Inducing Source Definitions for Web Service Composition

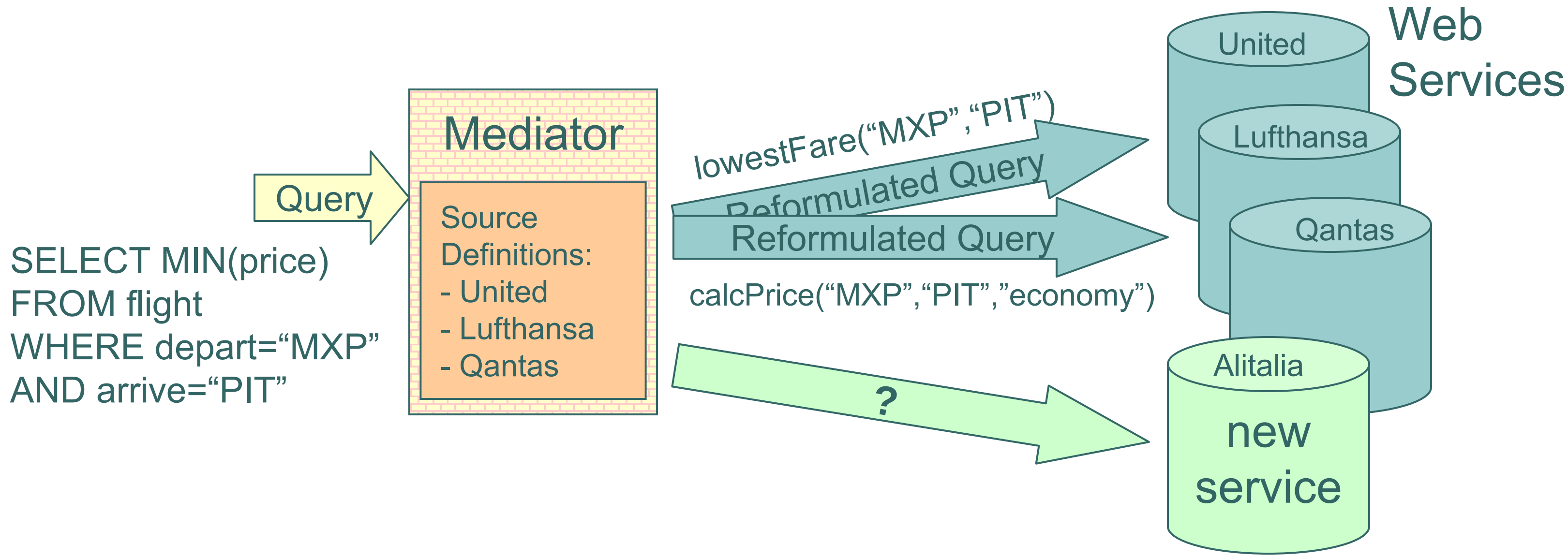
Mark Carman
Craig Knoblock

Overview of the Talk

- Mediators for composing services
- Inducing source definitions: A simple example
- Generation & test framework
- Case study & preliminary experiments
- Challenges & future work
- Related work

Mediators for Composing Web Services

- Provide uniform access to heterogeneous sources
- Source definitions are used to reformulate query
- New service, no source definition, no integration!
- Can we discover definitions automatically?



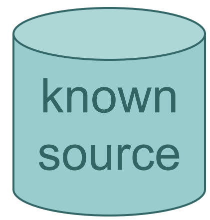
Inducing Source Definitions: A Simple Example

- Step 1: use metadata to classify input types ($)
- Step 2: invoke service and classify output types

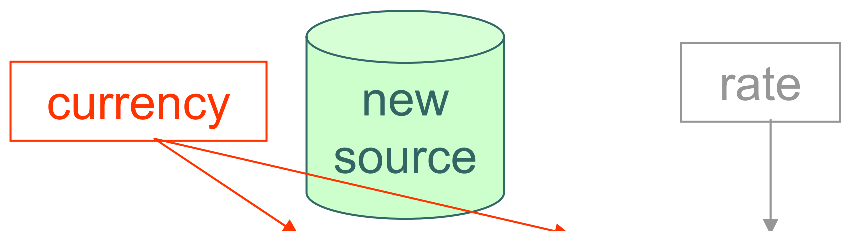
Mediator

Semantic Types:
currency ⊇ {USD, EUR, AUD}
rate ⊇ {1936.2, 1.3058, 0.53177}

Predicates:
exchange(currency,currency,rate)



LatestRates($country1,$country2,rate):-
exchange(country1,country2,rate)

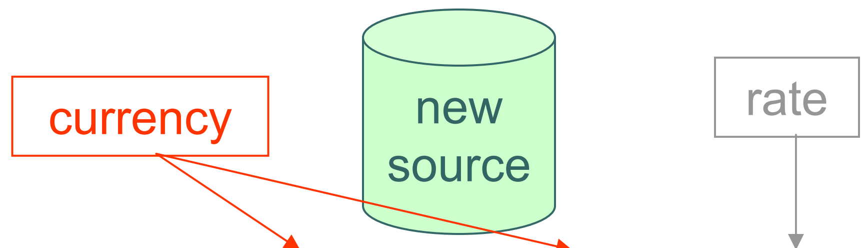


RateFinder($fromCountry,$toCountry,val):- ?

{<EUR,USD,1.30799>,<USD,EUR,0.764526>,…}

Inducing Source Definitions: A Simple Example

- Step 3: generate plausible source definitions
- Step 4: reformulate in terms of other sources



RateFinder($fromCountry,$toCountry,val):- ?

Candidate definitions

def_1($from, $to, val) :- exchange(from,to,val)

def_2($from, $to, val) :- exchange(to,from,val)

Reformulated definitions

def_1($from, $to, val) :- LatestRates(from,to,val)

def_2($from, $to, val) :- LatestRates(to,from,val)

Inducing Source Definitions: A Simple Example

- Step 5: invoke services and compare output

def_1($from, $to, val) :- exchange(from,to,val)

def_2($from, $to, val) :- exchange(to,from,val)

def_1($from, $to, val) :- LatestRates(from,to,val)

def_2($from, $to, val) :- LatestRates(to,from,val)

match

Input	RateFinder	Def_1	Def_2
<EUR,USD>	1.30799	1.30772	0.764692
<USD,EUR>	0.764526	0.764692	1.30772
<EUR,AUD>	1.68665	1.68979	0.591789

The Framework

Intuition: *Services often have similar semantics, so we should be able to use what we know to induce that which we don't*

Two phase algorithm

For each operation provided by the new service:

1. Classify its input/output data types
   - *Classify* inputs based on metadata similarity
   - *Invoke* operation & classify outputs based on data
2. Induce a source definition
   - *Generate* candidates via Inductive Logic Programming
   - *Test* individual candidates by reformulating them

Comparing Candidate Definitions

Sources may return *multiple tuples* for each input

&

Sources may be *incomplete*

- Use Record Linkage to discover common tuples
- Compare candidate definitions using:

$$score(def) = \frac{|src \cap def|}{|src| + |def|}$$

- Approximate score through sampling
- Terminate search when highest score converges:

$$\frac{mean(score(def_1) - score(def_2))}{\sqrt{variance(score(def_1) - score(def_2))/N}} \geq t\_value(0.05, N)$$

Use Case: Zip Code Data

- Single *real* zip-code service with multiple operations
- The first operation is defined as:

```
getDistanceBetweenZipCodes($zip1, $zip2, distance) :-
        centroid(zip1, lat1, long1),
        centroid(zip2, lat2, long2),
        distanceInMiles(lat1, long1, lat2, long2, distance).
```

- Goal is to induce definition for a second operation:

```
getZipCodesWithin($zip1, $distance1, zip2, distance2) :-
        centroid(zip1, lat1, long1),
        centroid(zip2, lat2, long2),
        distanceInMiles(lat1, long1, lat2, long2, distance2),
        (distance2 ≤ distance1),
        (distance1 ≤ 300).
```

- Same service so no need to classify inputs/outputs or match constants!

Generating definitions: ILP

- Want to induce source definition for:

```
getZipCodesWithin($zip1, $distance1, zip2, distance2)
```

- Predicates available for generating definitions:

  {centroid, distanceInMiles, ≤,=}

- New type signature contains that of known source
  - Use known definition as starting point for local search:

```
getDistanceBetweenZipCodes($zip1, $zip2, distance) :-
        centroid(zip1, lat1, long1),
        centroid(zip2, lat2, long2),
        distanceInMiles(lat1, long1, lat2, long2, distance).
```

Generating definitions: ILP

- Want to induce source definition for:

getZipCodesWithin($zip1, $distance1, zip2, distance2)

	Plausible Source Definition
1	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d1), (d2 = d1)
2	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d1), (d2 ≤ d1)
3	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), (d2 ≤ d1)
4	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), (d1 ≤ d2)
5	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), (d1 ≤ #d)
6	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), (lt1 ≤ d1)
	…
n	cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), (d2 ≤ d1), (d1 ≤ #d)

INVALID
d2 unbound!

#d is a
constant

UNCHECKABLE
lt1 inaccessible!

contained in
defs 2 & 4

Testing definitions

Checking definitions requires LOTS of queries to sources!

- Reformulation's binding constraints may be different:

```
def_1($zip1, $distance1, zip2, distance2) :-
        centroid(zip1, lat1, long1),
        centroid(zip2, lat2, long2),
        distanceInMiles(lat1, long1, lat2, long2, distance1),
        (distance2 = distance1).
```

```
def_1($zip1, distance1, $zip2, distance2) :-
        getDistanceBetweenZipCodes($zip1, $zip2, distance1),
        (distance2 = distance1).
```

- Should invoke operation with every possible zip code!
- Don't want to be banned from using the service!

Implementation:

1. Store output tuples for reuse across definitions & trials
2. Sample to estimate score for ∀-type queries

Preliminary Results

Settings:

- Number of zip code constants initially available: 6
- Number of samples performed per trial: 20
- Number of candidate definitions in search space: 5

Results:

- Converged on “almost correct” definition!!!

```
getZipCodesWithin($zip1, $distance1, zip2, distance2) :-
        centroid(zip1, lat1, long1),
        centroid(zip2, lat2, long2),
        distanceInMiles(lat1, long1, lat2, long2, distance2),
        (distance2 ≤ distance1),
        (distance1 ≤ 243).
```

- Number of iterations to convergence: 12, never, …
- Lesson learned: Need strategy for selecting inputs!

Active Input Selection

- Idea: *Select input tuples which best differentiate the two best performing candidates*
- Sometimes it is possible to select inputs that are guaranteed not to return tuples for one definition:

cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), ($d2 \leq d1$)

cen(z1,lt1,lg1), cen(z2,lt2,lg2), dIM(lt1,lg1,lt2,lg2,d2), ($d2 \leq d1$), ($d1 \leq 243$)

- Useful only if we can check this property *without* accessing any sources
  - the predicates involved must be interpreted

Challenges & Future Work

- Need methodology for selecting inputs
  - Random strategy results in very long convergence times
  - Actively select inputs to best differentiate candidates!
  - Take variable type into account (nominal or numeric?)
- Number of tuples needed for effective sampling
  - Depends on number of trials performed thus far
  - Possibly also on number of known constants
- Compare local and global ILP search
- Need methodology for assigning constants in definitions

Related Work

- Classifying Web Services
  (Hess & Kushmerick 2003), (Johnston & Kushmerick 2004)
  - Classify input/output/services using metadata/data
  - We learn semantic relationships between inputs & outputs
- Category Translation
  (Perkowitz & Etzioni 1995)
  - Learn functions describing operations available on internet
  - We concentrate on a relational modeling of services
- CLIO
  (Yan et. al. 2001)
  - Helps users define complex mappings between schemas
  - They do not automate the process of discovering mappings
- iMAP
  (Dhamanka et. al. 2004)
  - Automates discovery of certain complex mappings
  - Our approach is more general (ILP) & tailored to web sources
  - We must deal with problem of generating valid input tuples