

# Ontology-Assisted Data Transformation and Integration

Lucas Zamboulis, Alexandra Poulouvasilis, Jianing Wang  
School of Computer Science and Information Systems,  
Birkbeck, University of London, WC1E 7HX, London, U.K.  
{lucas,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 G global-as-view (GAV), local-as-view (LAV), global-local-as-view (GLAV) or both-as-view (BAV) rules [14, 16, 18], 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 [25, 26]. Depending on the mapping rules, one can use GAV, LAV or GLAV query processing techniques [15, 5] to process queries posed on virtual integrated schemas that are answered by accessing the source data.

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 [23], 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 [22, 24] to answer queries posed on the target or integrated ontology.

We argue that collaboration between these two overlapping research areas can be beneficial. In particular, this

paper focuses on the use of ontologies together with 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 gives a brief overview of the AutoMed schema-based data transformation/integration system, and then shows how RDFS and OWL-DL 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 discusses firstly the integration of ontologies using AutoMed, and then the assistance of schema-based data transformation through the semantic enrichment provided by ontologies. Section 4 describes the use of an ontology as an enriched interface to a relational virtual integrated resource. Section 5 gives our concluding remarks and directions of future work.

## 2. AUTOMED OVERVIEW

AutoMed ([www.doc.ic.ac.uk/automed](http://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 [17]). 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  $\langle\langle \dots \rangle\rangle$ . 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*

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '08, August 24-30, 2008, Auckland, New Zealand  
Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

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<sup>1</sup>.

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.

The queries supplied with primitive transformations can be used to generate GAV, LAV or indeed GLAV mappings, and to translate queries and data along a transformation pathway (see [18, 19, 20]). 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 [29], OWL-Lite and OWL-DL [28] 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  $\langle\langle c \rangle\rangle$ .
- An RDFS property  $p$  linking two classes  $c_1$  and  $c_2$  is identified by the scheme  $\langle\langle p, c_1, c_2 \rangle\rangle$ . 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 the IQL query language). 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  $\langle\langle rdfs : Literal \rangle\rangle$ , of which there is one occurrence in any RDFS ontology.
- A subclass constraint in RDFS states that a class  $c_{sub}$  is a subclass of another class  $c_{sup}$ . In the HDM, this is represented by a constraint stating that instances

of  $c_{sub}$  are also instances of  $c_{sup}$ , and identified by the scheme  $\langle\langle rdfs : subclassOf, c_{sub}, c_{sup} \rangle\rangle$ .

- A subproperty constraint in RDFS states that a property  $p_{sub}$  is a subproperty of another property  $p_{sup}$ . In the HDM, this is represented by a constraint stating that instances of  $p_{sub}$  are also instances of  $p_{sup}$ , and identified by the scheme  $\langle\langle rdfs : subPropertyOf, p_{sub}, p_{sup} \rangle\rangle$ .

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  $\langle\langle owl : Thing \rangle\rangle$ , 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  $\langle\langle c \rangle\rangle$ . 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  $\langle\langle c \rangle\rangle$  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  $c_1 = (c_2 \text{ union } c_3)$ .
- 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  $\langle\langle owl : sameAs, c_1, c_2 \rangle\rangle$  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

The semantic enrichment of heterogeneous data sources as a means of facilitating their transformation and integration is highly desirable, as it enhances scalability and reusability [1, 12]. This paradigm has not been explored in detail, with the exception of [3] and [31] in which web service inputs and outputs are enriched with semantics provided by an ontology, thus facilitating matching and mapping generation between heterogeneous services.

This section extends our earlier work in [31] by describing the use of multiple ontologies to enrich and transform web service data, and by showing the independence of our approach from the ontology language employed and also its ability to handle multiple ontology languages at the same time. 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.

<sup>1</sup>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].

We illustrate this via an application in lifelong learning, MyPlan. MyPlan ([www.lkl.ac.uk/research/myplan](http://www.lkl.ac.uk/research/myplan)) is a project aiming 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' databases is in general not possible, an approach that relies on reconciling and combining the services they 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](http://www.lkl.ac.uk/research/l4all)) system to the eProfile ([www.schools.bedfordshire.gov.uk/im/EProfile](http://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<sup>2</sup> ontology ([www.foaf-project.org](http://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, illustrated in Figure 2, is to (1) integrate ontologies L4ALL and FOAF into the global ontology LLO by means of the appropriate AutoMed 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 an AutoMed 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 that is compliant with service  $S_2$  from data output by service  $S_1$ . We discuss steps (1)-(4) in more detail next.

<sup>2</sup>FOAF is OWL-Full, but we only use its OWL-DL subset here.

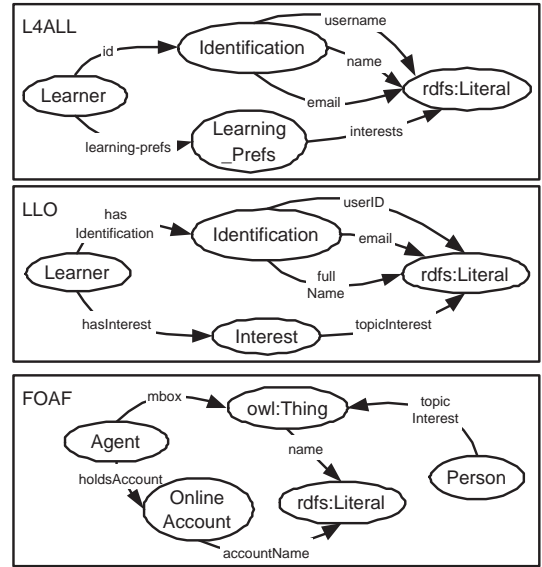


Figure 1: Ontologies L4ALL, LLO and FOAF.

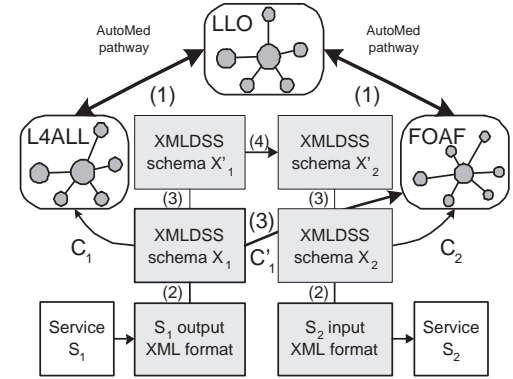


Figure 2: Reconciliation of Services  $S_1$  and  $S_2$ .

### 3.1 Integrating Ontologies using AutoMed

The integration of the L4ALL and FOAF ontologies with the LLO ontology using AutoMed requires the creation of two transformation pathways L4ALL $\rightarrow$ LLO and FOAF $\rightarrow$ LLO. L4ALL and 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 using AutoMed is straightforward: each one of the L4ALL RDFS constructs is transformed into an equivalent OWL construct. For example, to replace the RDFS class  $\langle\langle I4 : \text{Learner} \rangle\rangle$  with the equivalent OWL-DL class with the same name, the following transformations are applied to L4ALL:

```
add( $\langle\langle Ilo : \text{Learner} \rangle\rangle, \langle\langle I4 : \text{Learner} \rangle\rangle$ )
delete( $\langle\langle I4 : \text{Learner} \rangle\rangle, \langle\langle Ilo : \text{Learner} \rangle\rangle$ )
```

The first transformation above adds OWL-DL construct  $\langle\langle Ilo : \text{Learner} \rangle\rangle$  to L4ALL, specifying that it is equivalent to

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

---

... add steps for L4ALL→LLO...

① delete( $\langle\langle I4 : id, I4 : Learner, I4 : Identification \rangle\rangle, \langle\langle Ilo : hasIdentification, Ilo : Learner, Ilo : Identification \rangle\rangle$ )

② delete( $\langle\langle I4 : learning - prefs, I4 : Learner, I4 : Learning_Prefs \rangle\rangle, \langle\langle Ilo : hasInterest, Ilo : Learner, Ilo : Interest \rangle\rangle$ )

③ delete( $\langle\langle I4 : interests, I4 : Learning_Prefs, rdfs : Literal \rangle\rangle, \langle\langle Ilo : topicInterest, Ilo : Interest, rdfs : Literal \rangle\rangle$ )

④ delete( $\langle\langle I4 : Learner \rangle\rangle, \langle\langle Ilo : Learner \rangle\rangle$ )

⑤ delete( $\langle\langle I4 : email, I4 : Identification, rdfs : Literal \rangle\rangle, \langle\langle Ilo : email, Ilo : Identification, rdfs : Literal \rangle\rangle$ )

⑥ delete( $\langle\langle I4 : username, I4 : Identification, rdfs : Literal \rangle\rangle, \langle\langle Ilo : userID, Ilo : Identification, rdfs : Literal \rangle\rangle$ )

⑦ delete( $\langle\langle I4 : name, I4 : Identification, rdfs : Literal \rangle\rangle, \langle\langle Ilo : fullName, Ilo : Identification, rdfs : Literal \rangle\rangle$ )

... 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( $\langle\langle Ilo : userID, Ilo : Identification, rdfs : Literal \rangle\rangle, [\{ha, lit\} \{ag, oa\} \leftarrow \langle\langle foaf : holdsAccount, foaf : Agent, foaf : OnlineAccount \rangle\rangle; \{oa, lit\} \leftarrow \langle\langle foaf : accountName, foaf : OnlineAccount, rdfs : Literal \rangle\rangle]$ )

⑨ delete( $\langle\langle Ilo : fullName, Ilo : Identification, rdfs : Literal \rangle\rangle, [\{t, lit\} \{t, lit\} \leftarrow \langle\langle foaf : name, owl : Thing, rdfs : Literal \rangle\rangle; member t \langle\langle foaf : Agent \rangle\rangle]$ )

⑩ delete( $\langle\langle Ilo : email, Ilo : Identification, rdfs : Literal \rangle\rangle, [\{y, z\} \{x, y, z\} \leftarrow (generateProperty \langle\langle foaf : mbox, foaf : Agent, rdfs : Literal \rangle\rangle)]$ )

⑪ delete( $\langle\langle Ilo : hasIdentification, Ilo : Learner, Ilo : Identification \rangle\rangle, [\{x, y\} \{x, y\} \leftarrow (generateProperty \langle\langle foaf : Agent \rangle\rangle)]$ )

⑫ delete( $\langle\langle Ilo : topicInterest, Ilo : Interest, rdfs : Literal \rangle\rangle, [\{y, z\} \{x, y, z\} \leftarrow (generateProperty \langle\langle foaf : topic\_interest, foaf : Person, owl : Thing \rangle\rangle)]$ )

⑬ delete( $\langle\langle Ilo : hasInterest, Ilo : Learner, Ilo : Interest \rangle\rangle, [\{x, y\} \{x, y, z\} \leftarrow (generateProperty \langle\langle foaf : topic\_interest, foaf : Person, owl : Thing \rangle\rangle)]$ )

⑭ delete( $\langle\langle Ilo : Interest \rangle\rangle, [x]x \leftarrow (generateClass \langle\langle foaf : topic\_interest, foaf : Person, owl : Thing \rangle\rangle)]$ )

⑮ delete( $\langle\langle Ilo : Learner \rangle\rangle, \langle\langle foaf : Agent \rangle\rangle$ )

... more delete steps for LLO→FOAF...

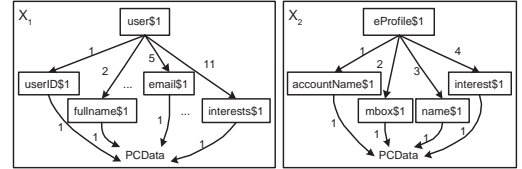
---

the RDFS construct  $\langle\langle I4 : Learner \rangle\rangle$ . The RDFS construct  $\langle\langle I4 : Learner \rangle\rangle$  can then be deleted, specifying that it is equivalent to the OWL-DL construct  $\langle\langle Ilo : Learner \rangle\rangle$ . Note that, since a number of properties reference the  $\langle\langle I4 : Learner \rangle\rangle$  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  $\langle\langle Ilo : Learner \rangle\rangle$  is equivalent to  $\langle\langle foaf : Agent \rangle\rangle$ . Since FOAF does not contain a class analogous to  $\langle\langle Ilo : Interest \rangle\rangle$  in the LLO, in ⑭ we use an IQL function `generateClass` to generate as many instances of class  $\langle\langle Ilo : Interest \rangle\rangle$  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 which may be generated



**Figure 3: XMLDSS schemas  $X_1$  and  $X_2$ .**

either manually (as here), or semi-automatically (as in 3.2 below) or automatically (as in 3.3). However, AutoMed does not (as yet) provide any facilities for verifying the correctness of such pathways.

### 3.2 XML Data Source Enrichment

XML data sources are represented in AutoMed using the XML DataSource Schema (XMLDSS) data model, which summarises the tree structure of XML documents, much like DataGuides [9]. An XMLDSS schema consists of four kinds of constructs: **Element**, **Attribute**, **Text** and **NestList** (see [31] 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  $\langle\langle i, e_p, e_c \rangle\rangle$  and  $\langle\langle i, e_p, Text \rangle\rangle$ , 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 two example documents given earlier in this section i.e. for 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

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

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

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

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

L4ALL ontology, and another set of correspondences,  $C_2$ , between  $X_2$  and its local FOAF ontology. As discussed in [31], 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  $\langle\langle c \rangle\rangle$ ; or to a path ending with a class-valued property of the form  $\langle\langle p, c1, c2 \rangle\rangle$ , or to a path ending with a literal-valued property  $\langle\langle p, c, \text{Literal} \rangle\rangle$ ; 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 [31] 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,  $\langle\langle\text{eProfile}\$1\rangle\rangle$  is renamed to  $\langle\langle foaf : \text{Agent} \rangle\rangle$  and  $\langle\langle\text{mbox}\rangle\rangle$  to  $\langle\langle\text{Agent.mbox.Literal}\rangle\rangle$ .

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  $L4ALL \rightarrow LLO \rightarrow FOAF$  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 [31], there is a proviso here that the new set of correspondences  $C'_1$  must conform syntactically to the correspondences accepted as input 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 [33, 31] for details of a schema restructuring algorithm (SRA) that automatically creates an AutoMed 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  $\langle\langle\text{email}\$1\rangle\rangle$  and  $\langle\langle\text{mbox}\$1\rangle\rangle$  from schemas  $X_1$  and  $X_2$ , which were replaced by elements  $\langle\langle foaf : \text{Agent.foaf : mbox.rdfs : Literal}\$1 \rangle\rangle$  and  $\langle\langle foaf : \text{Agent.foaf : mbox.owl : Thing}\$1 \rangle\rangle$  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  $X'_1 \rightarrow X'_2$ . When this is composed with the pathway  $X_1 \rightarrow X'_1$  and the reverse of the pathway

and  $X_2 \rightarrow X'_2$  generated from the previous data source enrichment process, an overall pathway  $X_1 \rightarrow X'_1 \rightarrow X'_2 \rightarrow X_2$  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 [32]. 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 DATA 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](http://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.

[24] advocates *ontology-based data access*, i.e. a setting where an ontology provides access to one or more data sources, as a means for describing a domain at a high-level of abstraction, separating users' knowledge of the domain from the data layer. Such a setting also has the advantage of allowing domain knowledge to be expressed in the form of logical formalisms, allowing inference mechanisms and ultimately converting a (possibly integrated) data source to a knowledgebase.

One of the systems that pioneered the interoperation of schema-based data sources with ontologies is TAMBIS [27]. This idea has been recently extended by [21], which also discusses the multiple roles of an ontology in such systems, i.e. a data source, a mediated schema and a system ontology.

In terms of ontology-based data access for relational data sources, OntoGrate [8] performs relational database integration and provides ontology-based data access with reasonable performance using query answering [22] by translating database schemas to ontologies and providing first order logic mappings between these ontologies and a global ontology. [24] follows a different integration strategy, by first federating all relational databases, then providing GLAV mappings between the federated schema and the global ontology. Given a query posed on the ontology, the GLAV mappings are used to generate SQL sub-queries that can be submitted to and evaluated by the relational data sources.

We have implemented an alternative approach in the ASSIST project that leverages AutoMed's existing schema-based data integration and distributed query processing capabilities. In particular, we have first integrated, using AutoMed, 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.

Compared with the approach of [24], our approach leverages existing schema-based data integration and query processing capabilities, while still allowing ontology query rewrites

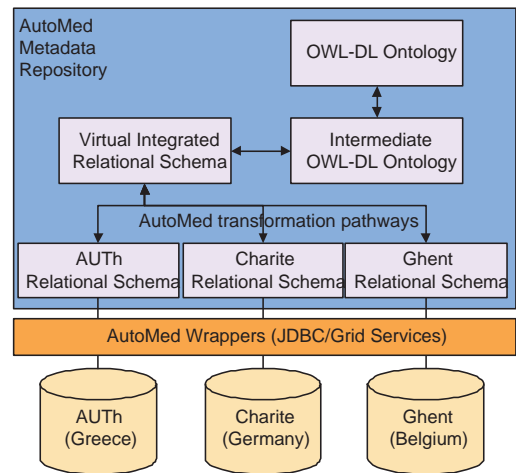


Figure 4: The ASSIST Integrated Resource.

ing techniques such as those of [24] to be used in order to generate from a query posed on the OWL-DL ontology suitable sub-queries targeted at the virtual integrated relational schema, which can then be evaluated using AutoMed's distributed query processing capabilities. Another advantage is that the transformation pathways from the relational data sources to the virtual integrated relational schema can be more complex than the GLAV mappings of [24], if required by the integration logic of the application.

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 [17] discusses how the relational data model can be encoded in the HDM, [18] gives several relational data transformation/integration examples, while [30] 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  $\langle\langle R \rangle\rangle$ ; 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  $\langle\langle R, a \rangle\rangle$  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,  $\text{patient}(\text{pid}, \text{birthdate}, \text{visitId})$ , where  $\text{visitId}$  references the primary key attribute,  $\text{vid}$ , of another relation,  $\text{visit}$ . Then,  $\text{patient}$  is represented in the HDM by a construct  $\langle\langle \text{patient} \rangle\rangle$ , three constructs  $\langle\langle \text{patient}, \text{pid} \rangle\rangle$ ,  $\langle\langle \text{patient}, \text{birthdate} \rangle\rangle$  and  $\langle\langle \text{patient}, \text{visitId} \rangle\rangle$ , a primary key construct  $\langle\langle \text{patient\_pk}, \text{patient}, \langle\langle \text{patient}, \text{pid} \rangle\rangle \rangle\rangle$ , and a foreign key construct  $\langle\langle \text{patient\_fk}, \text{patient}, \langle\langle \text{patient}, \text{visitId} \rangle\rangle, \text{visit}, \langle\langle \text{visit}, \text{vid} \rangle\rangle \rangle\rangle$ .

To illustrate the transformation of the ASSIST data sources into the virtual integrated schema, the following small example shows how  $\text{patient}$  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  $\langle R \rangle$  translates to a Class  $C$  in  $S_{Ont}$ , each of its attributes  $\langle R, a \rangle$  translates to a Property  $\langle a, C, rdfs : Literal \rangle$ , while the primary key of  $\langle R \rangle$  translates into another Class  $\langle C_{pk} \rangle$  and a Property  $\langle pk, C, C_{pk} \rangle$ . The second part (lines 9–17), translates the foreign key constraints of schema  $S_{Rel}$ . In particular, the algorithm creates two Class constructs,  $\langle C_{R_{fk}} \rangle$  and  $\langle C_{S_{fk}} \rangle$ , 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  $\langle fk, C_R, C_{R_{fk}} \rangle$ ,  $\langle fk, C_S, C_{S_{fk}} \rangle$  and  $\langle fk, C_{S_{fk}}, C_{R_{fk}} \rangle$  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  $S_{Ont}$  using the newly created ontology constructs to specify their extents, 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  $\langle R \rangle$ . 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  $\langle patient \rangle$ , and one Property construct per

---

### 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  $\langle rdfs : Literal \rangle$  to  $S_{Ont}$ 
3 for each relation  $R$  in  $S_{Rel}$  do
4   Add class  $\langle C \rangle$  to  $S_{Ont}$  and populate its extent using
   query  $[getLSID \langle R \rangle | r \leftarrow \langle R \rangle]$ 
5   for each attribute  $a$  of  $R$  do
6     Add property  $\langle a, C, rdfs : Literal \rangle$  to  $S_{Ont}$  and
     populate its extent using query
      $[ \{ (getLSID \langle R \rangle) r, a \} | \{ r, a \} \leftarrow \langle R, a \rangle ]$ 
7   Add class  $\langle C_{pk} \rangle$  to  $S_{Ont}$  and populate its extent using
   query  $[getLSID \langle R \rangle | r \leftarrow \langle R \rangle]$ 
8   Add property  $\langle pk, C, C_{pk} \rangle$  and populate its extent using
   query  $[ \{ (getLSID \langle R \rangle) r \} | r \leftarrow \langle R \rangle ]$ 
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
     $[ \{ r, \{ a_1, \dots, a_i, \dots, a_n \} \} | r \leftarrow \langle R \rangle; \{ r, a_1 \} \leftarrow$ 
     $\langle R, a_1 \rangle; \dots; \{ r, a_i \} \leftarrow \langle R, a_i \rangle; \dots; \{ r, a_n \} \leftarrow \langle R, a_n \rangle ]$ 
12    Let  $Q_2$  be
     $[ \{ s, \{ b_1, \dots, b_i, \dots, b_n \} \} | s \leftarrow \langle S \rangle; \{ s, b_1 \} \leftarrow$ 
     $\langle S, b_1 \rangle; \dots; \{ s, b_i \} \leftarrow \langle S, b_i \rangle; \dots; \{ s, b_n \} \leftarrow \langle S, b_n \rangle ]$ 
13    Add class  $\langle C_{R_{fk}} \rangle$  to  $S_{Ont}$  and populate its extent
    using query  $[ (getLSID \langle R \rangle) cr | \{ r, cr \} \leftarrow Q_1 ]$ 
14    Add class  $\langle C_{S_{fk}} \rangle$  to  $S_{Ont}$  and populate its extent
    using query  $[ (getLSID \langle S \rangle) cs | \{ s, cs \} \leftarrow Q_2 ]$ 
15    Add property  $\langle fk, C_R, C_{R_{fk}} \rangle$  to  $S_{Ont}$  and populate
    its extent using query  $Q_1$ 
16    Add property  $\langle fk, C_S, C_{S_{fk}} \rangle$  to  $S_{Ont}$  and populate its
    extent using query  $Q_2$ 
17    Add property  $\langle fk, C_{S_{fk}}, C_{R_{fk}} \rangle$  to  $S_{Ont}$  and populate
    its extent using query
     $[ \{ (getLSID \langle R \rangle) cs, (getLSID \langle S \rangle) cs \} | \{ s, cs \} \leftarrow Q_2 ]$ 
18 deleteRelationalConstructs( $S_{Ont}$ )

```

---

attribute,  $\langle id, patient, rdfs : Literal \rangle$ ,  $\langle birthdate, patient, rdfs : Literal \rangle$  and  $\langle visitId, patient, rdfs : Literal \rangle$ . The primary key of the relation is represented with Class  $\langle patient\_pk \rangle$  and Property  $\langle patient\_has\_pk, patient, patient\_pk \rangle$ , and the foreign key between relations `patient` and `visit` is represented with Class constructs  $\langle patient\_visit\_fk \rangle$ ,  $\langle visit\_patient\_fk \rangle$  and Property constructs  $\langle patient\_has\_fk, patient, patient\_visit\_fk \rangle$ ,  $\langle visit\_has\_fk, visit, visit\_patient\_fk \rangle$ , and  $\langle patient\_fk\_visit, patient\_visit\_fk, visit\_patient\_fk \rangle$ .

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.

## 5. CONCLUDING REMARKS AND FUTURE WORK

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 inter-

operability of different lifelong learning systems within the MyPlan project.

Secondly, we have presented an approach to ontology-based data access of relational data sources that leverages AutoMed's schema-based DTI and distributed query processing capabilities. This is currently being used to support the integration of heterogeneous medical databases under a predefined ontology. Although illustrated in the context of AutoMed, our approach is more generally applicable to other schema-based DTI and distributed query processing systems, and would allow their capabilities to be combined with ontology-based query rewriting and evaluation techniques. An interesting area of future investigation is to explore the relative expressiveness and performance trade-offs between the ontology-based data access approach of [24] and our proposed alternative.

## 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 Dikaos 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, et al. Ontology-based integration of XML web resources. In *Proc. Int. Semantic Web Conference*, pages 117–131, 2002.
- [2] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. International Workshop on the Web and Databases (WebDB'02)*, pages 89–94, 2002.
- [3] S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Proc. Data Integration in the Life Sciences (DILS'04)*, LNCS/LNBI 2994, 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. Int. Workshop on Semantic Web and Databases (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] D. Dou et al. Integrating databases into the semantic web through an ontology-based framework. In *Proc. Int. Workshop on Semantic Web and Databases (SWDB'06 at ICDE'06)*, pages 54–63, 2006.
- [9] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 436–445, 1997.
- [10] E. Jasper, A. Poulouvasilis, 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. Lakshmanan and F. Sadri. XML interoperability. In *Proc. International Workshop on Web and Databases (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. ACM Symposium on Principles of Database Systems (PODS02)*, pages 233–246, 2002.
- [15] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. ACM Symposium on Principles of Database Systems (PODS'95)*, pages 95–104, 2002.
- [16] J. Madhavan and A. Halevy. Composing mappings among data sources. In *Proc. Int. Conf. on Very Large Databases (VLDB'03)*, pages 572–583, 2003.
- [17] P. McBrien and A. Poulouvasilis. A uniform approach to inter-model transformations. In *Proc. Conf. on Advanced Information Systems Engineering (CAISE'99)*, LNCS 1626, pages 333–348, 1999.
- [18] P. McBrien and A. Poulouvasilis. Data integration by bi-directional schema transformation rules. In *Proc. International Conference on Data Engineering (ICDE'03)*, pages 227–238, 2003.
- [19] P. McBrien and A. Poulouvasilis. Defining Peer-to-Peer Data Integration using Both as View Rules. In *Proc. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03 at VLDB'03)*, LNCS 2944, pages 91–107, 2003.
- [20] P. McBrien and A. Poulouvasilis. P2P query reformulation over Both-as-View data transformation rules. In *Proc. Workshop on Databases, Information Systems and Peer-to-Peer Computing (at VLDB'06)*, LNCS 4125, pages 310–322. Springer, 2006.
- [21] P. Mork, R. Shaker, and P. Tarczy-Hornoch. The multiple roles of ontologies in the BioMediator data integration system. In *Proc. Data Integration for the Life Sciences (DILS'05)*, pages 96–104, 2005.
- [22] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.
- [23] N. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.
- [24] 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.
- [25] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
- [26] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics*, IV:146–171, 2005.
- [27] R. Stevens et al. TAMBIS: Transparent access to multiple bioinformatics information sources. *Bioinformatics*, 16(2):184–186, 2000.
- [28] W3C. OWL Web Ontology Language Reference.



Available at <http://www.w3.org/TR/owl-ref>,  
February 2004.

- [29] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Available at <http://www.w3.org/TR/rdf-schema>, February 2004.
- [30] L. Zamboulis, H. Fan, K. Belhajjame, J. Siepen, A. Jones, N. Martin, A. Poulouvassilis, S. Hubbard, S. M. Embury, and N. W. Paton. Data access and integration in the ISPIDER proteomics Grid. In *Proc. Data Integration in the Life Sciences (DILS'06)*, LNCS/LNBI 4075, pages 3–18, 2006.
- [31] L. Zamboulis, N. Martin, and A. Poulouvassilis. Bioinformatics service reconciliation by heterogeneous schema transformation. In *Proc. Int. Workshop on Data Integration in the Life Sciences (DILS'07)*, LNCS/LNBI 4544, pages 89–104, 2007.
- [32] L. Zamboulis and A. Poulouvassilis. Using AutoMed for XML data transformation and integration. In *Proc. Int. Workshop on Data Integration over the Web (at CAiSE'04)*, LNCS 3084, pages 58–69. Springer, 2004.
- [33] L. Zamboulis and A. Poulouvassilis. Information sharing for the Semantic Web - a schema transformation approach. In *Proc. Int. Workshop on Data Integration and the Semantic Web (at CAiSE'06)*, pages 275–289, 2006.