

# Using Web Ontologies for Web Service Composition

**Evren Sirin**

*MINDSWAP Research Group*

*University of Maryland, College Park*

## Why Web Service Composition?

- Buy all the books in Harry Potter series
- Make the travel arrangements to Waterloo
- Arrange doctor and hospital appointments for my mother

# Other Applications for WSC

- B2B Applications
  - Purchase the items ... from suppliers that satisfy ... and arrange the shipment so that delivery will be done in 3 days without exceeding the cost limit
- Grid computing
  - Retrieve the DNA data about ... apply the ... tests and create an output in ... format
- Pervasive computing
  - E-mail my presentation to all the people in the conference room and project it on the screen

## Web Services

- Self-contained, self-describing, modular applications
- Published, located, invoked across the Web
- May perform simple functions or complicated business processes
- Component oriented with loose coupling
- Interoperability between diverse applications

## Web Service Descriptions

- XML based descriptions
  - Messages also XML based (SOAP)
- Web Service Description Language (WSDL)
  - Define messages received/sent by the service
  - Abstract interfaces + Concrete bindings

# WSDL Example

- A simple web service
  - Input: ISBN number of a book
  - Output: Price of the book in USD

```
<message name="BNPriceRequest">  
  <part name="isbn" type="xsd:string" />  
</message>  
<message name="BNPriceResponse">  
  <part name="price" type="xsd:float" />  
</message>  
<portType name="BNPricePortType">  
  <operation name="getPrice">  
    <input message="ns1:BNPriceRequest" />  
    <output message="ns1:BNPriceResponse" />  
  </operation>  
</portType>
```

## Problems

- No semantics
  - What is an ISBN number? a book? price? US dollars? currency?
- No logical constraints between its input and output parameters
  - What is the value of output price if book is not in stock?

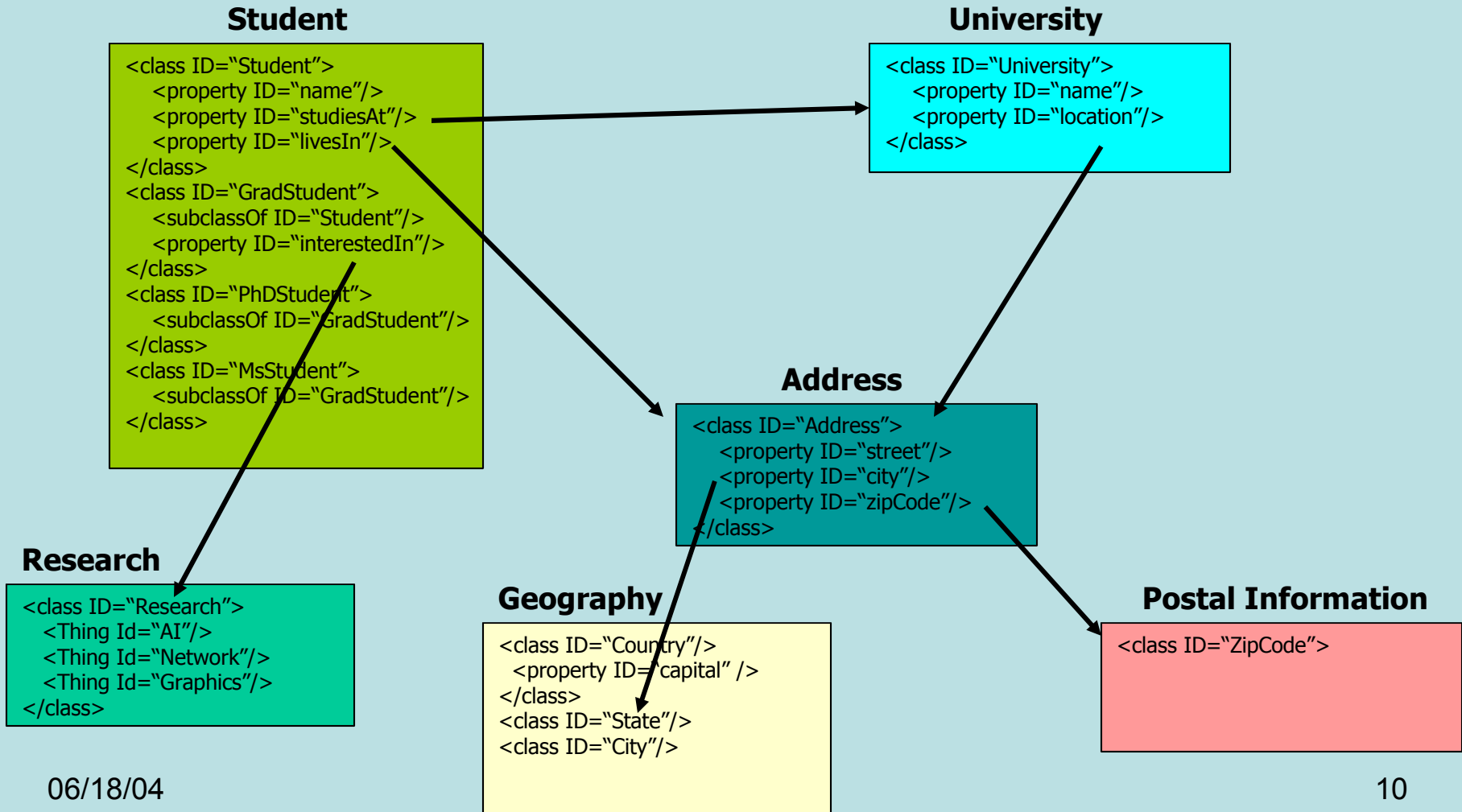
## Semantic Web

“an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation”

## SemWeb Vision

- Fill the gap between the concepts people use and the data computers interpret
- Make the data machine interpretable
- Loosely coupled, independently evolving Web Ontologies
- Provide a common understanding between heterogeneous systems
- Distributed, partially mapped and scalable

## Web Ontologies



# Applications on SemWeb

- FOAF project
  - Decentralized, distributed social network with millions of users
- Photo Markup
  - Better search for multimedia
  - Adobe's new XMP standard based on RDF
- eScience Projects
  - Collaboration between different areas of science (physics, chemistry and biology)

# SemWeb Languages

- Resource Description Framework (RDF)
  - Triples: Subject, predicate, object
- RDF Schema (RDF-S)
  - Frame-like technique to build taxonomies
  - Similar to semantic nets
- Web Ontology Language (OWL)
  - Extends RDF & RDF-S
  - Provides semantics based on Description Logics (DL)

# What can OWL do?

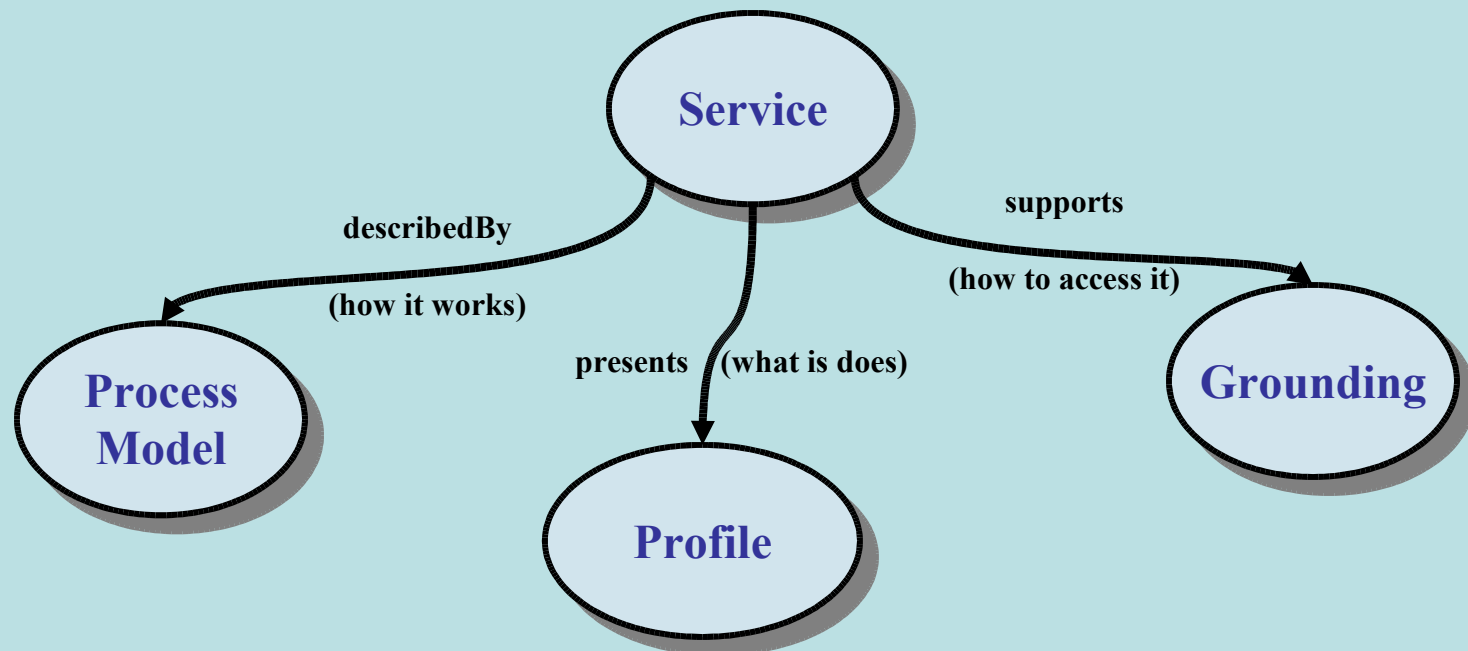
- Define what kinds of things are in the world (classes)
  - person, student, university, ...
- Define how these things relate (properties)
  - livesIn, studiesAt, ... (abstract)
  - firstName, birthDate, ... (concrete)
- Provide information about objects (individuals)
  - Evren, University of Maryland, ...
- Somewhat similar (but more general than) a DB Schema language plus a DB
  - Does not have closed-world assumption
  - Allows for uncertainty and vagueness

# OWL Examples

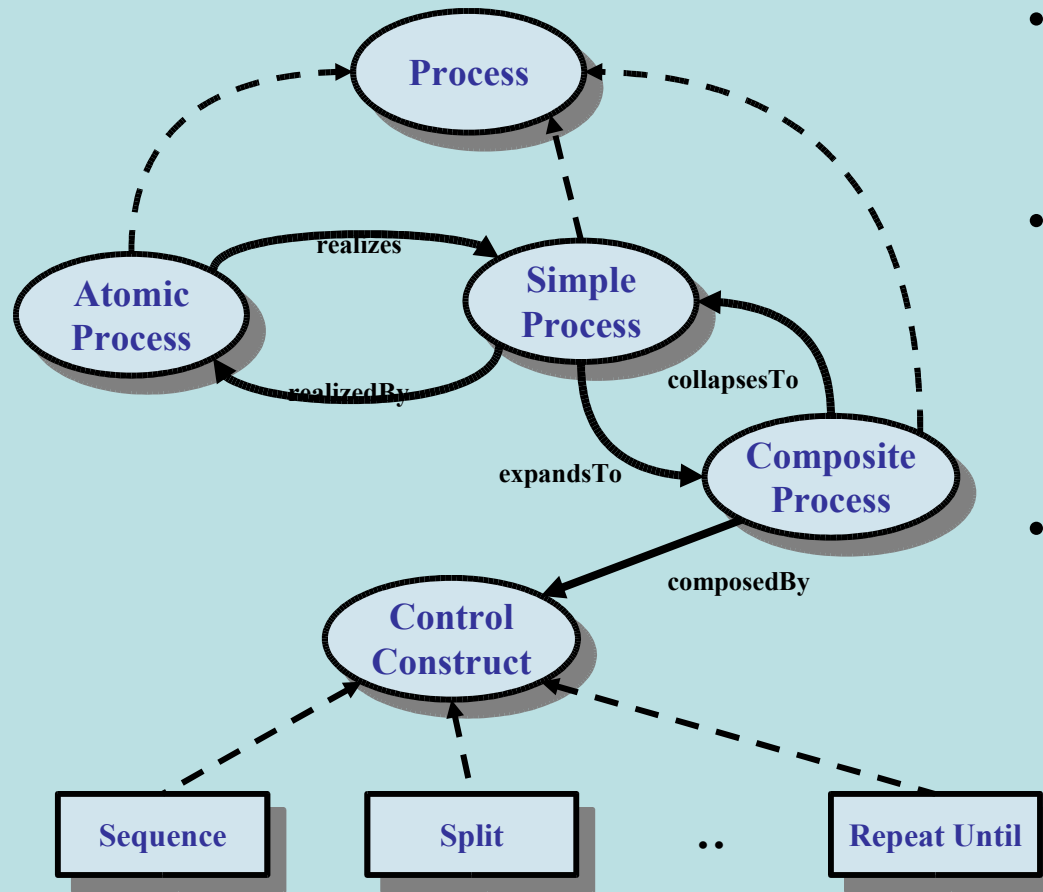
- Evren a GraduateStudent
  - GraduateStudent = Student  $\sqcap$   
 $\exists$ attends.GraduateSchool
- Evren livesIn Maryland
  - livesIn a FunctionalProperty
- Evren hasAdvisor JimHendler
  - hasAdvisor inverseOf isAdvisorOf
- Evren a  $\exists$ owns.(Car  $\sqcap$   $\exists$ hasColor.{Gray})
  - hasColor range Color
- Evren a  $\exists$ likes.(Basketball  $\sqcup$  Soccer)
  - Basketball  $\sqsubseteq$  Sport

# OWL-S

- Provide a set of ontologies for describing Web services
- Automate discovery, selection, composition and execution of services



# Process Model



- **Atomic processes**
  - directly invocable
  - black box
- **Composite processes**
  - consists of other processes
  - defined by a control construct
- **Simple processes**
  - abstract views, not executable
  - atomic process without a grounding
  - simplified representation of a composite process

# Matchmaking in OWL-S

- Use DL reasoning to match service requests with advertisements
  - Subsumption reasoning
  - Profile matching
- Example request

Service  $\sqcap$

$\exists$ isProvidedBy. $\exists$ hasRating.{High}  $\sqcap$

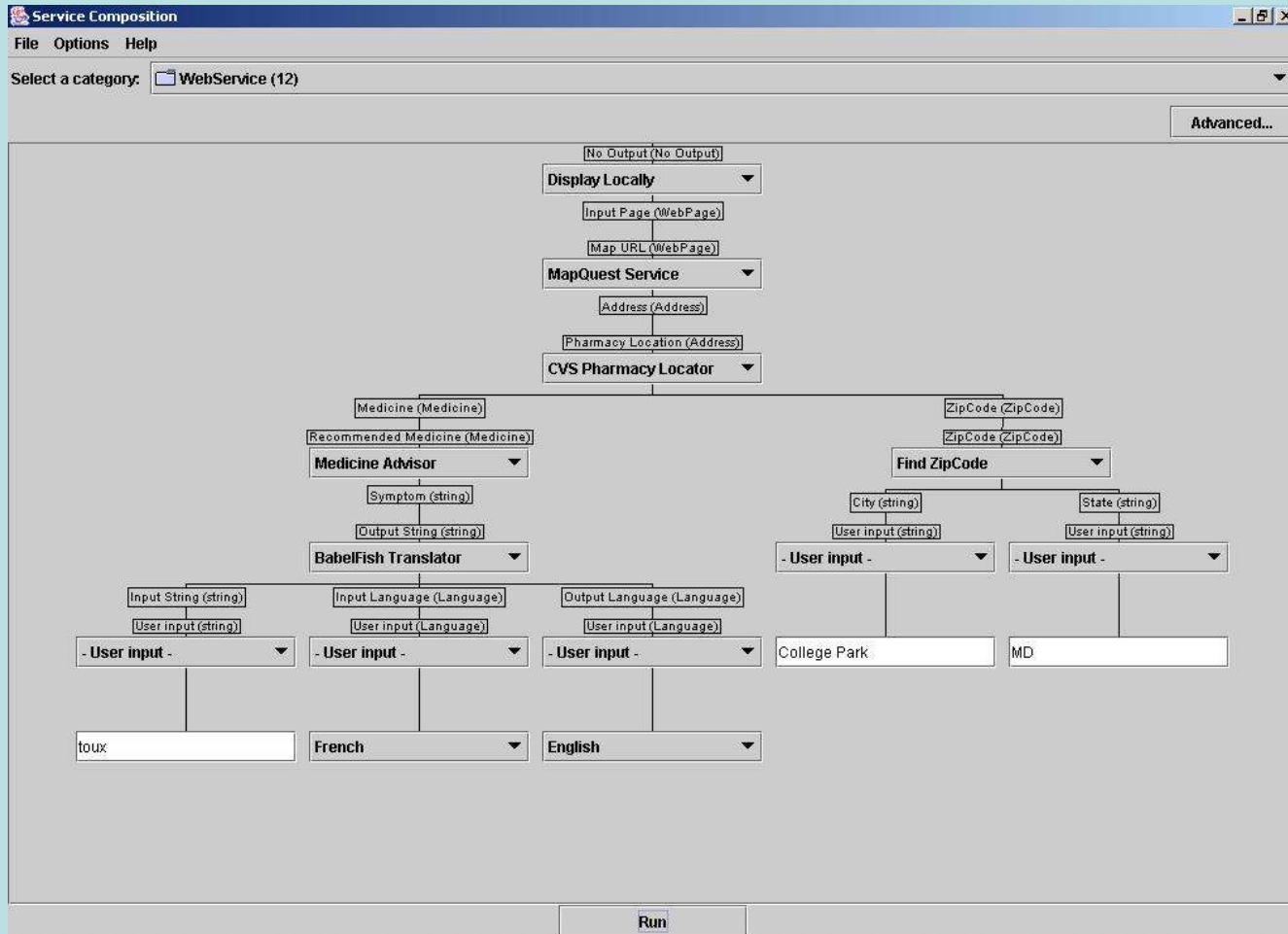
$\exists$ sells.(PC  $\sqcap$   $\exists$ hasCPU.IntelCPU)  $\sqcap$

$\exists$ canDeliver.{Maryland}

## Interactive Composition Approach

- Semi-automatic process
- Goal oriented
- Guide a user through steps of building a composition
  - Start with a service
  - Present matching services
  - Filter possibilities

## Matching By IOPEs



# Selection by Filtering

**Advanced options**

<b>MicrophonePickup</b>	sParamet...	Omnidirectional
<b>Location</b>	sParamet...	Latitude in the range 40 60
		Longitude equals 75
		Altitude equals
<b>Rating</b>	sParamet...	Good
	<b>Name</b>	LongName includes sensor
		ShortName equals
		Description equals
		SerialNumber equals
	<b>Type</b>	SensingMechanism - No constraint specified -
		SensingMeasureme... - No constraint specified -
		SensingMode Active
		TriggerMode - No constraint specified -

Close

# Shortcomings

- Not scalable to large number of services
  - Works well in a restricted domain
- Very simple service descriptions
  - Need to go beyond input/output
- Restricted workflows
  - Single output, no complex data flows
- Too much responsibility on user

## Automated Composition

- Automatically find a set of services to achieve a given objective
- Various different approaches
  - Program synthesis
  - Situation calculus + Logic programming
  - AI planning

# Planning

- Services – Actions – Operators
  - Parameters: Inputs of the service
  - Preconditions: Things that needs to be true before execution
  - Add list: Things that will become true after execution
  - Delete list: Things that will become false after execution
- An action describes a state transition
  - Planner has a state of the world
  - Generally represented as a set of atoms

# Planning Operator

- Map Web Service descriptions to planning operators

```
(:action register-course
:parameters (?student - Student ?course - Course)
:precondition
  (and (?course hasPrerequisite ?anotherCourse)
        (?student passed ?anotherCourse))
:effect (?student registered ?course))
```

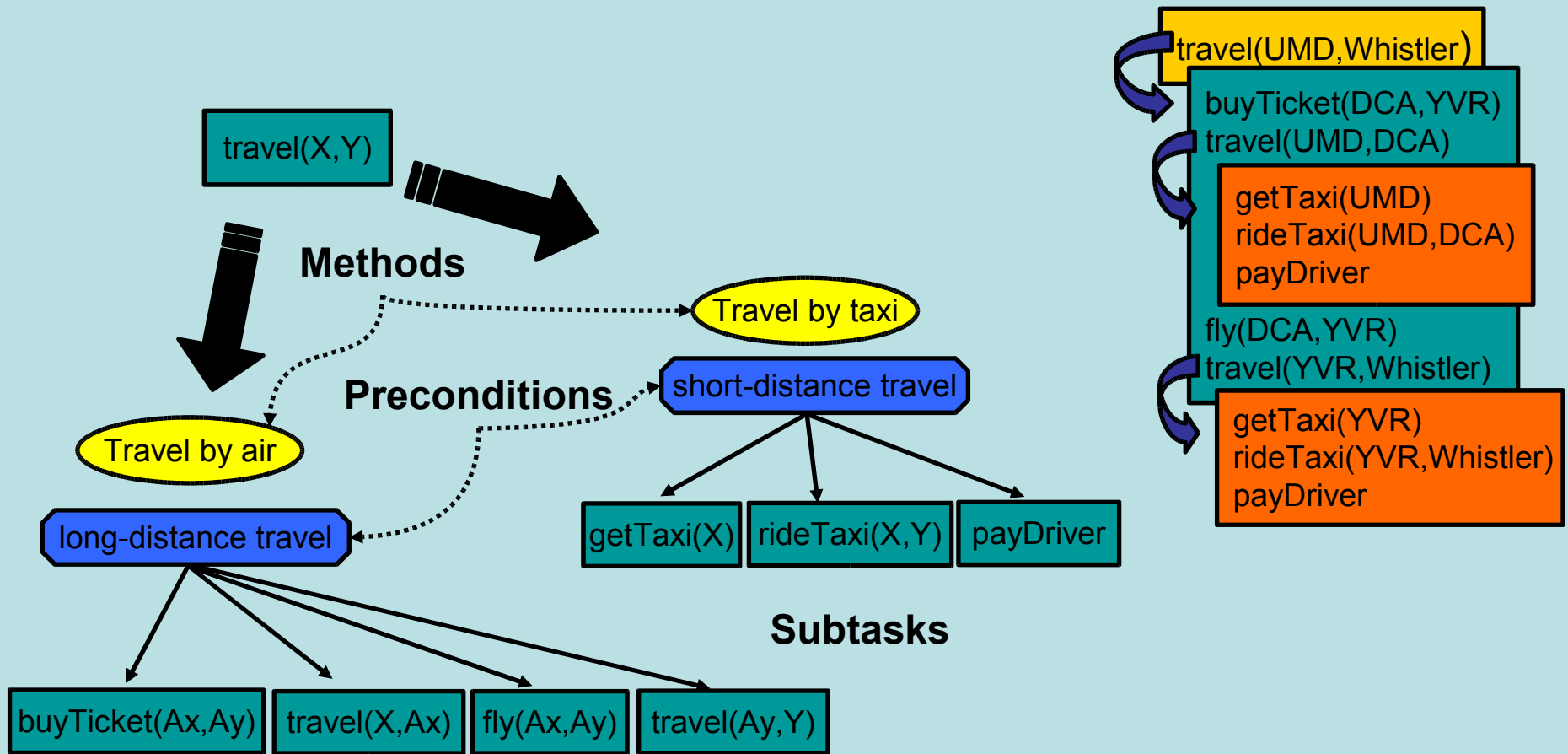
# Planning

- Produce a sequence of actions to achieve an objective
- Goal oriented
  - Make a logical statement true
  - Objective: (Evren owns aBook)
- Task oriented
  - Find primitive tasks to accomplish a composite task
  - Objective: BuyBook(aBook)

# HTN Planning

- Hierarchical Task Networks
- Objective is to perform a task
- *Operators* as usual
- *Methods* decompose a task into subtasks
  - Standard operating procedures
- SHOP2 is an efficient HTN planner
  - Won an award in IPC 2002
  - Ability to call external programs

## Decomposition in SHOP2



# From OWL-S to SHOP2

- Translate service-composition problems into planning problems for SHOP2
  - OWL-S processes  $\Rightarrow$  SHOP2's tasks
  - Descriptions of atomic processes  $\Rightarrow$  SHOP2's operators
  - Descriptions of composite processes  $\Rightarrow$  SHOP2's methods
- Correctness with respect to situation calculus semantics given in
  - S. McIlraith, T. Son, Adapting Golog for composition of semantic web services, KR2002

# Limitations of Classical Planning

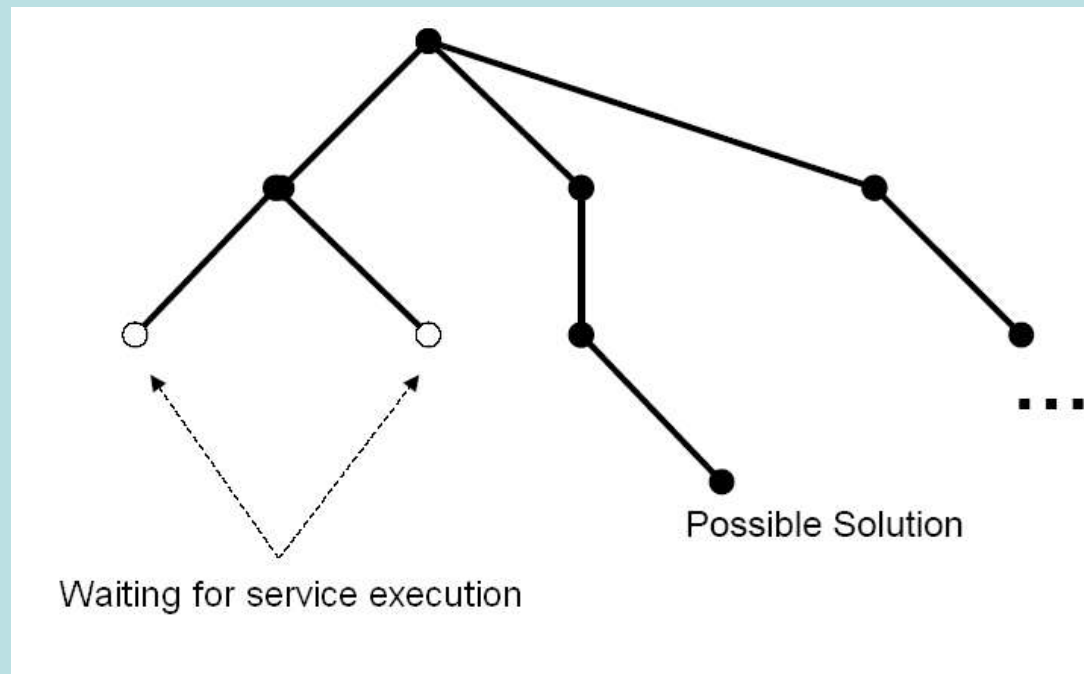
- Classical planning usually...
  - ...assumes complete information
    - Planner knows the complete initial state of the world
    - Planner has complete information at all times during planning
  - ...is done “off-line”
    - Plan generation and execution are two independent and sequential processes
- Not applicable to Web!

# Information Gathering

- Classical solutions
  - Gather all the information and then plan
  - Put sensing actions in your plan
    - Gather information during execution
- Interleaving planning and info gathering
- Gather information during planning
  - Execute info providing Web Services
  - Slows down planning process
    - Continue planning during execution

# Efficiency

- Issue a Web Service query
  - Continue with other possible decompositions



# What about reasoning?

- Planners typically support only fairly limited reasoning capabilities
  - State is a set of ground atoms
  - Closed world assumption
  - Inferencing limited to Horn clause axioms
  - Not nearly as expressive as OWL
    - OWL DL fragment corresponds to a very expressive Description Logic: SHION(D)

## Planner + Reasoner

- World state is represented as a KB
- Planner interacts with the state through a reasoner
  - Precondition evaluation
  - Applying effects
- Precondition is satisfied iff it is a logical consequence of the KB

# Example Service

- Schedule a Treatment
  - A *Person* trying to schedule an appointment in a hospital with a good *Rating* for a *Treatment*
  - *Hospital* should be supported by the *Insurance*. Both *Hospital* and *Person* should be available at the *TreatmentTime*.

# Example Method

(:method Schedule

:parameters (?Person ?Treatment ?Rating)

:precondition

**(and (?Person hasInsurance ?Insurance),  
(?Insurance supports ?Hospital),  
(?Hospital offers ?Treatment),  
(?Hospital hasRating ?Rating))**

:decomposition

(ordered

(GetAvailableTimes ?Hospital)

(MakeTheAppointment ?Hospital ?ApptTime)

(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>>

...

# Query Answering on SemWeb

- Open world semantics
  - Cannot use Negation As Failure
- Uncertainty in the KB
  - Answering disjunctive query is harder
  - Evren a  $\exists$ likes.(Basketball  $\sqcup$  Soccer)
    - Evren likes Basketball? *False*
    - Evren likes Soccer? *False*
    - Evren likes Basketball  $\sqcup$  Evren likes Soccer? *True*

# Numerical Computation

- Preconditions that requires numeric computation along with logical inferencing

```
(:action buy-book
```

```
  :parameters (?b - Book ?cc - CreditCard)
```

```
  :precondition (and (?b hasCost ?price)
```

```
                  (?cc hasAvailableLimit ?limit)
```

```
                  (?price < ?limit))
```

```
  :effect ...)
```

# 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

```
(:action make-me-the-president
  :parameters (?p - Person ?cc - CreditCard)
  :precondition (?cc hasAvailableLimit $10,000)
  :effect (?p president 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
- Integrate with SHOP2 planner

# 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  $\sqcap$
  - $\exists$ manufacturedBy.{IBM}  $\sqcap$
  - $\exists$ hasCPU. $\exists$ cpuType.{Pentium})).

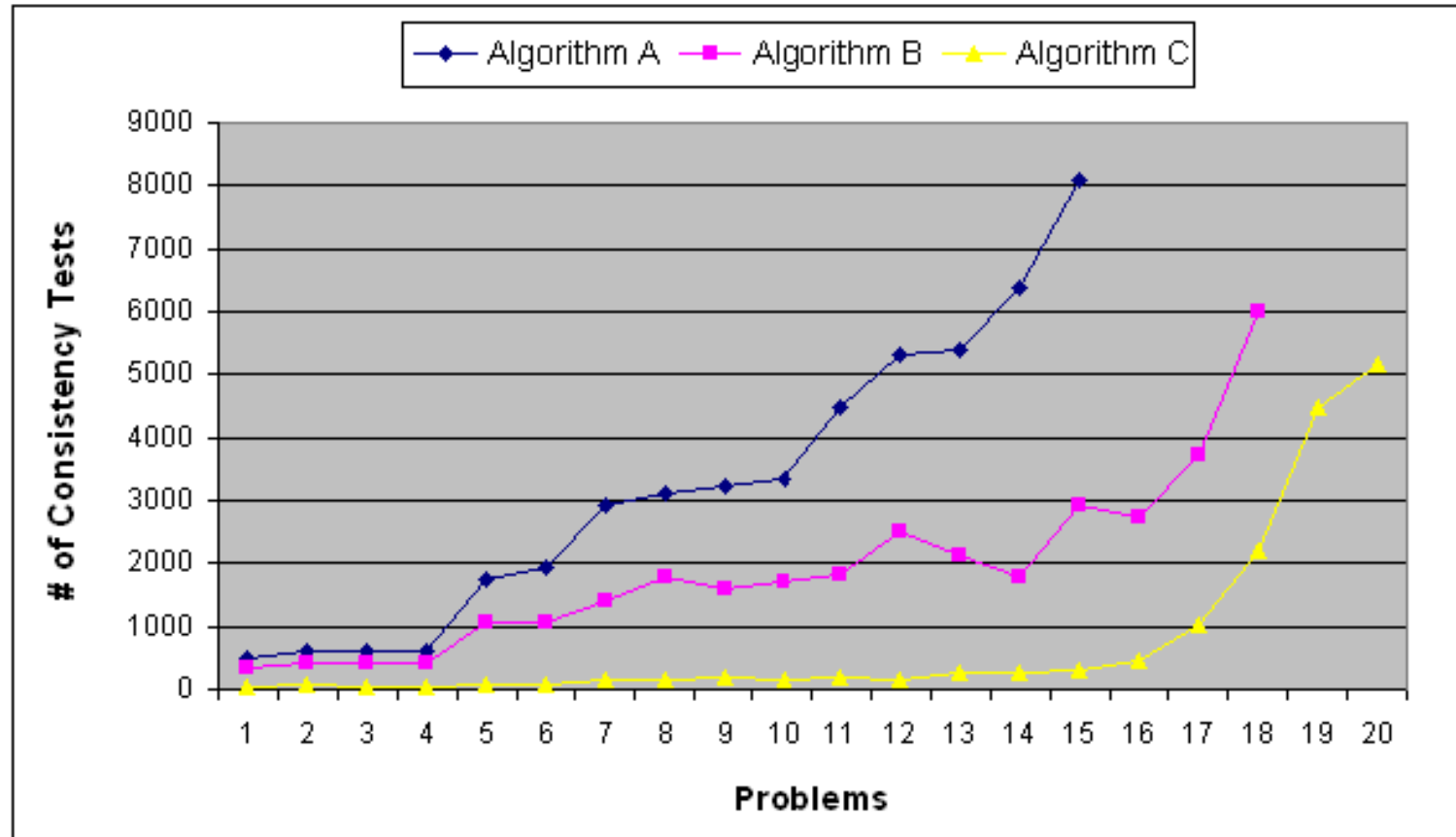
# Retrieving Variable Bindings

- For each variable
  - Substitute variable with a named individual
  - Roll up the query
  - Test if the query with no variables is entailed
- Optimization
  - Roll up the query for each variable separately
  - Retrieve likely candidates for the variable
  - Try only the likely candidates

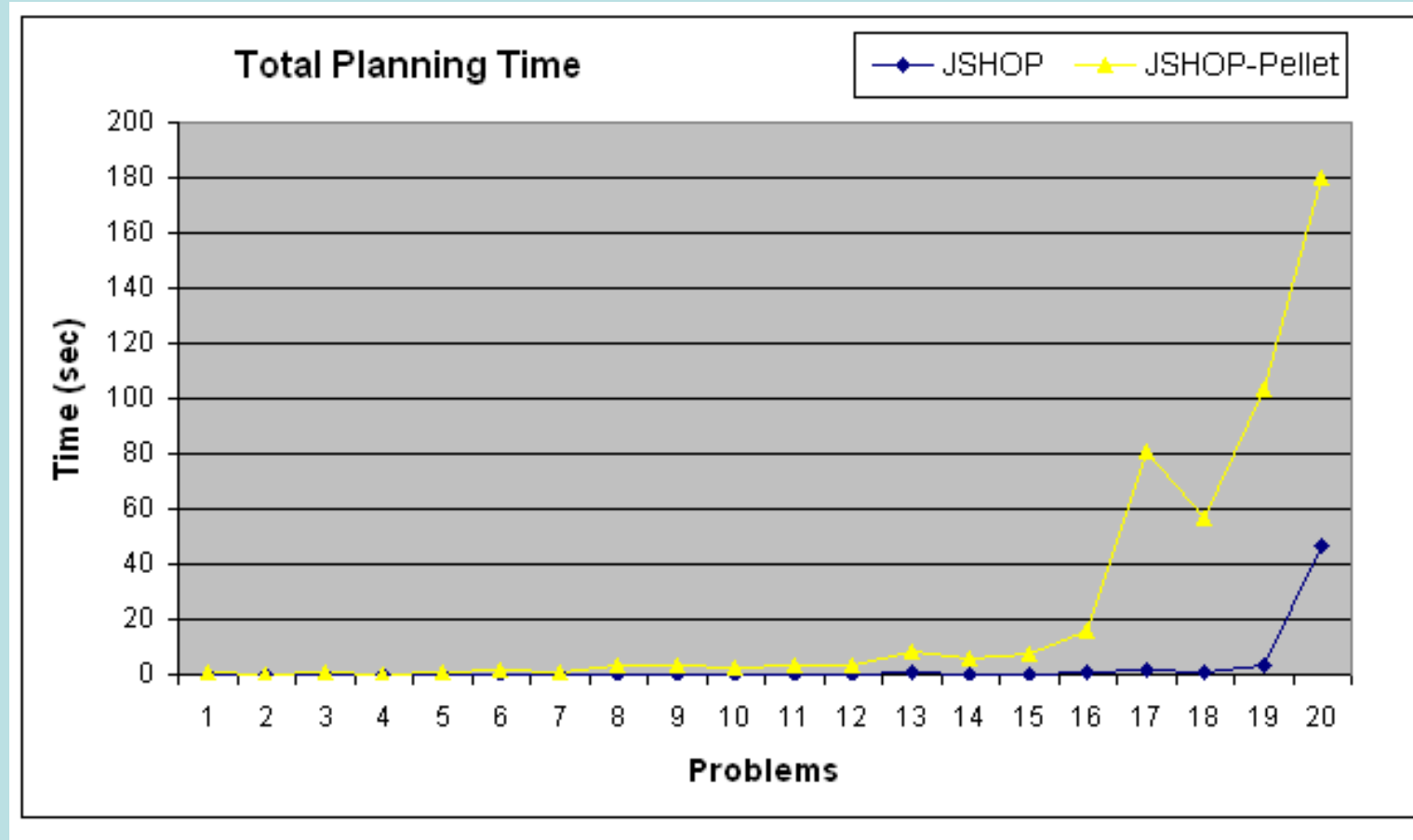
## Variable Dependencies

- 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



# Conclusions

- A framework where
  - Ontologies describe Web Service behaviors
  - AI planning is used to compose services
  - Description Logics for reasoning
- A lot of issues related to the nature of Web
  - Incomplete information
  - Open world semantics
  - Expressivity of OWL

## Future Directions

- Partial service descriptions
  - Composite descriptions with abstract components
  - Discover services on the fly that matches abstract descriptions
- QoS considerations for compositions
- User interaction
  - Mixed-initiative system

## Acknowledgements

- Maryland Information and Networks Dynamics Lab, Semantic Web and Agents Project (MINDSWAP)
  - Bijan Parsia and James Hendler
- SHOP2 Group
  - Ugur Kuter and Dana Nau
- Corporate Research Partners
  - Fujitsu Labs of America, College Park