Querying complex heterogeneous data sets

Alex Poulovassilis

Information Management and Web Technologies Group
Co-Director of the London Knowledge Lab

Workshop on Data Integration and Querying for Nuclear Forensics, Imperial College, July 2010

Outline of the talk

1. Fine-grained data integration and querying of distributed data sources
2. Ontology-enabled data integration
3. Tracing data lineage in integrated data resources
4. Flexible querying of heterogeneous graph-structured data
5. Other related recent and ongoing work

More details can be found at www.dcs.bbk.ac.uk/~ap including copies of papers

1. Fine-grained data integration and querying of distributed data sources

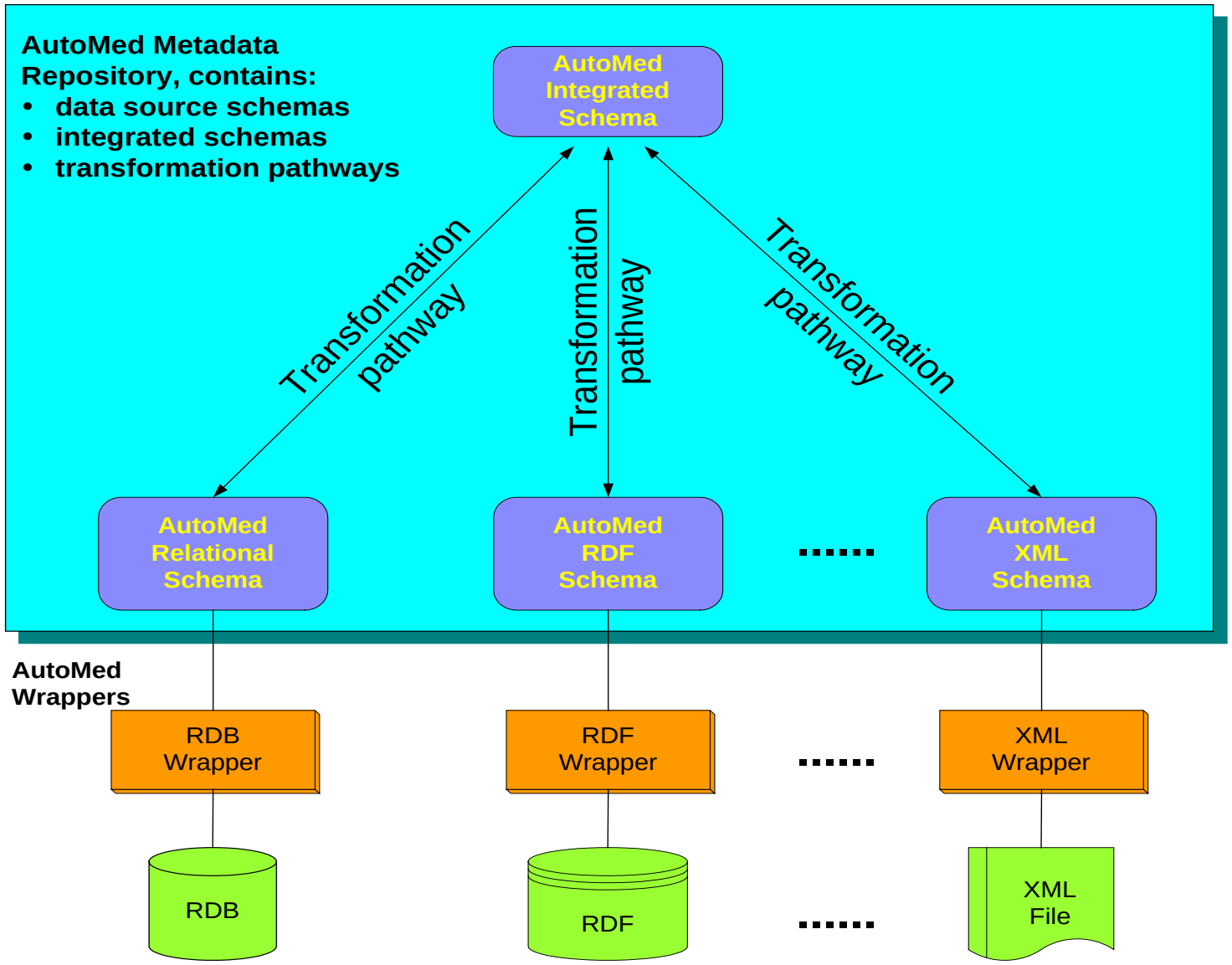
- ***Data integration*** is the process of creating an integrated resource which combines data from a number of data sources in order to support new queries and analyses
- The data sources may be different in their data model, query interfaces, query processing capabilities, database schema/data exchange format, data types used, nomenclature adopted etc.
- Integrating them to meet the needs of new users and applications requires reconciliation of this ***heterogeneity***
- ***Data sources may change their format and content*** without considering the impact on any integrated derived resources; so techniques are needed for maintaining the integrated resource in the face of changes in the data sources
- Integrated resources may themselves become data sources for higher-level integrations, resulting in a ***network of dependencies***

Fine-grained data integration and querying of distributed data sources – AutoMed

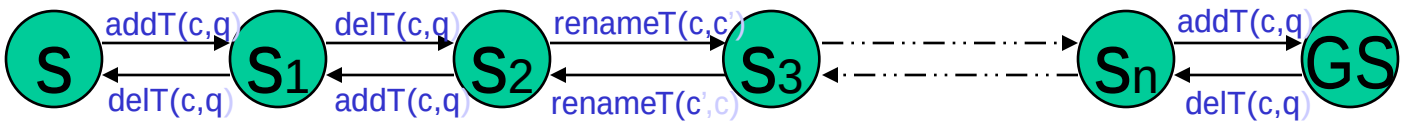
- Overview of the AutoMed data integration system:
  - Both-As-View (BAV) approach
  - Hypergraph-based meta model (HDM)
  - Transformations that transform both the schema and the data
  - Intermediate Query Language (IQL) for (i) specifying transformation queries (views) and (ii) querying a target transformed/integrated schema
  - Supports fine-grained data integration: individual attributes of data source schemas can be transformed and combined to form the extend of a new attribute of an entity in the integrated schema
  - Supports evolution of both source and target schemas
  - Allows networks of schemas – which may have data associated with them or be virtual – to be interconnected by transformation pathways

See McBrien and Poulovassilis, ICDE 2003 and other papers

AutoMed's Both-As-View Approach



Schema & Data Transformation Pathways



- addT / exendT (c, q);
- deleteT / contractT (c, q);
- renameT (c, c');

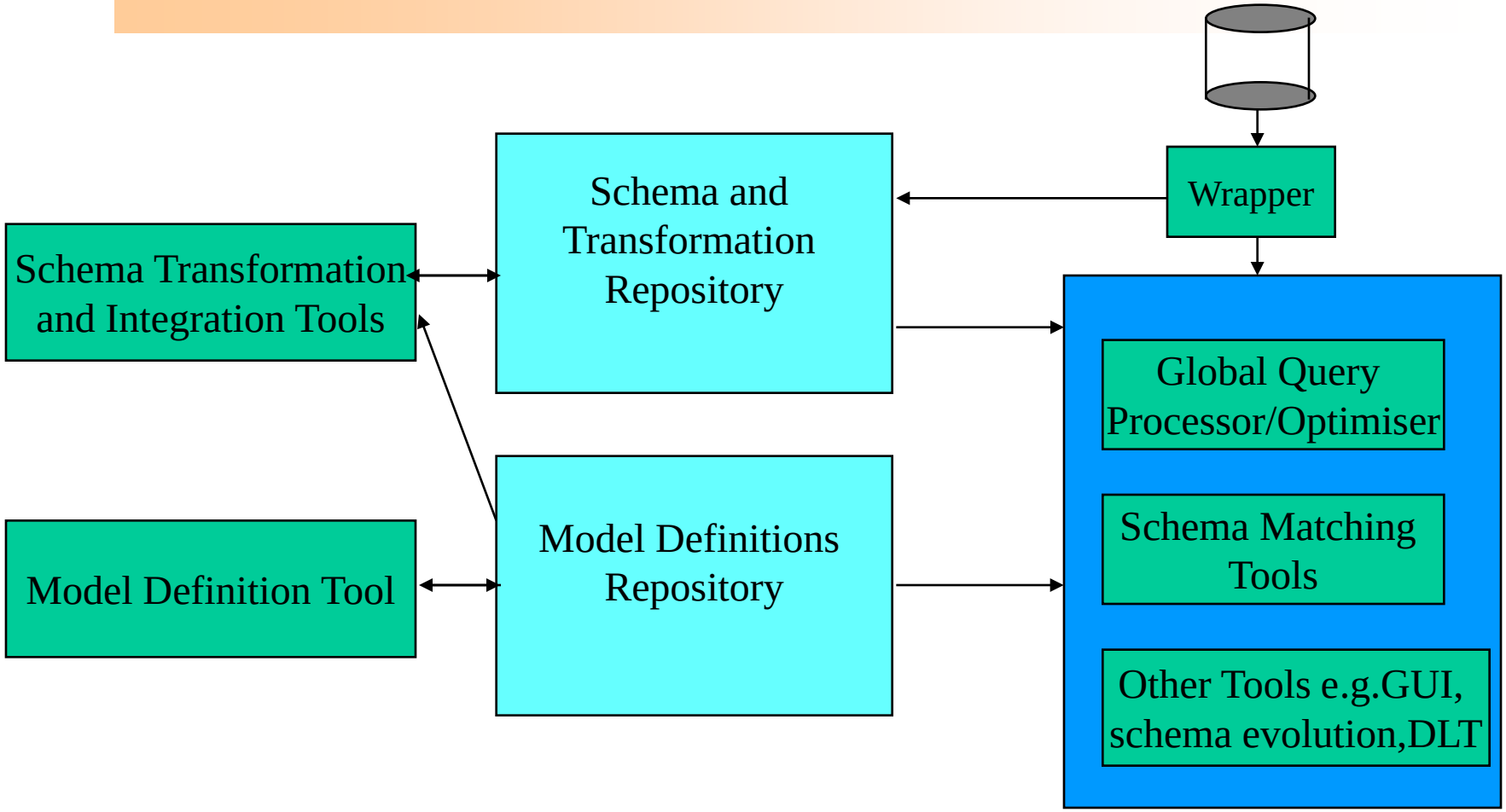
- For simple relational model, T can be Rel, Att, PKey, FKey
- For XML, T can be Element, Attribute, NestList
- For HDM, T can be Node, Edge, Constraint

IQL Query Language

IQL is a comprehensions-based functional query language. The syntax of IQL includes the following operators:

1. E1 ++ E2 ++ … ++ Er /* bag union*/
2. E1 -- E2 /* bag *monus* */
3. group E
   /* group a bag of pairs on their first component*/
4. sort E
5. distinct E /* remove duplicates*/
6. aggFun E
   /* aggFun = “max” | “min”
   |“count” | “sum” | “avg” */
7. gc aggFun E
   /*group and compute a bag of pairs on their first component and apply an aggregation function to the second component*/
8. [e| Q1; …; Qr] /*comprehension*/
9. map f E /*extended projection*/

AutoMed Architecture



Global Query Processing in AutoMed

- We handle query language heterogeneity by translation into/from IQL
- A query Q expressed in a high-level query language such as SQL on an integrated schema GS is first translated into IQL
- View definitions are then derived from the transformation pathways between GS and the data source schemas
- These view definitions are substituted into Q, reformulating it into an IQL query over source schema constructs
- Query optimisation then occurs
- One goal of this is to generate the largest possible sub-queries that can be submitted to data source Wrappers for translation into the data source query languages and evaluation by the data sources

Global Query Processing in AutoMed

- Query evaluation then follows, during which the AutoMed Evaluator submits to Wrappers sub-queries that they are able to translate into the data source query language

  (currently, AutoMed supports wrappers for SQL, OQL, XPath, XQuery and flat-file data sources)
- The Wrappers submit sub-queries to the data sources, and translate sub-query results back into the IQL type system
- The Evaluator then undertakes any further necessary query evaluation to combine sub-query results, and returns overall results to the user

Integrating AutoMed with OGSA-DAI and DQP

- Motivation:
  - Grid computing technologies enable distributed computational and data resources to be accessed in a service-based environment
  - This is particularly of value for applications requiring access to complex combinations of computational and data resources, as in the Life Sciences, for example
  - OGSA-DAI is an open-source, standard Grid data access middleware
  - OGSA-DQP provides distributed query processing capabilities over OGSA-DAI enabled data sources

Integrating AutoMed with OGSA-DAI and DQP

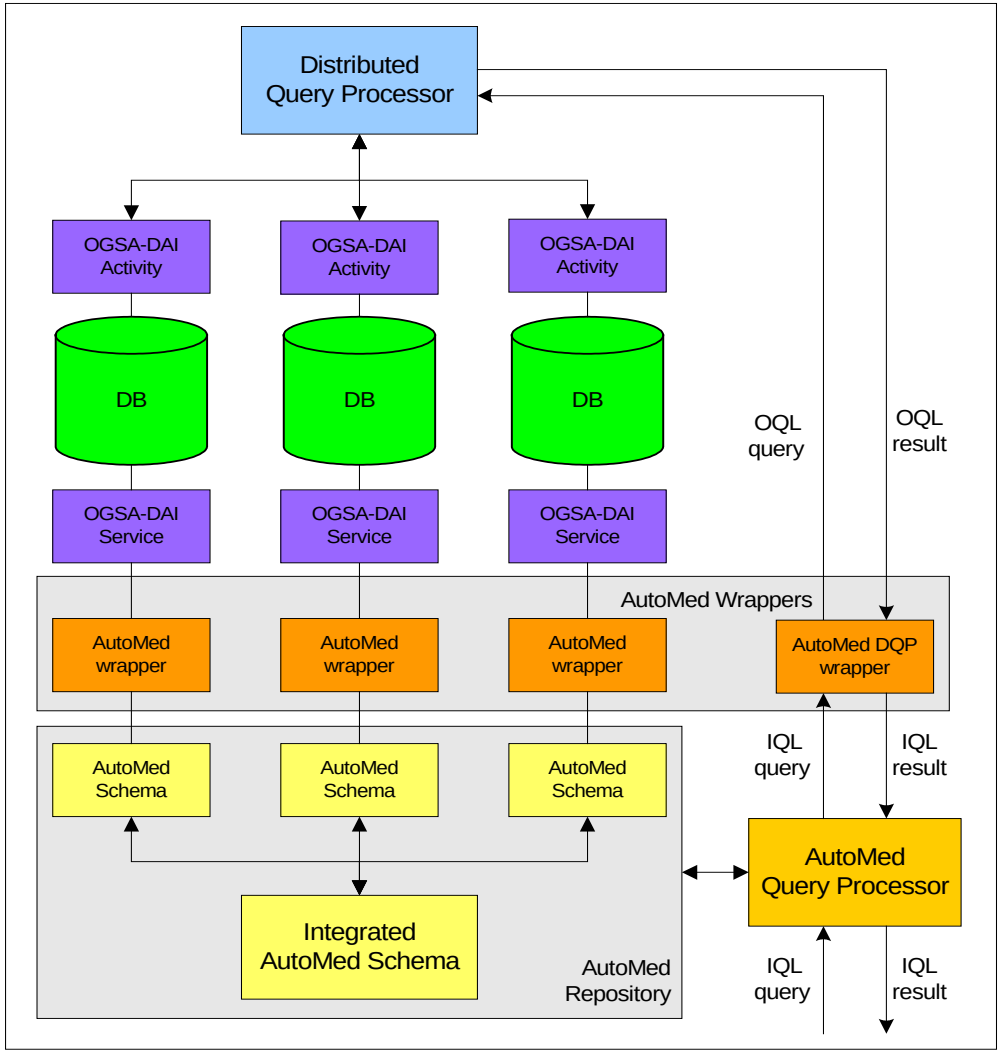
- Biological data sources are characterised by their ***high degree of heterogeneity***, in terms of data model, query interfaces, query processing capabilities, database schema or data exchange format, data types used, nomenclature adopted
- Hence, a promising area for investigation is the ***combination*** of Grid data access & distributed querying middleware and fine-grained data integration technologies
- We investigated this in the ISPIDER project, developing an architecture for virtual integration of Grid-enabled data sources that leverages the functionalities of AutoMed and the OGSA-DQP service-based distributed query processor

See:

Data access and integration in the ISPIDER proteomics Grid, Zamboulis et al., Proc. DILS 2006

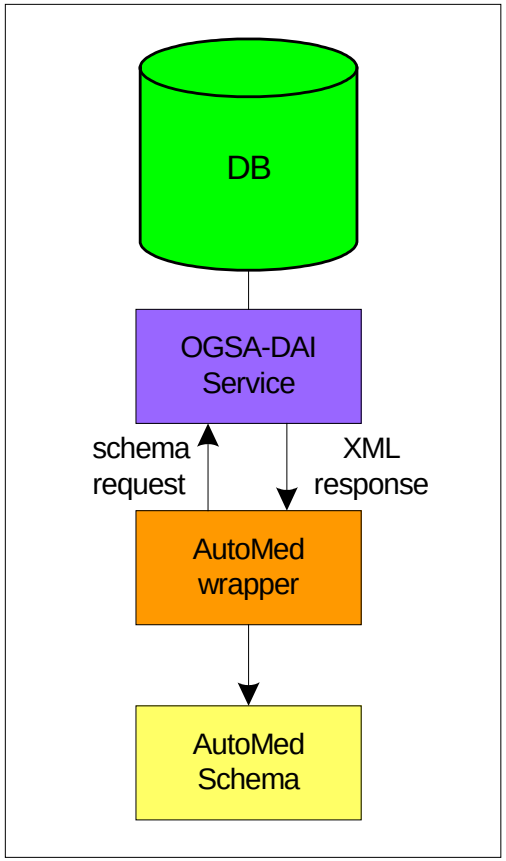
DAI/DQP/AutoMed Interoperability

- Data sources wrapped with OGSA-DAI
- AutoMed-DAI wrappers extract data sources' metadata
- Semantic integration of data sources using transformation pathways
- IQL queries submitted to an integrated schema are reformulated to IQL queries on the data sources, using the transformation pathways
- Subqueries are submitted to AutoMed-DQP wrapper for evaluation by DQP



The AutoMed-DAI Wrapper

- The AutoMed-DAI wrapper requests the schema of the data source using an OGSA-DAI service
- The service replies with the source schema encoded an in XML response document
- The AutoMed-DAI wrapper creates the corresponding schema in the AutoMed repository



The AutoMed-DQP Wrapper

- The AutoMed-DQP wrapper undertakes two tasks:
  - needs to inform AutoMed of the subset of IQL that it is capable of translating into OQL
  - is responsible for making interactions with OGSA-DQP transparent to the remainder of the AutoMed infrastructure
- On receiving an IQL query, the AutoMed-DQP wrapper first translates it into the equivalent OQL query
- The OQL query is then sent to OGSA-DQP for evaluation
- The reply from OGSA-DQP is in the form of an XML response document containing the query results
- The AutoMed-DQP wrapper translates these results into the IQL type system, and returns the result to AutoMed's evaluator for any further necessary evaluation

Experimental results

- Four autonomous proteomics databases were integrated using this architecture, as part of the ISPIDER project:
  - there are multiple schema-level and data-level modelling conflicts in the data sources, which needed to be resolved by transforming and combining the extents of individual attributes of the data source entities to form the extent of each attribute of each entity in the integrated schema
  - there were a mixture of large, community data sources (gpmDB, PepSeeker) and of smaller data sources produced by smaller groups of researchers (PEDRo, PRIDE)
  - there is an absence of commonly agreed identifiers for instances of the biological entities in the different data sources

Experimental results

- A set of biologically meaningful queries on the integrated resource were provided by our biologist partners at Manchester University
- Our experimental findings validated the design of our overall architecture showing that, as intended, it does indeed combine the respective advantages of AutoMed (fine-grained data transformation and integration) and OGSA-DQP (Grid-based distributed query processing)
- The AutoMed Query Processor is able to adequately handle all stages of query processing for SPU queries, plus also joins within data sources, but the OGSA-DQP distributed query processing capabilities are necessary in order to efficiently evaluate queries which contain joins between data sources
- Our architecture makes very few assumptions about its components and these perform functionality that, in principle, could be provided by a variety of other systems/products.

Experimental results

- A key general finding from our experiments is that it is indeed feasible to perform fine-grained data transformation/integration in the context of heterogeneous Grid data sources and achieve acceptable query performance, using our proposed architectural framework
- In the context of SPJ global queries and SPJU data transformation queries, we found that applying just four well-known equivalences within the data integration system's query optimiser is sufficient to provide acceptable query performance in conjunction with OGSA-DQP's query optimisation and query planning capabilities
- Also, as expected, supporting parallel and incremental evaluation within the data integration system can offer significant advantages in terms of overall query performance

See: Query Performance Evaluation of an Architecture for Fine-Grained Integration of Heterogeneous Grid Data Sources, Zamboulis, Martin, Poulovassilis; to appear in Future Generation Computer Systems, 2010

2. Ontology-enabled data integration

- Motivation:
  - When creating a virtual integrated resource from a number of data sources, the integrated schema may be defined using a standard data modelling language, or it may be a source-independent ontology defined in an ontology language
  - The latter has the advantage of describing the domain at a high-level of abstraction, using possibly already existing ontologies and separating users' knowledge of the domain from structural details of the source data
  - It also allows domain knowledge to be expressed using logical formalisms, enabling inference mechanisms to be applied, and ultimately converting the integrated resource into a knowledge base

.

Ontology-enabled data integration

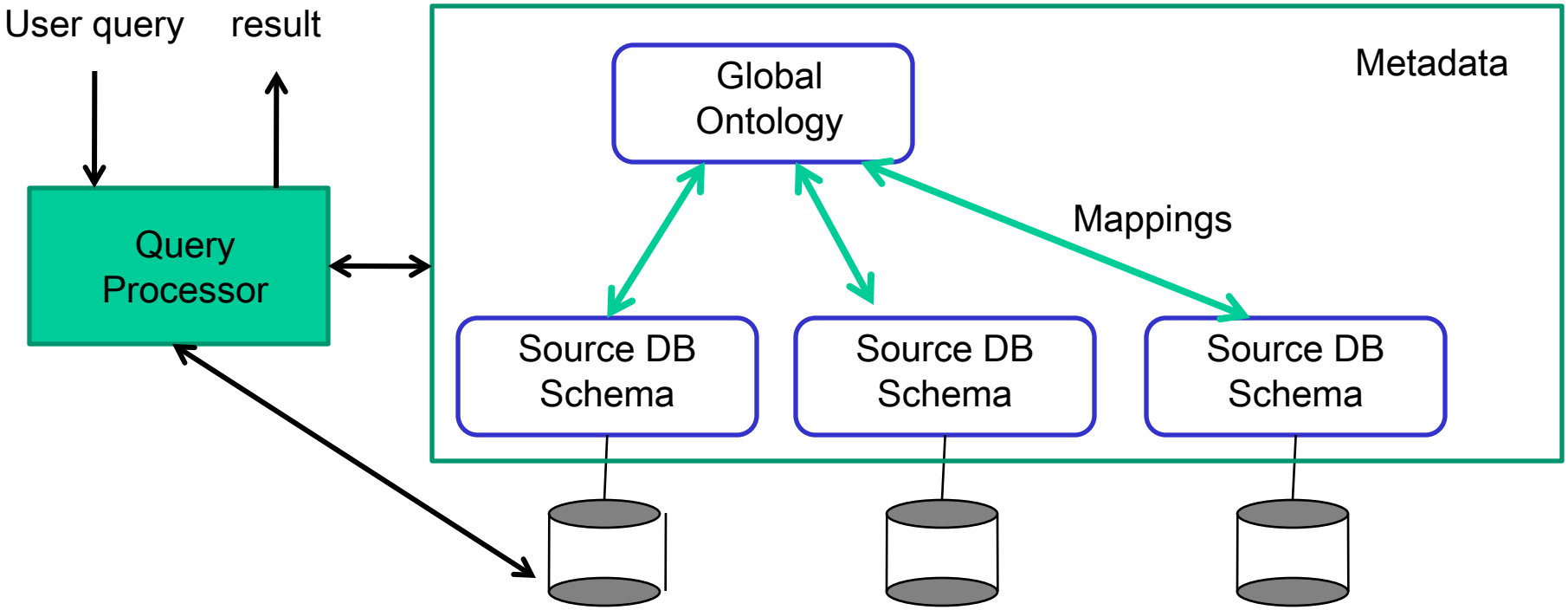
- The ASSIST project investigated cross-fertilisation between schema-based and ontology-based DI
- In particular, it focussed on the use of ontologies to enhance schema-based approaches, firstly as a means of providing richer semantics for schema constructs and secondly as a means of enabling schema-based DI approaches to be used together with ontology-based ones

See:

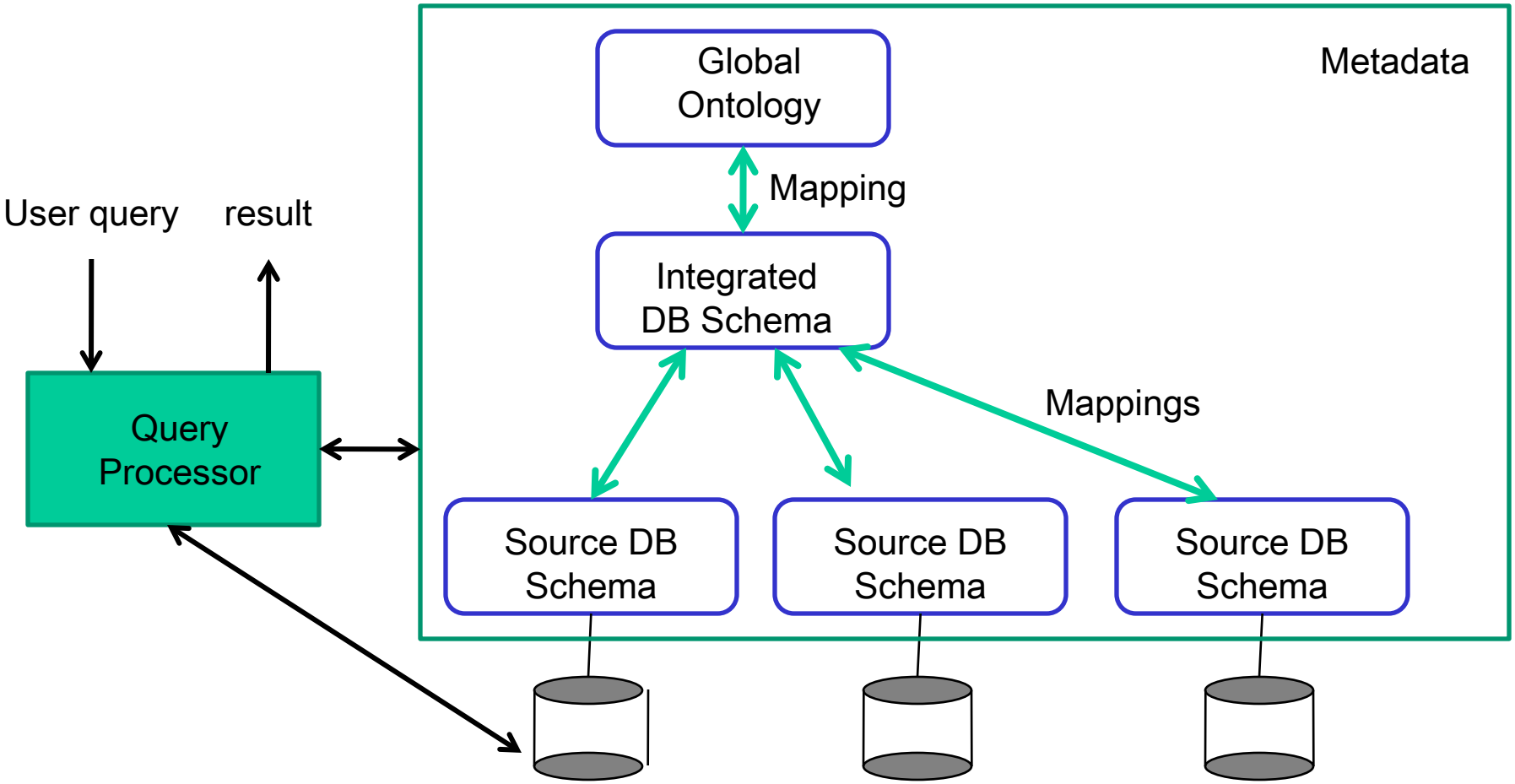
(1) Zamboulis, Poulovassilis, Wang, Ontology-Assisted Data Transformation and Integration, Proc. ODBIS 2008

(2) Zamboulis, Poulovassilis, Roussos, Flexible Data Integration and Ontology-Based Data Access to Medical Records, Proc. BIBE 2008

Ontology-based access to multiple databases, traditional approach



Ontology-based access to an Integrated Virtual Database Resource, as investigated in ASSIST



ASSIST

- The ASSIST project aims to facilitate cervical cancer research by integrating several medical databases containing information about patients and examinations they have undergone
- There is an existing domain ontology, expressed in OWL-DL, together with an additional set of medical rules expressed in EL++
- AutoMed is used to define the mappings shown in the diagram above i.e.
  - between the ontology and the (virtual) integrated database schema, and
  - between this integrated database schema and the data source schemas
- User queries are expressed with respect to the ontology
- Queries are first expanded according to the medical rules
- AutoMed is then used to evaluate the expanded queries

ASSIST Integration Strategy

- Step 1: integrate the data source schemas into a relational schema, R1; however, R1 may contain information that is not encompassed by the ontology
- Step 2: transform R1 into a relational schema R2, which
  - drops information from R1 that is not in the ontology; and
  - includes new concepts aligned with those of the ontology and defined in terms of concepts of R1
- Step 3: automatically translate R2 into OWL-DL
  - using a relational-to-RDF algorithm (Lausen, ODBIS'07)
- Step 4: transform this OWL-DL intermediate ontology into to the final ASSIST OWL-DL domain ontology

Advantages of this architecture

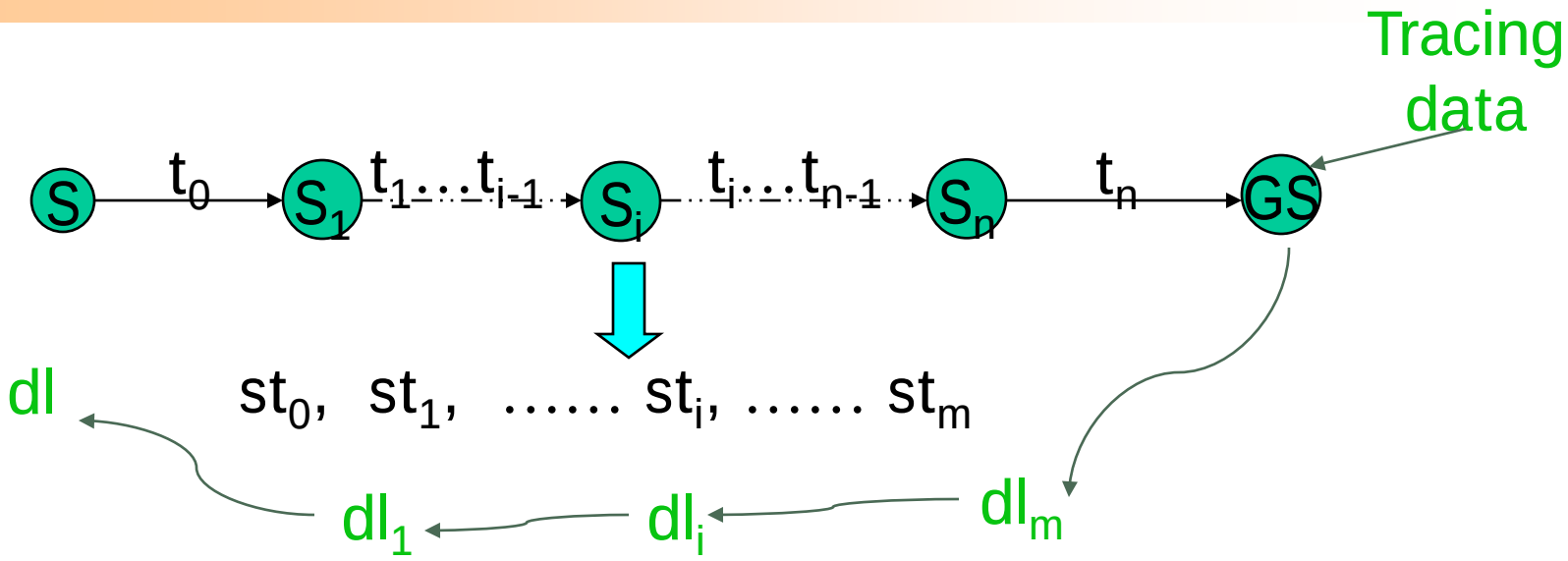
- Leverages both ontology-based query answering techniques, and schema-based distributed query processing capabilities over virtual integrated database schemas
- Provides increased maintainability of the integrated resource:
  - If the global ontology changes, we only need to amend the pathway from the integrated schema to the global ontology
  - If a data source schema changes, or is added or dropped, we address first the pathway from it to the integrated schema; if the latter changes as a result, then we address the impact on the global ontology as a separate second step

3. Tracing data lineage in transformed/ integrated data resources

- Motivation:
  - With data resources – virtual or materialised – that result from transforming/integrating heterogeneous data sources, it is sometimes necessary to trace how information in the integrated resource has been derived from the individual data sources:
    - from which data items in which data sources is an item of integrated data derived (its “origin-pool”) ?
    - which data items in the data sources have some influence on the integrated data item (its “affect-pool”) ?

- This can be extended to investigating how the results of a query, Q, submitted to a transformed/integrated schema, S, have been derived:
  - treat Q as defining a new derived construct c in a new schema S’ = S U c, where the extent of c is defined by Q

Tracing data lineage through BAV pathways



- Lineage data structure storing lineage data
- DLT formulae for each simple IQL query
- DLT procedure for a single simple step
- DLT procedure for a transformation pathway

See Fan, Poulovassilis, Using Schema Transformation Pathways for Data Lineage Tracing, Proc. BNCOD 2005; Fan, Data lineage tracing in data warehousing environments, Proc. BNCOD 2007

The Lineage Data Structure

- Lineage data structure:
  - the schema construct
  - isVirtual?
  - materialised lineage data
  - structure of virtual lineage data
  - constraint on virtual lineage data

E.g.

D|[{5,1},{5,2}] is an example of materialised lineage data for construct D

D|({x,y}, x=5) is an example of virtual lineage data

Affect-pool DLT formulae for MtMs (Materialised tracing data & Materialised schema data)

v	$DL(t)$
group D	$[\{x,y\}\|\{x,y\} \leftarrow D; x = \overline{a}]$
sort D	$D\|t$
distinct D	$D\|t$
aggFun D	D
gc aggFun D	$[\{x,y\}\|\{x,y\} \leftarrow D; x = \overline{a}]$
D_1 ++ D_2 ++ ... ++ D_n	$\forall i. D_i\|t$
D_1 -- D_2	$D_1\|t, D_2$
$[\overline{x}\|\overline{x_1} \leftarrow D_1; \ldots; \overline{x_n} \leftarrow D_n; C]$	$\forall i. [\overline{x_i}\|\overline{x_i} \leftarrow D_i; \overline{x_i} = ((\lambda \overline{x}.\overline{x_i})\ t)]$
$[\overline{x}\|\overline{x} \leftarrow D_1;$ member $D_2\ \overline{y}]$	$D_1\|t, [\overline{y}\|\overline{y} \leftarrow D_2; \overline{y} = ((\lambda \overline{x}.\overline{y})\ t)]$
$[\overline{x}\|\overline{x} \leftarrow D_1;$ not(member $D_2\ \overline{y})]$	$D_1\|t, D_2$
map $(\lambda \overline{x}.e)$ D	$[\overline{x}\|\overline{x} \leftarrow D, e = t]$

Table 1. DLT Formulae for MtMs

Affect-pool DLT formulae for MtVs (Materialised tracing data & Virtual schema data)

v	$DL(t)$
group D	$D\|(\{x,y\}, x = \overline{a})$
sort D	$D\|t$
distinct D	$D\|t$
aggFun D	$D\|(any, true)$
gc aggFun D	$D\|(\{x,y\}, x = \overline{a})$
D_1 ++ D_2 ++ ... ++ D_n	$\forall i. D_i\|t$
D_1 −− D_2	$D_1\|t, D_2\|(any, true)$
$[\overline{x}\|\overline{x_1} \leftarrow D_1; \ldots; \overline{x_n} \leftarrow D_n; C]$	$\forall i. D_i\|(\overline{x_i}, \overline{x_i} = ((\lambda\overline{x}.\overline{x_i})\ t))$
$[\overline{x}\|\overline{x} \leftarrow D_1;$ member $D_2\ \overline{y}]$	$D_1\|t, D_2\|(\overline{y}, \overline{y} = ((\lambda\overline{x}.\overline{y})\ t))$
$[\overline{x}\|\overline{x} \leftarrow D_1;$ not(member $D_2\ \overline{y})]$	$D_1\|t, D_2\|(any, true)$
map $(\lambda\overline{x}.e)$ D	$D\|(\overline{x}, e = t)$

Table 2. DLT Formulae for MtVs

Affect-pool DLT formulae for VtMs

Three kinds of virtual tracing data may arise:
(any,true), ({x,y}, x=a), (x, e=t)

v	$DL(t)$
group D	$[\{x,y\}\,\vert\,\{x,y\} \leftarrow D; member\ [first\ \overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t]\ x]$
sort D	$[\overline{x}\,\vert\,\overline{x} \leftarrow D; e=t]$
distinct D	$[\overline{x}\,\vert\,\overline{x} \leftarrow D; e=t]$
aggFun D	D
gc aggFun D	$[\{x,y\}\,\vert\,\{x,y\} \leftarrow D; member\ [first\ \overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t]\ x]$
$\mathtt{D}_1$ ++ $\mathtt{D}_2$ ++ ... ++ $\mathtt{D}_n$	$\forall i.[\overline{x}\,\vert\,\overline{x} \leftarrow D_i; e=t]$
$\mathtt{D}_1$ -- $\mathtt{D}_2$	$D_1\,\vert\,[\overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t], D_2$
$[\overline{\mathtt{x}}\,\vert\,\overline{\mathtt{x}_1} \leftarrow \mathtt{D}_1; \ldots; \overline{\mathtt{x}_n} \leftarrow \mathtt{D}_n; \mathtt{C}]$	$\forall i.[\overline{x_i}\,\vert\,\overline{x_i} \leftarrow D_i;$ $member\ (map\ (\lambda\overline{x}.\overline{x_i})\ [\overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t])\ \overline{x_i}]$
$[\overline{\mathtt{x}}\,\vert\,\overline{\mathtt{x}} \leftarrow \mathtt{D}_1; \mathtt{member}\ \mathtt{D}_2\ \overline{\mathtt{y}}]$	$[\overline{x}\,\vert\,\overline{x} \leftarrow D_1; member\ D_2\ \overline{y}; e=t],$ $[\overline{y}\,\vert\,\overline{y} \leftarrow D_2; member\ (map\ (\lambda\overline{x}.y)\ [\overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t])\ \overline{y}]$
$[\overline{\mathtt{x}}\,\vert\,\overline{\mathtt{x}} \leftarrow \mathtt{D}_1; \mathtt{not}(\mathtt{member}\ \mathtt{D}_2\ \overline{\mathtt{y}})]$	$D_1\,\vert\,[\overline{x}\,\vert\,\overline{x} \leftarrow \mathtt{v}; e=t], D_2$
map $(\lambda\overline{\mathtt{x}_1}.\mathtt{e}_1)$ D	$[\overline{x_1}\,\vert\,\overline{x_1} \leftarrow D; e=t]$

Table 3. DLT Formulae for VtMs with tracing data $(\overline{x}, e=t)$

DLT formulae for VtVs

- Similar to VtMs formulae, though in this case the schema data is unavailable
- Lineage would be untraceable if the virtual view itself, v, were used in the formulae
- So we use AutoMed's Global Query Processor (GQP) to materialise v
- Materialised tracing data can then be recovered and the situation reverts to MtVs earlier

DLT procedure for one transformation step

```
Proc DLT4AStep(td, ts)
{
  lpList = ∅;
  case MtMs:
    lpList ←DTL formulae for MtMs;
  case MtVs:
    lpList ←DTL formulae for MtVs;
  case VtMs:
    if (ts.result is required)
        //*recovering ts.result
      mv ← evaluate(ts.query);
        //*recovering td
      td.data ← mv:td.data;
      lpList ←DTL formulae for MtMs;
    else
      lpList ←DTL formulae for VtMs;
  case VtVs:
    if (td must be materialised)
        //*recovering ts.result
      mv ← GQP(ts.result);
        //*recovering td
      td.data ← mv:td.data;
      lpList ←DTL formulae for MtVs;
    else
      lpList ←DTL formulae for VtVs;
  return lpList;
}
```

DLT procedure for a transformation pathway

```
Proc oneDLT4APath(td, [ts_1, ..., ts_n])
{
  lpList = ∅;
  for i = n downto 1, do
    if (td.construct = ts_i.result)
      Num = i;
      lpList = DLT4AStep(td, ts_i);
      continue; //* End the for loop
  restTP = [ts_1, ..., ts_Num];
  return listDLT4APath(lpList, restTP);
}

Proc listDLT4APath([td_1, ..., td_m], [ts_1, ..., ts_n])
{
  lpList = ∅;
  for i = 1 to m, do
    lpList = merge(lpList,
                   oneDLT4APath(td_i, [ts_1, ..., ts_n]));
  return lpList;
}
```

Overall complexity is O(*n* x *m*), *n* being the number of add/extend transformations in the pathway and *m* the number of schema constructs referenced in the pathway

Example

$$
\begin{array}{ll}
\texttt{v1} & = [\{\texttt{'IS'},\texttt{k1},\texttt{k2},\texttt{x}\} \mid \{\texttt{k1},\texttt{k2},\texttt{x}\} \leftarrow \langle\!\langle \text{IStab}, \text{Mark} \rangle\!\rangle] \\
\texttt{v2} & = [\{\texttt{'MA'},\texttt{k1},\texttt{k2},\texttt{x}\} \mid \{\texttt{k1},\texttt{k2},\texttt{x}\} \leftarrow \langle\!\langle \text{MAtab}, \text{Mark} \rangle\!\rangle] \\
\langle\!\langle \text{Details}, \text{Mark} \rangle\!\rangle & = \texttt{v1} \mathbin{+\!\!+} \texttt{v2} \\
\texttt{v3} & = \texttt{map}\ (\lambda\{\texttt{k},\texttt{k1},\texttt{k2},\texttt{x}\}.\{\{\texttt{k},\texttt{k1}\},\texttt{x}\})\ \langle\!\langle \text{Details}, \text{Mark} \rangle\!\rangle \\
\texttt{v4} & = \texttt{gc avg v3} \\
\langle\!\langle \text{CourseSum}, \text{Avg} \rangle\!\rangle & = \texttt{map}\ (\lambda\{\{\texttt{x},\texttt{y}\},\texttt{z}\}.\{\texttt{x},\texttt{y},\texttt{z}\})\ \texttt{v4}
\end{array}
$$

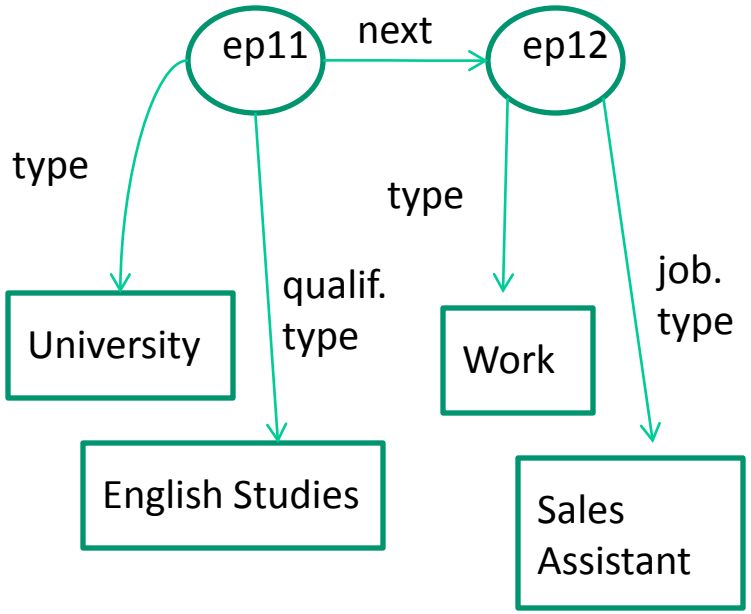
$$
\begin{array}{ll}
 & td = \langle\!\langle \text{CourseSum}, \text{Avg} \rangle\!\rangle\,|\{\texttt{'MA'},\texttt{'MAC01'},81\} \\
\overset{\text{MtVs}}{\Longrightarrow} & \texttt{v4}\,|\texttt{d1} = \texttt{v4}|\{\{\texttt{'MA'},\texttt{'MAC01'}\},81\} \\
\overset{\text{MtVs}}{\Longrightarrow} & \texttt{v3}\,|\texttt{d1} = \texttt{v3}|(\{\texttt{x},\texttt{y}\},\texttt{x}=\{\texttt{'MA'},\texttt{'MAC01'}\}) \\
\overset{\text{VtVs}}{\Longrightarrow} & \langle\!\langle \text{Details}, \text{Mark} \rangle\!\rangle|\texttt{d1} = \langle\!\langle \text{Details}, \text{Mark} \rangle\!\rangle|(\{\texttt{k},\texttt{k1},\texttt{k2},\texttt{x}\},\{\texttt{k='MA'};\texttt{k1='MAC01'}\}) \\
\overset{\text{VtVs}}{\Longrightarrow} & \texttt{v2}\,|\texttt{d1} = \texttt{v2}|(\{\texttt{k},\texttt{k1},\texttt{k2},\texttt{x}\},\{\texttt{k='MA'};\texttt{k1='MAC01'}\}), \\
 & \texttt{v1}\,|\texttt{d1} = \texttt{v1}|(\{\texttt{k},\texttt{k1},\texttt{k2},\texttt{x}\},\{\texttt{k='MA'};\texttt{k1='MAC01'}\}) \\
\overset{\text{VtMs}}{\Longrightarrow} & \langle\!\langle \text{MAtab}, \text{Mark} \rangle\!\rangle|\texttt{d1} = \langle\!\langle \text{MAtab}, \text{Mark} \rangle\!\rangle|(\{\texttt{k1},\texttt{k2},\texttt{x}\},\{\texttt{'MA'='MA'};\texttt{k1='MAC01'}\}) \\
 & \langle\!\langle \text{IStab}, \text{Mark} \rangle\!\rangle|\texttt{d1} = \langle\!\langle \text{IStab}, \text{Mark} \rangle\!\rangle|(\{\texttt{k1},\texttt{k2},\texttt{x}\},\{\texttt{'IS'='MA'};\texttt{k1='MAC01'}\})
\end{array}
$$

4. Flexible querying of heterogeneous graph-structured data

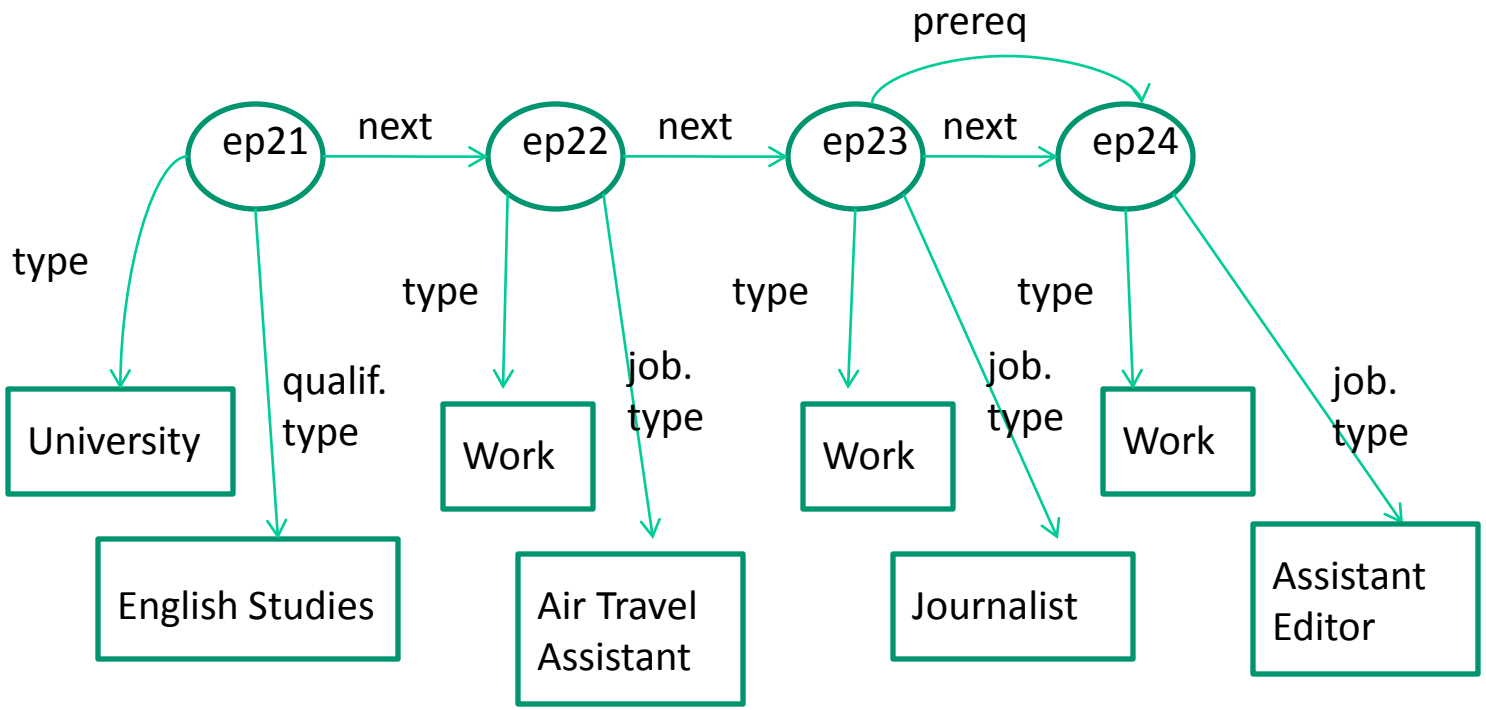
- Motivation:
  - when data resources have a complex, irregular structure (as may, for example, be the case in resources that result from federating or integrating multiple heterogeneous data sources), users may not be aware of the full structure and may need to be assisted in formulating queries that meet their information needs e.g.
    - by ***relaxing*** selected conditions of their query
    - by ***approximately matching*** specified relationships between entities in their query
  - We are developing query processing techniques for supporting efficient query relaxation and approximation while returning answers incrementally to the user ranked in order of "distance" to their original query

See Hurtado, Poulovassilis and Wood papers in ISWC'06, JoDS'08, ESWC'09, FQAS'09, SemHE Workshop at ECTEL'09

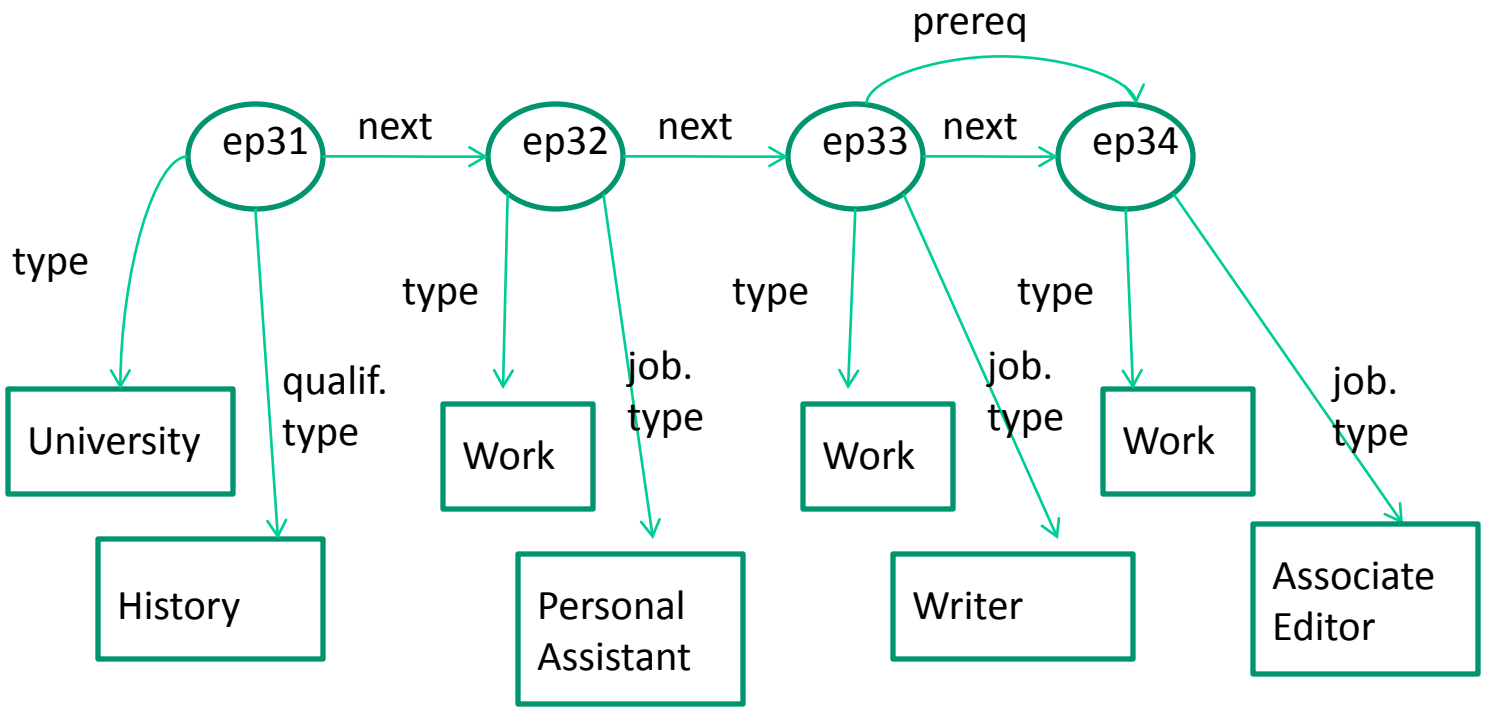
Example – from lifelong learning metadata domain



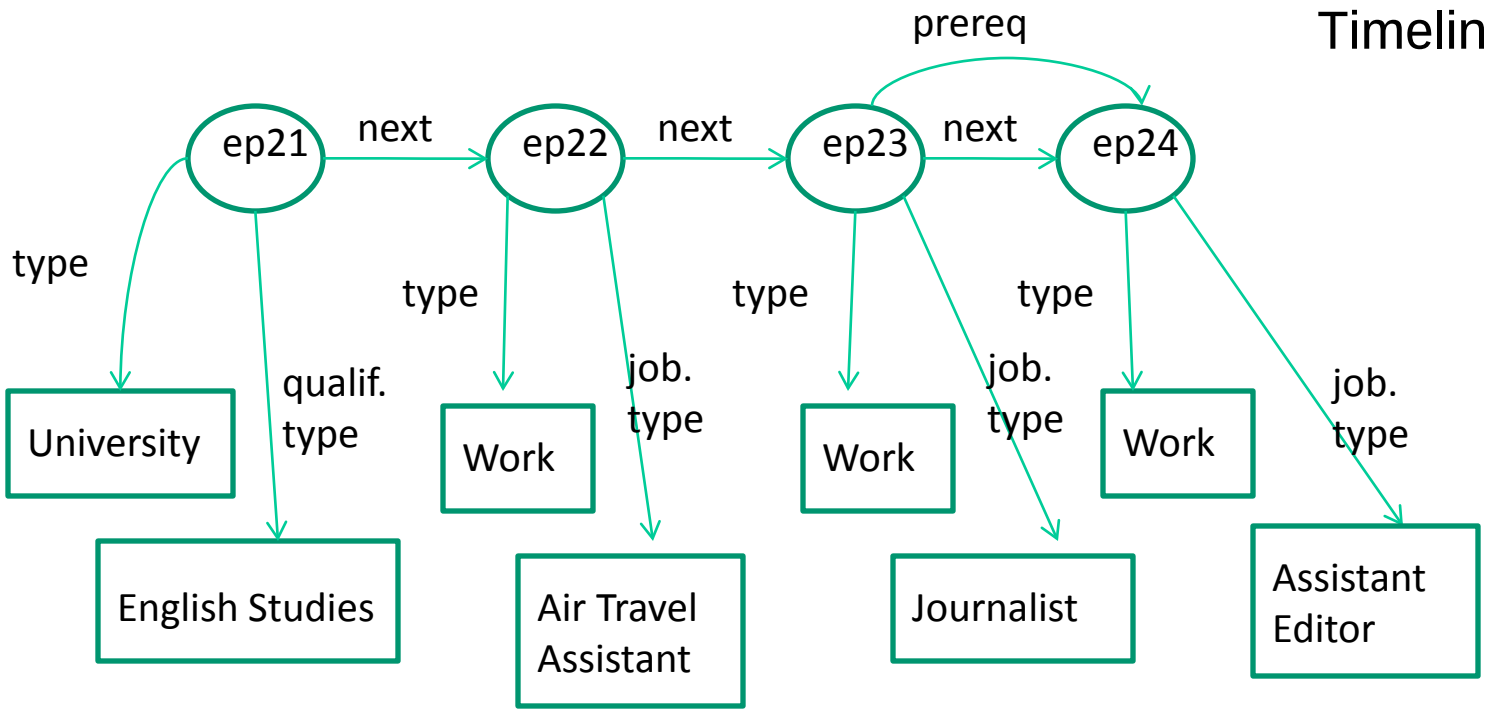
A fragment of **my timeline** data and metadata



Another user's timeline, User 2, where "prereq" indicates that this user believes that undertaking an earlier episode was necessary in order for them to be able to proceed to/achieve a later episode.



Another user's timeline, User 3

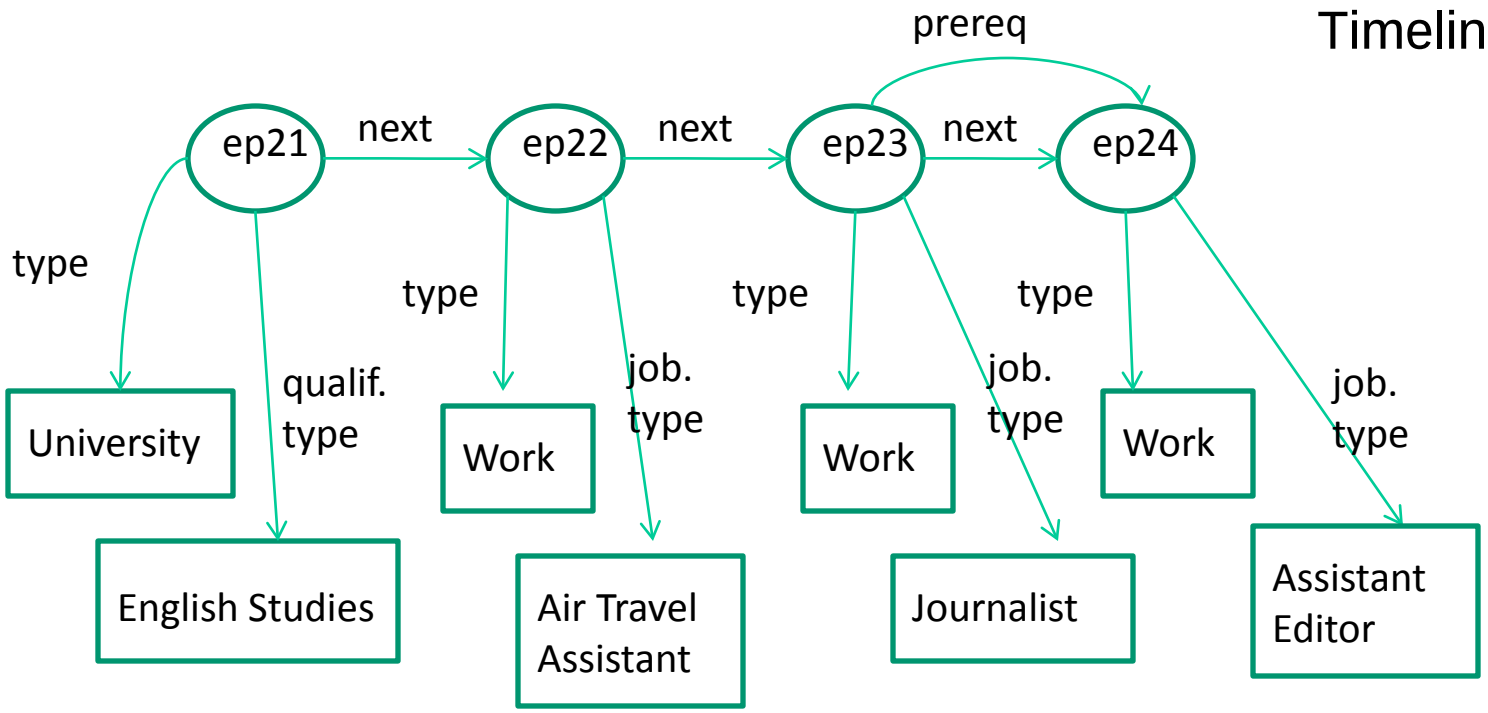


Query 1: I've done this so far; what work positions can I reach and how? E.g. selecting just the relevant prefix of my timeline (my English degree, rather than my temporary work as a Sales Assistant):

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
(?E1,prereq+,?E2),
(?E2,type,Work),(?E2,job.type,?P)

However, this will return no results relating to User 2

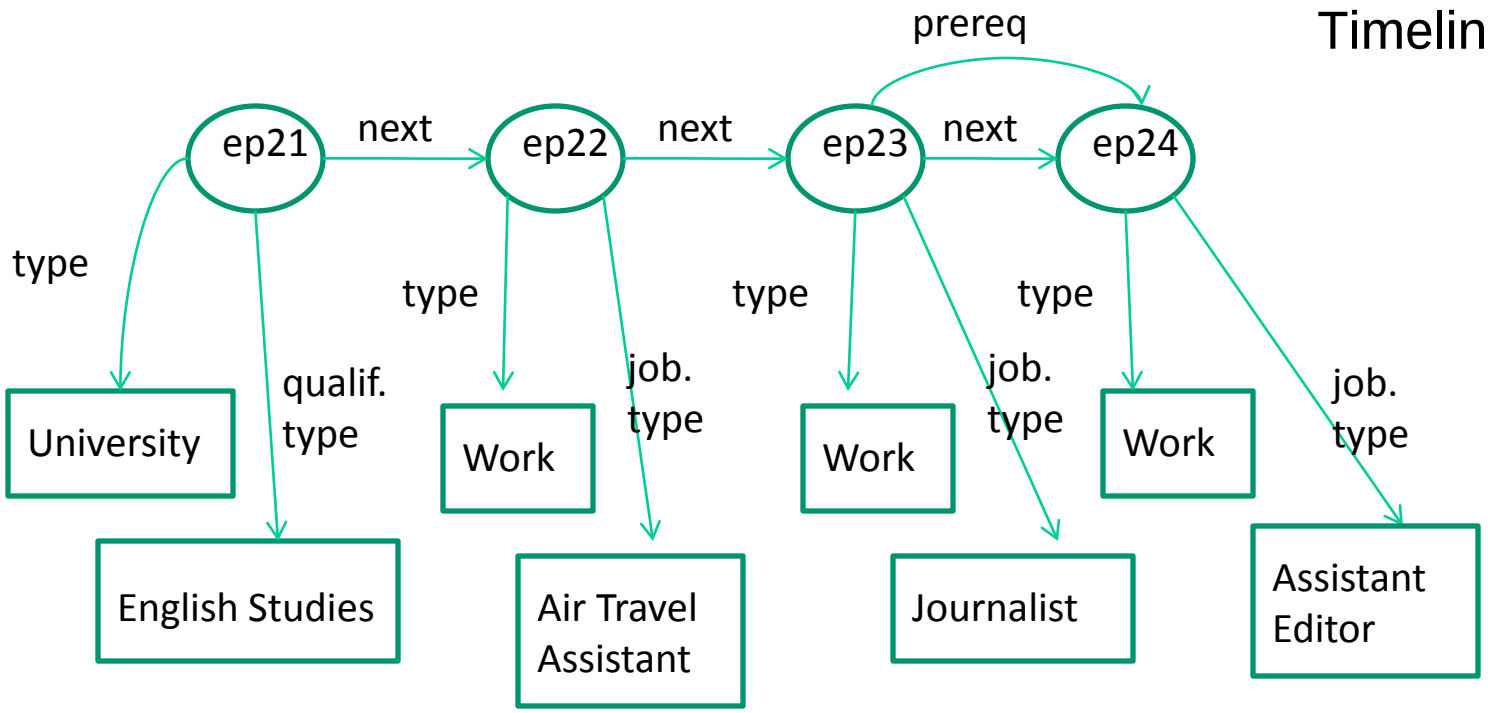
Timeline of User 2



Allowing query approximation can yield some answers. In particular, allowing replacement of the edge label “prereq” by label “next” at an edit cost of 1, the user can submit this query:

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,type,Work),(?E2,job.type,?P)

Timeline of User 2

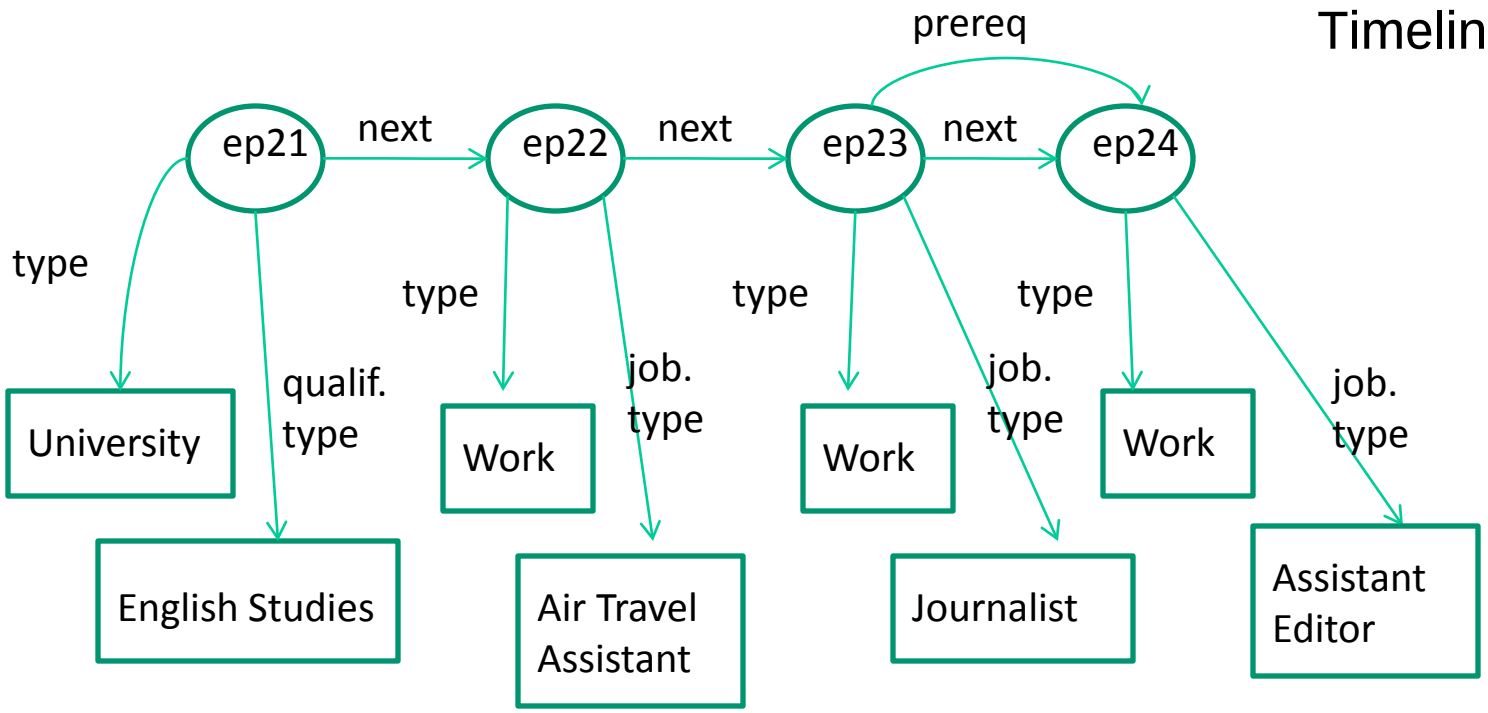


The regular expression prereq+ can be approximated by next.prereq* at edit distance 1. This allows the system to return

(ep22,AirTravelAssistant)

The user may judge this result to be not relevant and seek further results from the system.

Timeline of User 2

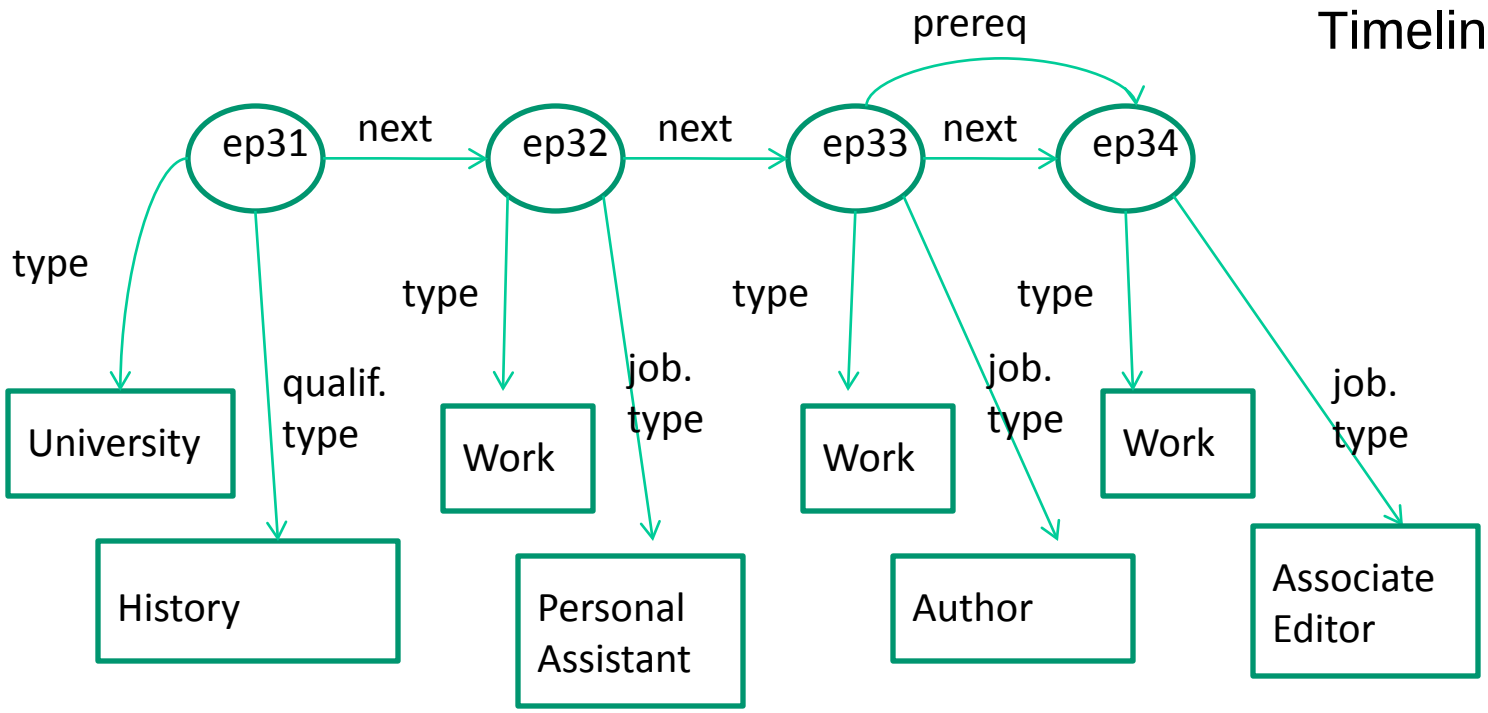


The regular expression next.prereq* can be approximated by next.next.prereq*, now at edit distance 2. This allows the following answers to be returned:

(ep23,Journalist), (ep24,AssistantEditor)

The user may judge both of these as being relevant, and can then request the system to return the whole of User 2's timeline to explore further.

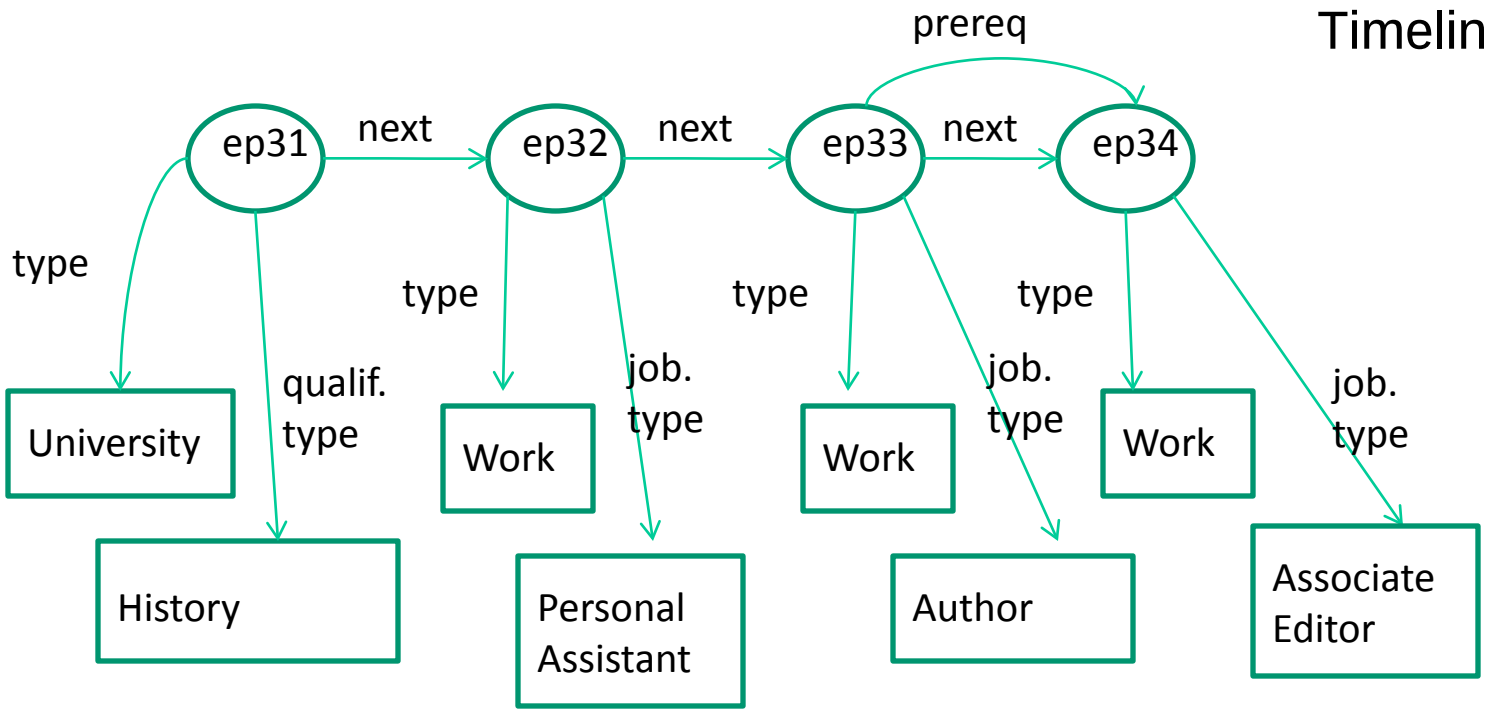
Timeline of User 3



Query 1, relaxed: What jobs are open to me if I study English or something similar at University?

(?E2,?P) ← (?E1,type,University),(?E1,qualif, ?D),
RELAX (?D,type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,type,Work),(?E2,job.type,?P)

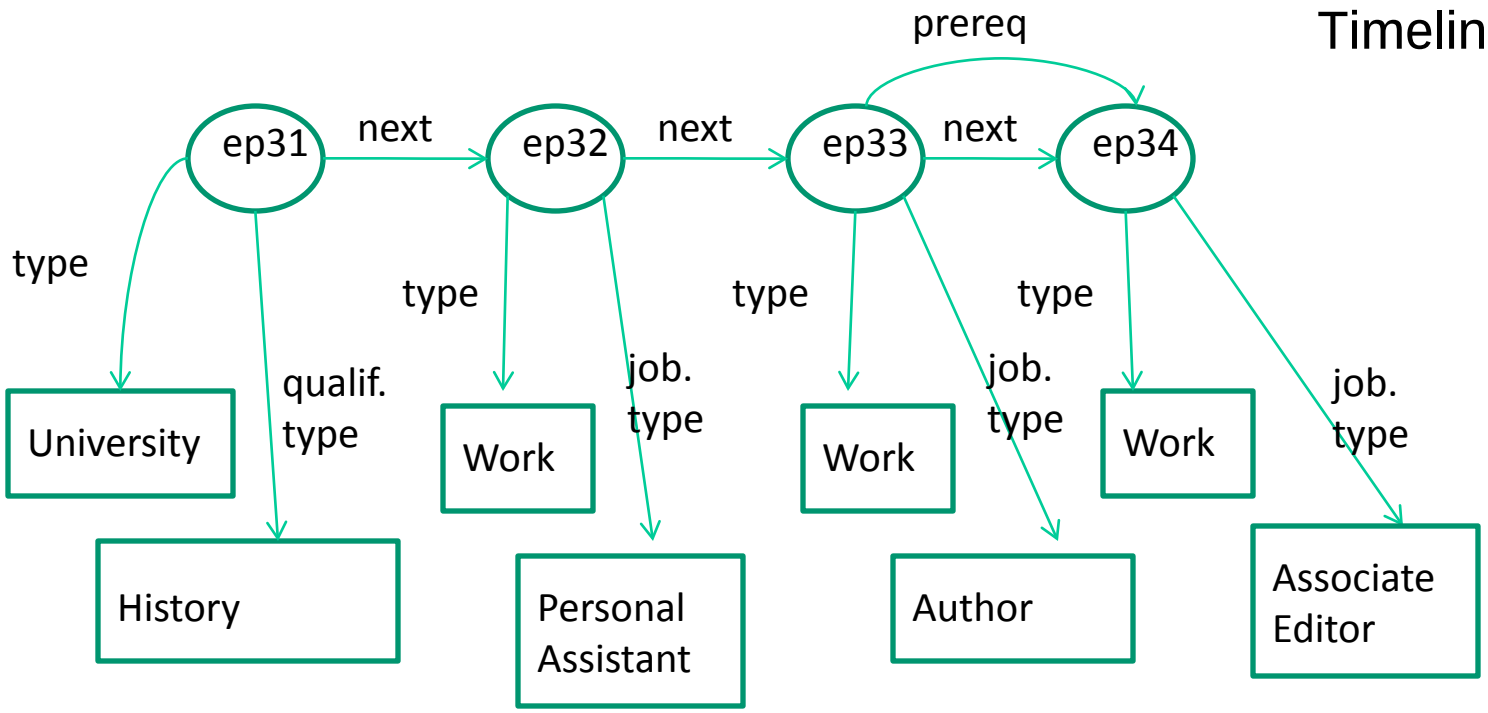
In addition to the answers obtained by the previous query, we also have answers returned from the timeline of User 3:



The regular expression prereq+ can be approximated by next.prereq* at edit distance 1, and English Studies can be relaxed to Humanities at relaxation distance 2 (see Fig. 1 of the SemHE'09 paper), thus encompassing History. This allows the system to return

(ep32,PersonalAssistant)

at overall distance 3 from the original query, assuming the same weighting is applied to approximation and relaxation.



next.prereq* can be approximated by next.next.prereq*, now at edit distance 2, with English Studies again relaxed to Humanities at relaxation distance 2 . This allows the following answers to be returned:

(ep33,Author), (ep34,AssociateEditor)

at overall distance 4 from the original query. The user may judge both of these as being relevant, and can then request the system to return the whole of User 3's timeline to explore further.

Query 2: I've done this so far, I want to attain to a certain work position, how might I do it? E.g. I've studied English at University, how might I become an Assistant Editor?

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,job.type,?P)
APPROX (?E2,prereq+,?Goal),
(?Goal,type,Work),(?Goal,job.type,AssistantEditor)

At distance 0 there are no results from the timeline of User 1.
At distance 1 there are still no results.
At distance 2, the following answers are returned:

(ep22,AirTravelAssistant), (ep23,Journalist)

the second of which gives the user potentially useful information on how to become an Assistant Editor having studied English at University

Query 2, relaxed: I've done this so far, I want to attain to a certain work position or something like it, how might I do it? E.g. I've studied English at University, how might I become an Assistant Editor or something similar?

(?E2,?P) ← (?E1,type,University),(?E1,qualif.type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,job.type,?P)
APPROX (?E2,prereq+,?Goal),
(?Goal,type,Work),(?Goal,job,?AG)
RELAX (?AG,type,AssistantEditor)

Query type 2, further relaxation: What jobs similar to Assistant Editor may be open to someone who's studied English or something similar, and how could they attain such jobs?

(?E2,?P) ← (?E1,type,University),(?E1,qualif,?D)
RELAX (?D,type,EnglishStudies),
APPROX (?E1,prereq+,?E2),
(?E2,job.type,?P)
APPROX (?E2,prereq+,?Goal),
(?Goal,type,Work),(?Goal,job,?AG)
RELAX (?AG,type,AssistantEditor)

In addition to the earlier results from User 2, English Studies can be relaxed to Humanities at distance 2 (thus encompassing History) and Assistant Editor to Editor at distance 1 (thus encompassing Associate Editor) – see Fig. 1 of the SemHE'09 paper – thus returning the following answers from User 3's timeline at distance 5 (edit distance 2 + relaxation distance 3):

(ep32,PersonalAssistant), (ep33,Author)

Single-Conjunct Regular Path Queries

- For exact regular path queries, the answer can be computed in polynomial time in the size of the database graph and the regular expression (Mendelzon & Wood, VLDB 1989)
- For APPROXed queries, our ESWC 2009 paper shows how the top-k answer can be computed in polynomial time in the size of the database graph and the regular expression
- This computation can be done incrementally
- For RELAXed queries, this is based on RDFS inference rules; we show in a forthcoming paper that such queries too can be computed in polynomial time in the size of the database and ontology graphs and the regular expression (this extends our earlier work on query relaxation for graph-pattern queries, JoDS 2008)
- Again, the computation can be done incrementally

General Approx/Relax Queries

- As we saw in the earlier examples, a general Approx/Relax query Q may comprise any number of the three above types of conjuncts on its right hand side, and any number of variables

  Z_1 ,..., Z_m on its left hand side

- Given any matching θ from variables to the nodes of graph G, the tuple $\theta(Z_1, ...,Z_m)$ has a ***distance to Q*** given by summing the total edit distance of the instantiated APPROXed conjuncts and the total relaxation distance of the instantiated RELAXed conjuncts (possibly with different weightings applied to each summand)

- The ***top-k answer of Q on G*** is a list containing the k tuples $\theta(Z_1, ...,Z_m)$ with minimum distance to Q, ranked in order of increasing distance to Q

General Approx/Relax Queries

- To ensure polynomial time evaluation, we require that the conjuncts of a query Q are *acyclic*
- This implies the existence of a join tree T induced by the conjuncts of Q, consisting of nodes denoting join operators and nodes representing single conjuncts
- Incremental evaluation proceeds by calling a function *getNext* with the root of T
- If its argument is a single query conjunct, *getNext* is as discussed earlier for single-conjunct regular path queries
- If its argument is a join operator, *getNext* is based on pipelined execution of any rank-join operator, such as the hash ripple join algorithm of Ilyas, Aref, Elmagarmid 2004

5. Other related recent and ongoing work

- Techniques for semi-automatic XML data transformation & integration, possibly ontology-assisted (Zamboulis)
- Information extraction from text data, and integration with semi-structured and structured information into a virtual global schema that can then be queried (Williams)
- Quality-driven data integration – incremental, user feedback-driven (Wang)
- Web information retrieval and integration (Demetriades, Naz, Alaseeri)
- Implementation, optimisation and evaluation of flexible query processing algorithms (Selmer)