Combining Data Integration with Natural Language Technology for the Semantic Web

Dean Williams, Alexandra Poulovassilis
{dean,ap}@dcs.bbk.ac.uk

School of Computer Science and Information Systems, Birkbeck College, University of London

Abstract. Current data integration systems allow a variety of heterogeneous structured or semi-structured data sources to be combined and queried by providing an integrated view over them. The Semantic Web also requires us to be able to integrate information from a variety of heterogeneous information sources. However, these information sources will also include natural language (e.g. web pages) and ontologies. In this paper we describe an architecture which combines the data integration approach with techniques from Information Extraction in order to allow information from ontologies and natural language sources to be integrated with other, semantically related, structured or semi-structured data. Our architecture is based on the AutoMed data integration system, and in this paper we describe several extensions which have been made to AutoMed in order to support this work, including adding RDF to the data sources supported by AutoMed and providing a repository for the data and metadata discovered by the Information Extraction process.

1 Introduction

The Semantic Web requires us to be able to integrate information from a variety of sources, including unstructured text from web pages, semi-structured XML data, structured databases, and metadata sources such as ontologies. Integration of heterogeneous data sources is a problem that has been addressed by several recent data integration systems, one of which is the AutoMed system being developed at Birkbeck and Imperial Colleges (`http://www.doc.ic.ac.uk/automed`). In data integration systems, several data sources, each with an associated local schema, are integrated to form a single virtual database with an associated global schema. If the data sources conform to different data models, then these need to be transformed into a common data model as part of the integration process. The AutoMed system uses a hypergraph-based data model, the **HDM** (see Section 2 below), as its common data model.

There is clearly potential for using this approach for information integration in the Semantic Web, but a number of extensions are required. In particular, while data in a wide range of structured and semi-structured formats has been dealt with by previous data integration systems, natural language sources and ontologies have not. In this paper we present a method of extracting data and metadata from natural language sources and integrating it with other structured and semi-structured data sources. We describe how existing metadata can be used to assist in this extraction process. We also describe a repository for storing the extracted data and metadata.

Our approach combines Information Extraction technology with the AutoMed data integration system. The resulting system, called **ESTEST**, makes use of existing metadata such as database schemas, natural language ontologies and domain-specific ontologies, to assist the Information Extraction process. Once new data and new metadata have been extracted from the text, this is integrated with the existing data and metadata. This extraction and integration process can then be reiterated as required.

The Resource Description Framework (RDF) is an emerging standard for sharing ontolological data, so in this paper we focus on this and show how it can be mapped into AutoMed's HDM common data model. Our ESTEST system will discover new data and metadata. While this data and metadata may be expressed in a variety of data models, these can all be mapped into the HDM, and hence we use a native HDM repository to store all the new data and metadata.

The remainder of this paper is stuctrured as follows. We first give a brief overview of the AutoMed system in Section 2. In Section 3 we show how RDF can be mapped into AutoMed's HDM common data model. Section 4 then describes the ESTEST system. Section 5 describes the HDM repository. We conclude in Section 6.

2 AutoMed

Up to now, most data integration approaches have been either **global as view (GAV)** [4, 30, 29] or **local as view (LAV)** [16, 14, 17]. In GAV, the constructs of a global schema are described as views over the local schemas. These view definitions are used to rewrite queries over a global schema into distributed queries over the local databases. However, one disadvantage of GAV is that it does not readily support the evolution of the local schemas, since a change in a local schema construct impacts on all the GAV view definitions referencing that construct. Similarly, the addition of a new local schema requires all the GAV view definitions to be reviewed in order to incorporate the new data source.

In LAV, the constructs of the local schemas are defined as views over the global schema, and processing queries over the global schema involves rewriting queries using views [15]. LAV confines changes to local schemas to impact only on the derivation rules defined for that schema. However, it has problems if one needs to change the global schema, since all the rules for defining local schemas as views of the global schema will need to be reviewed.

In contrast to GAV and LAV, AutoMed supports **both as view (BAV)** integration [22]. This is based on the use of reversible sequences of primitive schema transformations, called transformation **pathways**. From these pathways it is possible to extract a definition of the global schema as a view over the local schemas (GAV), and it is also possible to extract definitions of the local schemas as views over the global schema (LAV) (at present, AutoMed's global query processor extracts GAV views and uses these for global query processing [11, 12]).

As discussed in [21, 22], one advantage of BAV over GAV and LAV is that it readily supports the evolution of *both* global and local schemas, including the addition/removal of local schemas. Such evolutions can be expressed as extensions to the existing pathways, and the new view definitions can then simply be regenerated from the new pathways as needed for query processing.

The basis of AutoMed's BAV approach is a low-level **hypergraph-based data model (HDM)** [26, 18]. Facilities are provided for defining higher-level modelling languages in terms of this lower-level HDM: for example, previous work has shown how relational, ER, UML and XML data models can be so defined [19, 20].

An HDM schema consists of a set of nodes, edges and constraints, and so each modelling construct of a higher-level modelling language needs to be defined as some combination of HDM nodes, edges and constraints. For any modelling language $\mathcal{M}$ specified in this way (via the API of AutoMed's Model Definitions Repository [3]), AutoMed automatically 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 respectively add to, or delete from, the underlying HDM schema the corresponding set of nodes, edges and constraints. For those constructs of $\mathcal{M}$ which have textual names, there is also a rename primitive transformation.

One advantage of using a low-level common data model such as the HDM is that semantic mismatches between high-level modelling constructs are avoided. Another advantage is that the HDM provides a unifying semantics for higher-level modelling constructs and hence a basis for automatically or semi- automatically generating the semantic links between them — this is ongoing work being undertaken by other members of the AutoMed project.

In AutoMed, schemas are incrementally transformed by applying to them a sequence of primitive transformations $t_1, \ldots, t_r$. Each primitive transformation t_i makes a 'delta' change to the schema by adding, deleting or renaming just one schema construct. Thus, the intermediate schemas may contain constructs of more than one modelling language.

Each add or delete transformation is accompanied by a query specifying the extent of the new or deleted construct in terms of the rest of the constructs in the schema. This query is expressed in a functional **intermediate query language**, **IQL** [25]. Also available are contract and extend transformations which behave in the same way as add and delete except that they indicate that their accompanying query may only partially construct the extent of the new/removed schema construct. Moreover, their query may just be the constant Void, indicating that the extent of the new/removed construct cannot be derived even partially, in which case the query can be omitted. All source schemas, intermediate schemas and global schemas, and the pathways between them are stored in AutoMed's Schemas & Transformations Repository [3].

The queries present within transformations that add or delete schema constructs mean that *automatic translation* of data, queries and updates is possible between schemas [18]. The presence of such queries also means that each primitive transformation has an automatically derivable *reverse transformation*. In particular, each add/extend transformation is reversed by a delete/contract transformation with the same arguments, while each rename transformation is reversed by another rename transformation with the two arguments swapped. Hence, the bidirectional nature of the BAV integration approach.

3 Representing RDF in AutoMed

The Resource Description Framework (RDF) [13] is a language for describing information on the World Wide Web. RDF allows properties of Web resources to be stated in the form of subject-predicate-object triples e.g. a statement such as "Dean Williams is the author of the webpage http://www.xyz.com/index.html" can be represented by a triple where the subject is the webpage URL, the predicate is 'author' and the object is 'Dean Williams'.

Resources are identified using the Uniform Resource Identifier (URI) [2] web standard. URI's are a more general identifier than the Uniform Resource Locator (URL). While URIs cover defined, centralised schemes (such as the http part of a URL) they also allow for anyone to create their own URI naming schemes.

In the above example, the concept of 'author' could be used in different ways by different systems. For example, a web site maintenance system might use it in a different way from a book publishing house system. Referring to the concept using a URI with an explicit namespace allows the concept to be identified uniquely e.g. http://www.xyz.com/rdf/1.0/author

Values in RDF can be URIs, literals or unlabelled nodes, known as 'blank' nodes. Blank nodes can be used to structure property values e.g. by linking each line of an address. More generally, they represent concepts to which properties apply. Arbitrary identifiers are assigned to these blank nodes.

The subject component of a triple can be either a URI or a blank node. The predicate component of a triple must be a URI. The object component can be either a URI, or a blank node, or a literal.

Nodes and edges in a graph can be used to represent RDF statements. By convention, nodes are drawn as ovals and will be either labeled or blank, literals are drawn as rectangles, and edges as single-headed arrows linking nodes or literals.

Figure 1 shows the webpage author example above extended to allow for various properties of the person identified as the author to be related to that person.

3.1 Representing RDF in the HDM

A **schema** in the HDM data model [26, 19] is a triple $\langle Nodes, Edges, Constraints \rangle$. A **query** q over a schema is an expression whose variables are members of

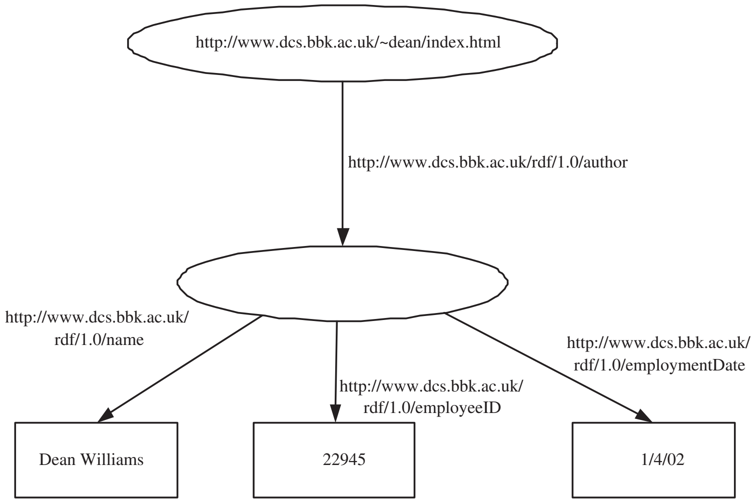

Fig. 1. Example RDF statement

$Nodes \cup Edges$. $Nodes$ and $Edges$ define a labelled, directed, nested hypergraph. It may be 'nested' in the sense that edges can link any number of both nodes and other edges. $Constraints$ is a set of boolean-valued queries over the schema which are satisfied by all instances of the schema. Nodes are uniquely identified by their names. Edges and constraints have an optional name associated with them. Edges need not be uniquely named within an HDM schema but are uniquely identified by the combination of their name and the components they link.

The constructs of any higher-level modelling language $\mathcal{M}$ are classified as either **extensional constructs**, or **constraint constructs**, or both. The **scheme** of a construct (delimited by double chevrons) uniquely identifies it within a schema. Extensional constructs represent sets of data values from some domain. Each such construct in $\mathcal{M}$ is defined as some combination of extensional constructs of the HDM i.e. of nodes and edges. There are three kinds of extensional constructs:

- **nodal** constructs may exist independently of any other constructs and are represented by an HDM node.
- **linking** constructs can only exist when certain other constructs exist. The extent of a linking construct is a subset of the Cartesian product of the extents of its dependent constructs. Linking constructs are represented by HDM edges.
- **nodal-linking** constructs are nodal constructs that can only exist when certain other constructs exist, and that are linked to these constructs (attributes in the relational model are an example). Nodal-linking constructs are represented by a combination of an HDM node and an HDM edge.

Constraint constructs represent restrictions on the extents of extensional constructs.

The RDF constructs URI, Literal and Blank are each represented by an HDM node and are all nodal constructs. The RDF construct Triple is represented by a combination of three HDM nodes, an edge, and three constraints; this construct is thus nodal, linking and constraint. This specification of RDF in the HDM is illustrated in Table 1. For any given RDF description, the HDM nodes ⟨⟨URI⟩⟩, ⟨⟨Literal⟩⟩ and ⟨⟨Blank⟩⟩ have as their extents the set of URIs, literals and blank nodes appearing in the description, respectively, while the HDM node ⟨⟨Triple⟩⟩ has as its extent the set of triples.

RDF Construct	HDM Representation
construct:RDFNode nodal scheme: ⟨⟨URI⟩⟩	node: ⟨⟨URI⟩⟩
construct:RDFNode scheme: ⟨⟨Literal⟩⟩	node: ⟨⟨Literal⟩⟩
construct:RDFNode scheme: ⟨⟨Blank⟩⟩	node: ⟨⟨Blank⟩⟩
construct:RDFEdge class: nodal, linking and constraint scheme: ⟨⟨Triple⟩⟩	nodes: ⟨⟨subject⟩⟩, ⟨⟨predicate⟩⟩, ⟨⟨object⟩⟩ edge: ⟨⟨Triple,subject,predicate,object⟩⟩ three constraints: ⟨⟨subject⟩⟩ $\subseteq$ (⟨⟨URI⟩⟩ $\cup$ ⟨⟨Blank⟩⟩) ⟨⟨predicate⟩⟩ $\subseteq$ ⟨⟨URI⟩⟩ ⟨⟨object⟩⟩ $\subseteq$ (⟨⟨URI⟩⟩ $\cup$ ⟨⟨Blank⟩⟩ $\cup$ ⟨⟨Literal⟩⟩)

Table 1. Definition of RDF model constructs

3.2 RDF Containers and Reification

The use of blank nodes to group together related properties in RDF was discussed above. RDF also provides for containers to refer to collections of resources. There are three types of container in RDF:

Bag: Unordered list with duplicates allowed.
Sequence: Ordered list, duplicates allowed.
Alternative: List of resources that represent alternatives for the single value of a property.

The 'rdf:type' property is used to specify the container and its type. Extending the employee example to show the use of the different container types:

A **bag** could be used to represent any relevant work related qualifications e.g. 'first aider', 'minibus driver'.
A **sequence** could be used to record of the job titles of posts an employee had held — in this case the order in which they had been held is significant and so a sequence is more suitable than a bag.

An **alternative** might be used if there could be more than one telephone in an employee's office and several numbers could be used to contact the employee.

Figure 2 shows these three containers appended to the original example. It is important to note that apart from the type names assigned, the schemas of these three container types are identical — the application program making use of the data will need to be able to interpret their semantic differences.

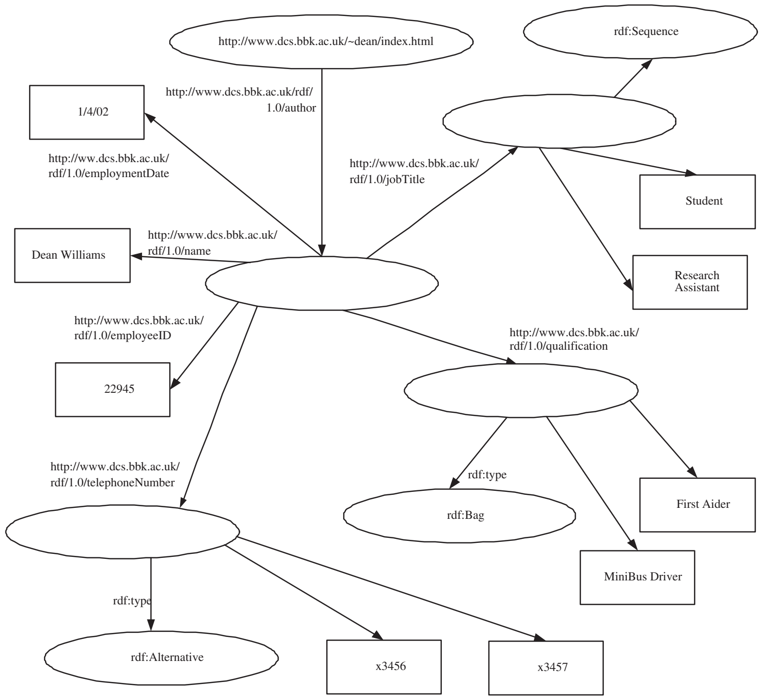

Fig. 2. Extended employee details example showing collections.

It is also important in RDF to be able to make statements about statements e.g. "Personnel says that Dean Williams' employee number is 22945". This is a fact about something the personnel department has said, not about Dean's employee number. In order to make a statement about this statement it is necessary to first remodel the original statement — this process of remodeling is known as reification. RDF has a method of modeling such statements that makes use of a specific value of the property 'rdf:type'. The statement has four properties:

- type of the statement is 'rdf:statement'
- subject is the resource described by the modelled statement i.e. 'Dean' in the example.
- predicate is the original property i.e. employee number.

– object is the object of the original property i.e. ‘22945’.

It is now possible to attach a property ‘informant’ to the reified statement as can be seen in Figure 3.

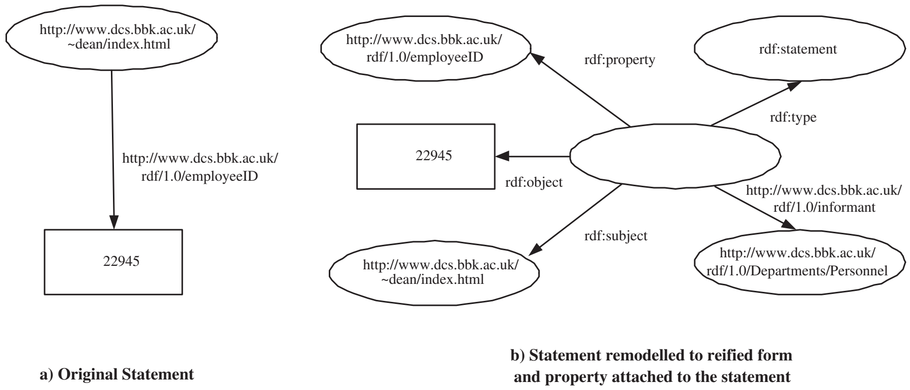

Fig. 3. Statement and its reified form with ’statement about a statement’ added.

The HDM specification for containers and statements is shown in Table 2. The members of containers may be any kind of resource, and resources are represented by an additional HDM node ⟨⟨Resource⟩⟩. There is one HDM edge ⟨⟨bag,Blank,Resource⟩⟩ whose extent is all the container/member associations for bag containers; one HDM edge ⟨⟨sequence,Blank,Resource⟩⟩ whose extent is all the container/member associations for sequence containers; and one HDM edge ⟨⟨alternative,Blank,Resource⟩⟩ whose extent is all the container/member associations for alternative containers.

For bag and sequence containers, an extra HDM node ⟨⟨Number⟩⟩ is used to model members’ cardinality and ordering, respectively (similarly to the way that order was represented for XML in [20]). In particular, we use an additional HDM edge from ⟨⟨bag,Blank,Resource⟩⟩ to ⟨⟨Number⟩⟩, and an additional HDM edge from ⟨⟨sequence,Blank,Resource⟩⟩ to ⟨⟨Number⟩⟩. For all instances of the edge ⟨⟨bag,Blank,Resource⟩⟩, there will be an edge to an instance of ⟨⟨Number⟩⟩ indicating the cardinality of that member in that bag. Similarly, for all instances of the edge ⟨⟨sequence,Blank,Resource⟩⟩, there will be one or more edges to an instance of ⟨⟨Number⟩⟩ indicating the position(s) of that member within that sequence[1].

See `http://www.dcs.bbk.ac.uk/~dean/eg/RDF.java` for a Java program which creates an AutoMed Model for RDF and an AutoMed Schema for storing RDF. Note that this is slightly unusual code for AutoMed as the schema will be the same for all RDF data sources, unlike say the relational model where

[1] We have not shown in Table 2 the cardinality constraints implied by these semantics but they can be expressed in IQL (the AutoMed query language) and we refer the reader to [32] for their definitions.

RDF Construct	HDM Representation
construct:RDFEdge class: nodal, linking and constraint scheme: ⟨⟨Statement⟩⟩	nodes: - edge: ⟨⟨Statement,Blank,subject,predicate,object⟩⟩ constraints: -
construct:RDFContainer class:nodal, linking and constraint scheme: $\langle\langle t\rangle\rangle$	nodes: ⟨⟨Resource⟩⟩, ⟨⟨Number⟩⟩ edge: $\langle\langle t$,Blank,Resource⟩⟩ constraint: ⟨⟨Resource⟩⟩ = (⟨⟨URI⟩⟩ ∪ ⟨⟨Blank⟩⟩ ∪ ⟨⟨Literal⟩⟩) if t = bag or t = sequence then edge: ⟨⟨_, $\langle\langle t$,Blank,Resource⟩⟩, ⟨⟨Number⟩⟩⟩⟩

Table 2. Definition of RDF model constructs —- containers and statements

generally different instances of Schema contain different tables (but all schemas conform to the one relational Model).

3.3 Primitive Transformations on RDF

After a modelling language $\mathcal{M}$ has been specified in terms of the HDM, a set of primitive transformations for schema constructs of $\mathcal{M}$ is then automatically available. One 'family' of primitive transformations is available for each different modelling construct. In the RDF specifications of Tables 1 and 2 there are three different modelling constructs: RDFNode, RDFEdge and RDFContainer, with, respectively, 3, 2, and 3 instances. The set of primitive schema transformations generated for the RDFNode construct from the specification in Table 1 is

- addRDFNode(scheme,query), extendRDFNode(scheme,query)
- deleteRDFNode(scheme,query), contractRDFNode(scheme,query)
- renameRDFNode(scheme,new-name)

where the 'scheme' parameter is one of ⟨⟨Blank⟩⟩, ⟨⟨URI⟩⟩ or ⟨⟨Literal⟩⟩, the 'query' parameter is an IQL query, and 'new-name' can be any name currently not in use (e.g. renameRDFNode(⟨⟨URI⟩⟩,MyURI) is valid but renameRDF(⟨⟨URI⟩⟩,Literal) is not).

The set of primitive schema transformations generated for the RDFEdge construct from the specifications in Tables 1 and 2 are

- addRDFEdge(scheme,query), extendRDFEdge(scheme,query)
- deleteRDFEdge(scheme,query), contractRDFEdge(scheme,query)
- renameRDFEdge(scheme,new-name)

where the 'scheme' parameter may be ⟨⟨Triple⟩⟩ or ⟨⟨Statement⟩⟩.

Finally, the set of primitive schema transformations generated for the RDFContainer construct from the specification in Table 2 are:

- addRDFContainer(scheme,query), extendRDFContainer(scheme,query)
- deleteRDFContainer(scheme,query), contractRDFContainer(scheme,query)
- renameRDFContainer(scheme,new-name)

where the 'scheme' parameter may be ⟨⟨Bag⟩⟩, ⟨⟨Sequence⟩⟩ or ⟨⟨Alternative⟩⟩.

3.4 An Example

We illustrate the above representation of RDF in the HDM by looking at a fragment of the WordNet [23] English language database. An RDF version of this database is available, as is an RDFS specification. Figure 4 shows a graph representation of part of the RDFS specification, including the class 'lexical concept' and its subclass 'noun'. The following properties are defined: 'word form' assigns a string to a lexical concept; 'hyponym of' creates an 'isa' hierarchy of lexical concepts; and 'glossary entry' gives a textual description of a lexical concept.

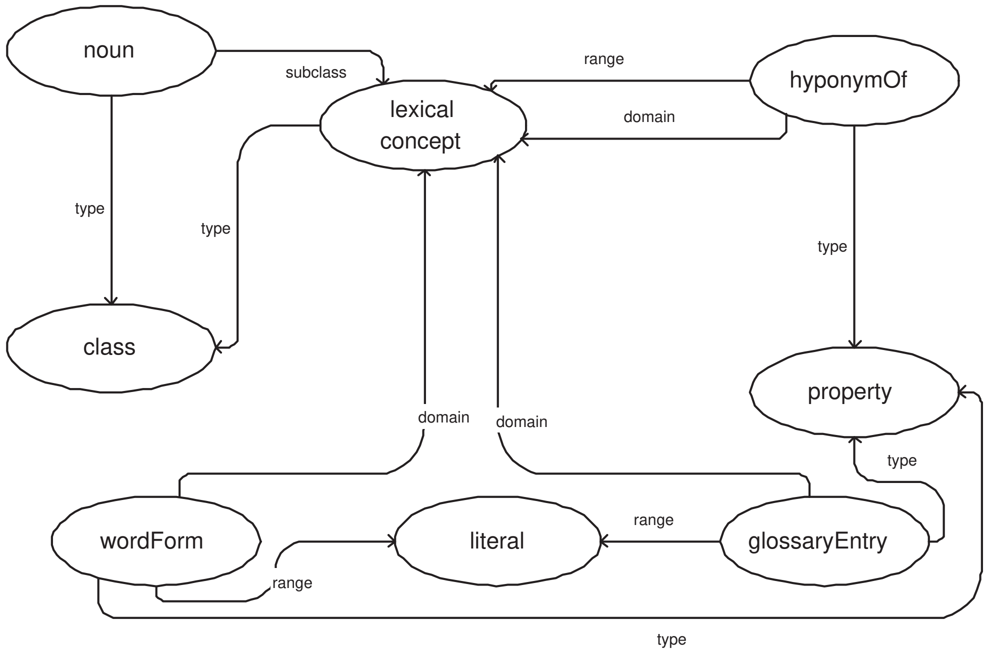

Fig. 4. Graph representation of part of the WordnNet RDFS

We can also represent RDFS in the HDM (see [32] for the details, which we have not given here for reasons of space).

Figure 5 shows in graph form a fragment of the WordNet RDF description for two concepts, "wheeled vehicle" and a hyponym of it, "locomotive". Several alternative word forms for the concept "locomotive" are given e.g. "engine" and "railway locomotive". Glossary entries for the two concepts are also listed.

This RDF model would be represented by following instances of the HDM nodes and edges specified on the right-hand column of Table 1:

```
<<Blank>>   = {}
```

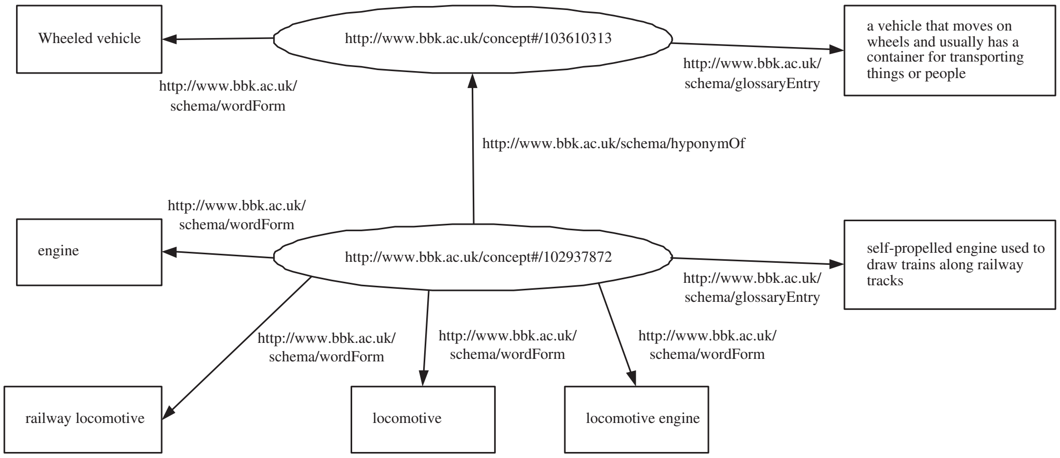

Fig. 5. Graph Based Representation of a fragment of WordNet RDF

```
<<URI>>     = {http://www.bbk.ac.uk/concept/103610313 ,
               http://www.bbk.ac.uk/concept/102937872,
               http://www.bbk.ac.uk/schema/wordForm ,
               http://www.bbk.ac.uk/schema/glossaryEntry ,
               http://www.bbk.ac.uk/schema/hyponymOf }
<<Literal>> = {"Wheeled Vehicle", "engine", "railway locomotive",
               "locomotive", "locomotive engine",
               "self-propelled engine used to draw trains along railway tracks",
               "a vehicle that moves on wheels and usually has a container for
                transporting things or people"}
<<subject>> = {http://www.bbk.ac.uk/concept/103610313 ,
               http://www.bbk.ac.uk/concept/102937872 }
<<predicate>>={http://www.bbk.ac.uk/schema/wordForm ,
               http://www.bbk.ac.uk/schema/glossaryEntry ,
               http://www.bbk.ac.uk/schema/hyponymOf }
<<object>> =  {http://www.bbk.ac.uk/concept/103610313,
               "Wheeled Vehicle", "engine", "railway locomotive",
               "locomotive", "locomotive engine",
               "self-propelled engine....tracks", "a vehicle...or people"}
<<Triple,subject,predicate,object>> = {
              (http://www.bbk.ac.uk/concept/103610313,
               http://www.bbk.ac.uk/schema/wordForm,
               "Wheeled Vehicle"),
              (http://www.bbk.ac.uk/concept/102937872,
               http://www.bbk.ac.uk/schema/wordForm,
               "engine"),
              (http://www.bbk.ac.uk/concept/102937872,
```

```
 http://www.bbk.ac.uk/schema/hyponymOf,
 http://www.bbk.ac.uk/concept/103610313),
(http://www.bbk.ac.uk/concept/102937872,
 http://www.bbk.ac.uk/schema/wordForm,
 "railway locomotive"),
(http://www.bbk.ac.uk/concept/102937872,
 http://www.bbk.ac.uk/schema/wordForm,
 "locomotive"),
(http://www.bbk.ac.uk/concept/102937872,
 http://www.bbk.ac.uk/schema/wordForm,
 "locomotive engine"),
(http://www.bbk.ac.uk/concept/102937872,
 http://www.bbk.ac.uk/schema/ glossaryEntry,
 "self-propelled engine....tracks"),
(http://www.bbk.ac.uk/concept/103610313,
 http://www.bbk.ac.uk/schema/glossaryEntry,
 "a vehicle...or people")}
```

A Java program to create a model for RDFS and build the above schema can be found at `http://www.dcs.bbk.ac.uk/~dean/eg/RDFS.java`

4 ESTEST

Our testbed for combining data integration and information extraction is called **Experimental Software for Extracting Structure from Text (ESTEST)**. ESTEST makes use of AutoMed for its data integration aspects and of an information extraction system to find structure in text. We begin this section with a review of related work. We then describe the ESTEST architecture. As the ESTEST software is still under development, we conclude the section with an extended example in order to illustrate the potential effectiveness of the ESTEST approach.

4.1 Related Work

Information extraction (IE) [1] is a branch of natural language processing that is concerned with extracting pre-defined entities from text and filling in a template with the extracted information. While information retrieval works by identifying and retrieving documents that are of interest, IE extracts data from within documents. Much of the original work on IE was based on participation in the TIPSTER programme which ran from 1991 to 1998, sponsored by DARPA. A key feature of the TIPSTER programme was the series of Message Understanding Conferences (MUC). At these conferences a number of IE systems competed to solve a set of IE tasks. The MUC tasks were: *named entity recognition* i.e. identifying and classify entities in text; *coreference resolution* i.e. identifying multiple references to the same entity including anaphoric references; and *template construction* i.e. filling in fixed format templates describing entities and their attributes (template elements) and identifying relations between template elements (scenario templates).

The degree of success achieved in IE is impressive with systems reaching levels of over 90% for the named entity recognition tasks, over 60% for coreference, over 80% from template elements and around 50% for scenario templates at MUC-7 [5]. As well as the prototype systems developed to compete in these tests (e.g. FASTUS [10], LOLITA [9], GATE [7]), there has been significant interest from industry and commercially available IE products include Quenza [27] for which one of the target markets is crime investigation.

In reviewing the available NLP techniques for our goal of extracting information from text and integrating it with other, semantically related, structured or semi-structured data, we noted the similarity between the IE approach and database schemas. Templates are used to define the entities that will be searched for and their attributes in IE, and these templates can be thought of as fragments of a schema which is to be populated by the IE process. For this reason we decided to investigate the suitability of IE techniques. We reviewed current IE systems against several criteria, including: the ability to integrate the IE components into a standalone system, the range of IE components available as standard, and the stability and extensibility of the software. As a result of this review, we selected the GATE 2 system from the University of Sheffield (`http://www.gate.ac.uk`) as the best base for developing ESTEST's IE functionality.

ESTEST is aimed at applications characterised by the need to integrate information present both in structured and in free-text form. Our testbed applications for ESTEST include crime investigation and road traffic accident reports, both of which collect large amounts of information in both structured and free-text form. In particular, road traffic accident reports in the U.K. are reported using a flat-file format known as STATS-20. In STATS-20, a record exists for each accident, and following this there are multiple records for the people and vehicles involved in the accident. There are also one or more lines of free text, containing information recorded by the Police Officer at the scene of the accident, for which the preceding structured records do not include any pre-defined fields. This includes information about the location of the accident, and a description of how it happened. An example is the following (fictitious) text description for accident number 23781:

```
23781A121 10K N DUNCESTER SOUTH
23781FOX RUNS INTO ROAD CAUSING VEHICLE TO SWERVE VIOLENTLY AND
23781LEAVE ROAD OFFSIDE
```

The first line says that the vehicle was proceeding southwards on the A121, and the accident look place 10 kilometres north of Duncester.

Previous research in analysing Road Traffic Accident Reports [33] investigated extracting location information from STATS-20 free text, using a deeper NLP technique based on description logic, and matching this against a predefined database schema recording known metadata about the U.K. road network. As a proof of concept of the IE approach, we developed a grammar for the locations part of the STATS-20 text and used this with the standard GATE 2 components in order to extract location information about accidents. Our results compared favourably with those from this earlier work, and the identification of locations from the reports was successful in every test case (our test data consisted of a whole year's accidents reports from one U.K. county).

We note however that this earlier work did not attempt the much more challenging task of extracting information from the text describing how the accident happened and this is where the ESTEST approach is aimed at.

Our goal of integrating discovered structure in text with existing known metadata can be thought of as a *schema matching* problem where one side of the match is all known metadata and the other is the collection of discovered facts. Schema matching [28] is concerned with identifying semantic relationships between elements of different schemas. A common feature of all schema matching approaches is the requirement for a match function which uses available information to provide the best match between two fragments of schema. A taxonomy of the approaches developed so far is given in [28]. This taxonomy is based on the types of evidence used in the match function e.g. is it based on instance or schema information, element level or structure level. However in the application areas envisaged for ESTEST, many of the clues available in conventional schema matching will not be present e.g. type information and field names, and so linguistic information has to be relied on.

Previous systems that are most relevant to the work proposed here include LSD [8], WHIRL [6] and TransScm [24]. LSD extracts data from semantically related sites which may have different schemas (e.g. a collection of estate agent websites). It uses a variety of learning algorithms for matching the schemas and providing a mediated schema. WHIRL uses Bayesian learners and bases its matching on textual similarity of names. TransScm makes use of schema matching to derive automatic data translation between to schema instances. This matching is particularly relevant in our context as the schemas are transformed into graphs and a graph-matching algorithm is used to match them.

4.2 ESTEST Architecture

Figure 6 illustrates the ESTEST system architecture and we now describe each of its components.

Initial Integration. The available data sources other than the text (e.g. structured, semi-structured, and domain ontologies) will first be integrated into a single virtual global schema using AutoMed transformation pathways. This global schema is then able to be queried by submitting queries expressed in AutoMed's IQL query language to the global query processor.

Create Data to Assist the IE Process. The global schema can now be used to provide data which assists the information extraction process. For example, lists of entities can be created by submitting queries to the global schema. These

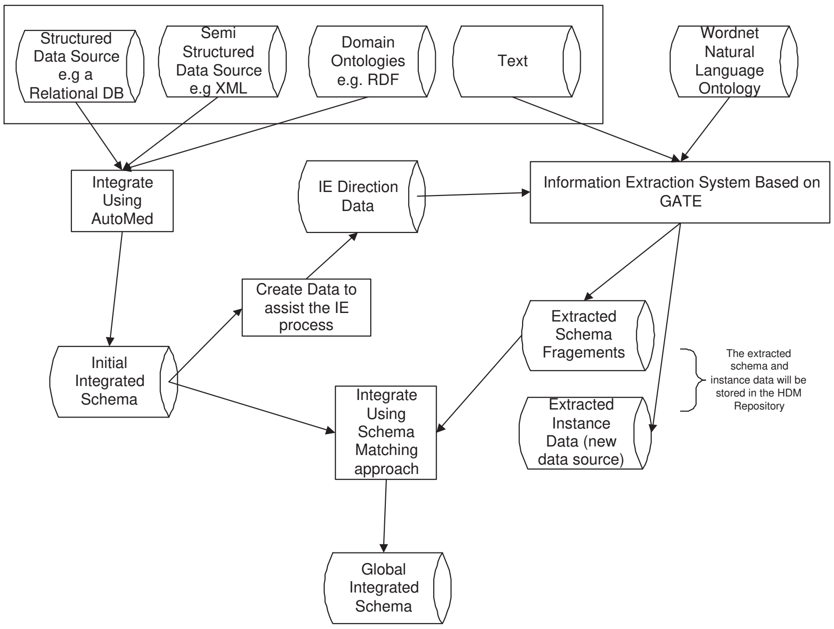

Fig. 6. Overview of the ESTEST System

lists can then be used by the 'named entity recogniser' components of the information extraction system (see below). From the global schema, we can also extract information to create templates for grammars. This is described in more detail in the next section, and is based on extracting text information from the global schema e.g. from table and column names in relational schemas.

Information Extraction System Based on GATE. GATE allows for a sequence of language processing components to be assembled and marks up annotations on the input text. GATE's language processing components include standard components such as sentence splitters and named entity recognisers. Bespoke components can also be constructed and integrated with the existing standard ones.

The GATE system provides a graphical user interface which can be used to test new information extraction components. Once developed and tested these components can then be run in a stand-alone configuration. 'Pipelines' of components can be defined, which allow each of a specified sequence of IE components to be applied in turn.

A pattern-matching language called JAPE is provided by GATE, and it is possible to develop grammars based on JAPE. We are developing new IE components which will generate templates for JAPE grammars, based on the assumption that the entities in the existing schema and domain ontologies will be at least a significant subset of the entity types for which we wish to extract information from the text.

There will need to be some human intervention in the development of the grammars, but we will make as much use as possible of the available schema information and the associated textual information e.g. table and column names in relational databases, and the associated textual description metadata that is provided by many database systems.

We are also developing a WordNet component which will make use of its synonym and hyponym structures to allow for alternative lists for words to be found in cases where the textual descriptions of schema elements is restricted, for example to a word in a column name.

Once the pipeline of GATE components have processed the input text, the result is a set of annotations over the text, each of which consists of the start and end positions in the text and the annotation type. For example for the text "dark blue Mondeo" and assuming the availability of some JAPE rules for handling colours and makes of cars, the resulting annotations might be:

```
00 03 string         ("dark")
04 04 separator      (space)
05 08 string         ("blue")
09 09 separator      (space)
10 15 string         ("mondeo")
00 03 colourShade    ("dark")
05 08 colour         ("blue")
00 08 colour         ("dark blue")
10 15 carModel       ("Mondeo")
00 15 car            ("dark blue Mondeo")
```

These annotations can be thought of as discovered fragments of schema. These fragments and the text to which they refer will be stored in the Extracted Schema Fragments and Extracted Instance Data store, respectively, both of which will be implemented using our HDM repository (see Section 5 below).

Integrate New and Existing Metadata Using Schema Matching A schema matching algorithm will take each new extracted schema fragment and find its best match with respect to the global schema, or allow for it to be appended to the global schema. Unlike many other schema matching applications, there will not be much structural information available to assist the matching algorithm and we will rely primarily on element names. However, we will also experiment with using the new instance values extracted, looking to see if these are already present in the extents of candidate schema entities and using any presence to provide evidence of a semantic match.

Once the new schema fragments have been integrated with the global schema, the data in the Extracted Data store can be treated as a new AutoMed data source and queries against the global schema will also be able to make use of this data.

4.3 An Example

Returning to the Road Traffic Accidents reports mentioned earlier, suppose that the structured information relating to all accidents in a particular region over a particular period has been stored in a relational database, AccDB, while the corresponding free text portions of the accident reports are available as a separate text file. In our simplified example here, AccDB consists of two tables

```
Accident(accNo,road,roadType)
Vehicle(accNo,vehNo,vehType)
```

In the Accident table, each accident is uniquely identified by an accNo, the road attribute identifies the road the accident occurred on, and roadType indicates the type of road. There may be zero, one or more vehicles associated with an accident, and information about each them is held in a row of the Vehicle table. Here vehNo uniquely identifies each vehicle involved in an accident and thus accNo,vehNo is the key of this table. (In the real data, there are also tables about the casualties involved in the accident, other people involved — the driver, passengers, etc., and information about the road and weather conditions.)

An example of the text entries collected in the separate text file might be:

```
23781FOX RUNS INTO ROAD CAUSING VEHICLE TO SWERVE VIOLENTLY AND
23781LEAVE ROAD OFFSIDE
```

Thus, in order to answer queries such as "on which roads were there accidents caused by animals and what animals caused them?" we would need to retrieve and integrate information from the text as well.

The WordNet ontology can first be used to extract information about possible animals. Suppose therefore we have an RDF representation of WordNet which contains triples of the following form describing animals:

```
concept_0, wordForm, animal
concept_1, hyponymOf, concept_0
concept_1, wordForm, carnivore
concept_2, hyponymOf, concept_1
concept_2, wordForm, canine
concept_3, hyponymOf, concept_1
concept_3, wordForm, feline
concept_4, hyponymOf, concept_2
concept_4, wordForm, dog
concept_5, hyponymOf, concept_2
concept_5, wordForm, fox
concept_6, hyponymOf, concept_3
concept_6, wordForm, cat
```

We now need to construct a (virtual) Integrated Schema which combines the information in the relational database, AccDB, with the information in this RDF data source. In our example, the Integrated Schema is a relational schema, but in practice it may be expressed in any data model supported by AutoMed. These are the tables of the Integrated Schema:

```
Accident(accNo,road,roadType)
Vehicle(accNo,vehNo,vehType)
Animals(animal)
AnimalInRoad(accNo,animal)
```

The information in Accident and Vehicle will be sourced from the relational database, the information in Animals from the RDF data source, and the information in AnimalInRoad from the Extracted Instance Data store created by the ESTEST information extraction system.

We need to define two pathways, one from the RDF data source, RDFDS, to the Integrated Schema, IS, and one from AccDB to IS. The following AutoMed pathway, RDFDS $\rightarrow$ IS, transforms RDFDS into IS. It first creates a new table Animals(animal) which contains the set of animals in RDFDS. Here, the IQL subquery `linkedTo('hyponymOf','concept_0')` returns the set of concepts in the RDF source which are recursively linked to `concept_0` (*i.e.* `animal`) by edges labelled `hyponymOf`. We assume that it can be translated by the RDFDS wrapper into a local query on the RDFDS. All the RDF information is then dropped from the schema, and contract transformations are used since this information cannot be reconstructed from the remaining schema constructs. A new table AnimalInRoad(accNo,animal) is then added the schema, and an extend transformation is used since this information cannot be derived from the RDFDS[2]. The transformation finally extends the schema to include the `Accident` and `Vehicle` tables (whose content will be sourced from AccDB):
RDFDS $\rightarrow$ IS:

```
addTable(<<Animals,animal>> [a | c<-linkedTo('hyponymOf','concept_0');
                                 (c,'wordForm',a) <- <<Triples>>]);
contractRDFEdge(<<Triples>>);
contractRDFNode(<<URI>>);
contractRDFNode(<<Literal>>);
contractRDFNode(<<Blank>>);
extendTable(<<AnimalInRoad,accNo,animal>>);
extendTable(<<Accident,accNo,road,roadType>>);
extendTable(<<Vehicle,accNo,vehNo,vehType>>);
```

The transformation pathway from AccDB to IS is simpler and consists of extend steps to include the missing `Animals` and `AnimalInRoad` tables:
AccDB $\rightarrow$ IS:

```
extendTable(<<Animals,animal>>);
extendTable(<<AnimalInRoad,accNo,animal>>);
```

[2] This is a simplified presentation of how relational schemas are actually encoded in AutoMed, which suffices for the purposes of this paper. We refer the reader to [19, 22] for the full details.

ESTEST can now use the Integrated Schema IS to extract information about animals in roads from the text. The IE system will have been given a named entity recogniser for animals, and grammar rules for detecting when animals cause accidents (as opposed to when they are victims of accidents, say). An example of the kind of JAPE grammar rule used might be:

```
Macro: CAUSEACTION
// e.g. "RUNS INTO", "RUNS ONTO", "WALKS IN FRONT OF"
(
  ({Token.string == "RUNS"} |
   {Token.string == "WALKS"} |
   {Token.string == "JUMPS"}  )

  (SPACE)?

  ({Token.string == "INTO"} |
   {Token.string == "ONTO"} |
   {Token.string == "IN FRONT OF"}  )

  (ANIMAL)
)
```

In this example rule, possible combinations of text such as "RUNS IN FRONT OF" or "WALKS INTO" are combined with the ANIMAL token which will be populated by the named entity recogniser based on a list of animal words from IS. The resulting text annotations will be extracted and, for this example, the annotations will be:

```
00 02 animal         ("FOX")
00 17 causeaction    ("FOX RUNS INTO ROAD")
```

The resulting Extracted Schema Fragments are as follows (expressed as edges in the HDM data model):

```
<<_,accNo,animal>>
<<_,accNo,causeaction>>
```

which respectively express the fact that there is an (unnamed) association between `accNo` and `animal`, and between `accNo` and `causeaction`. The corresponding Extracted Instance Data consists of one record

```
[23781,"FOX"]
```

within the extent of `<<_,accNo,animal>>`, and one record

```
[23781,"FOX RUNS INTO ROAD"]
```

within the extent of `<<_,accNo,causeaction>>`. It will now be straightforward for the schema matcher to identify that the HDM edge `<<_,accNo,animal>>`

matches the table AnimalInRoad(accNo,animal) in the Integrated Schema IS. A transformation pathway from the Extracted Schema Fragments (ES) to IS will then be automatically generated which contracts the <<_,accNo,causeaction>> HDM edge, replaces the the <<_,accNo,animal>> HDM edge by the AnimalInRoad(accNo,animal) table (by means of an add followed by a delete step), and uses a series of extend steps to add the missing Animals, Accident and Vehicle tables:
ES $\rightarrow$ IS:

```
contractHDMEdge(<<_,accNo,causeaction>>);
addTable(<<AnimalInRoad,accNo,animal>>, <<_,accNo,animal>>);
deleteHDMEdge(<<_,accNo,animal>>,(<<AnimalInRoad,accNo,animal>>);
extendTable(<<Animals,animal>>);
extendTable(<<Accident,accNo,road,roadType>>);
extendTable(<<Vehicle,accNo,vehNo,vehType>>);
```

The final result is thus three AutoMed transformation pathways from the three heterogeneous data sources to the virtual Integrated Schema IS: RDFDS $\rightarrow$ IS, AccDB $\rightarrow$ IS, ES $\rightarrow$ IS.

AutoMed's global query processor is able to traverse the reverse pathways from the IS to the three sources in order to create a view definition for each schema construct in IS in terms of the schema constructs in the three sources. When a query q on IS is submitted to the global query processor for evaluation, it substitutes these view definitions into q in order to reformulate it into a query over the data sources which, after some optimisation, can be evaluated over the local data sources [12] [3].

We can therefore pose this IQL query on IS to find "on which roads were there accidents caused by animals and what animals caused them?":

```
[(r,a)|(acc,a)<-<<AnimalInRoad,accNo,animal>>;
       (acc,r,t)<-<<Accident,accNo,road,roadType>>)]
```

The views generated for the global constructs <<AnimalInRoad,accNo,animal>> and <<Accident,accNo,road,roadType>> by AutoMed's global query processor are ES:<<_,accNo,animal>> OR AccDB:Void OR:RDFS:Void and ES:Void OR AccDB:<<Accident,accNo,road,roadType>> OR RDFS:Void — the prefix ES, AccDB or RDFS indicates the schema from which the construct is sourced. Standard query rewriting techniques can be applied to generated view definitions to remove instances of Void (see [12]). In the above case the two views simplify to just ES:<<_,accNo,animal>> and AccDB:<<Accident,accNo,road,roadType>>.

We note that the query above joins information from the AccDB and the Extracted Instance Data and could not be answered using either source alone.

[3] More specifically, after q has been reformulated and optimised, sub-queries of it are submitted to the appropriate data source wrappers for translation into the data source query languages and evaluation at the data sources. The wrappers translate sub-query results back into the IQL type system and the global query processor undertakes any further necessary post-processing of the query.

4.4 Supporting Schema Evolution

A final point to note is the ease with which successive data sources can be incrementally integrated with the Integrated Schema. This is something that cannot be done using the GAV approach to data integration where a change in the local data sources impacts, in the worst case, on all the integration rules defining global constructs in terms of local ones.

It could be done if the LAV data integration approach were used, but this approach would have problems if the Integrated Schema evolved, since all the rules for defining local schemas as views of the global schema would need to be reviewed. This is likely to happen in ESTEST (although it did not need to in the above example) since in general matching the Extracted Schema fragments may result in a semantic extension to the Integrated Schema.

Thus, we can conclude that AutoMed's Both-As-View integration approach is well-suited to the data integration requirements of the ESTEST system.

5 The HDM Repository

The ESTEST system will generate new data and metadata in its Extracted Instance Data and Extracted Schema fragments, respectively. While it would have been possible to use any of the data models already supported by AutoMed for this task, we have implemented a native HDM repository as the low-level HDM is well-suited to the task of storing both data and metadata simultaneously. We give a brief overview of this repository here, and refer the reader to [31] for more details.

We note that the ESTEST system will not extract constraints from the text, just extensional information. Thus, in our HDM repository, HDM schemas consist of nodes and edges e.g.⟪person⟫ is a node and ⟪worksIn,person,room⟫ is an edge. Edges can be of any arity. Edges can be named or unnamed e.g. ⟪_,person,room⟫ or ⟪worksIn,person,room⟫. Each component of an edge can be either a node or another edge e.g.
⟪livesAt,person,⟪address,houseNumber,road,town,postCode⟫⟫. Edges need not have unique names e.g. both ⟪worksIn,person,project⟫ and ⟪worksIn,person,room⟫ can be present in a schema. Nodes have an associated data type e.g. integer, string, date etc.

For example, suppose Dean, Mat, Hao and Edgar are people, Hao, Edgar and Dean work on the AutoMed project, Dean and Mat work on the Tristarp project, Mat and Dean sit in room B34E, Edgar and Hao sit in room BG26, and Dean lives at 64, Northdown Street, London N1 9BS. This data conforms to the following 'personnel' schema:

Nodes:
⟪person⟫, ⟪project⟫, ⟪room⟫, ⟪houseNumber⟫, ⟪road⟫, ⟪town⟫, ⟪postCode⟫
Edges:
⟪worksIn,person,project⟫, ⟪worksIn,person,room⟫,
⟪address,houseNumber,road,town,postCode⟫,

⟨⟨livesAt,person,⟨⟨address,houseNumber,road,town,postCode⟩⟩⟩⟩
For the purposes of this example, we assume that the data types of all the nodes are string except for ⟨⟨houseNumber⟩⟩ which is an integer.

HDM schema constructs are identified by their scheme enclosed within double-chevrons. To distinguish between metadata and data, we enclose HDM data in square brackets. For example, [dean,[64,Northdown Street,London,N1 9BS]] is an instance of ⟨⟨livesAt,person,⟨⟨address,houseNumber,road,town,postCode⟩⟩⟩⟩.

The HDM repository is implemented using the Postgres relational database. To allow for multiple users of the repository, users create logical HDM repositories of which any number can be housed in a single set of database tables. At the time of creation, an HDM schema is associated with an HDM repository which will be used to validate the data — this schema must already have been defined within the AutoMed Schemas & Transformations Repository (and must conform to the HDM Model within the AutoMed Models Definitions Repository). Node and edge data is stored as described above, with edges able to be of arbitrary length and edges able to connect other edges.

A Java API provides several facilities, including creation of the Postgres database, maintenance of the HDM repository, insertion/deletion of nodes and edges, and a set of query functions. A Parser takes text files containing schema and data and inserts them into the HDM repository. As the syntax for defining HDM data is quite verbose, the parser contains a number of syntax shortcuts which have been provided to make defining an HDM database as easy as possible. An example of input to the parser is as follows (note that the default data type assumed is string which can be overriden with a different type):

```
createdb;
newstore personnel birkbeck2;
use birkbeck2;
addmissingnodes;
settype houseNumber integer;
add <<worksIn,person,project>>
    [dean,tristarp] [mat,tristarp] [hao,automed] [edgar,automed]
    [dean,automed];
add <<worksIn,person,room>>
    [mat,B34E] [dean,B34E] [hao,NG26] [edgar,NG26];
add <<address>> [64,Northdown Street,London,N1 9BS] &1;
add <<address>> [12,Malet Street,London,WC1E 7HX] &2;
add <<livesAt>> [dean,&1];
add <<livesAt>> [edgar,&2];
```

Finally, a Wrapper has been implemented which allows an HDM repository to be accessed by the AutoMed global query processor and for data in the HDM store to thus participate in global query processing, as for any other data source supported by AutoMed.

6 Conclusion

In this paper we have described an architecture which combines the AutoMed data integration approach with Information Extraction technology in order to allow information from ontologies and natural language sources to be integrated with other, semantically related, structured or semi-structured data.

We believe that AutoMed is a promising system for performing data integration on the Semantic Web:

- The low-level, graph-based nature of the HDM lends itself naturally to modelling both structured and semi-structured information, and for discovering structure in text where both the schema and the instance data may be extended as part of the discovery process.
- AutoMed's bidirectional pathways result in easy support of schema evolution, both of local data sources, and of integrated virtual schemas; this is very likely to be needed in the dynamic environment of Web applications.
- AutoMed's fine-grained schema transformations make BAV pathways amenable to automatic or semi-automatic generation.

We have described extensions made to the current AutoMed system to support our architecture by handling RDF resources and by implementing a repository for HDM data and metadata.

We believe that Information Extraction is a promising approach for extending AutoMed to handle free-text information sources:

- There is a natural affinity between IE's discovery of pre-defined entities and with database schemas.
- IE's effectiveness has been demonstrated in the MUC tests mentioned earlier. Our experiments with GATE on Road Traffic Accident data have also demonstrated the effectiveness of the approach and toolset with respect to earlier work in this area.
- GATE provides an extensible, reliable toolset which can be used to develop the ESTEST Information Extraction system we have described above without having to reinvent already known components and methods.

The ESTEST system we have described here extends traditional data integration systems by using IE to handle natural language. We have shown related schema and ontology information (which are central to the Semantic Web) can be used to assist the IE process. We have also shown how ESTEST will make use of the extracted information and integrate it as a new data source with respect to a global schema.

We are currently implementing the ESTEST system and will test further its effectiveness in a number of application areas including Road Traffic Accidents and Operational Intelligence Police Reports. There are a number of research directions for further work within the approach described here, including the use of metadata to drive Information Extraction, and schema matching where only text and metadata is available.

References

1. D. Appelt. An introduction to information extraction. *Artificial Intelligence Communications*, 1999.
2. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. *The Internet Engineering Task Force*, 1998. http://www.ietf.org/rfc/rfc2396.txt.
3. M. Boyd, P.J. McBrien, and N. Tong. The automed schema integration repository. In *Proc. BNCOD02, LNCS 2405*, pages 42–45, 2002.
4. S.S. Chawathe *et al.* The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. 10th Meeting of the Information Processing Society of Japan*, pages 7–18, October 1994.
5. N. A. Chinchor. Overview of MUC-7/MET-2, 2001. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/overview.html.
6. W. W. Cohen. Integration of hetrogeneous databases without common domains using queries based on textual similarity. *Proc ACM SIGMOD*, pages 201–212, 1998.
7. H. Cunningham, R. Gaizauskas, K. Humphreys, and Y. Wilks. Experience with a language engineering architecture: Three years of gate. *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*, 1999.
8. A. Doan, P. Domingos, and A. Levy. Learning source descriptions for data integration. *WebDB (Informal Proceedings)*, 2000.
9. R. Garigliano, A. Urbanowicz, and D.J. Nettleton. Description of the Lolita System as used in MUC-7, 2001. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/dur_muc7.pdf.
10. J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. Fastus: A cascaded finite-state transducer for ex-tracting information from natural-language text. in finite state devices for natural language processing. *MIT Press*, 1996.
11. E. Jasper. Global query processing in the AutoMed heterogeneous database environment. In *Proc. BNCOD02, LNCS 2405*, pages 46–49, 2002.
12. E. Jasper, N. Tong, P. McBrien, and A. Poulovassilis. View generation and optimisation in the AutoMed data integration framework. Technical report, AutoMed Project, 2003.
13. O. Lassila and R.R. Swick. Resource description framework (RDF) model and syntax specification. *W3C Recommendation*, 1999. http://www.w3.org/TR/REC-rdf-syntax/.
14. A.Y. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.
15. A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS'95*, pages 95–104. ACM Press, May 1995.
16. A.Y. Levy, A. Rajamaran, and J.Ordille. Querying heterogeneous information sources using source description. In *Proc. VLDB'96*, pages 252–262, 1996.
17. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. In *Proc. VLDB'01*, pages 241–250, 2001.
18. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proc. ER'99, LNCS 1728*, pages 96–113, 1999.
19. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99, LNCS 1626*, pages 333–348, 1999.

20. P.J. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01, LNCS 2068*, pages 330–345, 2001.
21. P.J. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proc. CAiSE'02, LNCS 2348*, pages 484–499, 2002.
22. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03 (to appear)*, 2003.
23. G.A. Miller, R.Beckwith, C.Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.
24. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. *Proc. VLDB'98*, pages 122–133, 1998.
25. A. Poulovassilis. The AutoMed Intermediate Query Language. Technical report, AutoMed Project, 2001.
26. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
27. QUENZA. http://www.xanalys.com/quenza.html.
28. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10:334–350, 2001.
29. M.T. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for data sources. In *Proc. VLDB'97*, pages 266–275, Athens, Greece, 1997.
30. M. Templeton, H.Henley, E.Maros, and D.J. Van Buer. InterViso: Dealing with the complexity of federated database access. *The VLDB Journal*, 4(2):287–317, 1995.
31. D. Williams. The automed HDM data store. Technical report, Automed Project, 2003.
32. D. Williams and A.Poulovassilis. Representing RDF and RDF Schema in the HDM. Technical report, Automed Project, 2003.
33. J. Wu and B. Heydecker. Natural language understanding in road accident data analysis. *Advances in Engineering Software*, 29:599–610, 1998.