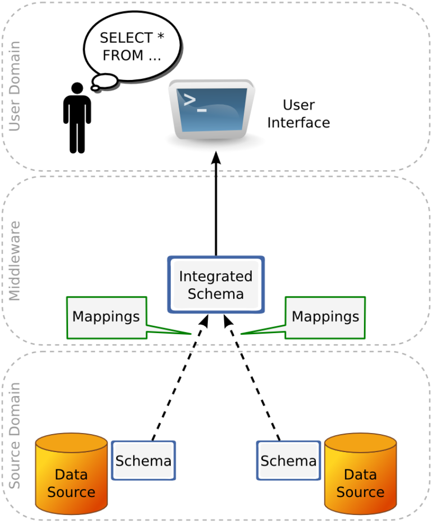
Integrating and Querying Web Service-Accessed Sources

Research Overview

Demian Lessa, Michalis Petropoulos,
Alin Deutsch (UCSD)

CSE Department, SUNY at Buffalo

Fall 2008



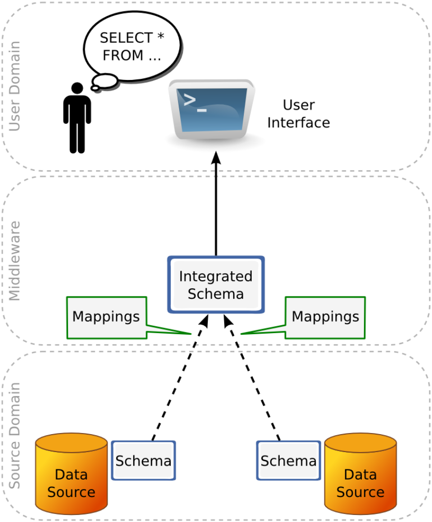
Classic Integration

The integrated schema represents the data of the underlying sources in a standard, uniform way.

Mappings express relationships between integrated schema and source schemas explicitly.

Typically,

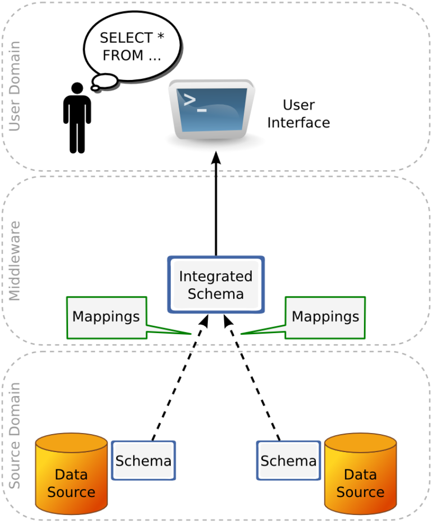
- one mapping per relation of the integrated schema
- mappings are union queries against the source schemas
- all mappings are executable



Users formulate queries against the integrated schema and submit them through the UI.

Users are oblivious of all the complexity involved in mapping source schemas to the integrated schema.

Users have no control or knowledge of which sources contribute in answering their queries.



Query answering semantics:

- user queries are expanded using the mappings
- individual queries in the expansion are sent to the appropriate source for execution
- the middleware combines the data from the sources

Let's see an example...

Schemas

Source A:

```
srcA_Specialists(LastName, FirstName, Specialty)
```

Source B:

```
srcB_Specialists(LastName, FirstName, Specialty, Phone)
```

Integrated:

```
Specialists(LastName, FirstName, Specialty)
```

Mapping

```
CREATE VIEW Specialists(LastName, FirstName, Specialty) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_Specialists
 UNION
 SELECT FirstName, LastName, Specialty
 FROM srcB_Specialists;
```

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

Unfolding

```
SELECT FirstName, LastName, Specialty
FROM
 (SELECT FirstName, LastName, Specialty
  FROM srcA_Specialists
  UNION
  SELECT FirstName, LastName, Specialty
  FROM srcB_Specialists)
WHERE LastName='Smith';
```

User Query

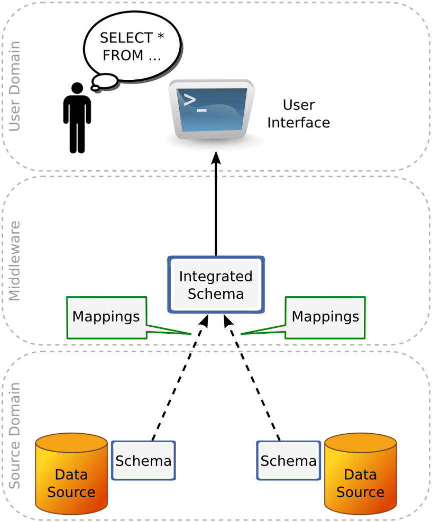
```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

Unfolding

```
SELECT FirstName, LastName, Specialty
FROM
 (SELECT FirstName, LastName, Specialty
  FROM srcA_Specialists
  UNION
  SELECT FirstName, LastName, Specialty
  FROM srcB_Specialists)
WHERE LastName='Smith';
```

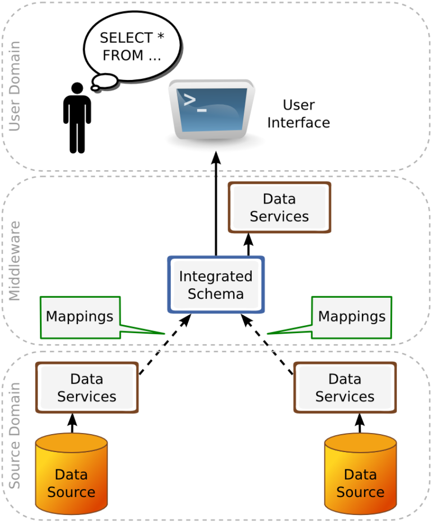
Query is Feasible!

The middleware pushes the selection down to each query in the union, sends the queries to the sources, and combines the results.



What is the problem with this approach?

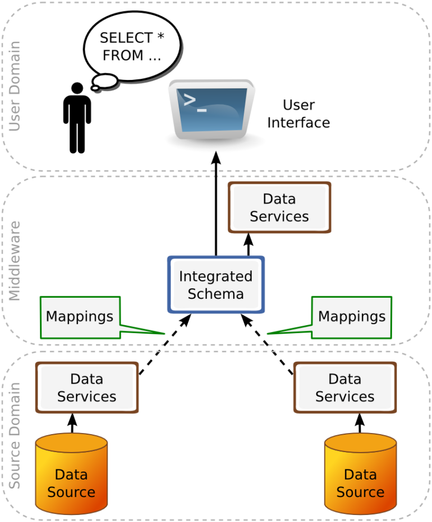
- not realistic for most scenarios: sources expose all their data!
- realistic approach: sources provide limited access to their data.



Yerneni et al (1999) explore the limited capabilities of underlying sources.

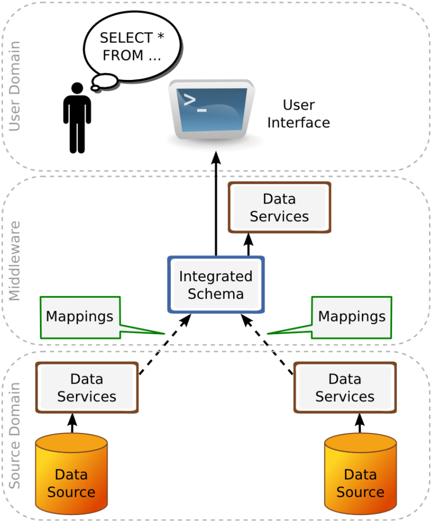
Their approach considers:

- sources that expose their data through data services
- mappings expressed against data services
- automatically computed integrated data services



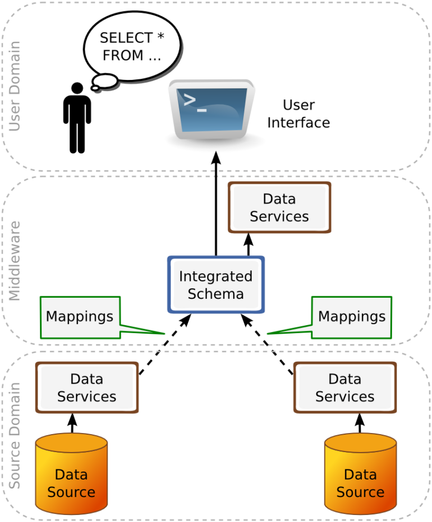
Source data services:

- queries with limited capabilities expressed as templates
- a template associates one adornment for every field of the data service
- an adornment determines if a field must be bound when executing the data service



Mappings:

- one mapping per relation of the integrated schema
- mappings are union queries against the data services
- mappings need not be executable
- the middleware computes templates for all mappings
- integrated data services are mappings combined with their respective templates



Query answering semantics:

- user queries are expanded using the mappings
- the middleware verifies if there is an executable query plan for the query
- if there is, individual data services in the expansion are sent to the appropriate source for execution
- the middleware combines the data from the sources

Let's see a couple examples...

Source Data Services

Source A:

srcA_getPhysicians(LastName^f, FirstName^f, Specialty^b, State^b)

Integrated Schema

Physicians(LastName, FirstName, Specialty)

Mapping

```
CREATE VIEW Physicians(LastName^f, FirstName^f, Specialty^b) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_getPhysicians(LastName, FirstName, Specialty, State);
```

Source Data Services

Source A:

srcA_getPhysicians(LastName^f, FirstName^f, Specialty^b, State^b)

Integrated Schema

Physicians(LastName, FirstName, Specialty)

Mapping

~~CREATE VIEW Physicians(LastName^f, FirstName^f, Specialty^b) AS~~
~~SELECT FirstName, LastName, Specialty~~
~~FROM srcA_getPhysicians(LastName, FirstName, Specialty, State);~~

Problem!

State must be bound in any call to srcA_getPhysicians due to its 'b' adornment. The mapping projects out the State field, so it is impossible to satisfy this capability!!!!

Source Data Services

Source A:
srcA_getPhysicians(LastName^f, FirstName^f, Specialty^b, State^b)

Integrated Schema

Physicians(LastName, FirstName, Specialty)

Mapping (alternate)

```
CREATE VIEW Physicians(LastName^f, FirstName^f, Specialty^b) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_getPhysicians(LastName, FirstName, Specialty, 'NY')
 UNION
 SELECT FirstName, LastName, Specialty
 FROM srcA_getPhysicians(LastName, FirstName, Specialty, 'CA')
 ...
```

Source Data Services

Source A:

srcA_getPhysicians(LastName^f, FirstName^f, Specialty^b, State^b)

Integrated Schema

Physicians(LastName, FirstName, Specialty)

Mapping (alternate)

```
CREATE VIEW Physicians(LastName^f, FirstName^f, Specialty^b) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_getPhysicians(LastName, FirstName, Specialty, 'NY')
 UNION
 SELECT FirstName, LastName, Specialty
 FROM srcA_getPhysicians(LastName, FirstName, Specialty, 'CA')
 ...
```

Problem!

Enumeration of all possible bindings!

Source Data Services

Source A:
srcA_getSpecialists(LastName^b, FirstName^f, Specialty^f)
Source B:
srcB_getSpecialists(LastName^f, FirstName^f, Specialty^b, Phone^u)

Integrated Schema

Specialists(LastName, FirstName, Specialty)

Mapping

CREATE VIEW Specialists(LastName^b, FirstName^f, Specialty^b) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_getSpecialists(LastName, FirstName, Specialty)
 UNION
 SELECT FirstName, LastName, Specialty
 FROM srcB_getSpecialists(LastName, FirstName, Specialty, Phone);

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

Query is not Feasible

The integrated service `Specialists` requires that both LastName and Specialty be bound in the user query, but the user did not satisfy the capability on Specialty!!!

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith' AND Specialty='Neurology';
```

Unfolding

```
SELECT FirstName, LastName, Specialty
FROM
 (SELECT FirstName, LastName, Specialty
  FROM srcA_getSpecialists(LastName, FirstName, Specialty)
  UNION
  SELECT FirstName, LastName, Specialty
  FROM srcB_getSpecialists(LastName, FirstName, Specialty, Phone))
WHERE LastName='Smith' AND Specialty='Neurology';
```

User Query

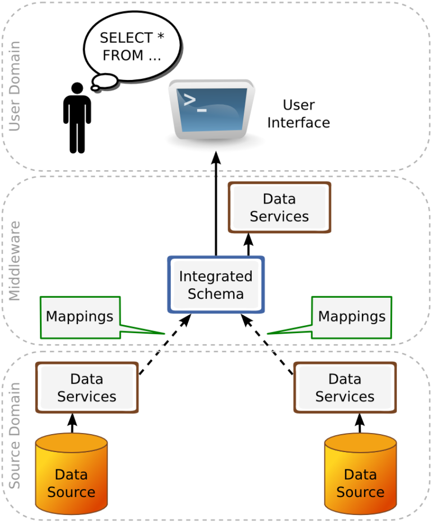
```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith' AND Specialty='Neurology';
```

Unfolding

```
SELECT FirstName, LastName, Specialty
FROM
 (SELECT FirstName, LastName, Specialty
  FROM srcA_getSpecialists(LastName, FirstName, Specialty)
  UNION
  SELECT FirstName, LastName, Specialty
  FROM srcB_getSpecialists(LastName, FirstName, Specialty, Phone))
WHERE LastName='Smith' AND Specialty='Neurology';
```

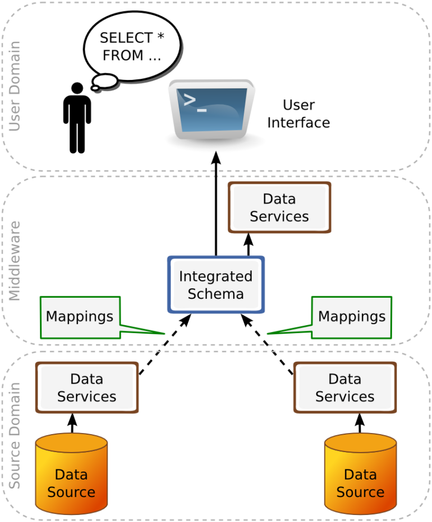
Query is Feasible!

Notice, however, that both capabilities are applied to all queries in the union and not selectively, where they are needed.



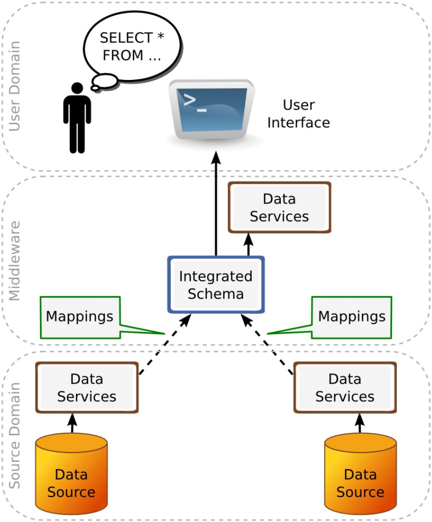
What is the problem with this approach?

- no support for mappings that project out fields with limited capabilities
- coupling of capabilities through mappings- i.e., capabilities of all sources must be satisfied
- no semantic relationship between data services and source schemas- i.e., there is no way of telling whether two services contribute subsets of the same data



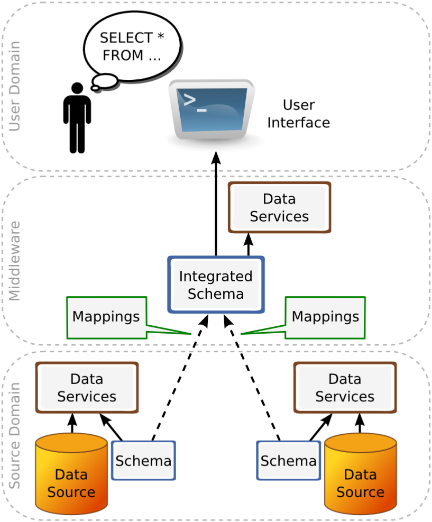
Further issues from the user's perspective:

- why is a query infeasible?
- can it be modified to become feasible?
- if so, are there capabilities that must be satisfied? which ones?
- in general, users have to go through a long trial-and-error process before formulating a feasible query



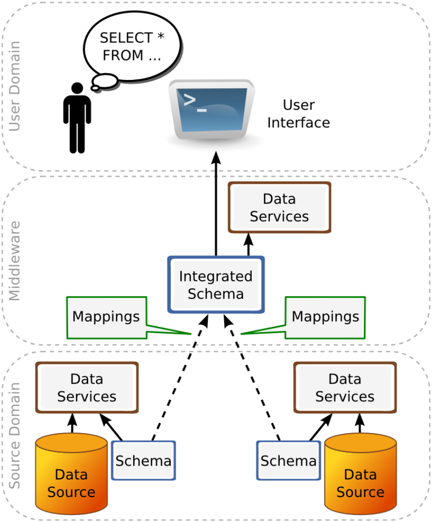
Even when users eventually obtain a feasible query, they are still unable to...

- select the sources that should contribute to the query during query formulation
- identify sources that contribute to the query after obtaining a feasible query
- extend their (feasible) queries in order to extract more data from the sources



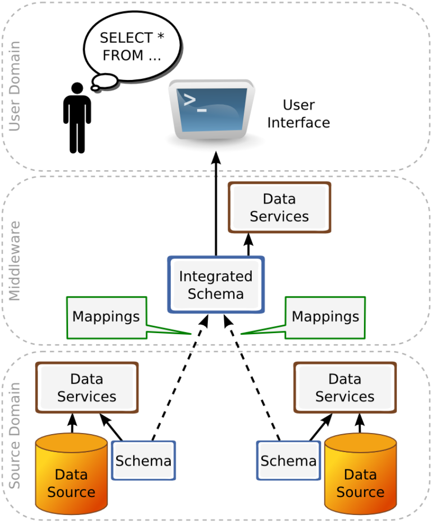
Source Domain Overview

- data sources expose their schema and publish a set of data services with limited capabilities
- each data service is a $CQ^{=p}$ query against a source schema
- data sources expose the actual query definition of every data service
- data sources only support queries through their data services interface



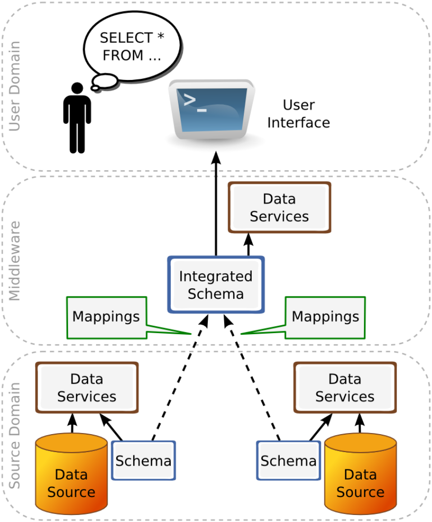
Middleware Overview

- integration engineers write a set of mappings per integrated schema relation
- this effectively decouples capabilities in the mappings
- each mapping is a $CQ^{=}$ query against the set of source schemas
- hence, engineers can focus on content integration



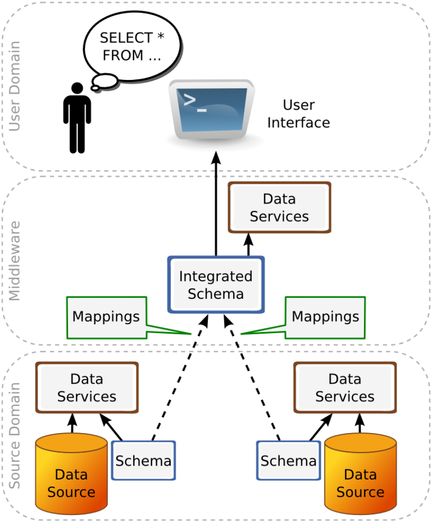
Middleware Overview

- the middleware computes the set of integrated data services automatically
- each integrated data service is a $CQ^{=p}$ query against the integrated schema



User Domain Overview

- users write (unions of) conjunctive queries against the integrated schema using a GUI
- the GUI provides visual suggestions to guide users in writing feasible queries and extracting data from all sources that have the potential to contribute



User Domain Overview

- users have fine-grained control over which sources must contribute to the query
- users are informed of all sources contributing to the query
- the GUI expresses unions of queries visually, in a compact form, using when needed annotations

Schemas

Source A:
`srcA_Specialists(LastName, FirstName, Specialty)`
Source B:
`srcB_Specialists(LastName, FirstName, Specialty, Phone)`
Integrated:
`Specialists(LastName, FirstName, Specialty)`

Source Data Services

```
CREATE VIEW srcA_getSpecialists(LastName^b, FirstName^f, Specialty^f) AS
 SELECT s.FirstName, s.LastName, s.Specialty
 FROM srcA_Specialists s;
```

```
CREATE VIEW srcB_getSpecialists(LastName^f, FirstName^f, Specialty^b) AS
 SELECT s.FirstName, s.LastName, s.Specialty
 FROM srcB_Specialists s;
```

Mappings

```
CREATE VIEW Specialists1(LastName, FirstName, Specialty) AS
 SELECT FirstName, LastName, Specialty
 FROM srcA_Specialists;

CREATE VIEW Specialists2(LastName, FirstName, Specialty) AS
 SELECT FirstName, LastName, Specialty
 FROM srcB_Specialists;
```

Integrated Data Services

```
CREATE VIEW getSpecialists1(LastName^b, FirstName^f, Specialty^f) AS
 SELECT s.LastName, s.FirstName, s.Specialty
 FROM Specialists s;

CREATE VIEW getSpecialists2(LastName^f, FirstName^f, Specialty^b) AS
 SELECT s.LastName, s.FirstName, s.Specialty
 FROM Specialists s;
```

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

User Query

```
SELECT FirstName, LastName, Specialty
FROM Specialists
WHERE LastName='Smith';
```

Query is Feasible!

The integrated service $\texttt{getSpecialists}_1$ can be used to answer the query. Using Yerneni's approach (Example #3), the integrated service for Specialists would require both LastName and Specialty to be bound.

Source Schema

```
srcA_Doctors(LastName, FirstName, Specialty, Address)
srcA_Hospitals(Hospital, Address)
```

Source Data Services

CREATE VIEW srcA_getAll(LastNm^f, FirstNm^f, Spec^f, Hosp^b, Addr^f) AS

```
 SELECT d.LastName, d.FirstName, d.Specialty, h.Hospital, h.Address
 FROM srcA_Doctors d, srcA_Hospitals h
 WHERE d.Address=h.Address;
```

Integrated Schema

```
Physicians(LastName, FirstName, Specialty, Address)
Hospitals(Hospital, Address)
```

Mappings

```
CREATE VIEW Physicians1(LastName, FirstName, Specialty, Address) AS
 SELECT d.LastName, d.FirstName, d.Specialty, d.Address
 FROM srcA_Doctors d;

CREATE VIEW Hospitals1(Hospital, Address) AS
 SELECT h.Hospital, h.Address
 FROM srcA_Hospitals h;
```

Integrated Data Services

CREATE VIEW getAll(LastNm^f, FirstNm^f, Spec^f, Hosp^b, Addr^f) AS

```
 SELECT p.LastName, p.FirstName, p.Specialty, h.Hospital, h.Address
 FROM Physicians p, Hospitals h
 WHERE p.Address=h.Address;
```

User Query

```
SELECT p.LastName, p.FirstName, p.Specialty
FROM Physicians p, Hospitals h
WHERE p.Address=h.Address AND h.Hospital='Mercy Hospital';
```

User Query

```
SELECT p.LastName, p.FirstName, p.Specialty
FROM Physicians p, Hospitals h
WHERE p.Address=h.Address AND h.Hospital='Mercy Hospital';
```

Query is Feasible!

The integrated service `getAll` can answer queries that join Physicians and Hospitals given that the Hospital capability is satisfied. Yerneni's approach would only support one mapping on Hospitals, so no query would be able to retrieve Physicians data.

Anatomy of the integrated services

- every integrated service must have an equivalent rewriting using the source data services
- if not, they would never produce executable query plans!
- further, every integrated service has an equivalent rewriting using the mappings (why?)
- so we are looking for queries that can be expressed equivalently using source data services and mappings, or bifeasible queries
- in the search process we should be the least restrictive possible to avoid excluding valid services

Anatomy of the integrated services

- the idea is to start from an initial set of queries, and compute syntactic extensions for each of them, while testing for bifeasibility
- boolean queries against the mappings and/or source data services are a good initial set of queries since these are the least restrictive queries that could be bifeasible
- for syntactic extensions, we apply maximally-contained rewritings using source data services and mappings- intuitively, this guarantees that candidate services are always expressible using either source data services or mappings

Let's see a step-by-step computation using the Example #5 scenario.

Let q be the initial query against Physicians$_1$:

```
SELECT {}
FROM Physicians1 d;
```

Let q be the initial query against Physicians$_1$:

```
SELECT {}
FROM Physicians₁ d;
```

Let q_0 be the query obtained by unfolding q:

```
SELECT {}
FROM srcA_Doctors d;
```

Let q be the initial query against Physicians$_1$:

```
SELECT {}
FROM Physicians₁ d;
```

Let q_0 be the query obtained by unfolding q:

```
SELECT {}
FROM srcA_Doctors d;
```

Maximally-contained rewritings using the source services:

```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

Let q be the initial query against Physicians$_1$:

```
SELECT {}
FROM Physicians_1 d;
```

Let q_0 be the query obtained by unfolding q:

```
SELECT {}
FROM srcA_Doctors d;
```

Maximally-contained rewritings using the source services:

```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

which unfolds to query $q^m_{0,1}$ below:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Let q be the initial query against Physicians$_1$:

```
SELECT {}
FROM Physicians₁ d;
```

Let q_0 be the query obtained by unfolding q:

```
SELECT {}
FROM srcA_Doctors d;
```

Maximally-contained rewritings using the source services:

```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

which unfolds to query $q^m_{0,1}$ below:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Since $q^m_{0,1} \neq q_0$, $q^m_{0,1}$ becomes a candidate for the next iteration.

Let $q_1 = q^m_{0,1}$ be the candidate query for this iteration:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Let $q_1 = q^m_{0,1}$ be the candidate query for this iteration:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Maximally-contained rewriting using the mappings:

```
SELECT {}
FROM Physicians₁ p, Hospitals₁ h
WHERE p.Address=h.Address;
```

Let $q_1 = q^m_{0,1}$ be the candidate query for this iteration:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Maximally-contained rewriting using the mappings:

```
SELECT {}
FROM Physicians1 p, Hospitals1 h
WHERE p.Address=h.Address;
```

which unfolds to query $q^m_{1,1}$ below:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Let $q_1 = q^m_{0,1}$ be the candidate query for this iteration:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Maximally-contained rewriting using the mappings:

```
SELECT {}
FROM Physicians_1 p, Hospitals_1 h
WHERE p.Address=h.Address;
```

which unfolds to query $q^m_{1,1}$ below:

```
SELECT {}
FROM srcA_Doctors d, srcA_Hospitals h
WHERE d.Address=h.Address;
```

Since $q^m_{1,1} \equiv q_1$, $q^m_{1,1}$ is an (expanded) integrated service.

Rewrite $q^{m}_{1,1}$ using the mappings to obtain q^{M}_{1}:

```
SELECT {}
FROM Physicians₁ p, Hospitals₁ h
WHERE p.Address=h.Address;
```

Rewrite $q_{1,1}^{m}$ using the mappings to obtain q_1^M:

```
SELECT {}
FROM Physicians_1 p, Hospitals_1 h
WHERE p.Address=h.Address;
```

Rewrite $q_{1,1}^{m}$ using the data services to obtain q_1^S:

```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

Rewrite $q_{1,1}^{m}$ using the mappings to obtain q_1^M:

```
SELECT {}
FROM Physicians₁ p, Hospitals₁ h
WHERE p.Address=h.Address;
```

Rewrite $q_{1,1}^{m}$ using the data services to obtain q_1^S:

```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

The integrated service is the query obtained from q_1^M by projecting the maximum projection list of q_1^M and q_1^S:

```
SELECT p.LastName, p.FirstName, p.Specialty, h.Hospital, h.Address
FROM Physicians₁ p, Hospitals₁ h
WHERE p.Address=h.Address;
```

Rewrite $q^m_{1,1}$ using the mappings to obtain q^M_1:

```
SELECT {}
FROM Physicians1 p, Hospitals1 h
WHERE p.Address=h.Address;
```

Rewrite $q^m_{1,1}$ using the data services to obtain q^S_1:

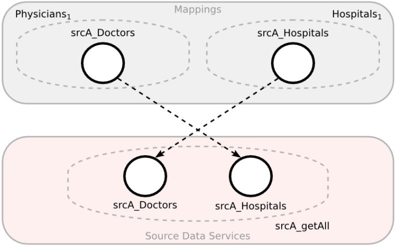
```
SELECT {}
FROM srcA_getAll(LastNm, FirstNm, Spec, Hosp, Addr) d;
```

The integrated service is the query obtained from q^M_1 by projecting the maximum projection list of q^M_1 and q^S_1:

```
SELECT p.LastName, p.FirstName, p.Specialty, h.Hospital, h.Address
FROM Physicians1 p, Hospitals1 h
WHERE p.Address=h.Address;
```

The capabilities for the integrated service are obtained from the algorithms in [Yerneni et al, 99].

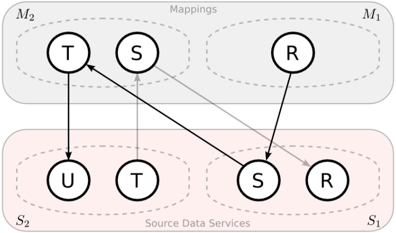
Relationship between Mappings and Source Data Services



- srcA_Doctors in Physicians_1 is covered by srcA_getAll and introduces srcA_Hospitals
- srcA_Hospitals in Hospitals_1 is covered by srcA_getAll and introduces srcA_Doctors

Let's see a more general picture...

Relationship between Mappings and Source Data Services



- rewriting M_1 using S_1 introduces subgoal S
- rewriting S_1 using M_1 and M_2 introduces subgoal T
- rewriting M_2 using S_1 and S_2 introduces subgoals U and R
- and so on...

Let's formalize this...

Bifeasible Query

Let V_1 and V_2 be two sets of views against a schema L, and q a query against L. Query q is bifeasible w.r.t. V_1 and V_2 if there are queries q_1 against V_1 and q_2 against V_2 s.t. the expansions q_1' and q_2' (of q_1 and q_2, respectively) are equivalent to q.

Maximally-Contained Bifeasible Query

Let V_1 and V_2 be two sets of views against a schema L, and q a query against L. Query q' is maximally-contained in q and bifeasible w.r.t. V_1 and V_2 if:

- q' is bifeasible w.r.t. to V_1 and V_2, and
- for every q'' s.t. $q' \sqsubseteq q'' \sqsubseteq q$, one of the following holds:
  a) q'' is not bifeasible w.r.t. V_1 and V_2, or
  b) $q'' \equiv q'$.

Set of Maximally-Contained Bifeasible Queries

Let V_1 and V_2 be two sets of views against a schema L, and Q the set of queries with empty projection lists obtained from $V_1 \cup V_2$. Q^B is the set of maximally-contained bifeasible queries of all queries $q \in Q$ w.r.t. V_1 and V_2 if, for every $q \in Q$, $q' \in Q^B$ if and only if q' is maximally-contained in q and bifeasible w.r.t. to V_1 and V_2.

What are the integrated services?

- views that capture capabilities of mappings and underlying sources using the vocabulary of the integrated schema
- expressed as $\mathrm{CQ}^{=p}$ queries against the integrated schema
- in particular, they are the set of maximally-contained bifeasible queries for the source data services (S) and mappings (M)
- if a user query is feasible w.r.t. the integrated services, then it is feasible

Algorithm Sketch (no capabilities)

- start the computation with a (candidate) query q_0 from M
- compute the maximally-contained rewritings of q_0 using S
- if q_0 has an equivalent rewriting, it is bifeasible
- if q_0 has no rewritings, it cannot yield bifeasible queries
- otherwise
  - q_0 has some maximally-contained rewriting q_0^m
  - any bifeasible $q_0^e \sqsubset q_0$ has an equivalent rewriting using S
  - a minimal extension of q_0 with this property is q_0^m
  - q_0^m becomes a candidate query for the next iteration
- the candidate queries for the next iteration have
  - equivalent rewritings using M
  - maximally-contained rewritings using S
- for the next iteration, we exchange S and M

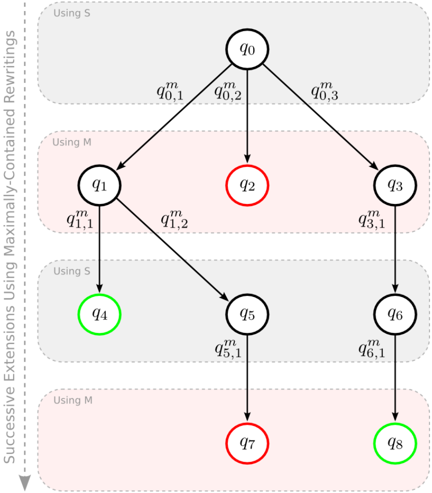
Algorithm 1: IntegratedServices($\mathcal{I}$)

$B \leftarrow \emptyset$
$W \leftarrow \emptyset$
$C \leftarrow$ queries in M with empty projection lists
Iterate(M, S, C, B)
for each $q \in B$ **do**
  for each $q_i \in EQ^M(q)$ **and** $q_j \in EQ^S(q)$ **do**
    $\overline{X_i} \leftarrow$ maximal projection list of q_i
    $\overline{Y_j} \leftarrow$ maximal projection list of q_j
    $W \leftarrow W \cup \{q_i(\overline{X_i} \cap \overline{Y_j})\}$
return minimized W

Algorithm 2: Iterate(V_1, V_2, C, B)

$C' \leftarrow \emptyset$
for each $q \in C$ **do**
  for each $q^{mc} \in MC^{V_2}(q)$ **do**
    $q^{exp} \leftarrow$ expansion of q^{mc} using V_2
    if $q \equiv q^{exp}$ **then**
      $B \leftarrow B \cup \{q^{exp}\}$
    else
      $C' \leftarrow C' \cup \{q^{exp}\}$
if $(C' \neq \emptyset)$ **then**
  Iterate(V_2, V_1, C', B)

Visualizing a Computation Thread

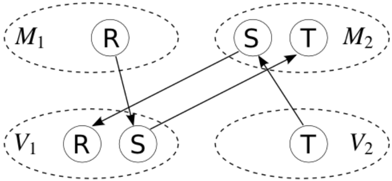


In order to use the above algorithm, certain restrictions apply to the input source data services (S) and mappings (M). In particular, these restrictions can be expressed in terms of a graph induced by S and M, which we call Views Graph.

Views Graph for S and M

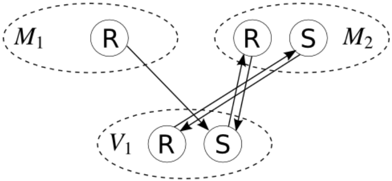
- directed, bipartite graph
- nodes represent subgoals of the services and mappings
- dashed edges represent an introduces relation with no fresh variables
- solid edges represent an introduces relation with fresh variables
- the views graph is weak acyclic if it has no cycle containing a solid edge

If the views graph is weak acyclic, the above algorithm terminates.



Views Graph

- fresh variables are introduced by all rewritings
- however, there are no cycles in the graph
- the algorithm terminates for these mappings and services



Views Graph

- fresh variables are introduced by some rewritings
- there are two cycles in the graph
- the algorithm will not terminate for these mappings and services

Open Problem: decidability of the computation of the set of maximally-contained bifeasible queries for V_1 and V_2.

If the problem turns out to be undecidable, no algorithm can solve the problem as stated.

Under the weak acyclicity restriction, our algorithm is guaranteed to terminate. It is sound by construction.

The completeness proof is currently under construction.

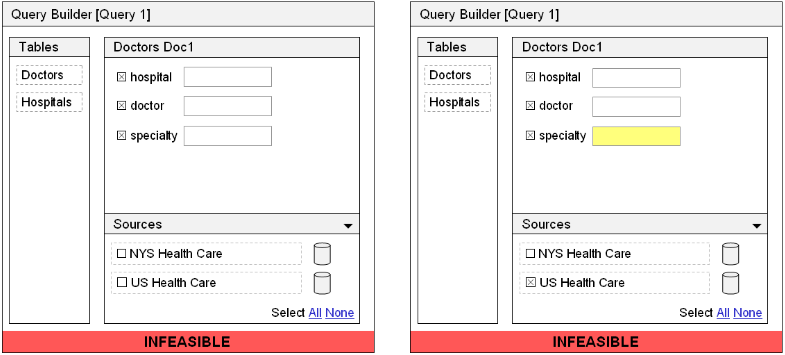
An alternative approach is to compute an upper bound on the size of the maximally-contained bifeasible queries, and let the algorithm run until it reaches this limit.

Show demo with a sample scenario.

Visual Query Interface

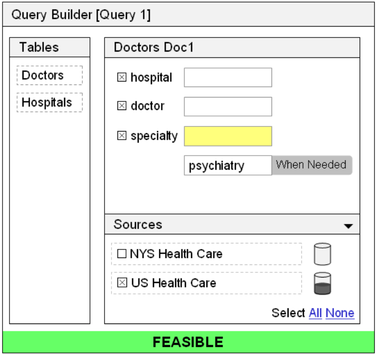
- the user formulates an inital, fixed query q
- the system displays current feasibility status
- for every table alias in q, the system displays a list of sources that can contribute
- sources selected by the user must contribute with the particular table in q
- suggestions are presented to the user as when needed actions
- required suggestions are necessary extensions to q
- optional suggestions offer a choice of alternate extensions to q

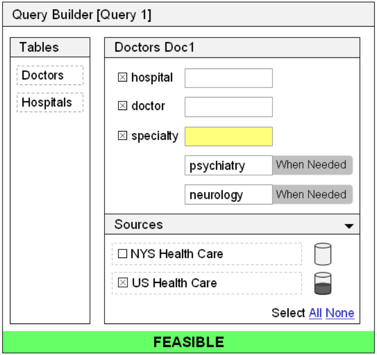
Let's see an example...



- query fixed
- no sources selected
- no source contribution
- no suggestions
- query infeasible

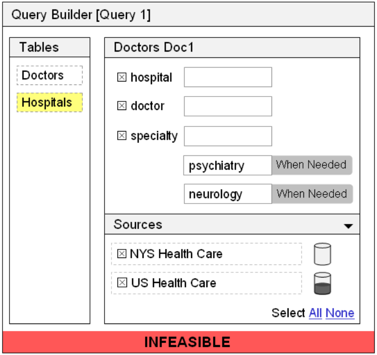
- source selected
- mandatory selection
- query infeasible
- no source contribution



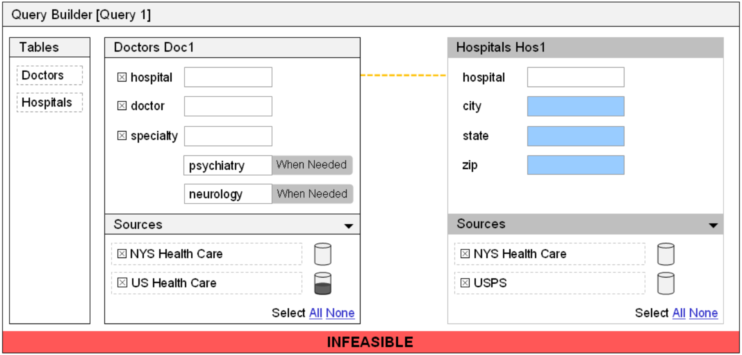


- selection applied
- query becomes feasible
- partial source contribution

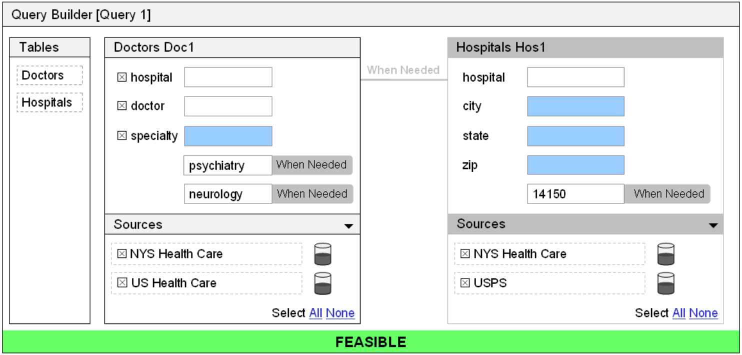
- new selection applied
- more data extracted from source
- intuition: when needed actions encode unions



- new source selected
- mandatory table action
- query infeasible



- new capabilities exposed
- mandatory join action
- at least one of the alternative selections
- query infeasible



- join and selection applied
- query feasible
- partial contribution from multiple sources
- several options to extract more data from sources

Intuition

- the user fixes a query q
- the middleware computes a set of maximally-contained rewritings for q using the integrated services
- a rewriting is feasible if it is equivalent to q
- a set of suggestions is computed for each strictly contained rewriting
- suggestions from all rewritings are combined and their required/optional attributes defined
- source annotations are processed against provenance information we compute for the rewritings
- this could possibly change query feasibility and filtering suggestions

Rewriting Suggestions

- let $\varphi : q \rightarrow r$ be the containment mapping from the fixed user query q to a maximally-contained rewriting r
- drop the rewriting if it does not satisfy some source annotation of q
- otherwise, compute the rewriting suggestions
- a subgoal not in the range of φ generates a table suggestion
- $\varphi(\texttt{var}) \rightarrow \texttt{const}$ generates a selection suggestion
- $\varphi(\texttt{var}_1) \rightarrow \texttt{var}_k$ and $\varphi(\texttt{var}_2) \rightarrow \texttt{var}_k$ generates a join suggestion

Combine Suggestions

- a suggestion is required if it is generated by all rewritings
- a suggestion is optional if it is generated by some but not all rewritings

Contributions

- Integration Architecture
- Visual Query Interface
- Prototypes