Ontology-Assisted Data Transformation and Integration

Lucas Zamboulis
School of Computer Science and Inf. Systems
Birkbeck, Univ. of London, WC1E 7HX, U.K.,
Benchmark Performance Ltd., CO16 9PT, U.K.
lucas@dcs.bbk.ac.uk

Alexandra Poulovassilis, Jianing Wang
School of Computer Science and Inf. Systems
Birkbeck, Univ. of London, WC1E 7HX, U.K.
{ap,jianing}@dcs.bbk.ac.uk

ABSTRACT

Schema-based data transformation and integration (DTI) has been an active research area for some time, while more recent advances in ontologies have led to significant research in ontology-based DTI. These two approaches present some overlaps and some differences, and in this paper we investigate possible synergies between them. In particular, we show how ontologies can enhance schema-based DTI approaches by providing richer semantics for schema constructs. We also illustrate one way in which schema-based DTI approaches can be used together with ontology-based approaches in a heterogeneous data integration setting.

1. INTRODUCTION

Schema-based data transformation and integration (DTI) is a well-studied research area. Mappings between source and target schemas can be expressed using global-as-view (GAV), local-as-view (LAV), global-local-as-view (GLAV) or both-as-view (BAV) rules [14, 15, 17], and it is also possible to define data-level mappings [2]. Mappings can be generated either manually or semi-automatically using a variety of schema matching techniques [23, 24]. Depending on the mapping rules, one can use GAV, LAV or GLAV query processing techniques [9, 5] to answer queries posed on virtual integrated schemas using the data sources.

Similarly, ontologies too may need to be transformed or integrated, and this requires ontology matching and mapping [11, 6]. Relationships between ontologies can be expressed in a variety of ways [21], e.g. using first order logic rules, using the schema-based approaches mentioned above, or using mapping ontologies, whose instances are used to define possibly complex mappings between ontologies. After specifying such mappings, one can use ontology-based query answering techniques [20, 22] to answer queries posed on the target or integrated ontology.

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 approach has the advantage of describing the domain at a high-level of abstraction, separating users' knowledge of the domain from the source data, and also allows domain knowledge to be expressed as logical formalisms, allowing inference mechanisms and ultimately converting the integrated resource into a knowledgebase. The integration strategy may vary significantly, e.g. [22] first federates a set of relational data sources, and then provides GLAV mappings between the federated schema and the global ontology, while [20] translates database schemas to ontologies and provides first order logic mappings between these ontologies and a global ontology.

We argue that cross-fertilisation between schema-based and ontology-based DTI can be beneficial. In particular, in this paper we focus on the use of ontologies to enhance schema-based approaches, firstly as a means of providing richer semantics for schema constructs and thus facilitating schema-based DTI, and secondly as a means of enabling schema-based DTI approaches to be used together with ontology-based ones.

Section 2 first gives an overview of the AutoMed schema-based DTI system, and then shows how the RDFS and OWL-DL languages can be represented within AutoMed. To date, AutoMed has been used extensively in schema-based DTI settings and so these extensions enable ongoing and future research into the synergies between schema-based and ontology-based approaches.

Section 3 describes firstly the integration of ontologies using AutoMed, and then schema-based transformation of data output by web services, assisted by semantic enrichment provided by mapping schemas to ontologies. As discussed in [1, 12], semantic enrichment of heterogeneous data sources as a means of facilitating their transformation and integration is desirable as it enhances scalability and reusability. However, this has not been explored in detail to date, with the exception of [3] and [28] in which web service input and output data are enriched with semantics provided by an ontology, thus facilitating matching and mapping generation between heterogeneous services.

Section 4 next describes the use of an ontology as an enriched interface to a relational virtual integrated resource. Compared with [22], our approach leverages existing schema-based data integration and query processing capabilities (in this case of AutoMed, but the approach is more generally applicable), while still allowing ontology query rewriting techniques such as those of [22] to be used to generate suitable sub-queries targeted at the virtual integrated relational

VLDB '08, August 24-30, 2008, Auckland, New Zealand

schema, to be evaluated using AutoMed's query processing capabilities, from a query posed on the ontology.

Section 5 gives our concluding remarks.

2. AUTOMED OVERVIEW

AutoMed (`www.doc.ic.ac.uk/automed`) is a heterogeneous DTI system which can handle virtual, materialised, and indeed hybrid data integration across multiple data models. It supports a hypergraph-based data model (the HDM) and provides facilities for specifying higher-level modelling languages in terms of this HDM (via the API of AutoMed's Model Definitions Repository). An HDM schema consists of a set of nodes, edges and constraints, and each modelling construct of a higher-level modelling language is specified as some combination of HDM nodes, edges and constraints (see [16]). For any modelling language, $\mathcal{M}$, specified in this way AutoMed provides a set of primitive schema transformations that can be applied to schema constructs expressed in $\mathcal{M}$. In particular, for every construct of $\mathcal{M}$ there is an `add` and a `delete` primitive transformation which add to/delete from a schema an instance of that construct. For those constructs of $\mathcal{M}$ which have textual names, there is also a `rename` primitive transformation.

Instances of modelling constructs within a particular schema are identified by means of their *scheme* enclosed within double chevrons $\langle\langle \ldots \rangle\rangle$. AutoMed schemas can be incrementally transformed by applying to them a sequence of primitive transformations, each adding, deleting or renaming just one schema construct (thus, in general, AutoMed schemas may contain constructs of more than one modelling language). A sequence of primitive transformations from one schema X_1 to another schema X_2 is termed a *transformation pathway* from X_1 to X_2. All source, intermediate, and integrated schemas, and the pathways between them, are stored in AutoMed's Schemas & Transformations Repository.

Each `add` and `delete` transformation is accompanied by a query specifying the extent of the added or deleted construct in terms of the rest of the constructs in the schema. This query is expressed in a comprehensions-based functional query language, IQL[1].

Also available are `extend` and `contract` primitive transformations which behave in the same way as `add` and `delete` except that they state that the extent of the new/removed construct cannot be precisely derived from the other constructs present in the schema. More specifically, each `extend` and `contract` transformation takes a pair of queries that specify a lower and an upper bound on the extent of the construct. The lower bound may be `Void` and the upper bound may be `Any`, which respectively indicate no known information about the lower or upper bound of the extent of the new construct.

Typically, a transformation pathway from a source schema X_1 to a target schema X_2 consists of a *growing phase*, in which schema constructs of X_2 that are missing from X_1 are added using `add` and `extend` transformations, followed by a *shrinking phase* in which schema constructs of X_1 not present in X_2 are removed using `delete` and `contract` transformations.

[1]Such languages subsume query languages such as SQL-92 and OQL in expressiveness [4]. IQL also provides a common query language for AutoMed that queries written in various high level query languages can be translated into and out of. Further details are given in [10].

The queries supplied with primitive transformations can be used to generate GAV, LAV or indeed GLAV mappings between source and target schemas, and to translate queries and data along a transformation pathway (see [17, 18, 19]). The queries supplied with primitive transformations also provide the necessary information for pathways to be automatically *reversible*, in that each `add`/`extend` transformation is reversed by a `delete`/`contract` transformation with the same arguments, while each `rename` is reversed by a `rename` with the two arguments swapped.

2.1 Representing Ontologies in AutoMed

We have extended AutoMed to support the RDFS [26], OWL-Lite and OWL-DL [25] languages. Below, we briefly describe the definitions of RDFS and OWL-DL in terms of AutoMed's HDM. The definition of OWL-Lite is a subset of that of OWL-DL and we therefore omit it.

Representing RDFS in the HDM:

- An RDFS class c is represented by a node in the HDM and is identified by the scheme $\langle\langle \mathsf{c} \rangle\rangle$.
- An RDFS property p linking two classes c_1 and c_2 is identified by the scheme $\langle\langle \mathsf{p}, \mathsf{c}_1, \mathsf{c}_2 \rangle\rangle$. In the HDM it is represented by an edge between nodes c_1 and c_2 and a cardinality constraint stating that each instance of c_1 is associated with precisely one instance of c_2 (HDM constraints can be specified in IQL). This representation in the HDM also captures implicitly the RDFS `rdfs:domain` and `rdfs:range` properties.
- Text in RDFS is represented by the `rdfs:Literal` construct. In the HDM, this is represented by a node and identified by the scheme $\langle\langle \mathsf{rdfs:Literal} \rangle\rangle$, of which there is one occurrence in any RDFS ontology.
- A subclass constraint in RDFS states that a class $\mathsf{c}_{\mathsf{sub}}$ is a subclass of another class $\mathsf{c}_{\mathsf{sup}}$. In the HDM, this is represented by a constraint stating that instances of $\mathsf{c}_{\mathsf{sub}}$ are also instances of $\mathsf{c}_{\mathsf{sup}}$, and identified by the scheme $\langle\langle \mathsf{rdfs:subClassOf}, \mathsf{c}_{\mathsf{sub}}, \mathsf{c}_{\mathsf{sup}} \rangle\rangle$.
- A subproperty constraint in RDFS states that a property $\mathsf{p}_{\mathsf{sub}}$ is a subproperty of another property $\mathsf{p}_{\mathsf{sup}}$. In the HDM, this is represented by a constraint stating that instances of $\mathsf{p}_{\mathsf{sub}}$ are also instances of $\mathsf{p}_{\mathsf{sup}}$, and identified by the scheme $\langle\langle \mathsf{rdfs:subPropertyOf}, \mathsf{p}_{\mathsf{sub}}, \mathsf{p}_{\mathsf{sup}} \rangle\rangle$.

Representing OWL-DL in the HDM:

- OWL-DL defines the class `owl:Thing` as a superclass of all classes. This is represented by a node in the HDM and identified by the scheme $\langle\langle \mathsf{owl:Thing} \rangle\rangle$, of which there is one occurrence in any OWL-DL ontology.
- Any other OWL-DL class c is also represented by a node in the HDM and is identified by the scheme $\langle\langle \mathsf{c} \rangle\rangle$. There is, in addition, an HDM constraint stating that all instances of c are also instances of `owl:Thing`.

  If c is a complex OWL-DL class, i.e. it is defined using other classes and set operators, there is also an HDM constraint specifying the extent of $\langle\langle \mathsf{c} \rangle\rangle$ with respect to these classes. For example, for class c_1 defined as the union of classes c_2 and c_3 using the `owl:unionOf` operator, the HDM constraint would be $\mathsf{c}_1 = (\mathsf{c}_2 \ \mathsf{union} \ \mathsf{c}_3)$.

- OWL-DL properties are represented in the same way as RDFS properties, and likewise for the `rdfs:Literal`, `rdfs:subClassOf` and `rdfs:subPropertyOf` constructs.

Finally, OWL-DL incorporates a large a number of constraints and we give below the representation of just one of these in the HDM. OWL-DL's other constraints are represented similarly.

- In OWL-DL a class c_1 may be asserted to be semantically identical to another class c_2. In the HDM this assertion is identified the scheme $\langle\langle \mathsf{owl:sameAs}, \mathsf{c}_1, \mathsf{c}_2 \rangle\rangle$ and is represented by two constraints, one stating that the instances of c_1 are also instances of c_2 and the other stating the converse.

3. ENRICHMENT AND TRANSFORMATION OF WEB SERVICE DATA

This section extends our earlier work in [28] by describing the use of multiple ontologies to enrich and transform web service data. This requires each service input/output to be mapped to a suitable ontology, and transformation pathways to be defined between the different ontologies to which service inputs/outputs are mapped. This use of multiple ontologies is discussed in detail here for the first time, as is the independence of our approach from the ontology language employed and its ability to handle multiple ontology languages concurrently. Section 3.1 discusses the integration of heterogeneous ontologies using AutoMed. Section 3.2 describes the semantic enrichment of services from different systems using different ontologies. Section 3.3 discusses matching and mapping generation for the enriched services, and finally data translation between them.

We illustrate this via an application in lifelong learning, MyPlan. The MyPlan project (`www.lkl.ac.uk/research/myplan`) aims to develop models of learners and to support them in planning their lifelong learning. One goal of MyPlan is to facilitate interoperability in a scalable fashion between existing systems targeted at the lifelong learner. Since direct access to these systems' repositories is in general not possible, an approach based on reconciling and combining the services the systems provide is being explored.

For our running example here, suppose we need to transfer learners' data from the L4All (`www.lkl.ac.uk/research/l4all`) system to the eProfile (`www.schools.bedfordshire.gov.uk/im/EProfile`) system. Each system is accompanied by an ontology. L4All uses the L4ALL RDFS ontology, developed specifically for the L4All system, while eProfile uses the Friend-Of-A-Friend OWL-DL[2] ontology (`www.foaf-project.org`). A Lifelong Learning Ontology, LLO (defined in OWL-DL), has also been developed as part of the MyPlan project which aims to encompass all concepts relating to lifelong learners. Figure 1 illustrates a portion of each of these ontologies.

Suppose now we need to transform the output of a service S_1 which retrieves data about a learner from L4All, to become the input of a service S_2 which inserts data about that learner into eProfile. Listed below are a sample output from S_1:

```
<user>
   <userID>John</userID>
   <fullname>John Smith</fullname>
   <age>1970</age> <gender>F</gender>
   <email>JohnS@bbk.ac.uk</email>
   <travel>15</travel> <location>London</location>
   <occupation>Technology Professional</occupation>
   <qual><![CDATA[PhD]]></qual>
   <skills><![CDATA[write good reports]]></skills>
   <interests><![CDATA[Sport]]></interests>
</user>
```

and a sample input for S_2:

```
<eProfile>
   <accountName>Mike2008</accountName>
   <mbox>Mike2008@yahoo.com</mbox>
   <name>Mike Jonson</name>
   <interest>sport</interest>
</eProfile>
```

Our approach (see Figure 2) is to (1) integrate ontologies L4ALL and FOAF into the global ontology LLO by means of the appropriate transformation pathways, (2) automatically extract XML schemas X_1 and X_2 for the output and input of services S_1 and S_2, respectively, (3) enrich these schemas using the L4ALL and FOAF ontologies, producing schemas X_1' and X_2', and (4) automatically transform X_1' into X_2'.

The result of this process is a transformation pathway $X_1 \leftrightarrow X_1' \leftrightarrow X_2' \leftrightarrow X_2$, which can then be used at run-time by the MyPlan service broker to automatically generate data compliant with service S_2 from data output by service S_1. We discuss steps (1)-(4) in more detail next.

Note that the XML documents consumed/produced by services may conform to a DTD or XML Schema, or may not be accompanied by a schema at all. For this reason, in step (2) above, an XMLDSS schema (see Section 3.2) is automatically extracted either from an accompanying DTD or XML Schema, or, if a schema does not exist, from sample input/output XML documents provided for the services.

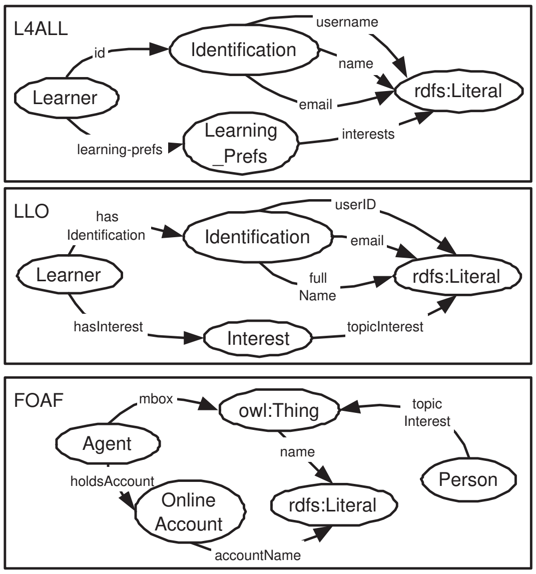

Figure 1: Ontologies L4ALL, LLO and FOAF.

3.1 Integrating Ontologies using AutoMed

The integration of the L4ALL and FOAF ontologies with LLO using AutoMed requires the creation of transformation pathways L4ALL→LLO and FOAF→LLO. L4ALL and

[2]FOAF is OWL-Full, but we only use its OWL-DL subset here.

Table 1: Fragment of the transformation pathway L4ALL→LLO→FOAF

. . . add steps for L4ALL→LLO. . .
① delete($\langle\langle$l4 : id, l4 : Learner, l4 : Identification$\rangle\rangle$,$\langle\langle$llo : hasIdentification, llo : Learner, llo : Identification$\rangle\rangle$)
② delete($\langle\langle$l4 : learning − prefs, l4 : Learner, l4 : Learning_Prefs$\rangle\rangle$,$\langle\langle$llo : hasInterest, llo : Learner, llo : Interest$\rangle\rangle$)
③ delete($\langle\langle$l4 : interests, l4 : Learning_Prefs, rdfs : Literal$\rangle\rangle$,$\langle\langle$llo : topicInterest, llo : Interest, rdfs : Literal$\rangle\rangle$)
④ delete($\langle\langle$l4 : Learner$\rangle\rangle$,$\langle\langle$llo : Learner$\rangle\rangle$)
⑤ delete($\langle\langle$l4 : email, l4 : Identification, rdfs : Literal$\rangle\rangle$,$\langle\langle$llo : email, llo : Identification, rdfs : Literal$\rangle\rangle$)
⑥ delete($\langle\langle$l4 : username, l4 : Identification, rdfs : Literal$\rangle\rangle$,$\langle\langle$llo : userID, llo : Identification, rdfs : Literal$\rangle\rangle$)
⑦ delete($\langle\langle$l4 : name, l4 : Identification, rdfs : Literal$\rangle\rangle$,$\langle\langle$llo : fullName, llo : Identification, rdfs : Literal$\rangle\rangle$)
. . . more delete steps for L4ALL→LLO. . .
. . . extend steps for L4ALL→LLO. . .
. . . contract steps for LLO→FOAF. . .
. . . add steps for LLO→FOAF. . .
. . . extend steps for LLO→FOAF. . .
⑧ delete($\langle\langle$llo : userID, llo : Identification, rdfs : Literal$\rangle\rangle$,[{ha, lit}|{ag, oa} ← $\langle\langle$foaf : holdsAccount, foaf : Agent, foaf : OnlineAccount$\rangle\rangle$; {oa, lit} ← $\langle\langle$foaf : accountName, foaf : OnlineAccount, rdfs : Literal$\rangle\rangle$])
⑨ delete($\langle\langle$llo : fullName, llo : Identification, rdfs : Literal$\rangle\rangle$,[{t, lit}|{t, lit} ← $\langle\langle$foaf : name, owl : Thing, rdfs : Literal$\rangle\rangle$; member t $\langle\langle$foaf : Agent$\rangle\rangle$])
⑩ delete($\langle\langle$llo : email, llo : Identification, rdfs : Literal$\rangle\rangle$,[{y, z}|{x, y, z} ← (generateProperty $\langle\langle$foaf : mbox, foaf : Agent, rdfs : Literal$\rangle\rangle$)])
⑪ delete($\langle\langle$llo : hasIdentification, llo : Learner, llo : Identification$\rangle\rangle$,[{x, y}|{x, y} ← (generateProperty $\langle\langle$foaf : Agent$\rangle\rangle$)])
⑫ delete($\langle\langle$llo : topicInterest, llo : Interest, rdfs : Literal$\rangle\rangle$, [{y, z}|{x, y, z} ← (generateProperty $\langle\langle$foaf : topic_interest, foaf : Person, owl : Thing$\rangle\rangle$)])
⑬ delete($\langle\langle$llo : hasInterest, llo : Learner, llo : Interest$\rangle\rangle$,[{x, y}|{x, y, z} ← (generateProperty $\langle\langle$foaf : topic_interest, foaf : Person, owl : Thing$\rangle\rangle$)])
⑭ delete($\langle\langle$llo : Interest$\rangle\rangle$,[x|x ← (generateClass $\langle\langle$foaf : topic_interest, foaf : Person, owl : Thing$\rangle\rangle$)])
⑮ delete($\langle\langle$llo : Learner$\rangle\rangle$,$\langle\langle$foaf : Agent$\rangle\rangle$)
. . . more delete steps for LLO→FOAF. . .

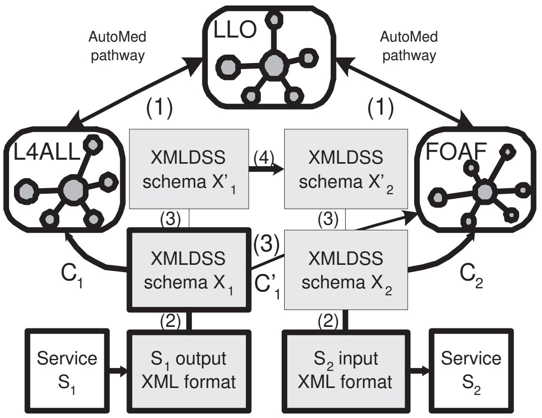

Figure 2: Reconciliation of Services S_1 and S_2.

LLO overlap significantly, but L4ALL is expressed in RDFS while LLO is expressed in OWL-DL. Both FOAF and LLO are expressed in OWL-DL, but FOAF is a general-purpose ontology while LLO targets lifelong learning.

Overcoming the modelling language heterogeneity problem between L4ALL and LLO is straightforward: each L4ALL RDFS construct is transformed into an equivalent OWL construct. For example, to replace the RDFS class $\langle\langle$l4 : Learner$\rangle\rangle$ with the equivalent OWL-DL class with the same name, the following transformations are applied to L4ALL:

add($\langle\langle$llo : Learner$\rangle\rangle$,$\langle\langle$l4 : Learner$\rangle\rangle$)
delete($\langle\langle$l4 : Learner$\rangle\rangle$,$\langle\langle$llo : Learner$\rangle\rangle$)

The first transformation above adds the OWL-DL construct $\langle\langle$llo : Learner$\rangle\rangle$ to L4ALL, specifying that it is equivalent to the RDFS construct $\langle\langle$l4 : Learner$\rangle\rangle$. This can then be deleted, specifying that it is equivalent to the OWL-DL construct $\langle\langle$llo : Learner$\rangle\rangle$. Note that, since a number of properties reference the $\langle\langle$l4 : Learner$\rangle\rangle$ class, in practice these would have to be deleted before deleting that class.

After translating the L4ALL ontology from RDFS to OWL-DL, its integration with LLO is completed by specifying the necessary `extend` transformations to "complete" L4ALL with those constructs from LLO that it is lacking. The upper part of Table 1 lists a fragment of the `delete` steps within the pathway L4ALL → LLO, as these will be referred to again in our running example.

Turning now to the integration of FOAF with LLO, the lower part of Table 1 lists a fragment of the `delete` steps within the pathway LLO → FOAF, as these will be referred to again — note that this pathway is the *reverse* of the pathway FOAF → LLO. We notice from ⑮ that $\langle\langle$llo : Learner$\rangle\rangle$ is equivalent to $\langle\langle$foaf : Agent$\rangle\rangle$. Since FOAF does not contain a class analogous to $\langle\langle$llo : Interest$\rangle\rangle$ in the LLO, in ⑭ we use an IQL function generateClass to generate as many instances of class $\langle\langle$llo : Interest$\rangle\rangle$ as there are instances of property $\langle\langle$foaf : topic_interest, foaf : Person, owl : Thing.$\rangle\rangle$. Similarly, the IQL function generateProperty generates the extent of a property. This function takes as input another property (if the property for which the extent is to be generated has a 1-n cardinality), or a class (if the property for which the extent is to be generated has a 1-1 cardinality). We also note that the LLO property userID maps to the join of FOAF properties holdsAccount and accountName (see ⑧). Finally, note that FOAF has a general-purpose name property, with domain and range owl:Thing and rdfs:Literal, respectively, whereas LLO only has a fullName property which is not general-purpose (see ⑨).

It should be stressed that AutoMed provides facilities for transforming/integrating schemas/ontologies by the specification of pathways between them that may be generated either manually (as here), or semi-automatically (as in Section 3.2) or automatically (as in Section 3.3). However, AutoMed does not (as yet) provide any facilities for verifying the correctness of such pathways.

Table 2: Correspondences between XMLDSS schema X_1 and the L4ALL Ontology

Construct:	Path:
⟨⟨user$1⟩⟩	[c|c ← ⟨⟨l4 : Learner⟩⟩]
⟨⟨userID$1⟩⟩	[id|{l, id} ← ⟨⟨l4 : id, l4 : Learner, l4 : Identification⟩⟩; {id, lit} ← ⟨⟨l4 : username, l4 : Identification, rdfs : Literal⟩⟩]
⟨⟨fullname$1⟩⟩	[id|{l, id} ← ⟨⟨l4 : id, l4 : Learner, l4 : Identification⟩⟩; {id, lit} ← ⟨⟨l4 : name, l4 : Identification, rdfs : Literal⟩⟩]
⟨⟨email$1⟩⟩	[id|{l, id} ← ⟨⟨l4 : id, l4 : Learner, l4 : Identification⟩⟩; {id, lit} ← ⟨⟨l4 : email, l4 : Identification, rdfs : Literal⟩⟩]
⟨⟨interests$1⟩⟩	[p|{l, p} ← ⟨⟨l4 : learning − prefs, l4 : Learner, l4 : Learning_Prefs⟩⟩; {p, lit} ← ⟨⟨l4 : interests, l4 : Learning_Prefs, rdfs : Literal⟩⟩]

Table 3: Correspondences between XMLDSS schema X_1 and the FOAF Ontology

Construct:	Path:
⟨⟨user$1⟩⟩	[c|c ← ⟨⟨foaf : Agent⟩⟩]
⟨⟨userID$1⟩⟩	[id|{l, id} ← (generateProperty⟨⟨foaf : Agent⟩⟩); {ag, oa} ← ⟨⟨foaf : holdsAccount, foaf : Agent, foaf : OnlineAccount⟩⟩; {oa, userlit} ← ⟨⟨foaf : accountName, foaf : OnlineAccount, rdfs : Literal⟩⟩]
⟨⟨fullname$1⟩⟩	[id|{l, id} ← (generateProperty⟨⟨foaf : Agent⟩⟩); id ← ⟨⟨foaf : Agent⟩⟩; {id, lit} ← ⟨⟨foaf : name, owl : Thing, rdfs : Literal⟩⟩]
⟨⟨email$1⟩⟩	[id|{l, id} ← (generateProperty⟨⟨foaf : Agent⟩⟩); {x, id, lit} ← (generateProperty⟨⟨foaf : mbox, foaf : Agent, rdfs : Literal⟩⟩)]
⟨⟨interests$1⟩⟩	[p|{l, p, z} ← (generateProperty⟨⟨foaf : topic$_i$nterest, foaf : Person, owl : Thing⟩⟩); {x, p, lit} ← (generateProperty⟨⟨foaf : topic$_i$nterest, foaf : Person, owl : Thing⟩⟩)]

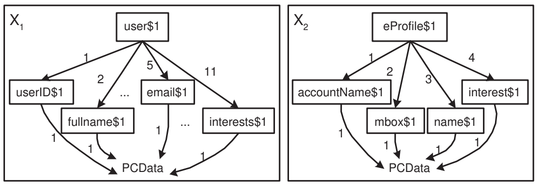

Figure 3: XMLDSS schemas X_1 and X_2.

3.2 XML Data Source Enrichment

XML data sources are represented in our approach using the XML DataSource Schema (XMLDSS) data model, which summarises the tree structure of XML documents, much like DataGuides [8]. An XMLDSS schema consists of four kinds of constructs: Element, Attribute, Text and NestList (see [28] for details of their specification in terms of the HDM). The last of these defines parent-child relationships either between two elements e_p and e_c or between an element e_p and the Text node. These are respectively identified by schemes of the form ⟨⟨i, e$_p$, e$_c$⟩⟩ and ⟨⟨i, e$_p$, Text⟩⟩, where i is the position of e_c or Text within the list of children of e_p. Note that, since the same name can be used for two or more elements with different semantics, element names are suffixed with $count, where count is incremented every time the same element name is encountered in a depth-first traversal of the schema. Figure 3 illustrates the XMLDSS schemas for the example documents given earlier in this section i.e. the output of service S_1 and the input of service S_2, respectively.

Our data source enrichment process requires the creation of a set of correspondences, C_1, between X_1 and its local L4ALL ontology, and another set of correspondences, C_2, between X_2 and its local FOAF ontology. As discussed in [28], a correspondence defines an Element, Attribute or NestList of an XMLDSS schema by means of an IQL query over a typed ontology (our correspondences are 'path-to-path' ones, in the terminology of [1]). In particular, an Element may map either to a class ⟨⟨c⟩⟩; or to a path ending with a class-valued property of the form ⟨⟨p, c1, c2⟩⟩, or to a path ending with a literal-valued property ⟨⟨p, c, Literal⟩⟩; additionally, the correspondence may state that the instances of a class are constrained by membership in some subclass. An Attribute may map either to a literal-valued property or to a path ending with a literal-valued property. A NestList between an Element and the Text construct may correspond to literal-valued property or to a path ending with such a property. This type of correspondence is used for reconciling data type incompatibilities between the XMLDSS schema and the ontology. In addition to these 1-1 correspondences, we also support 1-n correspondences as follows. An Element/Attribute may map to more than one path over the ontology. In this case, n correspondences are required, each associating the same XMLDSS Element/Attribute to a different path over the ontology, and specifying an expression that determines the part of the extent of the Element/Attribute to which the correspondence applies. This expression is in general a select-project IQL query. We note that these extended correspondences are GLAV rules and, since the example presented here does not make use of them, we refer the reader to [28] for further details.

Table 2 lists some of the correspondences, C_1, between X_1 and L4ALL. The correspondences between X_2 and FOAF, C_2, are similar, but are not listed due to lack of space.

Using the correspondences C_2, it is possible to automatically transform schema X_2 into a schema X'_2 that is semantically enriched since its element names use terms from the FOAF ontology. For example, ⟨⟨eProfile$1⟩⟩ is renamed to ⟨⟨foaf : Agent⟩⟩ and ⟨⟨mbox⟩⟩ to ⟨⟨Agent.mbox.Literal⟩⟩.

In order to enrich also X_1 with respect to X_2's ontology, we can automatically reformulate each correspondence in C_1 using the transformation pathway L4ALL→LLO→FOAF of Table 1. The new set of correspondences, C'_1 (see Table 3), now links X_1 with FOAF, and so can be used to transform X_1 into an enriched schema X'_1 that uses terms from FOAF. As discussed in [28], there is a proviso here that the new set of correspondences C'_1 must conform syntactically to the correspondence format accepted by the enrichment process.

3.3 Ontology-Assisted Schema and Data Transformation

Resulting from the above data source enrichment process are schemas X'_1 and X'_2 that both use the terminology of FOAF, as well as pathways $X_1 \rightarrow X'_1$ and $X_2 \rightarrow X'_2$. However, this is not, in general, enough for transforming data from one data source to the other:

First, X_1' and X_2' may be structurally different, e.g. X_1' may use attributes rather than elements to store text. This is not the case in our running example and the reader is referred to our earlier work in [30, 28] for details of a schema restructuring algorithm (SRA) that automatically creates a transformation pathway between two structurally heterogeneous XMLDSS schemas, provided elements are named according to the same terminology.

Second, even though both XMLDSS schemas use the same terminology, element names may contain subtle differences, due to sub-class and sub-property constraints in the ontology. For example, this is the case with elements $\langle\langle \mathsf{email\$1} \rangle\rangle$ and $\langle\langle \mathsf{mbox\$1} \rangle\rangle$ from schemas X_1 and X_2, which were replaced by elements $\langle\langle \mathsf{foaf:Agent.foaf:mbox.rdfs:Literal\$1} \rangle\rangle$ and $\langle\langle \mathsf{foaf:Agent.foaf:mbox.owl:Thing\$1} \rangle\rangle$ in X_1' and X_2'. The SRA algorithm is able to use input that specifies an element in the source schema to be a sub-class or super-class of an element in the target and vice-versa. Deriving this input involves splitting each path name in its constituent parts and comparing the corresponding classes and properties. For example, given path names A.B.C and A'.B'.C', we compare A with A', B with B' etc., to derive whether $A \equiv A'$, $A \subseteq A'$ or $A \supseteq A'$. The SRA algorithm can handle equivalence and subsumption relationships between elements, as well as union (two elements from schema X_1' may correspond to a single element in X_2'). However, the algorithm does not handle intersection and so ignores cases where e.g. A and C are super-classes of A' and C', but B is a sub-property of B'.

The result of the above ontology-assisted data transformation process is a pathway $X_1' \to X_2'$. When this is composed with the pathway $X_1 \to X_1'$ and the reverse of the pathway $X_2 \to X_2'$ generated from the previous data source enrichment process, an overall pathway $X_1 \to X_1' \to X_2' \to X_2$ is obtained.

This pathway can now be used to automatically transform data that is structured according to X_1 to be structured according to X_2, using the algorithm of [29]. For example, the output from S_1 given earlier would be translated into the following X_2-compliant data:

```
<eProfile>
    <accountName>John</accountName>
    <mbox>JohnS@bbk.ac.uk</mbox>
    <name>John Smith</name>
    <interest>Sport</interest>
</eProfile>
```

4. ONTOLOGY-BASED ACCESS TO AN INTEGRATED RESOURCE

We turn now to our use of an ontology for accessing an integrated relational resource. This work forms part of the EU ASSIST project (see `assist.iti.gr`) for which three relational databases containing patients' medical data need to be integrated. A predefined OWL-DL ontology will provide a high-level representation of the integrated resource, to which user queries will be submitted.

[22] terms such a setting "ontology-based data access" and describes a solution whereby relational databases are first federated, and then GLAV mappings are specified between the federated database and the ontology. Given a query posed on the ontology, the GLAV mappings are used to generate SQL sub-queries submitted to the relational data sources for evaluation. We have implemented an alternative approach in the ASSIST project that leverages AutoMed's

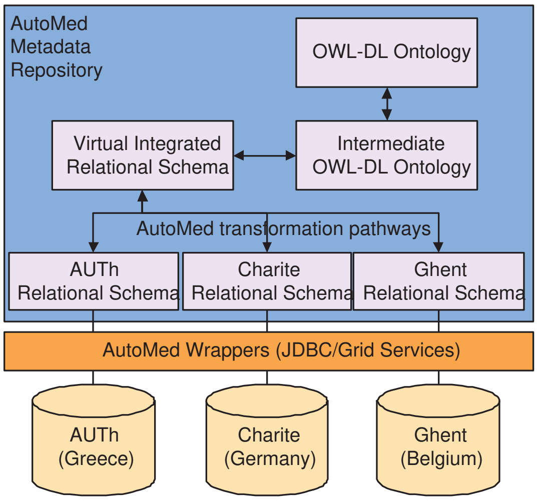

Figure 4: The ASSIST Integrated Resource.

existing schema-based data integration and query processing capabilities. In particular, we have first integrated the three relational databases into a virtual integrated relational schema. This schema is then automatically translated into an equivalent OWL-DL representation, and finally this is manually transformed into the predefined ASSIST OWL-DL ontology, enriched with appropriate medical expert knowledge. This architecture is illustrated in Figure 4.

In the rest of this section, Section 4.1 briefly discusses the integration of the relational databases under a virtual relational schema, while Section 4.2 describes the relational-to-OWL-DL translation algorithm.

4.1 Integrating the relational data sources

Reference [16] discusses how the relational data model can be encoded in the HDM, [17] gives several relational data transformation/integration examples, while [27] discusses a large scale integration of several relational proteomics databases using AutoMed.

Briefly, a relation `R` is represented by an HDM node and identified by a scheme $\langle\langle \mathsf{R} \rangle\rangle$; the extent of the node is the projection of `R` onto its primary key attributes. An attribute `a` of `R` is identified by a scheme $\langle\langle \mathsf{R}, \mathsf{a} \rangle\rangle$ and is represented by an HDM node (for the attribute) and an edge (between the relation and the attribute); the extent of the edge is the projection of `R` onto its primary key attributes plus `a` itself. There are also HDM constraint representations for primary keys and foreign keys. To illustrate, consider a relation from the virtual integrated schema of ASSIST, patient(<u>pid</u>,birthdate,visitId), where visitId references the primary key attribute, vid, of another relation, visit. Then, patient is represented in the HDM by a construct $\langle\langle \mathsf{patient} \rangle\rangle$, three constructs $\langle\langle \mathsf{patient}, \mathsf{pid} \rangle\rangle$, $\langle\langle \mathsf{patient}, \mathsf{birthdate} \rangle\rangle$ and $\langle\langle \mathsf{patient}, \mathsf{visitID} \rangle\rangle$, a primary key construct $\langle\langle \mathsf{patient\_pk}, \mathsf{patient}, \langle\langle \mathsf{patient}, \mathsf{pid} \rangle\rangle \rangle\rangle$, and a foreign key construct $\langle\langle \mathsf{patient\_fk}, \mathsf{patient}, \langle\langle \mathsf{patient}, \mathsf{visitId} \rangle\rangle, \mathsf{visit}, \langle\langle \mathsf{visit}, \mathsf{vid} \rangle\rangle \rangle\rangle$.

To illustrate the transformation of the ASSIST data sources into the virtual integrated schema, the following example shows how patient data is sourced from the AUTh database:

add(⟨⟨patient⟩⟩,[{'assist.auth.gr : patient', t}|t ← ⟨⟨patInfo⟩⟩])
add(⟨⟨patient, birthdate⟩⟩,[{{'assist.auth.gr : patInfo', t}, b}|
{t, b} ← ⟨⟨patInfo, BirthDate⟩⟩])
add(⟨⟨patient, visitId⟩⟩,[{{'assist.auth.gr : patInfo', t}, v}|
{t, v} ← ⟨⟨patInfo, visit⟩⟩])

4.2 Translation into OWL

The translation of the relational integrated schema into an equivalent OWL representation is undertaken using an algorithm based on [13], which describes the representation of relational databases in RDF. Similarly to [13], our translation of relational schemas into OWL can support both single-attribute and composite primary and foreign keys.

The algorithm, listed in Panel 1, takes an AutoMed relational schema S_{Rel} as input and outputs an AutoMed OWL schema S_{Ont}. The algorithm has three parts. The first part (lines 2–8), translates the relations of schema S_{Rel}. In particular, a relation ⟨⟨R⟩⟩ translates to a Class C in S_{Ont}, each of its attributes ⟨⟨R, a⟩⟩ translates to a Property ⟨⟨a, C, rdfs : Literal⟩⟩, while the primary key of ⟨⟨R⟩⟩ translates into another Class ⟨⟨$\mathsf{C_{pk}}$⟩⟩ and a Property ⟨⟨pk, C, $\mathsf{C_{pk}}$⟩⟩. The second part (lines 9–17), translates the foreign key constraints of schema S_{Rel}. In particular, the algorithm creates two Class constructs, ⟨⟨$\mathsf{C_{R_{fk}}}$⟩⟩ and ⟨⟨$\mathsf{C_{S_{fk}}}$⟩⟩, representing the set of attributes of relation R and the set of attributes of relation S that reference the former. The algorithm also creates Property constructs ⟨⟨fk, $\mathsf{C_R}$, $\mathsf{C_{R_{fk}}}$⟩⟩, ⟨⟨fk, $\mathsf{C_S}$, $\mathsf{C_{S_{fk}}}$⟩⟩ and ⟨⟨fk, $\mathsf{C_{S_{fk}}}$, $\mathsf{C_{R_{fk}}}$⟩⟩ that link the newly added Class constructs together with each other and with the Class constructs that represent relations R and S. The third part (line 18), which removes the relational schema constructs from schema S_{Ont} is straightforward and omitted.

Note that, as specified in the algorithm, the extent of a Class construct that represents a relation is generated by skolemising the extent of the corresponding relational construct. This is because all individuals in an ontology must be unique, and the values of a primary key of a relation are not necessarily unique across all values of all primary keys within a database. In our setting, which has the added requirement of uniqueness across data sources, we use IQL function getLSID that generates a tuple {sk, r} for each primary key value r, where sk is the LSID of relation ⟨⟨R⟩⟩. An LSID is a Life Sciences Research Uniform Resource Name (URN) specification that provides a standardised naming scheme for entities in the life sciences [7]. For example, the LSID `URN:LSID:assist.auth.gr.patients:126` refers to the row with primary key value 126 in table patients of the AUTh database — in this case, the LSID issuing authority is `assist.auth.gr`. The generality of the LSID naming scheme has rendered it useful in domains outside the life sciences as well.

The extent of a Property construct that represents an attribute is generated similarly, i.e. each tuple is of the form {{sk, r}, a}, where {sk, r} is generated as above, and a is the attribute value.

Note also that, although primary and foreign key constructs are modelled as constraints in the HDM representation of the relational data model, the corresponding constructs in the HDM representation of the OWL data model are extensional constructs, in the spirit of [13].

Referring to the example in Section 4.1, relation patient(id, birthdate, visitId) is represented in the OWL schema with a Class construct ⟨⟨patient⟩⟩, and one Property construct per attribute, ⟨⟨id, patient, rdfs : Literal⟩⟩, ⟨⟨birthdate,patient, rdfs:Literal⟩⟩ and ⟨⟨visitId, patient, rdfs : Literal⟩⟩. The primary key of the relation is represented with Class ⟨⟨patient_pk⟩⟩ and Property ⟨⟨patient_has_pk, patient, patient_pk⟩⟩, and the foreign key between relations patient and visit is represented with Class constructs ⟨⟨patient_visit_fk⟩⟩, ⟨⟨visit_patient_fk⟩⟩ and Property constructs ⟨⟨patient_has_fk, patient, patient_visit_fk⟩⟩, ⟨⟨visit_has_fk, visit, visit_patient_fk⟩⟩, and ⟨⟨patient_fk_visit, patient_visit_fk,visit_patient_fk⟩⟩.

After automatically translating the virtual integrated relational schema into an OWL-DL ontology, we manually transform this ontology into the predefined ASSIST OWL-DL domain ontology — see Figure 4.

Panel 1: Relational-to-OWL Translation

Input: AutoMed Relational Schema S_{Rel}
Output: AutoMed OWL Schema S_{Ont}
1 Copy S_{Rel} to S_{Ont}
2 Add class ⟨⟨rdfs : Literal⟩⟩ to S_{Ont}
3 **for** each relation R in S_{Rel} **do**
4 Add class ⟨⟨C⟩⟩ to S_{Ont} and populate its extent using query [getLSID ⟨⟨R⟩⟩ r|r ← ⟨⟨R⟩⟩]
5 **for** each attribute a of R **do**
6 Add property ⟨⟨a, C, rdfs : Literal⟩⟩ to S_{Ont} and populate its extent using query [{(getLSID ⟨⟨R⟩⟩ r), a}|{r, a} ← ⟨⟨R, a⟩⟩]
7 Add class ⟨⟨$\mathsf{C_{pk}}$⟩⟩ to S_{Ont} and populate its extent using query [getLSID ⟨⟨R⟩⟩ r|r ← ⟨⟨R⟩⟩]
8 Add property ⟨⟨pk, C, $\mathsf{C_{pk}}$⟩⟩ and populate its extent using query [{(getLSID ⟨⟨R⟩⟩ r), (getLSID ⟨⟨R⟩⟩ r)}|r ← ⟨⟨R⟩⟩]
9 **for** each relation R in S_{Rel} **do**
10 **for** each foreign key with label fk identifying attributes a_i of R being referenced by attributes b_i of S ($1 \leq i \leq n$) **do**
11 Let Q_1 be [{r, {$\mathsf{a_1}, \ldots, \mathsf{a_i}, \ldots, \mathsf{a_n}$}}|r ← ⟨⟨R⟩⟩; {r, $\mathsf{a_1}$} ← ⟨⟨R, $\mathsf{a_1}$⟩⟩; ... ; {r, $\mathsf{a_i}$} ← ⟨⟨R, $\mathsf{a_i}$⟩⟩ ... ; {r, $\mathsf{a_n}$} ← ⟨⟨R, $\mathsf{a_n}$⟩⟩]
12 Let Q_2 be [{s, {$\mathsf{b_1}, \ldots, \mathsf{b_i}, \ldots, \mathsf{b_n}$}}|s ← ⟨⟨S⟩⟩; {s, $\mathsf{b_1}$} ← ⟨⟨S, $\mathsf{b_1}$⟩⟩; ... ; {s, $\mathsf{b_i}$} ← ⟨⟨S, $\mathsf{b_i}$⟩⟩ ... ; {s, $\mathsf{b_n}$} ← ⟨⟨S, $\mathsf{b_n}$⟩⟩]
13 Add class ⟨⟨$\mathsf{C_{R_{fk}}}$⟩⟩ to S_{Ont} and populate its extent using query [(getLSID ⟨⟨R⟩⟩ cr)|{r, cr} ← $\mathsf{Q_1}$]
14 Add class ⟨⟨$\mathsf{C_{S_{fk}}}$⟩⟩ to S_{Ont} and populate its extent using query [(getLSID ⟨⟨S⟩⟩ cs)|{s, cs} ← $\mathsf{Q_2}$]
15 Add property ⟨⟨fk, $\mathsf{C_R}$, $\mathsf{C_{R_{fk}}}$⟩⟩ to S_{Ont} and populate its extent using query Q_1
16 Add property ⟨⟨fk, $\mathsf{C_S}$, $\mathsf{C_{S_{fk}}}$⟩⟩ to S_{Ont} and populate its extent using query Q_2
17 Add property ⟨⟨fk, $\mathsf{C_{S_{fk}}}$, $\mathsf{C_{R_{fk}}}$⟩⟩ to S_{Ont} and populate its extent using query [{(getLSID ⟨⟨R⟩⟩ cs), (getLSID ⟨⟨S⟩⟩ cs)}|{s, cs} ← $\mathsf{Q_2}$]
18 deleteRelationalConstructs(S_{Ont})

5. CONCLUDING REMARKS

In this paper we have discussed two possible synergies between schema-based and ontology-based approaches to data transformation/integration (DTI). Firstly, we have discussed the transformation of heterogeneous XML data by using multiple ontologies as a 'semantic bridge' between them. This entails first integrating the ontologies using AutoMed, then enriching the XML data sources with semantics provided by the ontologies, and then automatically undertaking ontology-assisted data restructuring. This functionality is currently being deployed in order to support the interoperability of different lifelong learning systems within the MyPlan project.

Secondly, we have presented an approach to ontology-based access of relational data sources that leverages AutoMed's schema-based DTI and query processing capabilities. This is currently being used to support the integration of heterogeneous medical databases under a predefined ontology in the ASSIST project. Although illustrated in the context of AutoMed, our approach is more generally applicable to other schema-based DTI and query processing systems, and would allow their capabilities to be combined with ontology-based query rewriting and evaluation techniques.

6. ACKNOWLEDGMENTS

This research has been partly funded by the MyPlan JISC project and the ASSIST EU FP7 project. The authors would like to thank Hassan Baajour and Dikaios Papadogkonas for their insight into the semantics of the MyPlan and the ASSIST data sources, respectively.

7. REFERENCES

[1] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-based integration of XML web resources. In *Proc. ISWC'02*, pages 117–131, 2002.

[2] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. WebDB'02*, pages 89–94, 2002.

[3] S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Proc. DILS'04*, pages 1–16, 2004.

[4] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.

[5] A. Cali. Query answering by rewriting in GLAV data integration systems under constraints. In *Proc. SWDB'04*, pages 167–184, 2004.

[6] N. Choi, I. Song, and H. Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.

[7] T. Clark, S. Martin, and T. Liefeld. Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, 5(1):59–70, 2004.

[8] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. VLDB'97*, pages 436–445, 1997.

[9] A. Y. Halevy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS'95*, pages 95–104, 2002.

[10] E. Jasper, A. Poulovassilis, L. Zamboulis, and H. Fan. Processing IQL queries and migrating data in the AutoMed toolkit. AutoMed Technical Report 20, 2006.

[11] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[12] L. V. S. Lakshmanan and F. Sadri. XML interoperability. In *Proc. WebDB'03*, pages 19–24, 2003.

[13] G. Lausen. Relational databases in RDF. In *Proc. Joint ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases*, 2007.

[14] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS02*, pages 233–246, 2002.

[15] J. Madhavan and A. Y. Halevy. Composing mappings among data sources. In *Proc. VLDB'03*, pages 572–583, 2003.

[16] P. J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, pages 333–348, 1999.

[17] P. J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238, 2003.

[18] P. J. McBrien and A. Poulovassilis. Defining Peer-to-Peer Data Integration using Both as View Rules. In *Proc. DBISP2P'03 (at VLDB'03)*, pages 91–107, 2003.

[19] P. J. McBrien and A. Poulovassilis. P2P query reformulation over Both-as-View data transformation rules. In *Proc. DBISP2P'06 (at VLDB'06)*, pages 310–322, 2006.

[20] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[21] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[22] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[23] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[24] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics*, IV:146–171, 2005.

[25] W3C. OWL Web Ontology Language Reference. Available at http://www.w3.org/TR/owl-ref, February 2004.

[26] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Available at http://www.w3.org/TR/rdf-schema, February 2004.

[27] L. Zamboulis, H. Fan, K. Belhajjame, J. Siepen, A. Jones, N. J. Martin, A. Poulovassilis, S. Hubbard, S. M. Embury, and N. W. Paton. Data access and integration in the ISPIDER proteomics Grid. In *Proc. DILS'06*, pages 3–18, 2006.

[28] L. Zamboulis, N. J. Martin, and A. Poulovassilis. Bioinformatics service reconciliation by heterogeneous schema transformation. In *Proc. DILS'07*, pages 89–104, 2007.

[29] L. Zamboulis and A. Poulovassilis. Using AutoMed for XML data transformation and integration. In *Proc. DIWeb'04 (at CAiSE'04)*, pages 58–69, 2004.

[30] L. Zamboulis and A. Poulovassilis. Information sharing for the Semantic Web - a schema transformation approach. In *Proc. DISWeb'06 (at CAiSE'06)*, pages 275–289, 2006.