

# Intelligent Technologies Module: Ontologies and their use in Information Systems Part II

Alex Poulouvassilis

November/December 2009

## In the previous lecture we discussed:

- What is an ontology?
- How are ontologies developed?
- Reasoning over ontologies
- Usage Scenarios for Ontologies in Information Systems:
  - Supporting personalisation

# Outline of this lecture

In this lecture we will look at two other major usage scenarios for ontologies in Information Systems:

- **Ontology as global schema**, and in particular:
  - integrating and querying heterogeneous databases under an ontology
- Enabling interoperability between different systems, via **ontology-assisted service reconciliation**

## 6. Ontology as Global Schema

- We will look at
  - the challenges faced in integrating heterogeneous data
  - the main approaches to heterogeneous data integration
  - our work at Birkbeck in this area
  - three case studies,
    - illustrating three different ways in which heterogeneous databases may be integrated under an ontology
    - and how querying is supported in each case

# Challenges of heterogeneous data integration

- Increasingly large volumes of data are being made available
- Data sources are often developed by different people with differing requirements for differing purposes
- Data sources may therefore be heterogeneous in terms of their:
  - data model
  - query interfaces
  - query processing capabilities
  - database schema or data exchange format
  - data types used
  - nomenclature adopted
- Integrating data sources to meet the needs of new users or new applications requires reconciliation of such heterogeneities

# Two main approaches to data integration (DI)

## ▪ **Materialised DI:**

- import the source data into a centralised “staging area”
- define a Data Warehouse schema, typically within a relational Database Management System (DBMS) such as Oracle, DB2, SQL Server etc.
- clean, transform and aggregate the imported data so that it conforms to the DW schema
- load the data into the DW
- query the DW via the DBMS’s querying facilities
- incrementally update the DW when data sources change (insertions, deletions, updates ...), utilising facilities that the DBMS provides

# Two main approaches to DI

## ▪ **Virtual DI:**

- define an integrated schema (IS)
- “wrap” the data sources (DSs), using [wrapper](#) software
- construct [mappings](#) between the DSs and the IS, using [mediator](#) software
- different mediator systems support different types of mappings e.g.
  - ***Global-as-view (GAV) mappings***, where each schema object  $o$  in the IS is defined as a view over the DSs:

$$o = / \subseteq / \supseteq \varphi(DS_1, \dots, DS_n) \quad \text{for some formula } \varphi$$

- ***Local-as-view (LAV) mappings***, where each schema object  $o$  in a DS is defined as a view over the IS:

$$o = / \subseteq / \supseteq \psi(\text{IS}) \quad \text{for some formula } \psi$$

- ***Global-local-as-view (GLAV) mappings***, where the mapping relates a view over a DS with a view over the IS:

$$\varphi(DS_i) = / \subseteq / \supseteq \psi(\text{IS}) \quad \text{for some formulae } \varphi, \psi$$

# Two main approaches to DI

- **Virtual DI (cont'd):**

- the integrated schema is queried by submitting queries to the mediator system
- the mediator coordinates query evaluation:
  - it uses the mappings to reformulate queries expressed on the IS to queries expressed on the DS schemas
  - it submits subqueries to the DS wrappers for evaluation at the data sources
  - it merges the subquery results returned by the wrappers into an overall query result for the original query

# Comparing the approaches

- Materialised and virtual DI both allow the integrated resource to be queried as though it were a single data source:
  - both support a single integrated schema
  - users/applications do not generally need to be aware of the data source schemas, formats or content
- A materialised DI approach is generally adopted for:
  - better query performance (centralised rather than distributed)
  - greater ease of data cleansing and data annotation (on materialised rather than virtual data)
- A virtual DI approach is generally adopted for:
  - lower cost of storing and maintaining the integrated resource (no replication of DSs' data)
  - greater currency of the integrated resource (the IS reflects the current content of the DSs)

# Virtual DI: the integrated schema

- The integrated schema may be defined in a standard data modelling language
- Or, it may be a source-independent *ontology*:
  - defined in a higher-level ontology language
  - serving as a “global” schema for multiple potential data sources, beyond the ones being integrated
- The integrated schema does not necessarily have to encompass *all* of the data in the data sources:
  - it may be enough to capture just the data that is needed for answering the main queries that users will want to submit
  - this avoids the possibly complex process of creating a complete integrated schema and a complete set of mappings between the data sources and the integrated schema

# Our work: AutoMed Project

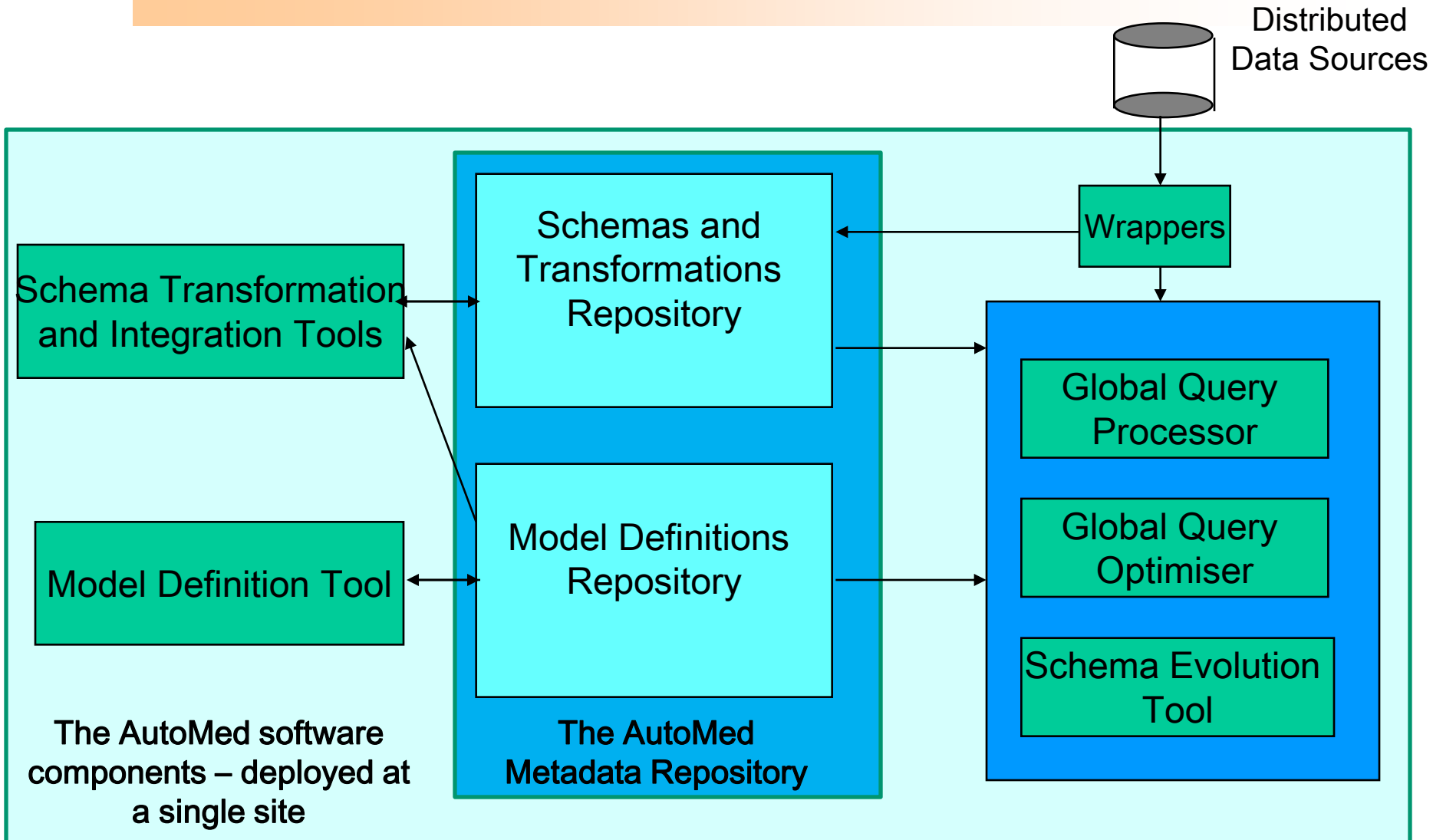
The AutoMed Project at Birkbeck and Imperial has developed tools for the semi-automatic integration of heterogeneous information sources:

- AutoMed supports a graph-based metamodel, the [Hypergraph Data Model](#) (HDM)
- Higher-level modelling languages can be defined in terms of this HDM, via the API of AutoMed's Model Definitions Repository
  - so the system is extensible with new modelling languages e.g. relational, XML, RDF/S, OWL
- After a modelling language has been defined in this way, a set of primitive transformations become available that can be applied to schemas expressed in that language
- Schemas may or may not have a data source associated with them i.e. they may be materialised or virtual schemas

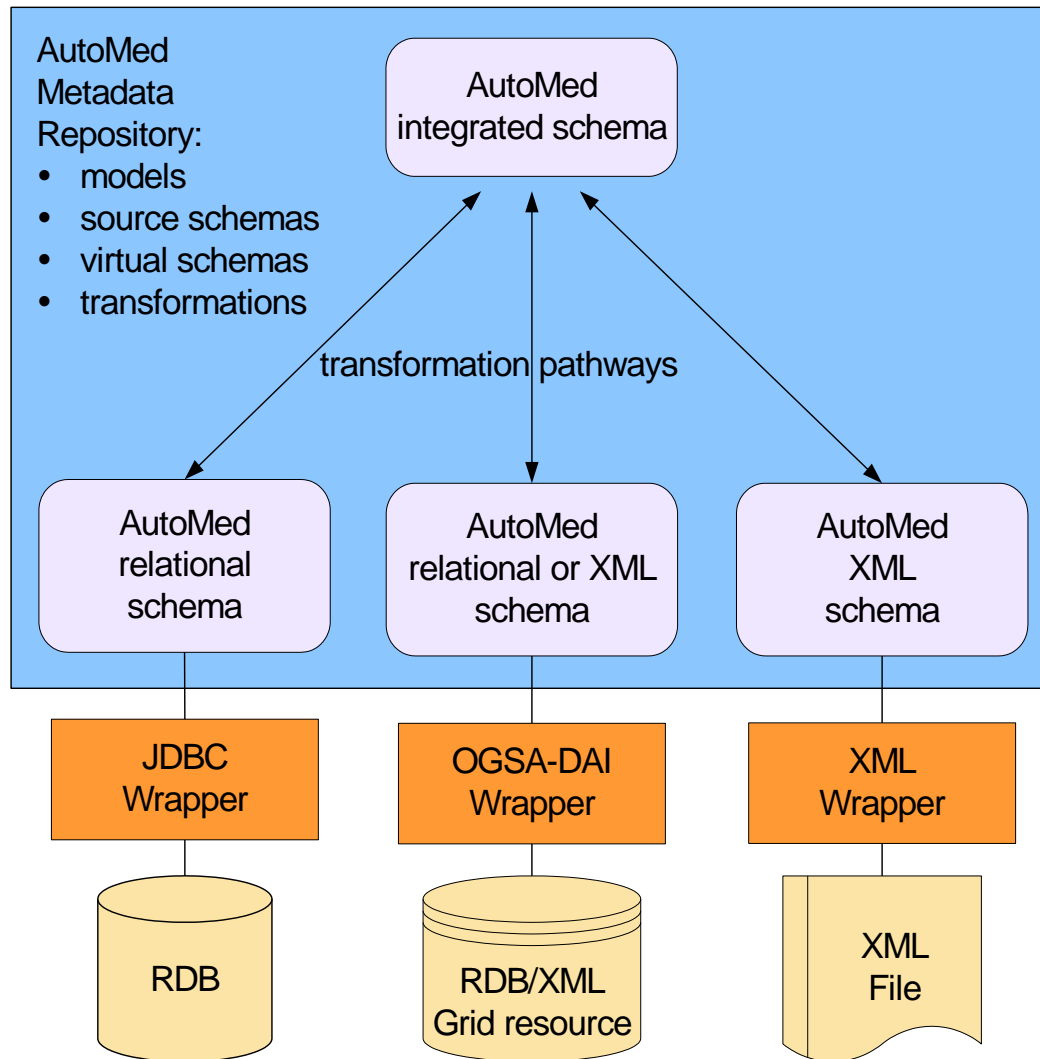
# AutoMed features

- Transformations are applied to schemas via the API of AutoMed's Schemas & Transformations Repository
- The set of primitive transformations supported is as follows:
  - `add(construct,query)`, `delete(construct,query)`
  - `extend(construct,Range q1 q2)`, `contract(construct,Range q1 q2)`
  - `rename(construct,old_name,new_name)`
- The queries within these transformations allow automatic **data** and **query translation** along sequences of transformations (which we term transformation pathways)
- Using sequences of AutoMed primitive transformations, we can specify GAV, LAV and GLAV mappings between an integrated schema and a set of data source schemas
- So we term AutoMed's mapping approach ***Both-As-View (BAV)***

# AutoMed Architecture



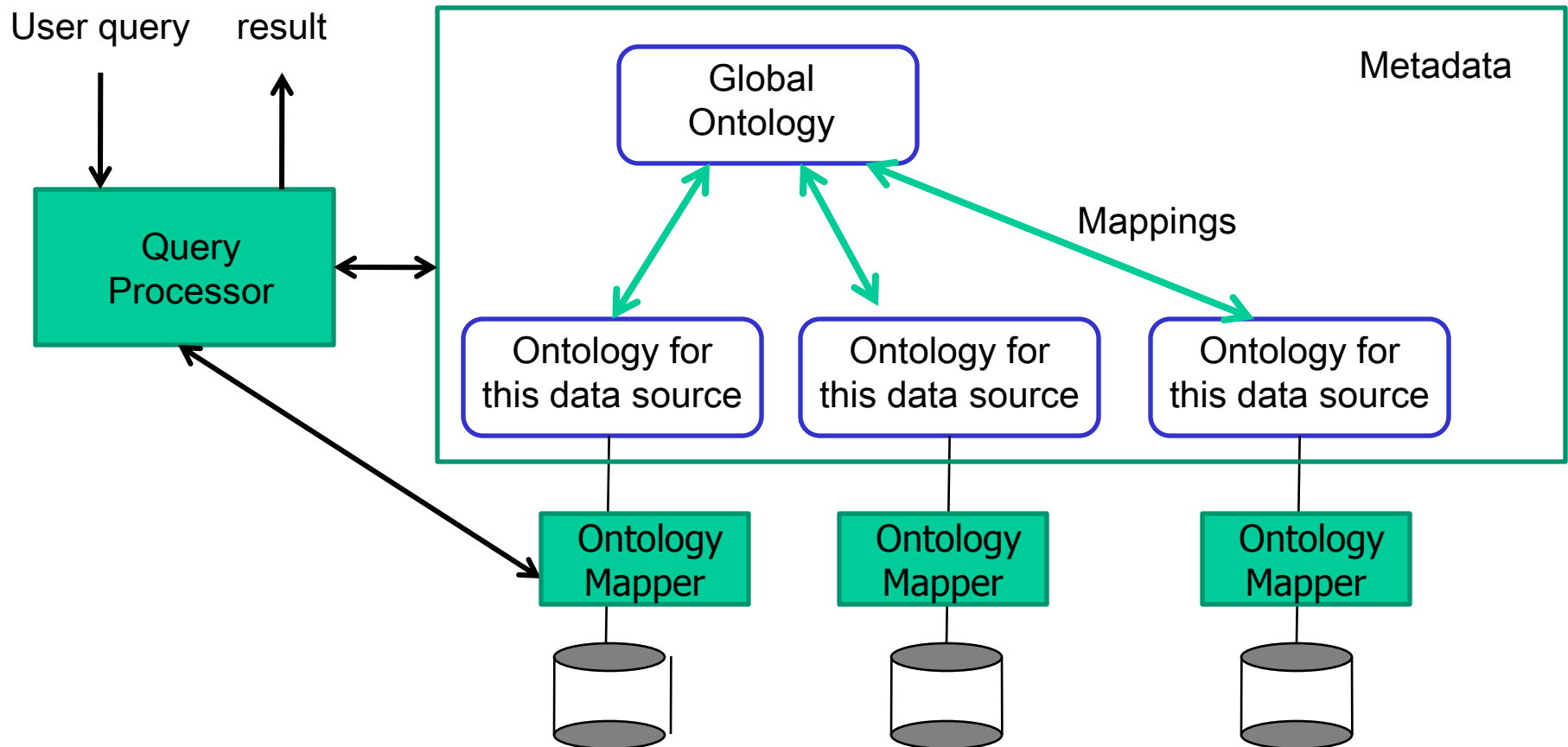
# Virtual Data Integration using AutoMed



# Integrating heterogeneous databases under an ontology

- There are several possible architectures:
  - ontology-based access to multiple ontology-mapped databases
  - ontology-based access to multiple databases
  - ontology-based access to an integrated virtual database resource

# Ontology-based access to multiple ontology-mapped databases



# Case Study E: SemWIK (Semantic Web Integrator and Query Engine)

- The aim of the SemWIK middleware is to support data sharing in distributed scientific communities
- An RDF mapping service (specifically D2R Server) interacts with each of the data sources
- The global ontology is expressed in OWL-DL
- The mappings consist of GAV views, which map terms in the ontology to the data sources' RDF representations
- Users' queries are expressed with respect to the ontology, using SPARQL
- The SemWIK Query Processor reformulates user queries to sub-queries expressed on the data sources' RDF representations, using the mappings
- The sub-queries are directed to the RDF mapping services over the data sources, which translate the SPARQL sub-queries to native queries that are submitted to the actual data sources for evaluation

# SPARQL – a query language for RDF

- See <http://www.w3.org/TR/rdf-sparql-query/> “SPARQL Query language for RDF”
- Here is an example SPARQL query from that document:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
```

```
PREFIX ns: <http://example.org/ns#>
```

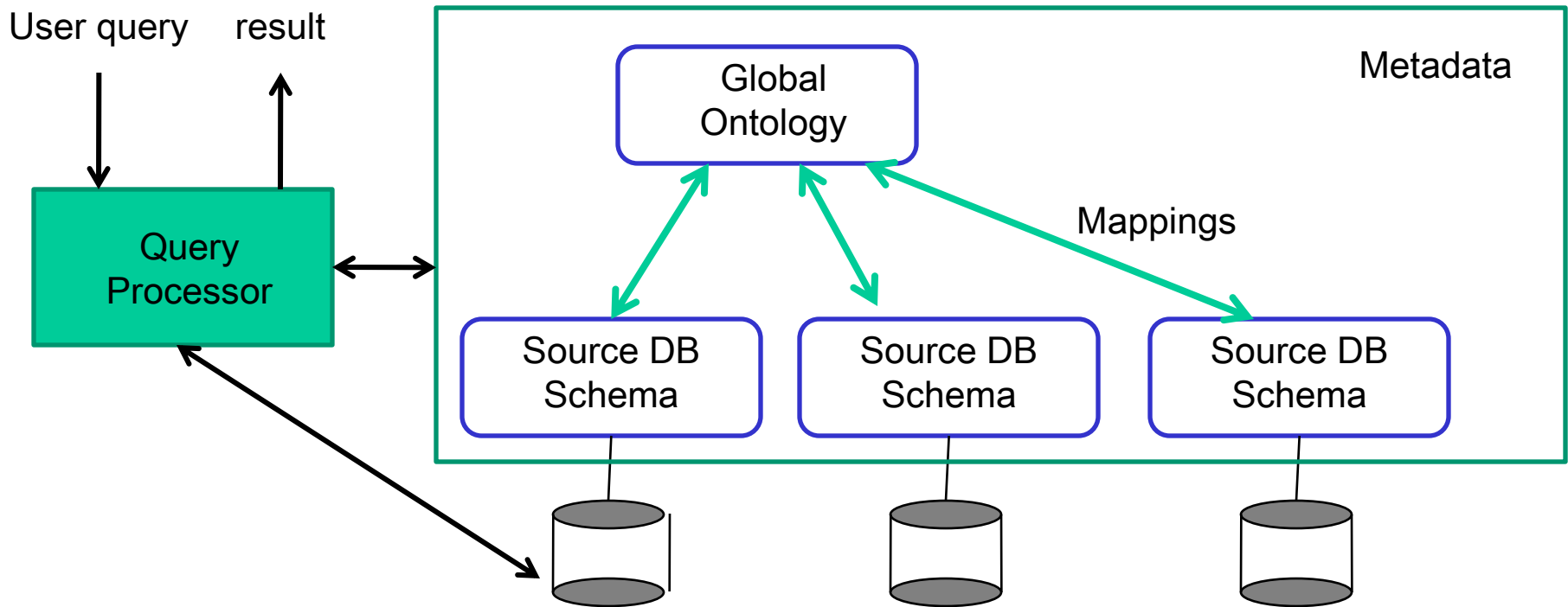
```
SELECT ?title ?price
```

```
WHERE { ?x ns:price ?price .
```

```
        FILTER (?price < 30.5)
```

```
        ?x dc:title ?title . }
```

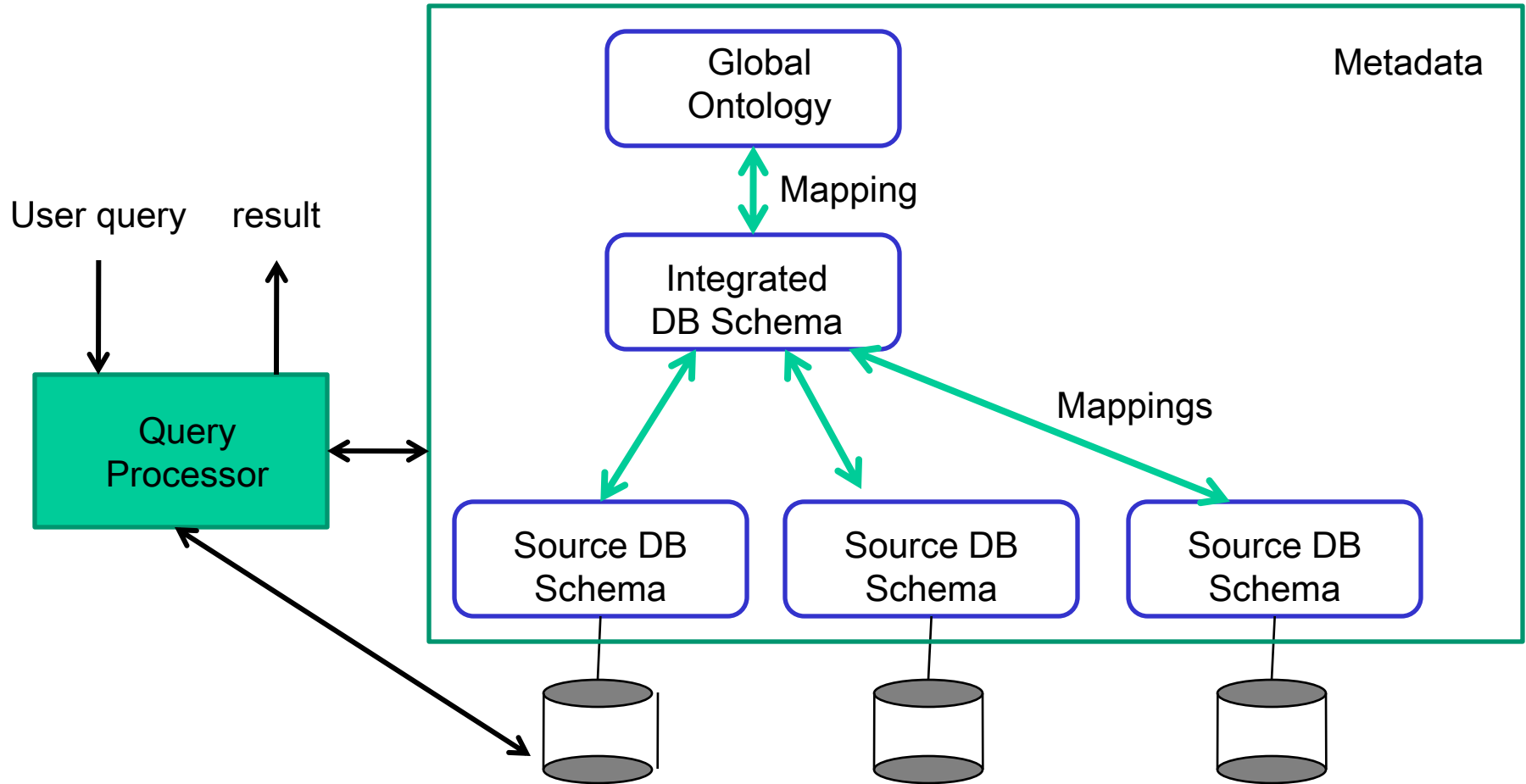
# Ontology-based access to multiple databases



# Case Study F: MASTRO

- The global ontology is expressed in the DL-Lite<sub>A</sub> language [Poggi et al., 2008]
- The data in the data sources is treated as instance data of the ontology
- Mappings consist of a query  $Q_D$  over a data source mapped to a query  $Q_O$  over the ontology (so these are GLAV mappings)
- User queries are expressed with respect to the ontology
- User queries are first expanded according to the constraints contained in the ontology
- The Query Processor then reformulates these queries to sub-queries expressed on the data sources, using the mappings
- The sub-queries are evaluated at the data sources, and their results are merged to give the overall answer to the original query

# Ontology-based access to an Integrated Virtual Database Resource



# Case Study G: 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

## 7. Enabling interoperability between systems

- A plethora of services are being made available in many domains: education, science, business ...
- Similar problems arise when combining such services into larger-scale workflows as when integrating heterogeneous data sources:
  - the necessary services are often created independently by different parties, using different technologies, data formats and data types
- Therefore additional code needs to be developed to transform the output of one service into a format that can be consumed by another
- Pointing to the need for **service reconciliation** techniques

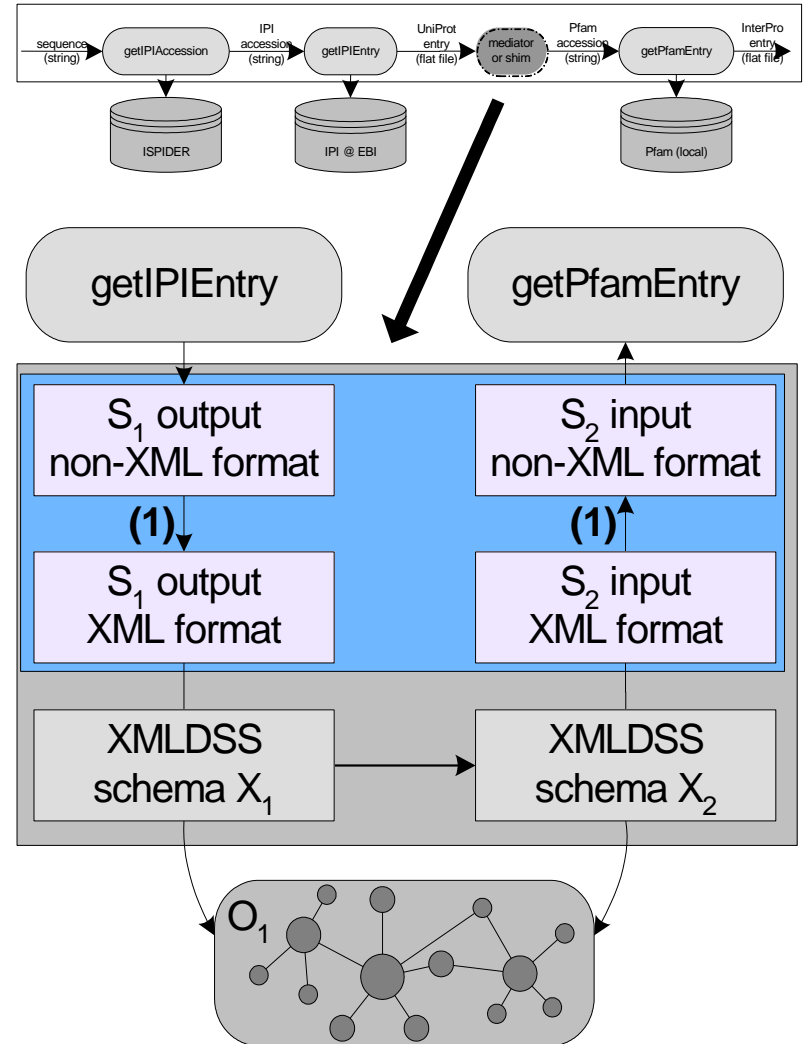
# Service reconciliation

- Previous approaches include:
  - **Shims**, developed as part of the *my*Grid project. These are services that act as intermediaries between specific pairs of services and reconcile their inputs and outputs. They need to be programmed individually
  - **Bowers & Ludäscher (DILS'04)** use mappings to an ontology for reconciling services. An XQuery query is automatically generated from these mappings. Applied to the output of one service, the XQuery query transforms it into a form that can be accepted by the next service

# Our approach

## (1) We adopt XML as the common representation format for service inputs and outputs:

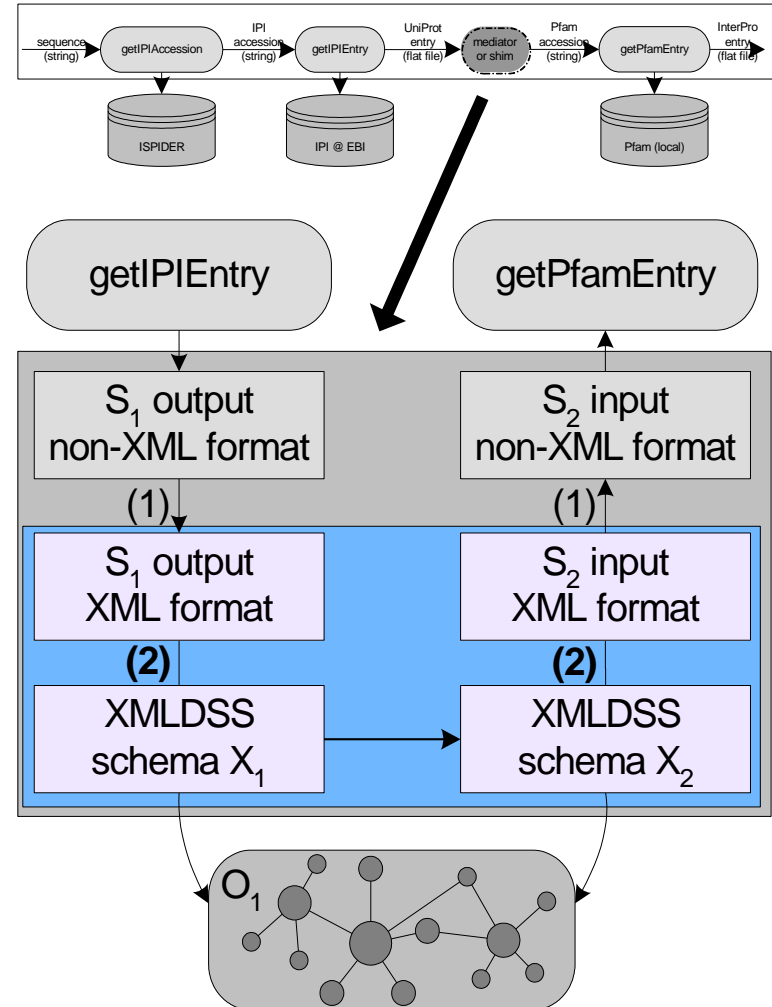
- Suppose we need to automatically translate the output of service S<sub>1</sub> (getIPIEntry) so that it can be passed to the next service S<sub>2</sub> (getPfamEntry)
- We assume the availability of a format converters that can
  - convert the output of service S<sub>1</sub> into XML, if it is not XML;
  - and similarly that can convert from an XML representation into the input format expected by service S<sub>2</sub>



# Our approach

## (2) We adopt XMLDSS as the schema type:

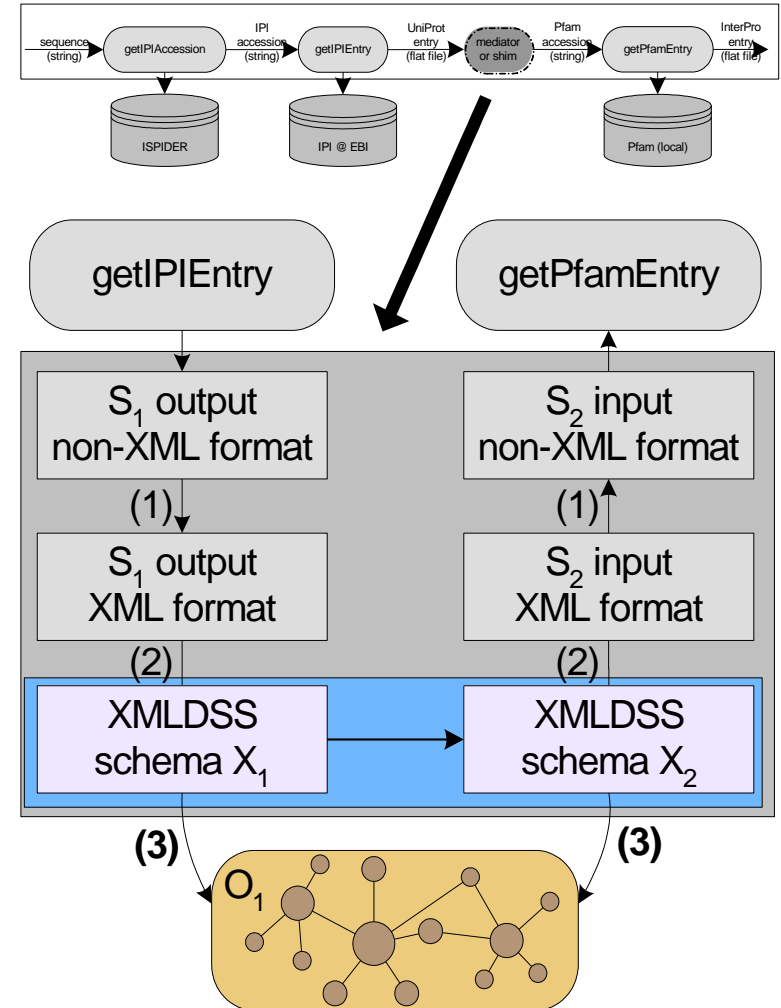
- We use our own **XMLDSS** schema type as the common schema type for XML data that is output from/input to services
- XMLDSS is a “structural summary” of an XML document
- This can be automatically derived from a DTD/XML Schema associated with the XML document, if this is available
- If not, an XMLDSS schema can be automatically extracted from an XML document – there is an AutoMed tool that can do that



# Our approach

## (3) We use correspondences to a domain ontology:

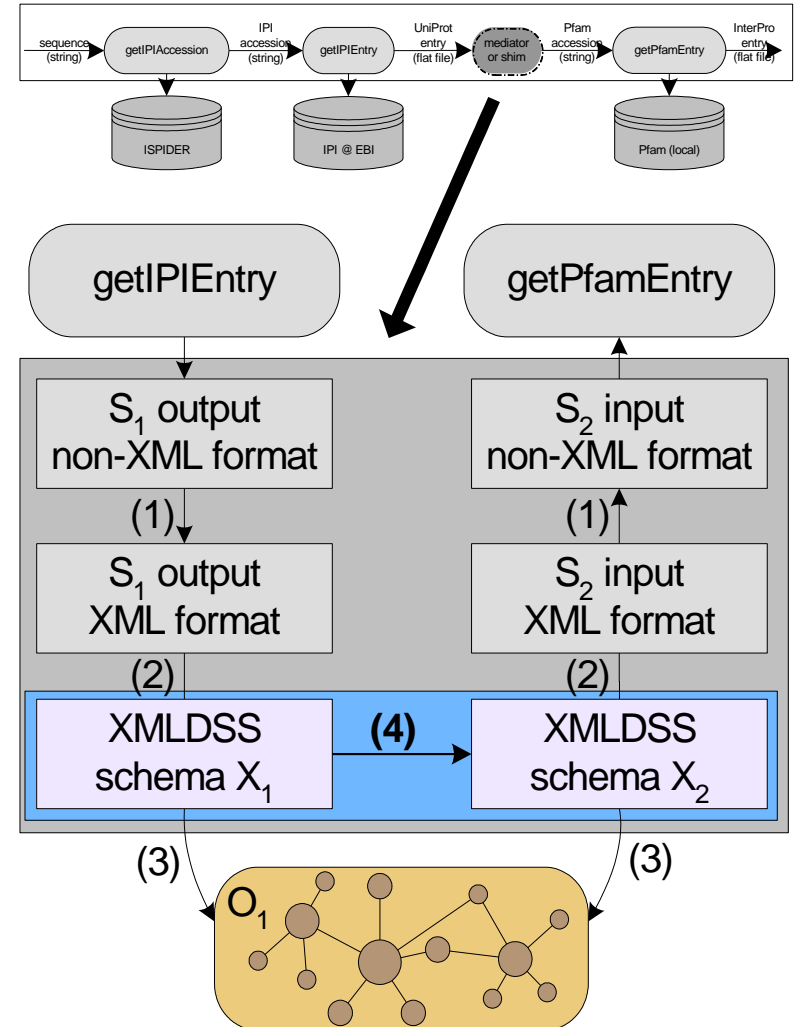
- We assume the availability of a set of correspondences between each XMLDSS schema and a domain ontology (a set of correspondences C1 for schema X1 and a set of correspondences C2 for schema X2)
- An XMLDSS element can correspond to some concept or path in the ontology
- An XMLDSS attribute can correspond to some literal-valued property, or to some path ending in such a property
- In general, there may be several correspondences for an element or an attribute with respect to the ontology



# Our approach

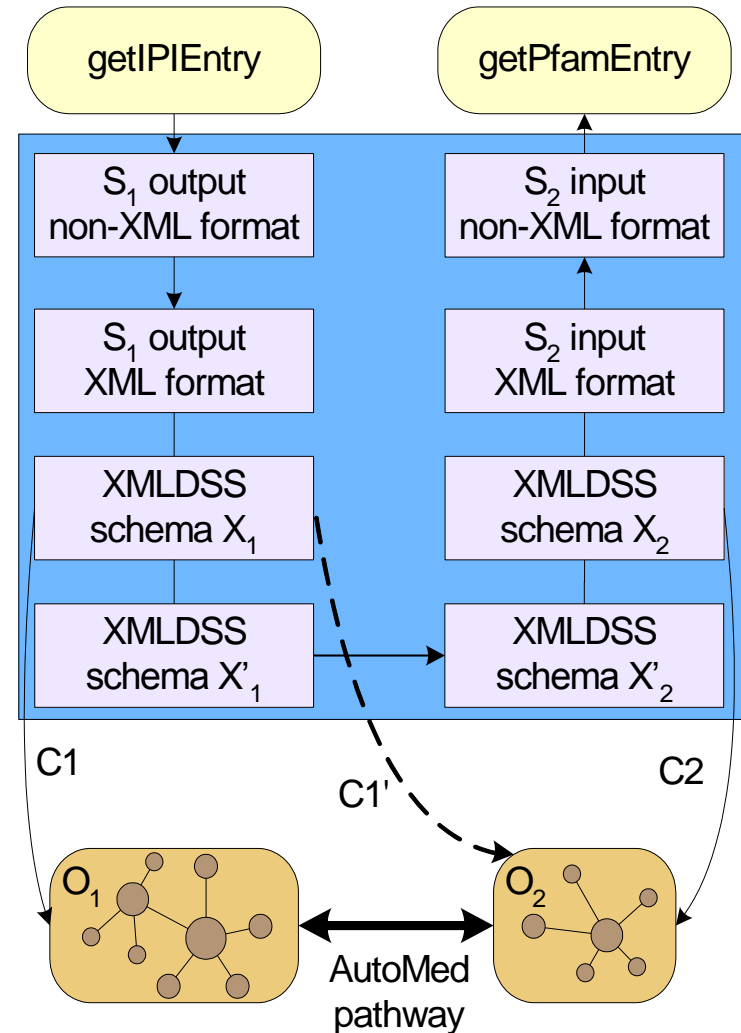
## (4) Schema and data transformation:

- A schema transformation pathway is then automatically generated that can transform  $X_1$  to  $X_2$ :
  - The correspondences are used to create pathways  $X_1 \leftrightarrow X_1'$  and  $X_2 \leftrightarrow X_2'$ , where intermediate schemas  $X_1'$  and  $X_2'$  are conformed w.r.t. the ontology
  - Our XMLDSS restructuring algorithm then automatically creates the pathway  $X_1' \leftrightarrow X_2'$
  - Hence obtaining an overall pathway  $X_1 \leftrightarrow X_1' \leftrightarrow X_2' \leftrightarrow X_2$ . This can now be used to transform data output from  $S_1$  into data that can be input to  $S_2$



# Correspondences to multiple ontologies

- In a setting where
  - a) X1 corresponds to an ontology O1 using a set of correspondences C1,
  - b) X2 corresponds to a *different* ontology O2 using a set of correspondences C2, and
  - c) there is an AutoMed transformation pathway  $O1 \leftrightarrow O2$
- Then, we can automatically generate a new set of correspondences C1' for X1 with respect to O2 (using the information in the pathway  $O1 \leftrightarrow O2$ )
- The setting is now identical the single-ontology setting earlier
- Proviso: the generated C1' must conform to our language for specifying correspondences



# Possible deployments

A Workflow Tool could use our approach either dynamically or statically:

## **a) Dynamically, as a run-time mediation service:**

- The workflow tool invokes a service S1 and receives its output
- The tool submits the output of S1, the schema of the next service S2, and the two sets of correspondences to the AutoMed XML data transformation service
- The AutoMed service transforms the output of S1 to a suitable input for submission to S2, and returns this back to the workflow tool
- The workflow tool invokes service S2 with this data

## **b) Statically, for shim generation:**

- AutoMed is used to generate a shim between services S1 and S2
- The AutoMed XML data transformation service can generate an XQuery query which, when applied to output from S1, creates data corresponding to the input that is expected by S2
- The Workflow tool can use these shims at run-time

# Case Study H: Interoperability in the MyPlan project

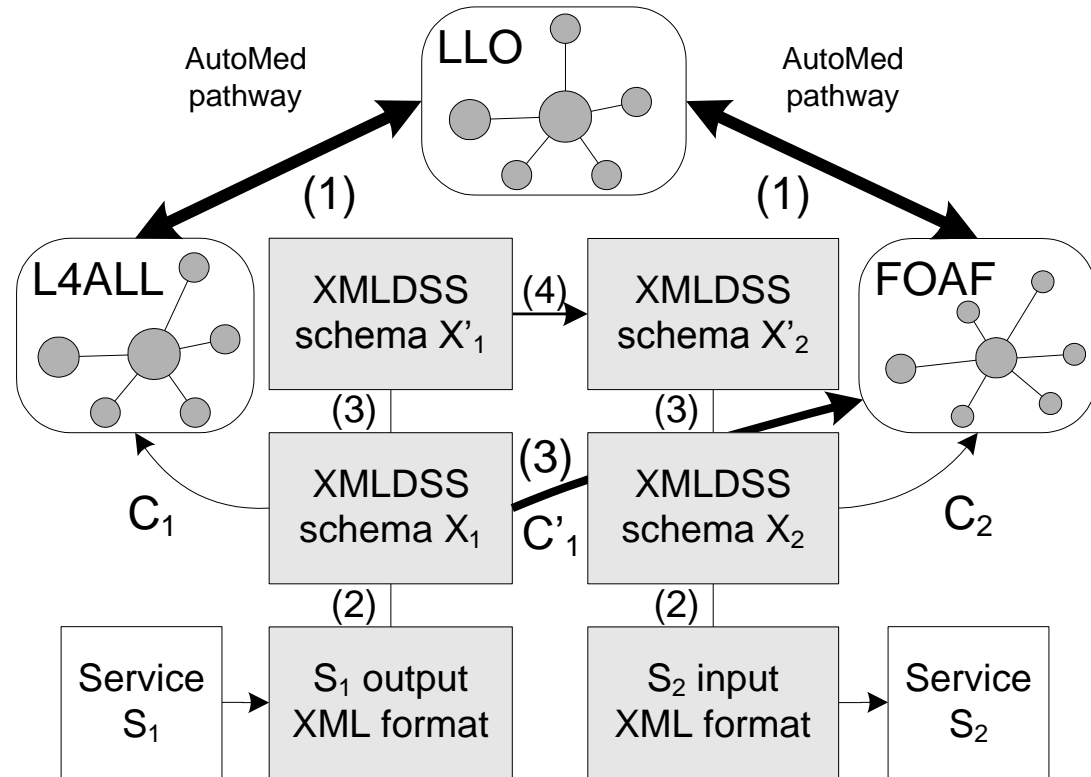
- The MyPlan project aimed to support learners in planning their lifelong learning
- One project goal was to facilitate interoperability between existing systems that target the lifelong learner
- In general, direct access to these systems' metadata repositories is not possible; but some systems do provide services that can be invoked by other systems
- Therefore, we applied our service reconciliation techniques described above in this setting.

# MyPlan project

- Our proof-of-concept implementation involves transferring learners' data between two systems:
  - L4A// ([www.lkl.ac.uk/research/l4all](http://www.lkl.ac.uk/research/l4all)) whose services' inputs and outputs correspond to the **L4ALL** ontology, defined in RDFS
  - eProfile ([www.schools.bedfordshire.gov.uk/im/EProfile](http://www.schools.bedfordshire.gov.uk/im/EProfile)) whose services' inputs and outputs correspond to the **FOAF** ontology, defined in OWL-DL
- In particular, we discuss next how data output from an L4A// service  $S_1$  which semantically corresponds to the L4ALL ontology, can be automatically transformed to data that semantically corresponds to FOAF and can be input to service  $S_2$  of eProfile

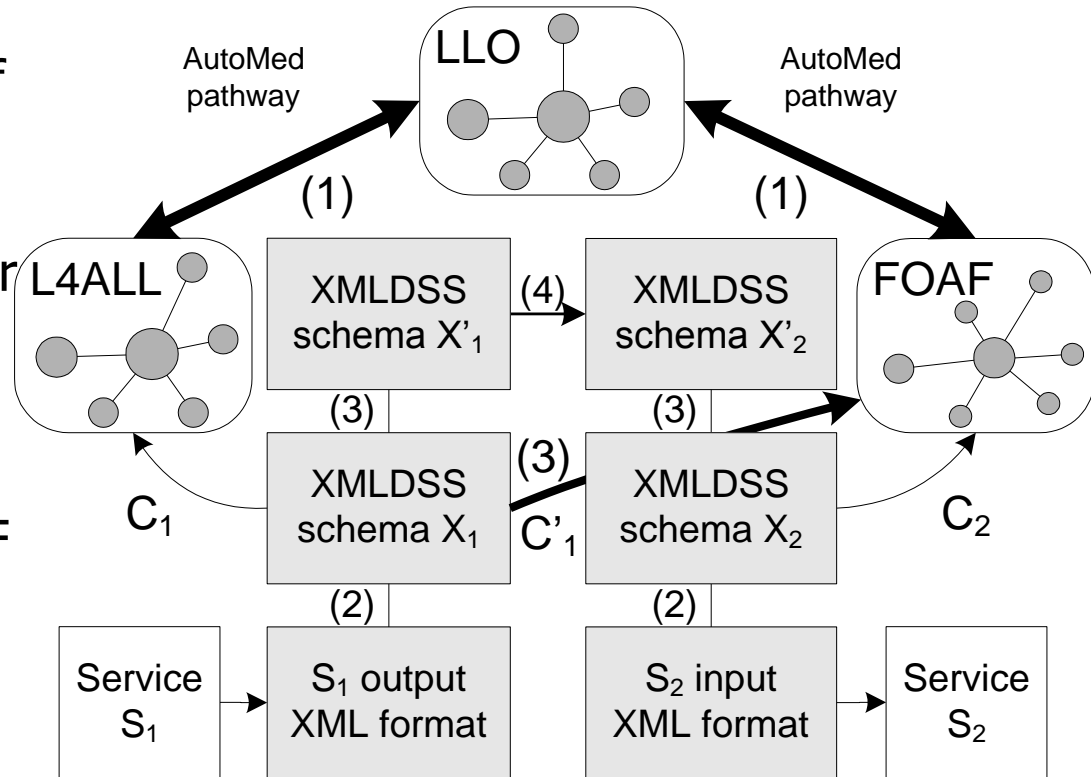
# Reconciling services $S_1$ of L4A// and $S_2$ of eProfile

1. Manually integrate, using AutoMed, the L4ALL and FOAF ontologies into a global ontology for the Lifelong Learning domain – the **LLO** ontology (defined in OWL-DL)
2. Automatically generate (using the AutoMed XMLDSS schema generation tool) a schema  $X_1$  for the output of service  $S_1$  and a schema  $X_2$  for the input of service  $S_2$

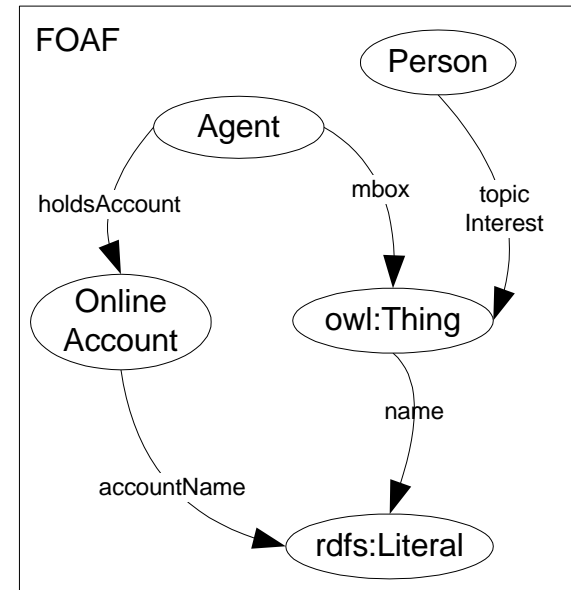
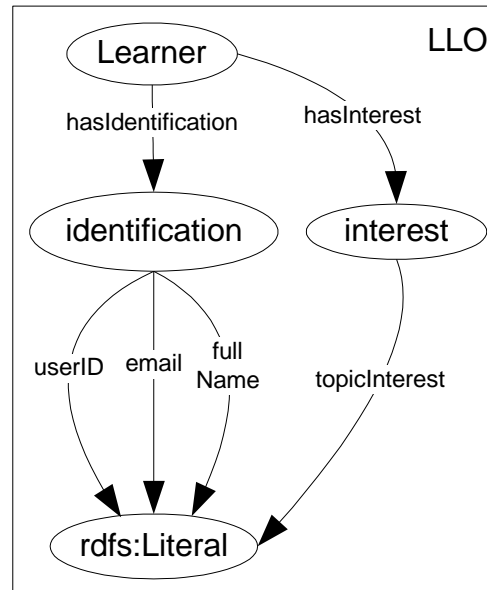
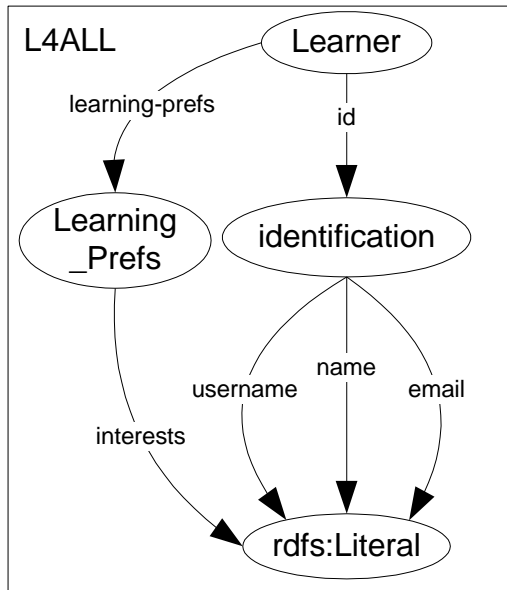


# Reconciling services $S_1$ of L4A// and $S_2$ of eProfile

4. Manually specify, or semi-automatically derive, a set of correspondences  $C_1$  for  $X_1$  with respect to L4ALL, and a set of correspondences  $C_2$  for  $X_2$  with respect to FOAF.
5. Automatically translate  $C_1$  to  $C'_1$ , now targeting FOAF, using the L4ALL $\leftrightarrow$ LLO $\leftrightarrow$ FOAF transformation pathway.
6. Automatically produce schemas  $X'_1$  and  $X'_2$  that are now conformed w.r.t. FOAF.
7. Automatically produce the transformation pathway  $X'_1 \leftrightarrow X'_2$



# Fragments of the L4ALL, FOAF and LLO ontologies



# Correspondences C1 between $X_1$ and L4ALL

<b>Construct</b>	<b>Path</b>
$\langle\langle\text{user}\$1\rangle\rangle$	$[c c\leftarrow\langle\langle\text{l4:Learner}\rangle\rangle]$
$\langle\langle\text{userID}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow\langle\langle\text{l4:id; l4:Learner; l4:Identification}\rangle\rangle;$ $\{id,lit\}\leftarrow\langle\langle\text{l4:username; l4:Identification; rdfs:Literal}\rangle\rangle]$
$\langle\langle\text{fullname}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow\langle\langle\text{l4:id; l4:Learner; l4:Identification}\rangle\rangle;$ $\{id,lit\}\leftarrow\langle\langle\text{l4:name; l4:Identification; rdfs:Literal}\rangle\rangle]$
$\langle\langle\text{email}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow\langle\langle\text{l4:id; l4:Learner; l4:Identification}\rangle\rangle;$ $\{id,lit\}\leftarrow\langle\langle\text{l4:email; l4:Identification; rdfs:Literal}\rangle\rangle]$
$\langle\langle\text{interests}\$1\rangle\rangle$	$[p \{l,p\}\leftarrow\langle\langle\text{l4:learning-prefs;$ $\text{l4:Learner;l4:Learning_Prefs}\rangle\rangle;$ $\{id,lit\}\leftarrow\langle\langle\text{l4:interests; l4:Learning Prefs; rdfs:Literal}\rangle\rangle]$

# Correspondences C1' between $X_1$ and FOAF

<b>Construct</b>	<b>Path</b>
$\langle\langle\text{user}\$1\rangle\rangle$	$[c c\leftarrow\langle\langle\text{foaf:Agent}\rangle\rangle]$
$\langle\langle\text{userID}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:Agent}\rangle\rangle);$ $\{id,oa\}\leftarrow\langle\langle\text{foaf:holdsAccount,foaf:Agent,foaf:OnlineAccount}\rangle\rangle;$ $\{oa,userlit\}\leftarrow\langle\langle\text{foaf:accountName,foaf:OnlineAccount,rdfs:Literal}\rangle\rangle]$
$\langle\langle\text{fullname}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:Agent}\rangle\rangle); id\leftarrow\langle\langle\text{foaf:Agent}\rangle\rangle$ $\{id,lit\}\leftarrow\langle\langle\text{foaf:name,owl:Thing,rdfs:Literal}\rangle\rangle]$
$\langle\langle\text{email}\$1\rangle\rangle$	$[id \{l,id\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:Agent}\rangle\rangle);$ $\{x,id,lit\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:mbox,foaf:Agent,rdfs:Literal}\rangle\rangle)]$
$\langle\langle\text{interests}\$1\rangle\rangle$	$[p \{l,p,z\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:topic\_interest,foaf:Person,owl:Thing}\rangle\rangle);$ $\{x,id,lit\}\leftarrow(\text{generateProperty } \langle\langle\text{foaf:topic\_interest,foaf:Person,owl:Thing}\rangle\rangle)]$

# Reading for this week

- Flexible data integration and ontology-based data access to medical records. L.Zamboulis, A.Poulovassilis, G.Roussos. Proc. BIBE'08, Athens, pp 1-6. At [http://www.dcs.bbk.ac.uk/~ap/pubs/BIBE\\_ZPR08.pdf](http://www.dcs.bbk.ac.uk/~ap/pubs/BIBE_ZPR08.pdf) [you can skip Data Cleansing Section III D, the Relational-to-OWL-DL translation Section III E (1), Query Processing Section III F]
- Ontology-Assisted data transformation and integration. L.Zamboulis, A.Poulovassilis, J.Wang. Proc. ODBIS'08, Auckland, pp 29-36. At <http://www.dcs.bbk.ac.uk/~ap/pubs/odbis08.pdf> [you can skip Sections 2.1, 3.1, 4]

## **Question for you to consider:**

What are the similarities and dissimilarities of the three ways of accessing heterogeneous databases through an ontology, as illustrated by Case Studies E, F and G ?

# References

- A semantic web middleware for virtual data integration on the Web, A.Langegger et al, Proc. ESWC 2008, pp 493-507
- Linking data to ontologies, A.Poggi et al, Journal of Data Semantics, 2008, pp 133-173
- Bioinformatics service reconciliation by heterogeneous schema transformation, L.Zamboulis, N.Martin, A.Poulovassilis, Proc. DILS'07, Philadelphia, pp 89-104. At <http://www.dcs.bbk.ac.uk/~ap/pubs/dils07.pdf>