Intersection Schemas as a Dataspace Integration Technique

Richard Brownlow
Department of Computer Science
Birkbeck, University of London
London, UK
richard@dcs.bbk.ac.uk

Alex Poulovassilis
Department of Computer Science
Birkbeck, University of London
London, UK
ap@dcs.bbk.ac.uk

ABSTRACT

This paper introduces the concept of *Intersection Schemas* in the field of heterogeneous data integration and dataspaces. We introduce a technique for incrementally integrating heterogeneous data sources by specifying semantic overlaps between sets of extensional schemas using bidirectional schema transformations, and automatically combining them into a global schema at each iteration of the integration process. We propose an incremental data integration methodology that uses this technique and that aims to reduce the amount of up-front effort required. Such approaches to data integration are often described as *pay-as-you-go*. A demonstrator of our technique is described, which utilizes a new graphical user tool implemented using the AutoMed heterogeneous data integration system. A case study is also described, and our technique and integration methodology are compared with a classical data integration strategy.

Keywords

Dataspaces, Pay-as-you-go, Data Integration, Bidirectional Schema Transformations

1. INTRODUCTION

Data integration is the process of taking data from several different data sources and bringing them together in a structured manner such that Data Services can be supported over the integrated resource. These data services could be Data Mining tools, search engines or simple querying tools. The data sources can be varied in type, for example relational databases, XML data, or web pages. The data sources might be located at different sites of a network, and each data source may be running different data management software. The overarching challenge in data integration is to design frameworks and methodologies that allow heterogeneous data sources to be accessed as a single integrated resource by data services.

Traditional data integration approaches [5] require all the mappings between the different data sources to be determined up-front, prior to being able to run any data services on the integrated resource. In order to establish the mappings, it is likely a data integration project will require a certain level of domain expertise. This approach to data integration means that the full cost of the integration must be committed up front. In order to do this accurately, the timescales for producing the mappings must be accurately estimated. As a result, data integration projects are often costly and risky undertakings [9].

An active area of research [8, 9] is how to reduce the amount of up front effort required in data integration. One approach is to develop a framework that presents all the data in a Common Data Model, but in an unintegrated format. Tools are then provided to allow the data integrator to incrementally identify the semantic relationships between the different data sources. The concept of a dataspace allows semantic integration of data to be undertaken incrementally [7]. Such an approach is often described as pay-as-you-go, since integration can proceed as resources (and budgets) allow. Data services can then be provided at each iteration, rather than waiting for all the integration to be completed up-front.

A theoretical discussion of data integration is presented in [12]. This includes a discussion of global-as-view (GAV) and local-as-view (LAV) approaches, along with a discussion of query processing in data integration settings. The data integration setting is formally defined in terms source schemas, global schemas and mappings between source and global schemas.

The data integration process can be considered as comprising three subprocesses. Firstly, schema matching and mapping, which includes the identification of correspondences between different schema objects and the definition of mappings between schemas. The work in [17] provides an overview of the schema matching process, while [3, 1, 4] discuss different approaches to schema mapping. Secondly, schema merging — the creation of an integrated schema based on the schema mappings. Thirdly, schema improvement which increases the quality of the integrated schema, for example by removing redundant information. The work in [2] discusses the schema improvement and refinement process. An incremental (or pay-as-you-go) data integration process results in an incomplete integration after each iteration of these three subprocesses.

Data integrators tend to fall into two groups [9]. The first

group are domain experts who are knowledgable in the application domain(s) of the data sources being integrated (often with limited data integration experience). The second group are data integration experts who are familiar with data integration environments (but are likely to have limited experience in the target application domain for a particular data integration project). These two groups of integrators must work closely together. It is therefore important to develop tools which can be used by both of these types of data integrators, to increase their productivity in the data integration environment, while still allowing maximum flexibility in the types of semantic mappings that can be supported.

This paper proposes a new technique for reducing the amount of up front effort required in data integration, utilizing bidirectional schema transformations. The focus of our research is developing *light–weight* techniques for creating mappings between data sources and in developing new data integration frameworks and methodologies that use such techniques. By *light–weight* techniques we mean techniques that can be used for rapid, incremental dataspace integration. In Section 2 of the paper we present in detail our approach. We present an overview of the AutoMed system in Section 2.1, our new intersection schema methodology in Section 2.2, a description of the methodology integration workflow in Section 2.3 and a worked example in Section 2.4. In Section 3 we present a case study contrasting our approach with a classical data integration approach in the context of a large-scale data integration project in the field of proteomics. We give our concluding remarks and directions of further work in Section 4.

2. OUR APPROACH

2.1 Overview of AutoMed

AutoMed[1] is a schema transformation and integration system that provides a low-level hypergraph-based data model (the *HDM*) as a common data model for representing heterogeneous data sources. Higher level modelling languages (e.g. relational, XML, RDF/S, OWL) can be defined in terms of the HDM using the API of AutoMed's Model Definitions Repository (MDR) [6].

For any modelling language M specified in terms of the HDM, AutoMed provides a set of primitive schema transformations that can be applied to schema constructs expressed in M. In particular, there is an *add* and a *delete* primitive transformation for adding/deleting any construct of M to/from a schema. For those constructs of M which have textual names, there is also a *rename* primitive transformation. Each *add* or *delete* transformation is accompanied by a query which defines the extent of the new or deleted schema object in terms of the rest of the objects in the schema (i.e. this query specifies a view definition). This query is expressed in IQL, a functional query language developed for the AutoMed system [10]. 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/deleted schema object cannot be precisely derived from the other schema objects. Each *extend* and *contract* transformation takes a query of the form *Range* q_l q_u where the subqueries q_l and q_u specify a lower and an upper bound on the extent of the new/deleted schema object. The lower bound may be the constant *Void*, equivalent to the empty collection, and the upper bound may be the constant *Any*, equivalent to the largest possible collection of the type of the schema object.

[1]http://www.doc.ic.ac.uk/automed

Schemas and their associated instances can be incrementally transformed by applying to them a sequence of primitive transformations. A sequence of primitive transformations transforming a schema S_1 to a schema S_2 is termed a *pathway from* S_1 to S_2 and denoted by $S_1 \rightarrow S_2$. All source, intermediate and integrated schemas, and the pathways between them, are stored in AutoMed's Schemas & Transformations Repository (STR) [6].

In addition to the five primitive schema transformations already mentioned, AutoMed also supports an *ident* primitive. This operates at the level of entire schemas allows the integrator to assert that two syntactically identical schemas, S and S', should be connected by a pathway. The AutoMed system automatically translates such an *ident* transformation into a sequence of *id* transformations from S to S', of the form $id(S : c, S' : c)$ for each schema object c appearing in S and S' (where $S : c$ denotes object c appearing in schema S).

A key property of AutoMed's pathways is that they are automatically reversible, in that from a pathway $S_1 \rightarrow S_2$ we can automatically derive a pathway $S_2 \rightarrow S_1$ by reversing the order of the transformations and replacing each *add* by a *delete* with the same arguments, and vice versa; each *extend* by a *contract* with the same arguments, and vice versa; and each *rename* or *id* by a *rename* or *id* with the arguments reversed.

Schema integration using AutoMed typically proceeds by forming *union-compatible* schemas, as illustrated in Figure 1. Firstly, the appropriate AutoMed Wrapper for each data source is used to extract metadata from the data sources and to produce a set of data source schemas, $DS_1, \ldots, DS_n$, stored within the AutoMed repository. In order to integrate these, each DS_i is first transformed into a union-compatible schema US_i. The n schemas US_1, ..., US_n are identical, and this is verified by injecting an *ident* transformation between each pair US_i and US_{i+1}. An arbitrary one of the US_i can then be selected by the user for further improvement and refinement into the global schema. In terms of the data extents of the objects in the global schema, AutoMed provides a number of options for deriving these from the extents of objects in the union–compatible schemas, e.g. bag union, set union, choice, intersection. The first of these is the default, and it is what we assume in this paper.

There may be information within a US_i which is not semantically derivable from the corresponding DS_i. This is indicated by *extend* transformation steps within the pathway $DS_i \rightarrow US_i$. Conversely, not all of the information within a data source schema DS_i need be transferred into US_i and this is indicated by *contract* transformation steps occurring within $DS_i \rightarrow US_i$.

The queries accompanying the primitive transformations are used by AutoMed's Query Processor [10] in order to refor-

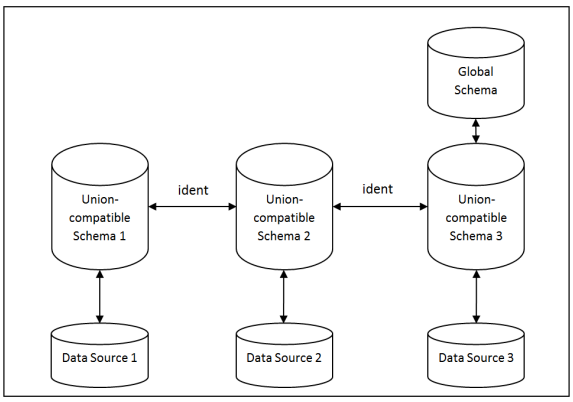

Figure 1: Data Integration via Union-Compatible Schemas

mulate users' queries expressed on a global schema to queries expressed on the source schemas. Query reformulation may be by means of query unfolding for GAV query processing, or by rewriting queries using views for LAV query processing [12], or by a combination of both GAV and LAV query processing techniques [15]. In [14], AutoMed's transformation approach was termed Both-As-View (BAV) since the add/extend steps in a pathway correspond to GAV mappings and the delete/contract steps to LAV mappings. However, BAV transformations capture a finer level of data integration granularity than do conventional GAV or LAV mappings, since BAV transformations are stated on the irreducible modelling constructs of a modelling language M (as determined by the definition of the language M in terms of the HDM), e.g. on individual columns of relational tables as opposed to on whole tables. It is also possible to express global-local-as-view (GLAV) mappings using BAV transformations [11].

A typical data integration workflow using the AutoMed system proceeds as follows: Firstly, each data source is wrapped to produce a data source schema. Secondly, the schema matching and mapping process is undertaken, with the help of AutoMed's Schema Matching tool [16] and the knowledge of domain experts. Each data source schema is transformed to a union-compatible schema via a series of primitive transformations. Thirdly, the union-compatible schemas are automatically merged by injecting *ident* transformations between them. Finally, one of these schemas is chosen as the source for further transformation, capturing any necessary improvements and refinements into the final global schema.

2.2 Intersection Schemas

We now propose a new methodology for lightweight data integration in an incremental pay-as-you-go environment, based on the concept of *Intersection Schemas*. The primary goal of this approach is to improve on existing data integration methodologies by increasing data integrators' productivity in the overall Data Integration process.

We demonstrate a lightweight technique that allows a domain expert to identify schema mappings that represent semantic intersections between different data source schemas. Using our technique, both the schema matching/mapping and the schema merging processes are undertaken iteratively.

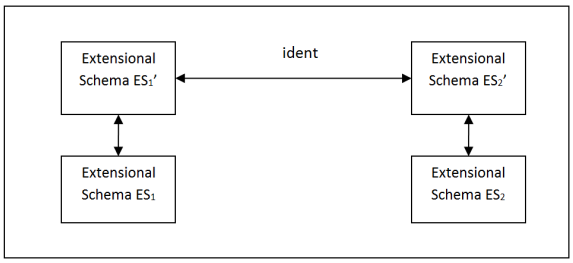

Figure 2: Intersection Schema

Three types of schema are encompassed within our methodology:

Extensional Schema: This is any schema in the AutoMed repository that is connected to a data source schema via a pathway.

Federated Schema: This is a combination of multiple schemas (of any kind), $S_1, ..., S_n$, into a single virtual schema F containing a union of the objects in $S_1, ..., S_n$, without undertaking any schema or data transformation or integration. Within F, the schema objects from each S_i are prefixed with the schema identifier of S_i so as to (i) be able to easily distinguish their provenance and (ii) disambiguate objects of the same name from different schemas. We write:

$$F = S_1 \cup S_2 \cup ... \cup S_n$$

Intersection Schema: This is a schema which contains only semantically overlapping content from a pair of extensional schemas ES_1 and ES_2 (see Figure 2). In more detail, there need to exist in the AutoMed repository two pathways $ES_1 \rightarrow ES_1'$ and $ES_2 \rightarrow ES_2'$ with ES_1' and ES_2' being union-compatible schemas, and each pathway consisting of a sequence of *add* and *delete* operations followed by a sequence of *contract* operations. Specifically, the pathway $ES_1 \rightarrow ES_1'$ will be of the form:

$add(o_1, f_1(ES_1)), add(o_2, f_2(ES_1)), ... , add(o_r, f_r(ES_1)),$
$del(c_1, g_1(ES_1')), del(c_2, g_2(ES_1')), ..., del(c_m, g_m(ES_1')),$
$contract(c_{m+1}, Range\ Void\ Any),$
$contract(c_{m+2}, Range\ Void\ Any),$
$contract(c_n, Range\ Void\ Any)$

where the c_i are schema objects of ES_1, the o_i are schema objects of ES_1', the f_i are IQL queries over ES_1 and the g_i are IQL queries over ES_1' (the pathway $ES_2 \rightarrow ES_2'$ has a similar form). The part of this pathway comprising the *add* and *delete* steps asserts the semantic equivalence of the set of schema objects $o_1, ...o_r$ of ES_1' and the set of objects $c_1...c_m$ of ES_1. There is also a pathway $ES_1' \rightarrow ES_2'$ comprising a single *ident* operation.

The schemas ES_1' and ES_2' can both be regarded as intersection schemas, and one of them can be explicitly renamed to reflect the fact that it is specifically chosen to be the intersection schema, I. As noted in Section 2.1, the data extents of the objects in schema I are formed from a bag-union of

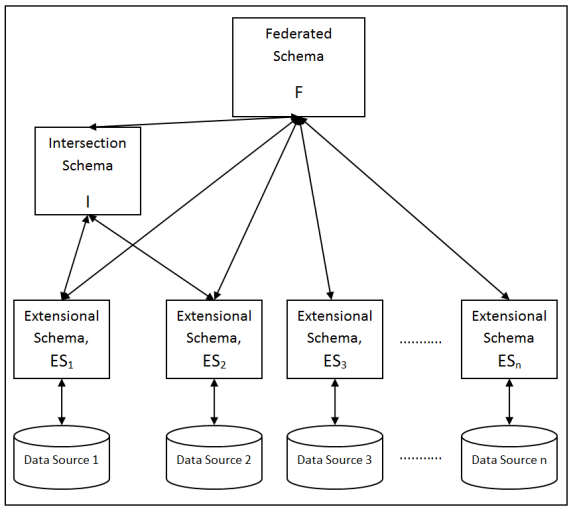

Figure 3: Integrated Intersection and Extensional Schemas

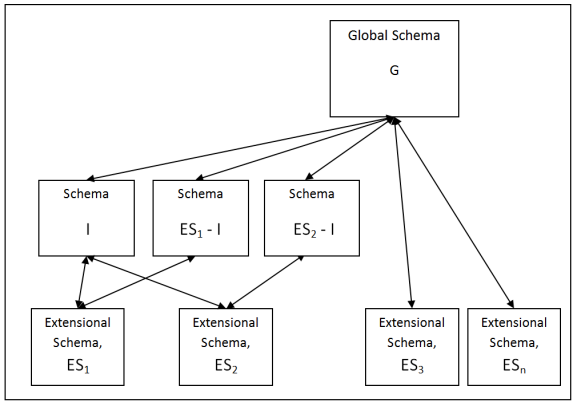

Figure 4: Global schema derived from Intersection and Extensional Schemas

the objects in schemas ES_1' and ES_2' from which they are derived.

In our approach a federated schema can be created from a set of extensional schemas $ES_1, ..., ES_n$ and a set of intersection schemas $I_1, ..., I_m$ (see Figure 3):

$$F = ES_1 \cup ... \cup ES_n \cup I_1 \cup ... \cup I_m$$

If the intersection schemas $I_1, ..., I_m$ have been derived from a subset of the extensional schemas $ES_1, ..., ES_n$ then it is possible to automatically determine objects that are now semantically redundant in F and our implemented tool (see Section 2.3 below) offers this as an option. For example, in Figure 4, the intersection schema I has been derived from the extensional schemas $ES_1$and ES_2, and the global schema G is defined as:

$$G = I \cup (ES_1 - I) \cup (ES_2 - I) \cup ES_3..... \cup ES_n$$

The $-$ operator here removes from the schema that is its first argument the subset of objects that are semantically equivalent with some subset of the objects in the schema that is its second argument. Operationally, given two extensional schemas ES_1 and ES_2 and an intersection schema I derived from them as described earlier, the schema $ES_1 - I$ is obtained from ES_1 by retaining only those schema objects of ES_1 which have been removed in the pathway $ES_1 \rightarrow I$ by a *contract* operation; likewise for the schema $ES_2 - I$. In more detail, if the pathway $ES_1 \rightarrow I$ is of the following form (possibly with a sequence of *ident* transformations added at the end as well):

$add(o_1, f_1(ES_1)), add(o_2, f_2(ES_1)), ..., add(o_r, f_r(ES_1)),$
$del(c_1, g_1(I)), del(c_2, g_2(I)), ..., del(c_m, g_m(I)),$
$contract(c_{m+1}, Range\ Void\ Any),$
$contract(c_{m+2}, Range\ Void\ Any),$
$contract(c_n, Range\ Void\ Any)$

where the c_i are schema objects of ES_1, the o_i are schema objects of I, the f_i are IQL queries over ES_1 and the g_i are IQL queries over I, then the pathway $ES_1 \rightarrow ES_1 - I$ is automatically derived to be:

$contract(c_1, Range\ Void\ Any), ..., contract(c_m, Range\ Void\ Any)$

2.3 Integration Workflow

Our techniques for creating federated, intersection and global schemas have been described in the previous subsection. These techniques now need to be incorporated into an overall workflow through which the global schema is produced iteratively. We describe the workflow below. Before doing so, we briefly describe the Intersection Schema Tool that has been developed to support this workflow. This tool firstly creates a single federated schema from a set of extensional schemas — this also serves as the first version of the global schema. The tool then allows a data integrator to incrementally identify semantic intersections between pairs of extensional schemas, and create a schema representing their intersection. Each intersection schema can be integrated with the current global schema using the tool, producing a new global schema.

The steps of the workflow proceed as follows:

1. Identify the extensional schemas representing the set of data sources that are to be integrated.
2. Initially a federated schema is created from the schemas identified in Step 1. Data Services can immediately be supported by this schema e.g. AutoMed's Query Tool [13].
3. Inspect the schemas identified in Step 1 and select two of them from which to derive an intersection schema.
4. Identify mappings between these two schemas and the intersection schema. For each Intersection Schema, a mappings table is maintained by the Intersection Schema Tool, which shows the IQL query correspondences between objects in the Intersection Schema and the current global schema. The Intersection Schema tool allows mappings to be added and edited by the data integrator. Other existing tools supported by AutoMed can be used to assist with this, for example the

Extent Tool which allows the extent of any schema object to be displayed [13] and the Schema Matching Tool [16] which aims to provide suggestions for schema mappings.

5. When all the mappings have been identified for this cycle of the workflow, the user can ask for an intersection schema to be generated. A new Global Schema is created automatically from the Intersection Schema and the extensional schemas by our tool, as described in Section 2.3. The user may optionally elect for any redundant objects in the new Global schema to be dropped.

6. The user may test the Intersection schema or Global schema at this stage by running queries on it using the AutoMed Query Tool and verifying that the results are as expected.

The data integrator can now proceed from Step 3 again, identify another pair of extensional schemas from which to create a new intersection schema, test this schema, generate a new global schema and, optionally, ask for any redundant objects to be removed from it. Any number of intersections between any pair of extensional schemas. ES_i and ES_j can be created at each iteration of the process.

2.4 Example

A large-scale data integration project that used AutoMed within a traditional data integration process was the iSpider project, which integrated data from several specialist Proteomics relational databases under a single virtual relational schema [19, 20]. In the iSpider project, all of the integration work was done "up front", before any data services were deployed and the integration took several person months to accomplish.

Two of the source databases used in iSpider were Pedro[2] and PepSeeker[3] and we now illustrate how our intersection-schema based tools and methodology can be used to (partially) integrate them. We were able to do this by examining the set of schema transformations generated for the original iSpider project by the domain experts and data integrators working on that project. These are listed in Appendix E of [18].

An initial Federated schema is first generated, prior to the creation of any Intersection schemas. Upon inspection of the Pedro and Pepseeker fragments of this federated schema, the user identifies that $\ll proteinhit, db\_search \gg$ from Pedro and $\ll proteinhit, fileparameters \gg$ from PepSeeker are semantically equivalent concepts and should be represented by a new concept $\ll UProteinHit, dbsearch \gg$ in an intersection schema between them[4]. The user creates an intersection schema in the tool as illustrated in Figure 5. The left hand panel of the tool shows the source schemas for Pedro and PepSeeker and allows the user to select objects from each source schema. The bottom panel shows the transformation queries for each subset of objects selected, and the name of the new Global schema object. If only a single object is selected from a source schema then, by default, the tool automatically creates a tranformation query consisting of just that object; the user is free to edit this query. The current Global schema is shown in the right hand panel. Once all the transformation queries in the "forwards" direction have been specified, a similar screen is presented to the user in order to specify the transformation queries in the reverse direction. This time, the top panel shows the newly defined Global schema objects on the left hand side and the source schemas on the right. Suggested transformation queries are presented in the bottom panel. Where possible, these queries are automatically generated by the tool from the user-specified queries in the forwards direction; for more complex transformations, the default query *Range Void Any* is used, which the user may edit. In this particular example, both the forwards and the reverse transformation queries consist of the same schema object. Once the user is satisfied with the new Global schema objects and the transformation queries, the user then requests the creation of the intersection schema.

The Intersection Schema is then automatically integrated with the original federated schema to create a new Global schema. If the user requests this, the schema objects $\ll proteinhit, db\_search \gg$ from Pedro and $\ll proteinhit, fileparameters \gg$ from PepSeeker can be dropped from the resulting federated schema, since their extents are included in the extent of $\ll UProteinHit, dbsearch \gg$ in the Intersection Schema. Queries can now be run by the user over the resulting schema to verify the data integration, and in particular to check that the extent of $\ll UProteinHit, dbsearch \gg$ is as expected.

3. CASE STUDY

We have re-examined the iSpider documentation and the original set of schema transformations that were produced by that project in order to see how our intersection-schema based approach could have been used in the context of a large-scale real-world data integration project to undertake an incremental, pay-as-you-go integration that would have allowed query services to be supported in a gradual fashion. In addition to Pedro and PepSeeker, a third database that had been integrated with them in the iSpider project was gpmDB[5].

The original iSpider project domain experts had identified a set of queries of high priority that the integrated resource should be able to answer (see Chapter 7 of [18]). Despite

[2]http://pedro.man.ac.uk/

[3]http://nwsr.smith.man.ac.uk/pepseeker

[4]AutoMed gives considerable flexibility for configuring how a construct m of a modelling language M is represented in the HDM, in general identifying the construct using a *scheme* of the form $\ll M, m, s_1, ..., s_n \gg$ where the s_i are either literals, or schemes representing other constructs. In this paper, we consider that AutoMed is configured to use a scheme of the form $\ll sql, table, t \gg$ to represent an SQL table named t, and a scheme of the form $\ll sql, column, t, c \gg$ to represent a column c of an SQL table t. Where the context of using a scheme would not cause ambiguity, the user may omit the modelling language M or the construct type m from the scheme, or both. Thus, in the context of the examples in this paper, where only relational tables are being considered, we refer to a table by a scheme of the form $\ll t \gg$, and to a column of a table by a scheme of the form $\ll t, c \gg$.

[5]http://www.thegpm.org/

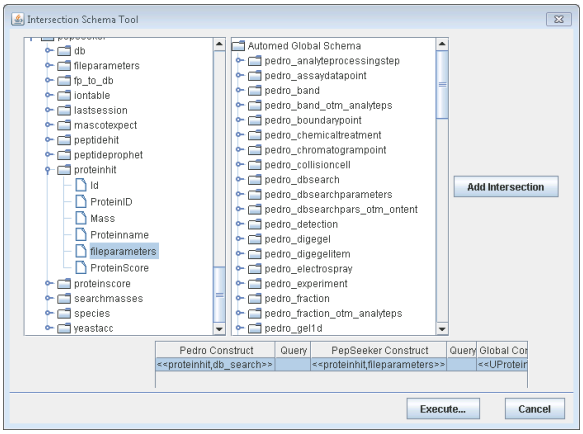

Figure 5: Intersection Schema Tool - identifying mappings

this fact, the iSpider project team elected to undertake a complete "up-front" integration of the data sources rather than using the set of queries to prioritise their integration effort.

We have used this set of queries in order to undertake an intersection schema-based integration, using the priority ordering of the queries to drive each iteration of the process. The set of queries are as follows, with higher priority queries being listed before lower priority ones:

1. Retrieve all protein identifications for a given protein accession number
2. Retrieve all protein identifications for a given group of proteins
3. Retrieve all protein identifications for a given organism
4. Retrieve all protein identifications given a certain peptide and their related amino acid information
5. Retrieve all identifications of a given protein given a certain peptide
6. Retrieve all peptide-related information for a given protein identification
7. Retrieve all ion related information

To answer query 1, we need to integrate $\ll protein, accession\_num \gg$ from Pedro, $\ll proseq, label \gg$ from gpmDB and $\ll protein \gg$ from PepSeeker, to create an entity $\ll UProtein, accession\_num \gg$ in the first intersection schema. These integrations are achieved by the following 6 transformations defined by the user in the Intersection Schema tool[6] (as well additional transformations automatically created by the tool[7]):

[6]The general syntax of the IQL queries within these *add* transformations is that of a *comprehension* [...|...] in which the expressions on the right hand side of the | are iterators and filters over one or more collections, and the expression on the left hand side of the | constructs a new collection.

[7]The additional *contract* transformations are automatically generated. Where possible, the *delete* transformations in the pathway from a data source schema to an intersection schema are also automatically generated from the user-specified *add* transformations to complete the bidirectional pathway. For more complex transformations, the user's input is needed to specify the *delete* transformations.

Q^1:	[{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; an = 'ENSP00000339074']
Q^2:	[{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; member [lsid|{lsid, d} ← ⟨⟨UProteinHit, description⟩⟩; like d '%Actin%'] lsid]
Q^3:	[{an, lsid}|{lsid, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; member [lsid|{lsid, o} ← ⟨⟨UProteinHit, organism⟩⟩; like o '%sapiens%'] lsid]
Q^4:	[{pr, sc}|{lsid1, pr} ← ⟨⟨UProteinHit, protein⟩⟩; {lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; seq = 'ATLTSDK'; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid2 = pepID; lsid1 = protID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩; {aid, pid} ← ⟨⟨GS1_aa, GS1_pepid⟩⟩; pid = pepID]
Q^5:	[{an, lsid1, sc}|{lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; seq = 'LVNELTEFAK'; {lsid1, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; an = 'gi—229552'; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid2 = pepID; lsid1 = protID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩]
Q^6:	[{an, seq, sc, pr, dbs}|{lsid1, an} ← ⟨⟨UProteinHit, accession_number⟩⟩; lsid1 = {'URN:LSID:ispider.man.ac.uk:pedro', 1069}; {pepID, protID} ← ⟨⟨UPeptideHitToProteinHit_mm⟩⟩; lsid1 = protID; {lsid2, seq} ← ⟨⟨UPeptideHit, sequence⟩⟩; lsid2 = pepID; {lsid2, sc} ← ⟨⟨UPeptideHit, score⟩⟩; {lsid2, pr} ← ⟨⟨UPeptideHit, probability⟩⟩; {lsid1, dbs} ← ⟨⟨UProteinHit, dbsearch⟩⟩]
Q^7:	[{ionid, mat, imm}|{ionid} ← ⟨⟨GS1_iontable⟩⟩, {ionid, mat} ← ⟨⟨GS1_iontable, GS1_Matches⟩⟩, {ionid, imm} ← ⟨⟨GS1_iontable, GS1_Immon⟩⟩]

Table 1: Case Study Queries

$add \ll UProtein \gg [\{'PEDRO', k\} \mid k \leftarrow\ll protein \gg]$
$add \ll UProtein \gg [\{'gpmDB', k\} \mid$
$\quad k \leftarrow\ll proseq \gg]$
$add \ll UProtein \gg [\{'pepSeeker', x\} \mid$
$\quad \{k, x\} \leftarrow\ll proteinhit, ProteinID \gg]$
$add \ll UProtein, accession\_num \gg [\{'PEDRO', k, x\} \mid$
$\quad \{k, x\} \leftarrow\ll protein, accession\_num \gg]$
$add \ll UProtein, accession\_num \gg [\{'gpmDB', k, x\} \mid$
$\quad \{k, x\} \leftarrow\ll proseq, label \gg]$
$add \ll UProtein, accession\_num \gg [\{'pepSeeker', k, x\} \mid$
$\quad k \leftarrow\ll uprotein \gg]$

To answer query 2, we need additionally to create an attribute $\ll UProtein, description \gg$ from $\ll protein, description \gg$ in Pedro. This is achieved by the following transformation defined by the user[8]:

$add \ll UProtein, description \gg [\{'PEDRO', k, x\} \mid$
$\quad \{k, x\} \leftarrow\ll protein, description \gg]$

To answer query 3, we need additionally to create an attribute $\ll UProtein, organism \gg$ from $\ll protein, organism \gg$ in Pedro. This is achieved by the following transformation defined by the user:

$add \ll UProtein, organism \gg [\{'PEDRO', k, x\} \mid$
$\quad \{k, x\} \leftarrow\ll protein, organism \gg]$

To answer query 4, we additionally need to integrate the following:

[8]Note, this is a refinement of the intersection schema-based methodology described in Section 2.3 — our Intersection Schema tool allows ad-hoc transformations of a single schema as well, as part of the iterative integration process.

- $\ll proteinhit, protein \gg$ from Pedro, $\ll protein, proseqid \gg$ from gpmDB and $\ll proteinhit, proteinid \gg$ from PepSeeker to create an attribute $\ll UProteinHit, protein \gg$;
- $\ll peptidehit \gg$ from Pedro, $\ll peptide \gg$ from gpmDB and $\ll peptidehit \gg$ from PepSeeker to create an entity $\ll UPeptideHit \gg$;
- $\ll peptidehit, sequence \gg$ from Pedro, $\ll peptide, seq \gg$ from gpmDB and $\ll peptidehit, pepseq \gg$ from PepSeeker to, create an attribute $\ll UPeptideHit, sequence \gg$;
- $\ll peptidehit, score \gg$ from Pedro and $\ll peptidehit, score \gg$ from PepSeeker to create attribute $\ll UPeptideHit, score \gg$;
- $\ll proteinhit, db\_search \gg$ from Pedro and $\ll proteinhit, fileparameters \gg$ from PepSeeker to create an attribute $\ll UProteinHit, dbsearch \gg$.

We also create an attribute $\ll UPepTideHit, dbsearch \gg$ from $\ll peptidehit, db\_search \gg$ in Pedro, and an entity $\ll UPeptideHitToProteinHit\_mm \gg$ from the join of $\ll UPeptideHit, dbsearch \gg$ and $\ll UProteinHit, dbsearch \gg$ already in the global schema. This is achieved by the following 15 transformations defined by the user:

$add \ll UProteinHit, protein \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll proteinhit, protein \gg]$
$add \ll UProteinHit, protein \gg [\{'gpmDB', k, x\} \mid \{k, x\} \leftarrow\ll protein, proseqid \gg]$
$add \ll UProteinHit, protein \gg [\{'pepSeeker', k, x\} \mid \{k, x\} \leftarrow\ll proteinhit, proteinid \gg]$
$add \ll UPeptideHit \gg [\{'PEDRO', k\} \mid k \leftarrow\ll peptidehit \gg]$
$add \ll UPeptideHit \gg [\{'gpmDB', k\} \mid k \leftarrow\ll peptide \gg]$
$add \ll UPeptideHit \gg [\{'pepSeeker', k\} \mid k \leftarrow\ll peptidehit \gg]$
$add \ll UPeptideHit, sequence \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, sequence \gg]$
$add \ll UPeptideHit, sequence \gg [\{'gpmDB', k, x\} \mid \{k, x\} \leftarrow\ll peptide, seq \gg]$
$add \ll UPeptideHit, sequence \gg [\{'pepSeeker', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, pepseq \gg]$
$add \ll UPeptideHit, score \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, score >]$
$add \ll UPeptideHit, score \gg [\{'pepSeeker', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, score \gg]$
$add \ll UProteinHit, dbsearch \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll proteinhit, db\_search \gg]$
$add \ll UProteinHit, dbsearch \gg [\{'pepSeeker', k, x\} \mid \{k, x\} \leftarrow\ll proteinhit, fileparameters \gg]$
$add \ll UPeptideHit, dbsearch \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, db\_search \gg]$
$add \ll uPeptideHitToProteinHit_m m \gg [\{k1, k2\} \mid \{k1, x\} \leftarrow\ll upeptidehit, dbsearch \gg; \{k2, y\} \leftarrow\ll uproteinhit, dbsearch \gg; x = y]$

To answer query 5, no further concepts need to be integrated. To answer query 6, we need to integrate $\ll peptidehit, probability \gg$ from Pedro, $\ll peptide, expect \gg$ from gpmDB and $\ll peptidehit, expect \gg$ from PepSeeker to create the attribute $\ll UPeptideHit, probability \gg$ in the next intersection schema. This is achieved by the following 3 transformations defined by the user:

$add \ll UPeptideHit, probability \gg [\{'PEDRO', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, probability \gg]$
$add \ll UPeptideHit, probability \gg [\{'gpmDB', k, x\} \mid \{k, x\} \leftarrow\ll peptide, expect \gg]$
$add \ll UPeptideHit, probability \gg [\{'pepSeeker', k, x\} \mid \{k, x\} \leftarrow\ll peptidehit, expect \gg]$

Finally, to answer query 7, no further concepts need to be integrated. For completeness, we list in Table 1 examples of the 7 queries (expressed in IQL) formulated over the resulting global schema.

In summary, we see that a total of 6+1+1+15+3 = 26 manually defined transformations are required to be able to answer all seven of these high priority queries.

By way of comparison, in the original iSpider integration (see Chapter 7 and Appendix E of [18] for full details, including listings of all the transformations), three successive versions of the global schema were produced, GS1, GS2, GS3. GS1 was defined to be identical with the Pedro schema due to the rich content of this schema compared with the gpmDB and PepSeeker schemas. AutoMed transformation pathways were then defined from the three data sources to GS1. Since all GS1 schema constructs have a trivial identify derivation from Pedro, we can consider the integration effort to comprise the manually defined transformations from gpmDB and PepSeeker to GS1. Also, we ignore any transformations whose query part is just *Range Void Any*. There are 19 non-trivial transformations from gpmDB to GS1 and 35 non-trivial transformations from PepSeeker to GS1.

The next version of the global schema, GS2, improved on GS1 by adding concepts that are supported by the gpmDB data source but not by Pedro and that therefore where not present in GS1. This required an additional 41 non-trivial transformations from PepSeeker to GS2 (note that all the additional transformations from Pedro to GS2 would have query parts *Range Void Any*).

The final version of the global schema, GS3, improved on GS2 by adding concepts that are supported by the PepSeeker data source but not by Pedro or gpmDB and that therefore were not present in GS2. This required no more non-trivial transformations (all the additional transformations from Pedro and gpmDBto GS3 would have query parts *Range Void Any*). Hence there are a total of 19+35+41=95 non-trivial transformations required by the original iSpider integration effort.

Of course, using the number of manually defined transformations as a comparison metric is rather crude; and moreover, as stated earlier, the original iSpider integration did not attempt to undertake a query-driven integration. Therefore, we are planning for the near future a more detailed evaluation of our intersection schema-based data integration methodology compared with traditional ones.

4. CONCLUSIONS

In this paper we have introduced a data integration methodology based on the concept of *Intersection Schemas*, using the AutoMed data integration framework. We have demonstrated the technique on a real-world data integration scenario, adopting a query-driven approach, and have seen that the number of user-defined steps required to perform the integration is significantly reduced compared to the original data integration methodology used by the domain experts on that project.

This work has been carried out in the context of the AutoMed data integration framework, which supports bidirectional schema and data transformations but which, up till now, has been used only for "up-front" data integration. In this paper we have shown how the AutoMed toolkit can be used to underpin a new light-weight data integration technique within an incremental pay-as-you-go data integration process, and hence how AutoMed can be applied within a dataspace environment.

Our future work includes extending the methodology so that intersections can be created between any number of source schemas at each iteration of the process, rather than just two as at present. We will also undertake a more detailed evaluation of our intersection-schema based integration approach with traditional integration methodologies in the context of further real-world large-scale data integration settings. For these investigations, we will take in both cases a query-driven approach and we will assess the productivity benefits arising using our approach. Since our techniques and tools are intended to be used by both domain experts and data integration experts, we propose a user evaluation which will consider two groups of users. The first group will consist of people familiar with the application domain but with limited knowledge of data integration processes, while the second group will be familiar with data integration processes but have limited knowledge of the application domain. Each group will be split randomly into two equally-sized subgroups. Each subgroup of a given group will be asked to undertake the same, query-driven, integration of a given set of data sources, guiding one subgroup through an intersection schema-based methodoloty and the other subgroup through a more traditional methodology, such as the 'ladder' approach [5]. A set of metrics will be measured for each subgroup, for example the time taken to complete the integration and the number of key clicks required within the toolset.

5. REFERENCES

[1] B. Alexe, B. T. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Trans. on Database Systems*, 36(4):23, 2011.

[2] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan. Muse: Mapping understanding and design by example. In *Proc. ICDE*, pages 10–19. IEEE, 2008.

[3] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[4] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proc. ACM SIGMOD*, pages 133–144. ACM, 2011.

[5] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[6] M. Boyd, C. Lazanitis, S. Kittivoraviktul, P. Mc Brien, and N. Rizopoulos. An overview of the AutoMed Repository. *Technical Report, AutoMed Project*, 2004.

[7] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[8] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *Proc. VLDB*, pages 9–16. VLDB Endowment, 2006.

[9] C. Hedeler, K. Belhajjame, N. W. Paton, A. Campi, A. A. Fernandes, and S. M. Embury. Flexible dataspace management through model management. In *Proc. EDBT/ICDT Workshops*, pages 114–134. Springer, 2010.

[10] E. Jasper, A. Poulovassilis, L. Zamboulis, and H. Fan. Processing IQL queries and migrating data in the automed toolkit. *Technical Report, AutoMed Project*, 2003.

[11] E. Jasper, N. Tong, P. McBrien, and A. Poulovassilis. Generating and optimising views from both as view data integration rules. In *Proc. DBIS'04*, volume 972, pages 13–30, 2004.

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

[13] P. McBrien. AutoMed in a nutshell. *Technical Report, AutoMed Project*, 2006.

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

[15] P. Mcbrien and A. Poulovassilis. P2P query reformulation over both-as-view data transformation rules. In *Proc. DBISP2P*, pages 310–322. Springer, 2006.

[16] N. Rizopoulos. Automatic discovery of semantic relationships between schema elements. In *Proc. ICEIS (1)*, pages 3–8, 2004.

[17] B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Schema mappings and data examples. In *Proc. EDBT*, pages 777–780. ACM, 2013.

[18] J. Wang. A Framework and Architecture for Quality Assessment in Data Integration. `http://www.dcs.bbk.ac.uk/research/recentphds/jwang.pdf`, 2012. [Online; accessed 01-December-2013].

[19] L. Zamboulis, H. Fan, K. Belhajjame, J. Siepen, A. Jones, N. Martin, A. Poulovassilis, 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*, pages 3–18. Springer, 2006.

[20] L. Zamboulis, N. Martin, and A. Poulovassilis. Query performance evaluation of an architecture for fine-grained integration of heterogeneous grid data sources. *Future Generation Computer Systems*, 26(8):1073–1091, 2010.