Planning for Web Services

Evren Sirin & Bijan Parsia

MINDSWAP Research Group

University of Maryland, College Park
Objective

• Automated composition of Web Services
  – Using OWL-S

• AI planning has proven useful

• Identify the challenges of…
  – Describing Web Services using ontologies
  – Using planning for composition
  – Complexity of reasoning during planning
How does AI planning work?
How does AI planning work?

• State of the world

Facts known about the world

A
B
C

Initial State
How does AI planning work?

- **State of the world**
- **Planning Operators**

Actions that change the state

### Initial State

- A
- B
- C

### Operator 1

- Pre: A
- Del: B
- Add: D

### Operator 2

- Pre: A "\(\land\)" B
- Del: C
- Add: B

OWL-S AtomicProcess
How does AI planning work?

- State of the world
- Planning Operators

Initial State

OWL-S AtomicProcess

Actions that change the state
How does AI planning work?

- **State of the world**
- **Planning Operators**
- **Goal formula**

OWL-S AtomicProcess

Operator 1

- Pre: A
- Del: B
- Add: D

Operator 2

- Pre: A ∧ D
- Del: B
- Add: E

Logical formula that needs to be true in the final state

A  C  D

A  B  C  G

Initial State
HTN Planning

• Hierarchical Task Networks

• Plan with tasks not goals
  – Primitive task $\rightarrow$ Operator $\rightarrow$ AtomicProcess
  – Compound task $\rightarrow$ Method $\rightarrow$ CompositeProcess

• *Methods* decompose a task into subtasks
  – Standard operating procedures

• Find a decomposition that is executable starting from the initial state
OWL-S Processes as Planning Operator

• Map OWL-S descriptions to planning operators

(:atomic-process register-course
 :inputs (?student – Student ?course - Course)
 :precondition
   (and (?course hasPrerequisite ?anotherCourse)
       (?student passed ?anotherCourse))
 :effect (?student registered ?course))
Classical Planning

• Planners typically support only fairly limited reasoning capabilities
  – State is a set of ground atoms
  – Closed world assumption is used
  – Inference limited to Horn clause axioms

• Not nearly as expressive as OWL
  – OWL DL corresponds to a very expressive Description Logic: SHION(D)
Planning with OWL-S

- Preconditions and effects expressed in OWL
  - Atoms of SWRL (not SWRL rules)
- World state is represented as an OWL KB
- Planner interacts with the state through an OWL reasoner
  - Evaluate preconditions
    - Precondition is satisfied iff it is a logical consequence of the KB
  - Apply effects
    - Modify the KB accordingly
Example Service

• Schedule a Treatment
  – A Person trying to schedule an appointment for a medical Treatment in a hospital with a good Rating
  – Hospital should be supported by the health Insurance
  – Person should be available at the appointment time Hospital offers
Example Description

(:composite-process ScheduleTreatment
  :inputs (?Person ?Treatment ?Rating)
  :precondition
    (and (?Person health:hasInsurance ?Insurance)
        (?Insurance insurance:supports ?Hospital)
        (?Hospital medical:offers ?Treatment)
        (?Hospital zagat:hasRating ?Rating))

  {
    perform GetAvailableTimes(?Hospital);
    perform MakeTheAppointment(?Hospital ?ApptTime);
    perform UpdatePersonalCalendar(?ApptTime)
  }
Example Query

SELECT ?Hospital
WHERE
   (?Person health:hasInsurance ?Insurance),
   (?Insurance insurance:supports ?Hospital),
   (?Hospital medical:offers ?Treatment),
   (?Hospital zagat:hasRating ?Rating)
USING
   health FOR <http://.../health-ont>
...

11/22/04
Distinguished Variables in Queries

• Initial KB: \{Parent = \exists \text{hasChild}. \top, 
  \text{John}:\text{Parent}\}\}

• Query: SELECT ?x 
  WHERE (?x \text{hasChild} ?y) 
  Answer: \{?x \leftarrow \text{John}\}

• Query: SELECT ?x, ?y 
  WHERE (?x \text{hasChild} ?y) 
  Answer: \emptyset
Expressive Preconditions

- **Negated expressions**
  - \((\text{not}(\text{?x rdf:type Registered}))\)
  - \((\text{?x rdf:type } \neg\text{Registered})\)

- **Universally quantified variables**
  - Requires closed world interpretation

- **Disjunctive conditions**
  - Disjunctive queries

- **Numerical comparison/computation**
  - Built-in functions of SWRL
Applying Effects

• Each service may have +/- effects
• Simulate the action by applying the effects to the current state
• Operational meaning
  – Add positive effects to KB
  – Remove negative effects from KB
• Logical meaning
  – New state entails the positive effects
  – New state does not entail the negative effects
Positive Effects

• Add the statements to KB
  – This may cause inconsistency

(:atomic-process make-me-the-president
 :inputs (?p - Person ?cc - CreditCard)
 :precondition (?cc hasAvailableLimit $10,000)
 :effect (?p presidentOf USA))

• Incorrect description? Incompatible services?
Negative Effects

• Deleting cannot cause inconsistency
• Deleting one assertion may not be enough
  – Same fact inferred from other facts
• Example Service
  – Unregister ?person
  – Delete (?person memberOf W3C)
• Other assertions
  – SubProperty: (X boardMemberOf W3C)
  – InverseProperty: (W3C hasMember X)
  – Class Restrictions: (X rdf:type W3CMember)
Implementation

• Investigate the efficiency of the system
• Use OWL DL Reasoner Pellet
  – Based on tableaux algorithms for very expressive DLs
  – Supports conjunctive queries
• Integrate with SHOP2 planner
  – Efficient HTN planner
  – Tests done with Java version JSHOP
Query Answering

• Reduced to KB consistency test
• Not efficient for finding variable bindings
  – Multiple consistency tests for each possible variable binding
• Relatively less studied in DLs
Rolling Up

• Roll up the query into one concept description

  The query

  (?c rdf:type Computer),
  (?c manufacturedBy IBM),
  (?c hasCPU ?cpu), (?cpu cpuType Pentium)

• The concept

  Computer ⊨
  ∃manufacturedBy.{IBM} ⊨
  ∃hasCPU.∃cpuType.{Pentium})).
Retrieving Variable Bindings

• For each combination of variable bindings
  – Substitute variables with named individuals
  – Roll up the query
  – Test if the query with no variables is entailed

• Simple Optimization
  – Roll up the query for each variable separately
  – Retrieve likely candidates for the variable
  – Try only the likely candidates
Variable Dependencies

• Not all likely candidates are relevant
  – Binding value for a variable depends on the binding of another variable

• Generate likely candidates iteratively
  – Avoid expensive consistency checks
  – Find failing bindings early
Comparison of Algorithms
Comparison with SHOP2

![Graph showing Total Planning Time for JSHOP and JSHOP-Pellet. The graph indicates a sharp increase in planning time for JSHOP-Pellet starting from problem 15 onwards.]
Conclusions

• Investigate planning with Semantic Web Services
  – Modeling issues
    • Precondition and effect descriptions
  – Efficiency issues
    • Extra complexity

• Future Work
  – Focus on real-world Web Service domains
  – Different query optimization techniques