschreiben.
- In OWL-S werden Web-Services als Prozesse modelliert.
- Ein Prozess kann haben:
  - Vorbedingungen
  - Inputs (d.h., Wissen-Vorbedingungen)
  - Outputs (d.h., Wissen-Effekte)
  - (bedingte) Effekte
- Es gibt drei Arten von Prozessen in OWL-S:
  - Atomic-Prozesse
  - Composite-Prozesse
  - Simple-Prozesse.

Quelle: http://www.w3.org/Submission/OWL-S/

OWL-S (2) - Beispiel

Ein Verkauf-Service kann verlangen:

- eine gültige Kreditkarte als Vorbedingung
- Kreditkartennummer und Ablaufdatum als Input.

  - Die Wissen-Vorbedingung des Services ist, dass Software-Agent (z.B.: planning engine) die Kreditkartennummer und das Ablaufdatum „wissen“ muss.

- Verkauf-Service erzeugt eine Quittung als Output.

  - Und der Wissen-Effekt des Services ist, dass der Software-Agent die Quittung weiß

- Der Effekt ist, dass die Kreditkarte belastet wird

  - Der Effekt ist bedingt, wenn das Effekt-Auftreten von irgendeiner Bedingung abhängt

KI-Planung

- KI Planung ist eine Technik der künstlichen Intelligenz
- Planung ist eine Schlüsselfähigkeit für intelligente Systeme
  - erhöht ihre Autonomie und Flexibilität durch den Aufbau von Reihenfolgen von Aktionen, ihre Ziele zu erzielen
- Planungstechniken sind in einer Vielzahl der folgenden Aufgaben angewendet worden
  - Robotics
  - process planning
  - web-based information gathering
  - autonomous agents
  - spacecraft mission control.
- Planung umfasst
  - die Darstellung von Aktionen und Weltmodelle
  - Inferieren (reasoning) über die Effekte von Aktionen
  - Techniken für die leistungsfähige Suche des Raumes der möglichen Pläne

WSK als Plannungsproblem

- Web-Services als einfache oder komplizierte Aktionen mit (Wissen-) Vorbedingungen und (Wissen-) Effekte
- Einsatz von Semantik ermöglicht die Inferenz über Vorbedingungen und Effekte
- WSK als die Planung- und Durchführungsaufgabe mit den folgenden Merkmale von Web Services und vom WSK-Problem
  - Offene Welt Plannung
    - mehrere Information-Sammlung-Services werden bei der Planung verlangt
  - Web-Services haben Wissen-Vorbedingungen (Input) und Wissen-Effekte (Output)
  - Outputs können Inputs eines nachfolgenden Web-Service sein
  - Viele Web-Services haben ähnlichen (Wissen-) Effekte
  - Kompositionen (d.h. Pläne) von Web-Services umfassen nicht nur atomare Aktionen (Services), sondern auch komplexe mit den Programmierkonstrukten (z.B.: If … Then … Else; While … Loop).
  - Pläne sind oft kurz daher Plan-Suche-Raum ist kurz und breit.

METHODOLOGIE (1)

- [1] benutzt Template-Plan um den wünschenswerten Plan aufzubauen.
- ***Template plans are reusable, high-level generic procedures***
- Vorteil:

  - keine Schwierigkeiten für das Aufbauen des Template-Planes

  - beschleunigt Suchgeschwindigkeit

  - wiederverwendbar
- [1] hat ein ConGolog Interpret unter Verwendung von **Situation Calculus** und **ConGolog** zur Unterstützung der Information-Sammlung-Services verstärkt
- [1] kombiniert Online-Ausführung von Informationen-Sammlung-Web-Services mit Offline-Simulation von Welt-Änderung-Web-Services.

[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.

Template Plan für das Motivation-Beispiel

```
proc(travel(Person,Ddate,Rdate,Origin,Destination,TripType),

   getDrivingTime(Origin,Destination) : ?(drivingTime(Origin,Destination,Dr)) : ?
(preferDrivingTime(PDr)) :

   if(lessThan(Dr,PDr),

     car(Person,Ddate,Rdate,Origin,Origin,TripType),

      ticket(Person,Ddate,Rdate,Origin,Destination,TripType) :

      car(Person,Ddate,Rdate,Destination,Destination,TripType)

   ) : hotel(Person,Ddate,Rdate,Destination,TripType) :

   getNearestPrinter(Name, Printer) : printTo(Person, Printer, Rinfo)

   ).
```

Situation Calculus (1)

- Situation Calculus ist eine logische Sprache für das Spezifizieren und die Implementierung der dynamischen Systeme
- Zustand der Welt wird durch Funktionen und Relationen (*fluents*) entsprechend einer Situation s beschrieben. Z.B.: F(x,s)
  - Eine Situation s ist eine Geschichte der primitiven Aktionen, die bei einer anfänglichen Situation S0
- Die Funktion do(a,s) bildet eine Situation und eine Aktion in eine neue Situation s´ ab.
  - s´ = do(a,s)
- eine Aktionentheorie mit Information-Sammlung-Aktionen enthält die folgende Menge von Axiomen
  - action precondition axioms definieren Poss(a,s)
  - successor state axioms für functional und relational fluents,
- Beispiel:
  - 3 Aktionen: pickup(x), putdown(x) und drop(x)
  - 3 fluents: holding(x), broken(x) und hot(x)

Situation Calculus (2)

- Precondition axioms:

$$Poss(pick(x), s) \equiv \neg holding(x, s)$$

$$Poss(drop(x), s) \equiv holding(x, s)$$

$$Poss(putdown(x), s) \equiv holding(x, s)$$

- Successor state axioms:

$$holding(x, do(a, s)) \equiv a = pickup(x) \vee holding(x, s) \wedge a \neq putdown(x) \wedge a \neq drop(x)$$

$$broken(x, do(a, s)) \equiv a = drop(x) \vee broken(x, s)$$

$$hot(x, do(a, s)) \equiv hot(x, s)$$

Golog und ConGolog (1)

- ConGolog ist eine erweiterte Version von Golog
- Beide basieren auf Situation Calculus durch die zusätzliche Bereitstellung von logischen Konstrukte für die Versammlung der primitiven „Situation Calculus“-Aktionen zu komplizierter Aktion.
- Golog benutzt „fluent“ Do, um ein Programm δ bei der Verwendung eines Formular $Do(\delta, s, s')$ auszuwerten
- ConGolog hat die Fähigkeit zum Arbeiten mit den gleichzeitigen Aktionen, die durch 2 fluent Trans und Final repräsentiert werden.
- $Do(\delta, s, s')$ wird in ConGolog neu definiert, wie folgt:

$$Do(\delta, s, s') \stackrel{def}{=} \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$$

- Wichtige Bausteine der ConGolog Aktionstheorie:
  - die Definitionen von primitiven und komplizierter Aktionen
  - action precondition axioms
  - successor state axioms

Beispiel für die Bausteine der ConGolog Aktionstheorie

- primitive actions
  - primitive_action(getDrivingTime(_Origin,_Destination)).
- complex actions
  - proc(travel(Person,Ddate,Rdate,Origin,Destination,TripType),
- preconditions for primitive actions
  - poss(getNearestPrinter(_Name, _Printer), _S).
  - poss(printTo(Name, Printer, _Info), S):-

    prove(nearestPrinter(Name, Printer,S)).
- successor state axioms

  drivingTime(Origin, Destination, Dr, do(E, S)) <=>

  (E = getDrivingTime(Origin, Destination) &
  execute(get_driving_time(Origin, Destination, Dr))) v

  (-(E = getDrivingTime(Origin,Destination)) & drivingTime(Origin,
  Destination, Dr, S)).

ConGolog Ansatz – ConGolog Anpassung (1)

ConGolog Programme generisch machen

- [1] definiert ein neues Konstrukt, das als Order genannt und durch „:“ Symbol gekennzeichnet wird
- Eine Order von zwei Aktionen a1:a2 wird definiert als:

$$a_1; \text{while}(\neg \text{Poss}(\text{a}_2))do(\pi a)[Poss(a)?; a]\text{endWhile};\text{a}_2$$

- Vorteil des neuen Order-Konstrukts im Vergleich mit dem Reihenfolge-Konstrukt (*sequence construct*)

$$buyAirTicket(\vec{x}); rentCar(\vec{y})$$

$$buyAirTicket(\vec{x}) : rentCar(\vec{y})$$

  - $Poss(rentCar(\vec{y}))$ erlegt auf, dass die Kreditkarte des Benutzers Begrenzung nicht überschreitet
  - Das zweite Programm hat die Flexibilität, eine Reihenfolge von Aktionen durchzuführen, um die Kreditkarte-Balance (Kontostand) zu reduzieren, um die Vorbedingung $Poss(rentCar(\vec{y}))$ zu erfüllen

[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.

ConGolog Ansatz – ConGolog Anpassung (2)

ConGolog Programme an Kundenwünschen anpassbar machen

- [1] führt ein neues bemerkenswertes „fluent" $Desirable(a,s)$ in „Situation Calculus" ein. D.h. Aktion a ist wünschenswert in der Situation s
- eine Aktion ist durchführbar, indem nicht nur $Poss(a,s)$ zutrifft, sondern auch $Desirable(a,s)$

$$Desirable(A(\vec{x}),s) \equiv \Omega_A \wedge R[C(do(A(\vec{x},s))]$$

  - $\Omega_A = \omega_1 \vee \omega_2 \vee ... \vee \omega_n$ sind die Bedingungen, die in der Situation, in der die Aktion durchgeführt wird, erfüllt werden müssen.
  - $C(do(A(\vec{x}),s))$ sind die Bedingungen, die in der Situation, nachdem die Aktion durchgeführt wurde, erfüllt werden müssen.
  - $R[C(do(A(\vec{x},s))]$ ist „repeated regression rewriting". Das bedeutet, $C(do(A(\vec{x}),s))$ zum Ausdruck $\Omega(s)$ neuegeschrieben wird, der in der Situation s bewiesen werden kann.

[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.

ConGolog Ansatz – ConGolog Anpassung (3)

Beispiel für das fluent $Desirable(a,s)$

- Die Studentin will fliegen, wenn die Fahrzeit mit dem Auto mehr als 3 Stunden beträgt. So haben wir:

$$\Omega_A = gt(DrivingTime(o,d),3,s) \quad \textbf{(1)}$$

- Außerdem hat sie Zeiten, in denen sie zu Hause sein muss und sie nicht wegfahren darf. So haben wir:

$$R[C(do(A(\vec{x}),s))] =$$
$$R[\neg(Away(dt,do(buyAirTicket(o,d,dt),s)) \wedge$$
$$MustbeHome(dt,do(buyAirTicket(o,d,dt),s)))]$$

ConGolog Ansatz – ConGolog Anpassung (4)

- Dieser Ausdruck wird unter Nutzung von „successor state axioms” für „fluent“ $Away(dt,s)$ und $MustbeHome(dt,s)$ neugeschrieben. z.B.:

$$Away(dt, do(a,s)) \equiv$$
$$[(a = buyAirTicket(o,d,dt) \wedge d \neq Home)$$
$$\vee (Away(dt,s) \wedge \neg(a = buyAirTicket(o,d,dt) \wedge d = Home))]$$

und $MustbeHome(dt, do(a,s)) \equiv MustbeHome(dt,s)$

- Von diesem stellt man fest:

$$R[\neg(Away(dt, do(buyAirTicket(o,d,dt),s)) \wedge \quad \textbf{(2)}$$
$$MustbeHome(dt, do(buyAirTicket(o,d,dt),s)))] =$$
$$d = Home \vee \neg MustbeHome(dt,s)$$

- Von (1) und von (2) haben wir:

$$Desirable(buyAirTicket(o,d,dt),s)) \equiv$$
$$gt(DriveTime(o,d),3,s) \wedge$$
$$(d = Home \vee \neg MustbeHome(dt,s))$$

ConGolog Ansatz – ConGolog Anpassung (5)

ConGolog Programme verwendbar machen

- Im Kontext der WSK werden Information-Sammlung-Services verwendet, wenn :
  - das Planungssystem unvollständiges Wissen des Anfangszustandes hat.
  - exogene Aktionen existieren, die die Welt auf die Weise, die das Planungssystem nicht vorher sagt, ändern
- Daher hat [1] ConGolog Programmen zu definieren, die durch eine Vielzahl von verschiedenen Planungssysteme verwendet werden können
  - [1] führt das Prädikat $ssf(\delta, s)$ (*short for self-sufficient*) ein
  - alle Vorbedingungen für die Aktionen, die die Programme durchzuführen versuchen, werden innerhalb des Programms realisiert, oder sind anerkannt als eine explizite Vorbedingung des Programms.

[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.

Middle Ground ConGolog

- *McIlraith, S., und Son* verwendet Programmiertechniken in Prolog zur Kombination der Online-Durchführung von Information-Sammlung-Services mit Offline-Simulation von Welt-Änderung-Services.
  - der Aufruf einer externen Funktion (d.h. Information-Sammlung-Service) wird in „successor state axioms" eines „fluent" $F(\vec{x}, s)$ in der folgenden Form dargestellt:

$$F(\vec{x}, do(A(\vec{x}), s)) \equiv exec(A(\vec{x}), s)$$

  - Durch diesen Aufruf kann der Wahrheitswert von einem bestimmten „fluent" $F(\vec{x}, s)$ festgestellt werden
  - Beispiel:

drivingTime(Origin, Destination, Dr, do(E, S)) <=>

(E = getDrivingTime(Origin, Destination) & execute(get_driving_time(Origin, Destination, Dr))) v

(-(E = getDrivingTime(Origin,Destination)) & drivingTime(Origin, Destination, Dr, S)).

  - Nachteil: ein Information-Sammlung-Service kann mehrfach aufgerufen werden, bis der wünschenswerte Plan gefunden wird

Zusammenfassung (1)

- Motivation-Beispiel vorgestellt
- Erklärung von OWL-S
- Einführung in KI-Planung
- WSK als Plannungsproblem beschrieben
- Situation Calculus, Golog und ConGolog erklärt
- ConGolog Ansatz von McIlraith, S., and Son

Zusammenfassung (2)

ConGolog Ansatz

- Der ConGolog-Ansatz bietet ein natürliches Formalismus zur automatischen Komposition von Web-Services auf dem Web.
- Ein klarer Ansatz durch Kombination **[1] und [2]**

  1. Gebe eine Menge von OWL-S Beschreibungen der (Atomic- oder Complex-) Web-Services

  2. Übersetze diese Beschreibungen in einer ConGolog-Aktionstheorie

  3. Für eine spezifische Aufgabe stellt ConGolog Interpret mit dem entsprechenden generischen Programm eine Reihenfolge der Welt-Änderung-Services für Durchführung fest.

- **kann nicht das WSK-Problem komplett lösen**
  - der ConGolog Ansatz kann die Pläne mit zeitlichen Begrenzungen nicht erzeugen
  - Wie alle anderen K.I. Ansätze kann ConGolog Ansatz auch das Problem über Zeitwert der unvollständigen Informationen nicht lösen

[1] McIlraith, S., and Son, T. C. Adapting Golog for Composition of Semantic Web Services. Proc. KRR, 482-493, 2002.

[2] Phan Minh, Fumio Hattori. Automatic Web Service Composition Using ConGolog. DABI. Portugal, 2006