Integration operators for generating RDF/OWL-based user defined mediator views in a grid environment

Abdel-Rahman H. Tawil · Adel Taweel · Usman Naeem · Matthew Montebello · Rabih Bashroush · Ameer Al-Nemrat

Abstract Research and development activities relating to the grid have generally focused on applications where data is stored in files. However, many scientific and commercial applications are highly dependent on Information Servers (ISs) for storage and organization of their data. A data-information system that supports operations on multiple informa-tion servers in a grid environment is referred to as an interoperable grid system. Different perceptions by end-users of interoperable systems in a grid environment may lead to dif-ferent reasons for integrating data. Even the same user might want to integrate the same distributed data in various ways to suit different needs, roles or tasks. Therefore multiple mediator views are needed to support this diversity. This paper describes our approach to supporting semantic interoperability in a heterogeneous multi-information server grid envi-ronment. It is based on using Integration Operators for generating multiple semantically rich RDF/OWL-based user defined mediator views above the grid participating ISs. These views support different perceptions of the distributed and heterogeneous data available. A set of grid services are developed for the implementation of the mediator views.

Keywords:
Integration operators for generating RDF/OWL-based user defined mediator views in a grid environment

1 Introduction

One way to support interoperability in a multi-ISs grid environment is to define a single centralized global view conforming all participating schemas of the individual ISs to a common data model, query language and management system. In this approach the integrated schema view is a consistent, non-redundant and a semantically complete representation of all the distributed and heterogeneous information available (Sheth and Larson 1990). However, a single view has a number of serious disadvantages. For example, the global schema may grow very large and becomes rigidly complex, especially if the participating ISs are numerous, with individually large schemas of structural and semantic heterogeneity. Also, the system integrator, not the users, normally identify and resolve all conflicts among all participating ISs even though some might never arise. As such, the creation of such a schema is complex, and is not suitable for users with diverse requirements nor for applications that requires frequent modifications to their local schemas and, consequently, to the composite global integrated schema. In a grid environment, conforming databases to a single semantic interpretation of the underlying information, as is usually the case in the global schema approach, is rarely acceptable to grid end users. Different end users may have different reasons for integrating data, and each user might wish to define his own mediator views of the underlying servers, without having to conform to some other user's views.

In Tawil et al. (2002, 2008) we argue that a better, more flexible approach to support heterogeneous ISs interoperability would result from creating user defined mediator views above the distributed databases of the ISs - with each mediator view tailored towards a certain group of specific users' requirements or application programs functionalities. These user-defined mediators are created by applying a set of semantically-rich integration operators, capable of reconciling the semantic and structural differences that are likely to arise when integrating distributed, heterogeneous and autonomous servers. The idea of mediating different heterogeneous ISs using local semantic relationships was explored in federated and interoperable multi-database approaches (e.g., Rivero et al. 2011a; Ricci et al. 2010; Koehler and Benkner 2009; Brezany et al. 2004; Chen et al . 2004; Halevy et al. 2003; Nejdl and Wolf 2002; Ludascher et al. 2001; Manolescu et al. 2001; Papakonstantinou et al. 1996; Arens et al. 1993). There, it is assumed that each mediator view in a multi-database environment provides integrated access to data stored in multiple autonomous and heterogeneous ISs and the focus is on mapping between the stored relations across the multi-databases. In a sense, a mediator is a view of data found in one or more ISs. Data does not exist at the mediator but one may query the mediator as if it is where the data is stored.

The Piazza peer data management system (Halevy et al. 2004) supports decentralized data management in a network of loosely coupled mappings between peers (nodes) schemas. Query processing is based on reformulating mediator queries and integrating query results using mapping rules. In the Edutella (Qu and Nejdl 2004) system, nodes can either agree to use the same schema or use different schemas. The Edutella mediator is responsible of handling mappings between the different schemas and uses them in order to translate queries from one schema to the other. However, as the semantics of the individual schemas are not explicitly defined mapping rules are difficult to maintain. When for example a new node joins the network, new mapping rules should be created and several current mapping rules should be changed. Similar problems appear in case a node leaves the network. The

SQPeer system (Kokkinidis and Christophides 2004) uses the Semantic Overlay Networks (SONs) technology, according to which peers sharing the same schema information about a community domain are clustered together to the same SON. The active schema of the peer is a subset (view) of a unique global SON schema for which all classes and properties are populated in the peer base. The system identifies peers that can answer a query by maintaining indexes on schema information on peers. Thus, query processing is not based on query reformulation. RDFSculpt (Kaoudi et al. 2005), introduces integration operators to manipulate RDF schemas. A key feature of their work is that such integration is based on set-like semantics. They define three binary operators (union, intersection, difference) that can be applied on existing RDF schema graphs as a whole, and produce new ones. A unary operator is also defined that can be applied on one RDF schema graph and return a part (subset) of it. Defined operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. Recently, grids are emerging as the infrastructure and set of protocols that enables integrated, collaborative management and use of inter-connected hardware and software resources (Mukherjee and Watson 2012; Young-Jin and Thottan 2011; Manuali and Grif 2011). This raises the question as to how ISs can be deployed and interoperate in such a new paradigm where a large amount of decentralized independently administrated sources can be involved in the grid-scale sharing cycle. Within a grid environment interoperable server systems are usually heterogeneous, this means that there are heterogeneity problems that needs to be resolved, where heterogeneity may arise at the hardware, operating system, data model, database management system, query language and also at the semantic interpretation of the stored information.

In our approach, data sources do not have to follow strict modeling restrictions imposed by the unique global schema. This is due to using integration operators capable of generating multiple semantically rich RDF/OWL-based user defined mediator views above the grid participating ISs. These views support different perceptions of the distributed and heterogeneous data available. Also, a set of grid services is developed for the implementation of the mediator views. Accordingly, this paper describes the design and implementation of a language for generating user defined mediator views based on a set of integration operators that are particularly suitable in grid-settings. The integration operators are built upon several semantic web standards and grid technologies and are capable of reconciling conflicting schema elements and constructing homogenized, customized and semantically-rich mediator views. It is the responsibility of the mediator to provide the service of going to its sources and find the answer to the query. The paper is organized in the following way. In the next section we describe a high level view of our prototype system (called MetaMed) service-based architecture, and a brief description of its components. Section 3 contains an in depth discussion of the integration language operator services required for semantic interoperability by our system. The predefined semantics of each integration operator service in terms of its inputs (e.g. local integration units) and outputs (global classes) are discussed, also the *semantic preservation rules* service for validating user decisions when applying these operators are defined. In Section 4, we present an integration scenario to show how our set of integration language services are used to define a mediator global schema, that is a mediator view of grid data stored in local ISs. Section 5 portrays how the system exploits the information on the global schema to enable "mediated access" services to data stored in Grid-enabled ISs. Lastly, Section 6 concludes this paper.

2 The MetaMed service-based architecture

The MetaMed mediation system provides integrated access via a general service-based architecture approach (Fig. 1) that is common to many grid projects. A high level view of the MetaMed system architecture shows that our system consists of the following three distinct layers: the *ISs layer*, the *grid services layer*, the *Ontology* and *mediation services layer*:

- **The information servers layer** consists of (i) the local ISs which will eventually be available to the end (global) users (ii) a set of low level application programming interfaces (i.e. *JDBC/ODBC drivers* (Orfali and Harkey 1996)) to provide direct access to the underlying ISs, and (iii) wrappers designed to be used in an information grid infrastructure that may initially rely upon the OGSA-DAI (Jackson et al. 2011) grid middleware. Wrappers (also called *translators*) (Papakonstantinou et al. 1995) provide a common query language and common data model for extracting the information needed to bring data uniformly back into the grid.
- **The grid services layer** consists of specific functionalities particularly related to grid facilities, is provided by using the standard Globus Toolkit Services. The Open Grid Services Architecture Data Access and Integration (OGSA-DAI) as well as grid functionality and database services (Muppavarapu et al. 2010) offered by other flavours of

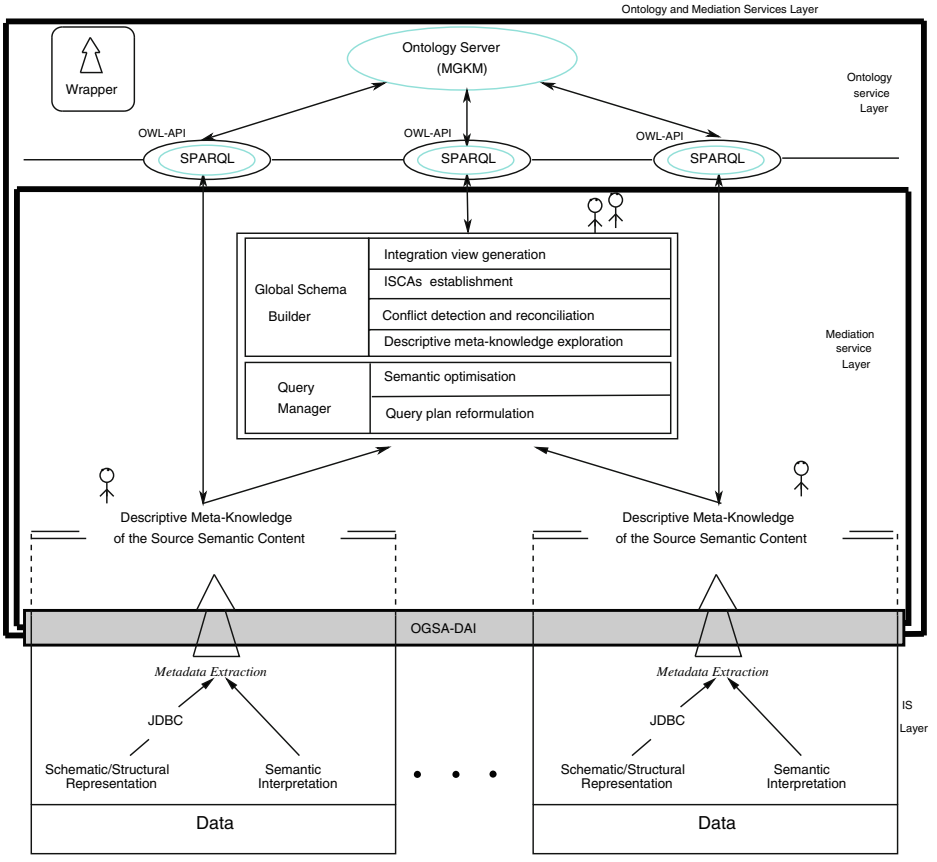


Fig. 1 High level view of metamed system architecture

grid middleware is shown on this level. OGSA-DAI is a middleware product that allows ISs, such as relational or XML databases, to be accessed in a data grid. Furthermore, it provides some of the base capabilities that would be required to run higher-level data integration services. Using OGSA-DAI interfaces disparate, heterogeneous data resources may be accessed in a uniform manner. It is important to note though that the underlying data model of each source is not hidden - a relational data resource remains a relational resource and the semantic and structural heterogeneity across grid databases will not masquerade. OGSA-DAI does not support database query languages heterogeneity and does not provide a query language that is not already supported by the data resource or through some other third party application.

- **The mediation services layer** consists of the core system modules which collectively provide the mediation services and it is composed of two main modules:

  1. Global Schema Generator (GSG): this includes the set of modules which are responsible for generating the global schema to be provided for the user, starting from the *source descriptive meta-knowledge* formulated in terms of the generic RDF/OWL-based domain model ontology (called the MGKM).[1] The GSG provide the following services: (1) descriptive meta-knowledge exploration, (2) conflict detection and (3) reconciliation, (4) interschema correspondence assertions establishment, (5) integration views generation. Moreover, the module performs (6) semantic validation of integration decisions taken by the user during the building of the integrated views. The result of the overall process is the so-called Mediator Global Schema view (see Fig. 2).
  2. Query Manager (QM): this is the query management module. It generates OQL_{I^3} (Hillairet et al. 2009) language queries to be sent to wrappers starting from each query posed by the user on the global schema. The QM provides the service of automatically optimising and reformulating global queries into a corresponding set of sub-queries for the individual ISs.

- **The ontology Services Layer:** A major source of knowledge for mediation comes from the *ontology layer* (Rivero et al. 2011b; Vidal et al. 2011). The MetaMed Ontology services targets two basically different application requirements. First the grid itself needs an ontology service. This service provides meta information about grid entities. Secondly, the MetaMed network Terminology Server (MTS) provides common terms (vocabulary) which are used to construct metadata descriptions that characterize the underlying ISs semantic content or to specify an information request, refer to Tawil et al. (1998) for a more detailed discussion about this layer. All interactions with this layer are achieved via OWL API (Horridge and Bechhofer 2011) abstraction layer which sits above an SPARQL/Jena query interface (Kollia et al. 2011).

Registered services at the grid layer are published in a registry. The grid middleware can query the registry to discover information on behalf of applications. These services about services are rather well known and deal with established web services concepts. A meta-registry *knows* the technical aspects about other services and responds to questions about their properties. Software modules at the mediation services layer communicates with both the ontology and the ISs layers using OWL API calls and JDBC API calls respectively. Wrappers convert data from each source into a common model. A wrapper for a particular source converts queries submitted in the common grid format and query language into

[1] MGKM stands for the MetaMed Generic Knowledge Model

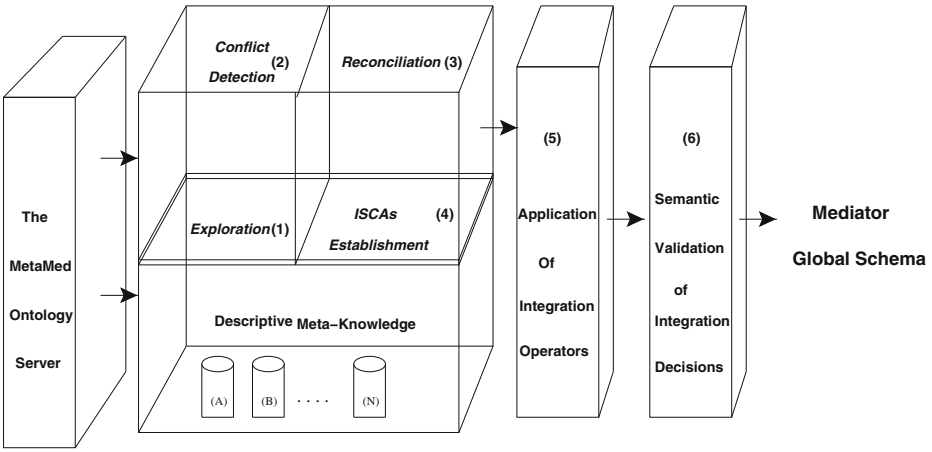


Fig. 2 Mediator global schema generation

queries understandable by that source. The wrapper also receives the results from a source, and converts them into a model understood by the application. Applications can access data directly through wrappers, but they may also go through *integrated views*. In MetaMed wrappers are used to make local ISs comply with the internal MGKM representation of our system. Wrapping services act as transducers between the local ISs and the MetaMed high level integration services and connect to the ISs through JDBC/ODBC drivers, and to Ontology knowledge base services through OWL API-SPARQL drivers.

In the next section we describe the set of RDF/OWL-based representation format for our integration operators used in generating user-defined mediator views. The generated global views consist of virtual classes, that derives their instances from their corresponding local integration units.

3 RDF/OWL-based integration operators for generating global views

RDF/OWL is being promoted as a standard for web-ontology language. Consequently, a considerable number of ontologies are and will continue to be developed based on OWL. The language was originally created to enable not only the representation of the data itself, but also its interpretation, i.e. knowledge about its format, usage or embodiment in a specific framework. This work aims at modelling the representation of both ISs data and schema components for grid-enabled servers using integration operators mediation services based on OWL. MetaMed uses a schema integration language capable of reconciling conflicting schema elements and constructing homogenized, customized and semantically-rich integrated views (Tian et al. 2008; Al-Mourad et al. 2003; Duwairi 1997). The language is based on a small set of operators that support the creation of integrated schema views by manipulating knowledge expressed in the MGKM formalism. The integration language supports selectivity by allowing the integration expert to choose the information of interest (e.g. the content of the integrated schema view), and customizability as the same local information can be integrated differently in alternative user-defined integrated schema views to support multiple semantics. Supporting multiple semantics through offering users selectivity and

customizability has several advantages as it reduces the complexity of the mediator generation process because only the relevant parts of the participating ISs schemas are integrated. The size of a generated mediator view is smaller compared with the case when a single global integrated schema is defined. This saves the user the burden of considering large global schema to locate information from multiple distributed and heterogeneous servers. Providing a schema integration language that aids in defining views with multiple semantics saves users the effort of manually integrating the selected information to create his/her views as is the case with the multidatabase language approach. Selectivity allows for information to be mediated from a large number of participating grid ISs. While customizability, on the other hand, means that the integrator can accurately indicate how to view selected information, and provision multiple semantic perspectives of the participating ISs at the mediator level.

Four major groups of integration operators are defined: Importation, Generalisation, Specialisation and Merger operators. In what follows Att is a function that returns the attributes of its argument, Ext is function that returns the extension of its argument, D are a set of derived (user defined) properties, that do not exist in the local classes but can be derived based on existing attributes, Gc are global classes, Lc are their corresponding local classes.[2] A detailed description of the generated interschema correspondence establishment rules following the execution of the integration operators is given in Tawil et al. (2008).

3.1 Importation operator

This group consists of operators: *ImportOne*, *ImportSubset*, and *ImportAll*. It is responsible for importing local classes into a global view without actually integrating them. Table 1 shows the services provided by each of the importation operators.

When a local class is imported, the set of attributes (Att) and all the descriptions associated with that class are also imported. For example, in terms of the relational model, it is possible to import a single table (*ImportOne*), a subset of a table (*ImportSubset*) or a table and all other tables referenced by it (*ImportAll*). Also, the extension (Ext) of the globally imported classes equals the extension of their local counterparts. The semantics of the importation operators are described explicitly in an RDF/OWL-based *operators* (*opt*) ontology. Below is the representation of the *ImportOne* operator:

```
<owl:ObjectProperty rdf:ID="ImportOne">
  <rdfs:subPropertyOf rdf:resource="#ImportationOperator" />
  <rdfs:domain>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
         <modl:LocalClass rdf:about="#Lc" />
         <modl:UserDefinedAttribute rdf:about="#D" />
       </owl:unionOf>
     </owl:Class>
   </rdfs:domain>
   <rdfs:range rdf:resource="#GlobalClass" />
   <owl:Restriction>
```

[2]The structural representation of our local and global schema views are specified internally in terms of the MetaMed ODL (modl) ontology.

```
      <owl:allValuesFrom rdf:resource="#Lc"/>
      <owl:allValuesFrom rdf:resource="#D"/>
    </owl:Restriction>
    <rdfs:label xml:lang="en">ImportOne</rdfs:label>
    <rdfs:comment xml:lang="en">Imports a single local class to a
    global view.</rdfs:comment>
</owl:ObjectProperty>
```

Table 1 shows how this operator, *ImportOne*, could be used to import the enriched Pnem-Patient class to the global user-defined level, (::) denotes the subclass relationship (e.g. *Pnemonia-Patient*) and the (:) denotes the semantic types of the attributes (e.g. *Patient-Identity-Number*, *Person-Full-Name*, *Date-Of-Birth*, ..). Both the subclass relationship and the semantic types of the attributes are extracted from the MGKM semantic model.

Each imported class is represented internally using the functions Import-Class and Import-Attribute relations of the *Importation* operator. The Import-Class function takes a local class *Pnem-Patient* as input and generates a global one *g-Pnem-Patient* as its output, e.g.

```
<owl:class rdf:ID="g-Pnem-Patient">
  <rdf:type rdf:resource="&opt;GlobalClass" />
  <opt:Import-Class rdf:resource="#Pnem-Patient" />
  <rdfs:subClassOf rdf:resource="&mgkm;Pneumonia-Patient" />
  <opt:Imported-Attribute rdf:resource="#g-Pnem-Patientg.g-Id"/>
  <modl:isIdentifiedBy>
     <modl:PrimaryKey>
       <modl:Attribute rdf:resource="#g-Pnem-Patientg.g-Id"/>
     </modl:PrimaryKey>
  </modl:isIdentifiedBy>
  ....
</owl:class>
```

The Import-Attribute function takes a local attribute *Pnem-Patient.Id* as input and generates a global imported *g-Pnem-Patient.g-Id* attribute as output, e.g.

```
<owl:DatatypeProperty rdf:ID="g-Pnem-Patient.g-Id">
  <rdf:type rdf:resource="&opt;GlobalAttribute" />
  <rdfs:subPropertyOf rdf:resource="&mgkm;Person-Identity
    -Number"/>
  <opt:Import-Attribute rdf:resource="#Pnem-Patientg.Id" />
</owl:DatatypeProperty>
```

3.2 Generalisation operator

This group consist of operators: *UnionOne*, *Union*, and *UnionAll*. These integrate local classes by linking them to a common superclass. The local classes have to be semantically *overlapping*. Table 2 shows the services of each generalisation operator.

When a set of intentionally related[3] local classes are *Unioned* three virtual classes are created to the global view, namely: the generated superclass and its two subclasses. It is possible to add only the generalised superclass (*UnionOne*) , the generated superclass and its

[3]Two local classes are considered to be *intentionally equivalent* when they represent the same concept

Table 1 Importation operator semantics

Importation Operator	Semantics	Extension	Properties	Validation Condition
Gc=(ImportOne Lc)	Import a single Local Class to a Global View.	Ext(Gc)=Ext(Lc)	Att(Gc)=Att(Lc) U D	Any local class
Gc=(ImportSubset Lc)	Import the subset of the Lc that satisfies the selection predicate.	Ext(Gc)=Subset of Ext(Lc)	Att(Gc)=Subset of Att(Lc) U D	Any local class
Gc=(ImportAll Lc)	Import a class and all other classes referenced by it.	Ext(Gc)=Ext(Lc) for all Gc derived from Lc	Att(Gc)=Att(Lc) U D for all Gc derived from Lc	Any local class

Example

Enriched Local Class	Global Class
Pnem−Patient::Pneumonia−Patient Id:Person−Identity−Number Name:Person−Name Age:Person−Age Address:Person−Address	g−Pnem−Patient::Pneumonia−Patient g−Id:Person−Identity−Number g−Name:Person−Name g−Age:Person−Age g−Address:Person−Address g−Pnem−Patient = (ImportOne Pnem−Patient) Ext(g−Pnem−Patient) = Ext(Pnem−Patient) *The global view (g−) is generated by using (ImportOne) operator.*

two subclasses (*Union*) or the generated superclass its two subclasses and all other classes referenced by them (*UnionAll*). The set of attributes and the associated descriptions that belong to the union of both local classes are upward-inherited by the generated superclass. Below is the RDF/OWL representation of the *UnionOne* operator:

```
<owl:ObjectProperty rdf:ID="UnionOne">
  <rdfs:subPropertyOf rdf:resource="#UnionOperator" />
  <rdfs:domain>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
         <modl:LocalClass rdf:about="#Lc1" />
         <modl:LocalClass rdf:about="#Lc2" />
         <modl:UserDefinedAttribute rdf:about="#D" />
       </owl:unionOf>
     </owl:Class>
   </rdfs:domain>
   <rdfs:range rdf:resource="&opt;GlobalClass" />
   <owl:Restriction>
     <owl:allValuesFrom rdf:resource="#Lc1"/>
     <owl:allValuesFrom rdf:resource="#Lc2"/>
     <owl:allValuesFrom rdf:resource="#D"/>
   </owl:Restriction>
   <rdfs:label xml:lang="en">ImportOne</rdfs:label>
```

```
    <rdfs:comment xml:lang="en">A Table has a set of Columns.
    </rdfs:comment>
</owl:ObjectProperty>
```

In the example of Table 2 the attributes g-Name, g-Age, g-Address are upward inherited, and the description associated with the class g-Patient is that of patients suffering from either *Cancer* or *Pnemonia*.

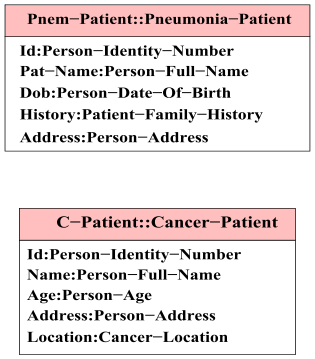
```
<owl:class rdf:ID="g-Patient">
  <rdf:type rdf:resource="&opt;GlobalClass" />
  <rdfs:subClassOf rdf:resource="&mgkm;Patient" />
  <owl:oneOf rdf:parseType="Collection">
    <mgkm:pathology rdf:about="&mgkm;Cancer" />
    <mgkm:pathology rdf:about="&mgkm;Pnemonia" />
  </owl:oneOf>
</owl:class>
```

Table 2 Generalisation operator semantics

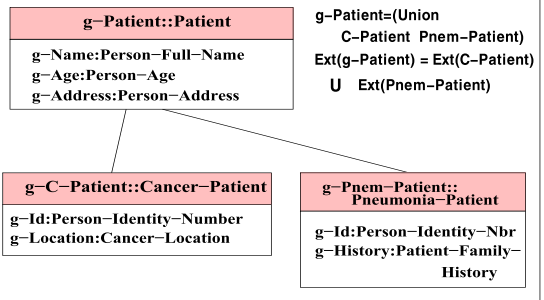
Generalisation Operator	Semantics	Extension	Properties	Validation Condition
Gc=(UnionOne Lc1 Lc2)	Generate a superclass for two local classes. Only the superclass is included in the view.	Ext(Gc)=Ext(Lc1) U Ext(Lc2)	Att(Gc)=Att(Lc1) U Att(Lc2) U D	Overlapping or Equivalent classes
Gc=(Union Lc1 Lc2)	Generate a superclass for two local classes. All three classes are added to the view.	Ext(Gc)=Ext(Lc1) U Ext(Lc2) Ext(Gc1)=Ext(Lc1) Ext(Gc2)=Ext(Lc2)	Att(Gc)=Att(Lc1) U Att(Lc2) U D Att(Gc1)=Att(Lc1) U D Att(Gc2)=Att(Lc2) U D	Overlapping or Equivalent classes
Gc=(UnionAll Lc1 Lc2)	Generate a superclass for two local classes. All three classes are added to the view, and all other classes referenced by them.	Ext(Gc)=Ext(Lc1) U Ext(Lc2) Ext(Gc1)=Ext(Lc1) Ext(Gc2)=Ext(Lc2) For all Gc derived from Lc	Att(Gc)=Att(Lc1) U Att(Lc2) U D Att(Gc1)=Att(Lc1) U D Att(Gc2)=Att(Lc2) U D For all Gc derived from Lc	Overlapping or Equivalent classes

Example

Enriched Local Classes



Global Classes



The rest of the inherited global classes and attributes are imported to their respective classes using the importation operator. Also, the extension of the generated superclass is the union of the extension of both local classes. While the extension of the other global classes (e.g. g-C-Patient and g-Pnem-Patient) is that of their local counterparts.

In the above example, the two property sets, namely: *Id, Pat-Name, Dob, History, Address* and *Id, Name, age, Address, Location* which belong to the classes Pnem-Patient and C-Patient respectively, may have semantic conflict. For instance, while g-Patient.g-Age can be safely retrieved from the C-Patient.Age attribute, a function is required to calculate the age of the patient from his *date of birth* when it is retrieved from Pnem-Patient.Dob. Also, equivalent properties may disagree on their syntax as well. For instance, the *Name* property maybe spelt Name in one class and Pat-Name in another. These conflicts are identified and delt with during the conflict detection and resolution phase (see Tawil et al. 2008) of the global schema generation process. Establishing interschema correspondence relationships and defining proper transformation to reconcile any heterogeneity is only possible when the semantic description of each property is explicitly defined to our system and the correspondence establishment rules to resolve conflicts are specified.

Another important issue that is demonstrated by the current example is establishing the set of correspondences between instances from the extensions of classes C-Patient and Pnem-Patient to determine the correct instances for the generated global class g-Patient. This achieved by using *global integrity constraints* that are defined on both classes, since primary keys (OIDs) are usually insufficient to determine this relationship. While primary keys are sufficient to uniquely identify objects at the local schema they are insufficient to determine relationships between retrieved instances from different distributed grid ISs at the mediator level. In this example, a patient object of one database represents the same real object of another if they have the same *name*, *age* and *address*:

$$
\begin{aligned}
(\Rightarrow\ \ & (=_{Ext}\ \mathrm{C-PatientPnem-Patient}) \\
& (\mathrm{and}(=_{Value}\ \mathrm{C-Patient.namePnem-Patient.name}) \\
& (=_{Value}\ \mathrm{C-Patient.age}(\mathrm{Date-Of-Birth-To-AgePnem-Patient.Dob?age})) \\
& (=_{Value}\ \mathrm{C-Patient.addressPnem-Patient.address})))
\end{aligned}
$$

In the above asserted OKBC knowledge => stands for implication, $=_{Ext}$ stands for extension equivalence, $=_{Value}$ stands for attribute values equivalence, and Date-Of-Birth-To-Age is a function that calculates the age of a person from its date of birth. The generalised (unioned) superclass is represented internally using the Union-Class and Union-Attribute functions of the *generalisation* operator. The Union-Class function takes two local classes as input *Pnem-Patient, C-Patient* and generates a global one *g-Patient* as its output, e.g.

```
<owl:class rdf:ID="g-Patient">
  <rdf:type rdf:resource="&opt;GlobalClass" />
  <opt:Union-Class>
     <modl:LocalClass rdf:resource="#Pnem-Patient"/>
     <modl:LocalClass rdf:resource="#C-Patient"/>
  </opt:Union-Class>
  <rdfs:subClassOf rdf:resource="&mgkm;Patient"/>
</owl:class>
```

While the Union-Attribute function takes two local attributes *Pnem-Patient.Pat-Name, C-Patient.Name* as input and generates one global imported attribute *g-Patient.g-Name* as output, e.g.

```
<owl:DatatypeProperty rdf:ID="g-Patient.g-Name">
  <rdf:type rdf:resource="&modl;Attribute"/>
  <rdfs:domain>
     <opt:Union-Attribute>
       <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
           <modl:LocalAttribute rdf:about="#Pnem-Patient.
            Pat-Name" />
           <modl:LocalAttribute rdf:about="#C-Patient.Name" />
         </owl:unionOf>
       </owl:Class>
     </opt:Union-Attribute>
  </rdfs:domain>
  <rdfs:subClassOf rdf:resource="&mgkm;Person-Identity-Number"/>
  <rdfs:range rdf:resource="#g-Name"/>
</owl:DatatypeProperty>
```

The associated global subclasses of the unioned superclass are mapped internally using the importation operator relations as described above. In our semantic representations we are trying to model real world concepts. In the real-world a concept can have multiple parents and can inherit properties along multiple paths. While, polyhierarchies are difficult to maintain, however when providing the correct set of axioms, an automated reasoner such as (pellet) can be used to maintain such relationships. The effort is considerable requiring richer axioms but worthwhile as the automated reasoner is able to maintain the whole structure, avoiding errors. The more expressive axiomatization also enables richer queries.

3.3 Specialisation operator

This group consists of two operators: *Intersect*, and *IntersectAll*. These integrate a set of overlapping classes by generating a common subclass. Table 3 shows the services contributed by each operator of this group.

The *Specialisation* operator group integrates a set of intentionally related local classes by specialising them into a common subclass. This adds three virtual classes to the grid global view, these are the generated subclass and its two superclasses. The set of attributes belonging to the two superclasses are imported from their respective local classes using the *Include* operator. The attributes that belong to the generated subclass are downward-inherited by the generated subclass. The extension of the generated global subclass is the intersection of the extension of both local classes. While the extension of the other global classes is that of their local counterparts.

Table 3 shows how the classes Doctor and C-Patient are integrated, using specialisation operators to generate the class Doctors-With-Cancer of hospital doctors who are suffering from cancer. To calculate the treatment expenses of doctors who are patients, we need to override the g-Expenses property of the g-C-Patient class with a user-defined property *special expenses* Spec-Expenses that has a computed expression equals to

Table 3 Specialisation operator semantics

Specialisation Operator	Semantics	Extension	Properties	Validation Condition
Gc=(Intersect Lc1 Lc2)	Generate a subclass for two local classes. Only the subclass is added to the view.	Ext(Gc)=Ext(Lc1) ∩ Ext(Lc2)	Att(Gc)=Att(Lc1) U Att(Lc2) U D	Overlapping classes
Gc=(IntersectAll Lc1 Lc2).	Generate a common subclass for two local classes. All three classes are added to the view.	Ext(Gc)=Ext(Lc1) ∩ Ext(Lc2) Ext(Gc1)=Ext(Lc1) Ext(Gc2)=Ext(Lc2)	Att(Gc)=Att(Lc1) U Att(Lc2) U D Att(Gc1)=Att(Lc1) U D Att(Gc2)=Att(Lc2) U D	Overlapping classes

Example

Enriched Local Classes

Doctor::Doctor
Id:Person–Identity–Number
Name:Person–Full–Name
Dob:Person–Date–Of–Birth
Salary:Salary

C–Patient::Cancer–Patient
Id:Person–Identity–Number
Name:Person–Full–Name
Dob:Person–Date–Of–Birth
Location:Cancer–Location
g–Expenses:Treatment–Expenses

Global Classes

g–Doctor::Doctor
g–Id:Person–Identity–Number
g–Name:Person–Full–Name
g–Dob:Person–Date–Of–Birth
g–Salary:Salary

g–C–Patient::Cancer–Patient
g–Id:Person–Identity–Number
g–Name:Person–Full–Name
g–Dob:Person–Date–Of–Birth
g–Location:Cancer–Location
g–Expenses:Treatment–Expenses

g–Doctors–With–Cancer::Doctor, Cancer–Patient
g–Id:Person–Identity–Number
g–Name:Person–Full–Name
g–Salary:Salary
g–Location:Cancer–Location
g–Spec–Expenses:Treatment–Expenses

Ext(g–C–Patient) = Ext(C–Patient)

Ext(g–Doctor) = Ext(Doctor)
Ext(g–Doctors–With–Cancer) =Ext(Doctor) ∩ Ext(C–Patient)

$\left(g - C - Patient.g - Expenses\text{-}\frac{g-C-Patient.g-Expenses*40}{100}\right)$ stating that these patients gets a 40 % discount. The OKBC representation of this rule is:

$$\begin{aligned}&(\text{and} = (\text{User} - \text{Defined} - \text{Attributeg} - \text{Spec} - \text{Expenses})\\&\quad(\text{Attribute} - \text{Ofg} - \text{Spec} - \text{Expensesg} - \text{Doctor} - \text{With} - \text{Cancer})\\&\quad(=_{\text{value}}\ \text{g} - \text{Spec} - \text{Expenses}(-\text{g} - \text{Expenses}(*0.4\text{g} - \text{Expenses})))\end{aligned}$$

The specialised subclass is represented internally using the Intersect-Class, and Intersect-Attribute functions of the *specialisation* operator. The Intersect-Class function takes two classes as input and generate an intersected subclass as output. The associated global superclasses of the specialised subclass are mapped internally using the importation operator relations.

3.4 Merger operator

This group consists of two operators: *Merge*, and *MergeAll*. The service offered by these operators is to integrate *equivalent* local integration units by merging them into one global class. Table 4 shows the effects of each of this group of operators.

The *Merger* operator group supports the integration of two intentionally equivalent local classes, by taking those classes as input and generating a *combined* integrated global class as output. The local integration units must be intentionally equivalent to be merged and the extension of the combined global class is the union of the extension of the locally combined classes. Table 4 shows how the classes *C-Patient* and Pnem-Patient are merged together to generate a single global class, namely the class g-Hospital-Patient. The concepts description

Table 4 Merger operator semantics

Merger Operator	Semantics	Extension	Properties	Validation Condition
Gc=(Merge Lc1 Lc2)	Merges two integration units into one global class.	Ext(Gc)=Ext(Lc1) U Ext(Lc2)	Att(Gc)=Att(Lc1) U Att(Lc2) U D	Equivalent integration units
Gc=(MergeAll Lc1 Lc2).	Merges two classes and all classes referenced by them.	Ext(Gc)=Ext(Lc1) U Ext(Lc2) The same holds for the rest of the referenced classes.	Att(Gc)=Att(Lc1) U Att(Lc2) U D The same holds for the rest of the referenced classes.	Equivalent integration units

Example

Enriched Local Classes

C–Patient::Cancer–Patient
- Id:Person–Identity–number
- Name:Person–Full–Name
- Dob:Person–Date–Of–Birth
- Address:Person–Address

Pnem–Patient::Pneumonia–Patient
- Id:Person–Identity–Number
- Name:Person–Full–Name
- History:Patient–Familly–History

Global Classes

g–Hospital–Patient::Cancer–Patient, Pneumonia–Patient
- g–Id:Person–Identity–Number
- g–Name:Person–Full–Name
- g–Dob:Person–Date–Of–Birth
- g–Address:Person–Address
- g–History:Patient–Family–History

Ext(g–Hospital–Patient) =Ext(C–Patient) U Ext(Pnem–Patient)

This view is generated by (Merge C–Patient Pnem–Patient)

of the global class is the union of the descriptions associated with its local *merged from* classes. Accordingly, the description of the g-Hospital-Patient class is that of patients who are either suffering from *Pnemonia* or *Cancer*, or from both.

Finally, the combined global class is represented internally using the Merge-Class and Merge-Attribute functions of the *merger* operator. The Merge-Class function takes two local classes *Pnem-Patient, C-Patient* as input and generate one global class *g-Hospital-Patient* as its output.

```
<owl:class rdf:ID="g-Hospital-Patient">
  <rdf:type rdf:resource="&opt;GlobalClass" />
  <opt:Merge-Class>
     <modl:LocalClass rdf:resource="#Pnem-Patient"/>
     <modl:LocalClass rdf:resource="#C-Patient"/>
  </opt:Merge-Class>
  <rdfs:subClassOf rdf:resource="&mgkm;Pneummonia-Patient"/>
  <rdfs:subClassOf rdf:resource="&mgkm;Cancer-Patient"/>
</owl:class>
```

While the Merge-Attribute function takes two local attributes *Pnem-Patient.Name, C-Patient.Name* as input and generates a global one *g-Hospital-Patient.g-Name* as its output.

```
<owl:DatatypeProperty rdf:ID="g-hospitaPatient.g-Name">
  <rdf:type rdf:resource="&modl;Attribute"/>
  <rdfs:domain>
     <opt:Merge-Attribute>
       <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
```

```
            <modl:LocalAttribute rdf:about="#Pnem-Patient
            .Name" />
            <modl:LocalAttribute rdf:about="#C-Patient.Name" />
          </owl:unionOf>
        </owl:Class>
      </opt:Merge-Attribute>
  </rdfs:domain>
  <rdfs:subClassOf rdf:resource="&galen;Person-Full-Name"/>
  <rdfs:range rdf:resource="#g-Name"/>
</owl:DatatypeProperty>
```

It is worth mentioning, here, that it is possible for the integration expert to override any of the global attributes for a virtual class by overriding (customizing) global attributes or processing the results of the retrieved values. This can be implemented in a variety of ways: these range from simple arithmetic manipulations (e.g. defining a function to convert degrees Centigrade to Fahrenheit) to complex operations which can only be realized using external programs. The reason behind allowing for user-specified options is to give users more control over the global view definition process and to support customizability.

3.5 Cost of integration operators

Open and dynamic grid environments pose additional challenges to the provision of cost-related metadata, since cost models require cost-related information to be consistently obtainable, i.e. over all data sources. In a grid environment quantitative cost estimates are difficult to derive, when the data to be retrieved is computed by complex queries, that retrieves and integrates data from different heterogeneous and grid distributed ISs. The cost of both *Union* and *Intersection* operators in the grid depends on network cost of query execution time and the scale of data retrieved to be processed. Accordingly, we can find the overall execution time by adding to the query execution cost the flat overheads. To overcome the difficulties of cost-based query planning, we advocate a qualitative approach. We assume that the cost to reformulation, query translation, and the system overheads are stable, which entails that if the size of data to be processed is determined we are able to estimate (with the help of a cost model, or based on previous experience) the execution time of each integration operation.

We estimate the cost of a distributed plan purely in terms of its execution time. We also assume that the execution time is dominated by the tuple transfer costs, and integration operators used. Thus ignoring the local processing costs. Accordingly, the execution costs are computed in terms of the response times offered by the various sources that makes up the (distributed) plan. The response time of a server is proportional to the number of times that server is called, and the expected number of tuples transferred over each call.

3.5.1 Case study

The experiment described in this subsection calculates the cost of using the integration operators of the MetaMed system when evaluated against example queries on the mediator global schema discussed in Fig. 4. Queries are run in various different configurations, and with varied parameters to test the different operators including the Importation, Generalisation, Specialisation and Merger operators. For the purpose of this experiment only the query

response time was taken into consideration when calculating the cost of execution for each operator.

The case study has been conducted on a homogeneous collection of dedicated MySQL server machines with no direct user access at King's College and University of Malta, while the global mediator schema is run from the University of East London. All machines running Windows 7 operating system and had the same specifications (CPU 2.60 GHz, 4 GB RAM). All three universities are connected via the (https://www.ja.net/products-services/janet-connect/janet-connections!). Janet is high-speed network for the UK research and education community. A connection to Janet gives a reliable and high bandwidth network.

The data (1 Gigabyte in total) is distributed over two volumes located on both sites. In itself, then, these data can be accommodated in memory on any of the machines, so that the execution time of a mediator calls should be dominated by CPU processing. The queries submitted to the mediator global schema view are executed at the local server where the data is located and retrieves tuples, from which part of the data are joined tuples across both sites, some satisfies the aggregation criterion while other tuples required functional conversions. Attributes are either generalized or specialized and the rest of the data is imported. As the cost of calls to MetaMed is several orders higher than a normal query execution between the two local sources, the query cost is expected to be dominated by the processing of returned data when a standard MultiDatabase (MDB) approach is used as opposed to a intelligent query planning and formulation when data are requested against the MetaMed (MM) mediator schema.

In our case study, all machines have wrappers implemented with APIs that can support the MetaMed global mediator queries and are capable for translating from and to the mediator global schema. The CPU cost of a MetaMed call for a given stored dataset varies significantly depending on the operator's required for specific query requests. Since the CPU speeds and usage cost may vary, there is a range of options within a grid environment for increasing the parallelism in the operation calls. Two strategies for parallelizing operators calls maybe considered in our integration context based on prior knowledge of individual IS specification and network speed. One strategy could be to target high-speed processors, the parallelization of the operation by the mediator through requesting larger data sets from the fastest processor depending on the distance and network speed. A second strategy may opt to target, slowest processor-first which could be adopted if the user is charged for using Grid resources, and the cost of a resource depends on its power. However, neither of these strategies are considered in our experiment.

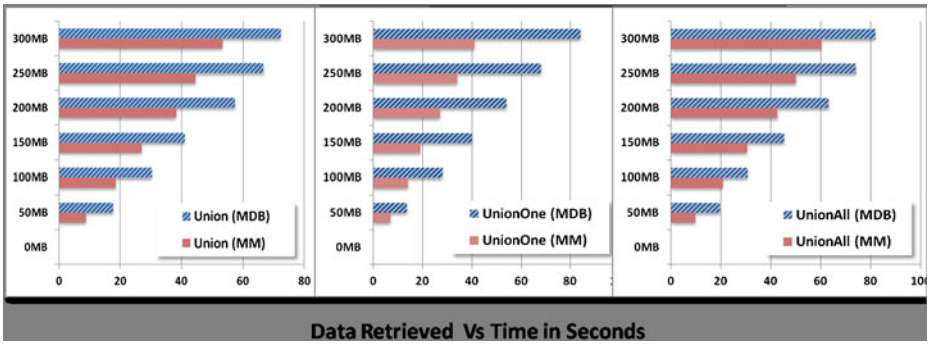


Fig. 3 Union operator cost

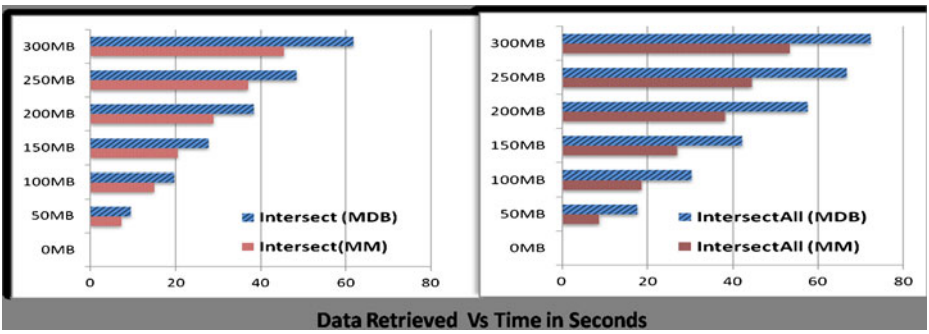


Fig. 4 Intersect operator cost

Much of the cost of the integration process occurring is spent while enriching the local schema and in the building of the integrated schema. The cost of using the importation operator including all of it's versions (ImportOne, ImportSubset and ImportAll) is equivalent to executing a direct query request to the participating ISs as the data is not integrated when using this operator.

The cost of the Generalisation operator depends on which version of the operator is used (UnionOne, Union or UnionAll). The cost of the operator depends on the data retrieved by the query when executed against the generalized classes of the MetaMed mediator global schema and is evaluated by executing several Multi-database (MDB) queries (Smith et al. 2002; Lu et al. 1992) requesting data from the generalized class. In our mediator schema example the Union operators were used and a superclass with the associated subclasses are generated representing the integration of equivalent attributes. Any remaining attributes are imported to their respective global mediator classes. The cost is determined based on query execution time when the same results are requested using queries against the mediator schema and then the same data is retrieved and integrated using MDB queries against the individual ISs schemas, see Fig. 3.

The cost of the specialisation operator is evaluated based on the size of the intersected data when integrated using either one of the operators (Intersect or IntersectAll). The query execution time is significantly reduced when the data is actually imported from either one of the local databases rather than filtering the data at the global schema. The same data is retrieved and integrated using queries against the mediator global schema and also on MDB queries against the individual ISs schemas, see Fig. 4.

Both Merger operators (Merge and MergeAll) significantly reduce the execution time. Prior to query execution MetaMed identifies the server from which the data are to be retrieves and complete the Merging of the data at the mediator global schema. The cost is evaluated based on the size of the data retrieved that is part of the merged classes from the global schema, see Fig. 5.

3.6 Verification of integration decisions

Each integration operator has predefined semantics in terms of its inputs and outputs, which explicitly states its capabilities and grid integration services offered. A MetaMed integration operator can only be applied if the semantics of the domain knowledge provided as input for the operator are consistent with its predefined semantics and it is expected to produce legal results (output). The decision of which operator to apply to a given set of integration units

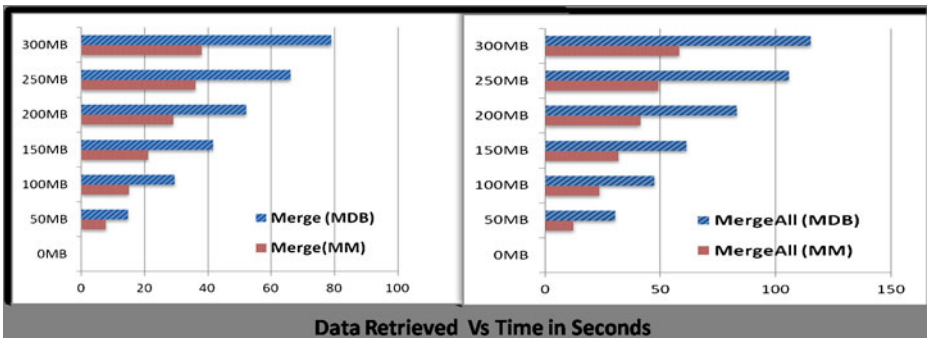


Fig. 5 Merge operator cost

depends on the interschema semantic relationships among these units (e.g. equivalence, overlap, .. etc), and user preferences, see (Li et al. 2010; Tawil et al. 2008; Li and Ling 2004). Semantic preservation rules for validating user decisions have been defined. Based on the description of the individual ISs, the interschema semantic relationships and the description of the conditions under which the integration operators can be applied, the verification of integration decisions can be validated using semantic preservation rules. The descriptions of the data elements involved in the execution of an integration operator can either match the integration rules associated with this operator or not. These are enforced implicitly by controlling user choices, i.e. a user cannot make an erroneous selection because s/he is presented with the valid set of properties to choose from. Whenever a choice is made by a user at the grid level, all the properties that may create a conflict if they were to be chosen during subsequent interactions are masked automatically by the system.

The *domain* (the concept(s) with which an operator is associated) and the *range* (the type of values it can have) of the internal relations associated with each operator are explicitly defined (constrained) in the system. For example, the Import-class and Import-Attribute relations of the importation operator can be described as follows:

$$
\begin{aligned}
(\Rightarrow \quad & (\textbf{Import} - \textbf{Class}\ ?Lc ?Gc) \\
& (\text{and} = (\text{Interface}\ ?Lc) \\
& \qquad (\text{Interface}\ ?Gc) \\
& \qquad (=_{Ext}\ ?Gc\ ?Lc) \\
& \qquad (\text{or} = (= \text{Att}(?Gc)\ \text{Att}(?Lc)) \\
& \qquad\qquad (\text{Subset} - \text{Of}\ \text{Att}(?Gc)\ \text{Att}(?Lc))).
\end{aligned}
$$

The local class ?Lc can be imported to a global class ?Gc only if both ?Lc and ?Gc are of type interface, and the associated attributes of ?Gc are equal to or are a subset of those of ?Lc, and the extension of the global imported classes ?Gc are equal to those of their local counterparts ?Lc.

$$
\begin{aligned}
(\Rightarrow \quad & (\textbf{Import} - \textbf{Attribute}\ ?Att\_Lc ?Att\_Gc) \\
& (\text{and} = (\text{Attribute}\ ?Att\_Lc) \\
& \qquad (\text{Attribute}\ ?Att\_Gc) \\
& \qquad (=_{Ext}\ ?Att\_Gc\ ?Att\_Lc))).
\end{aligned}
$$

A local attribute ?Att_Lc can only be imported to a global one ?Att_Gc if both ?Att_Lc and ?Att_Gc are of type attribute, and the extension of the global attributes equals those of their local counterparts. For the merger operator this translates to the following rules:

$$
\begin{aligned}
(\Rightarrow\ & (\textbf{Merge} - \textbf{Class}\ ?Lc1\ ?Lc2\ ?Gc) \\
& (\text{and} = (\text{Interface}\ ?Lcl) \\
& \qquad (\text{Interface}\ ?Lc2) \\
& \qquad (\text{Interface}\ ?Gc) \\
& \qquad (\text{Intentionally} - \text{Equivalent} - \text{With}\ ?Lc1\ ?Lc2) \\
& \qquad (= \text{Att}(?Gc)\ (\text{Union} = \text{Att}(?Lc1) \\
& \qquad\qquad \text{Att}(?Lc2))))).
\end{aligned}
$$

The local classes ?Lc1 and ?Lc2 can be merged to a global class ?Gc only if ?Lc1, ?Lc2, ?Gc are of type Interface, both ?Lc1 and ?Lc2 are intentionally equivalent, and the associated attributes of ?Gc are the union of the intentionally equivalent attributes of ?Lc1 and ?Lc2.

$$
\begin{aligned}
(\Rightarrow\ & (\textbf{Merge} - \textbf{Attribute}\ (\text{and}\ (?Att\_Lc1)\ (?Att\_Lc2))\ ?Att\_Gc) \\
& (\text{and} = (\text{Attribute}\ ?Att\_Lc1) \\
& \qquad (\text{Attribute}\ ?Att\_Lc2) \\
& \qquad (\text{Attribute}\ ?Att\_Gc) \\
& \qquad (\text{Intentionally} - \text{Equivalent} - \text{With}\ ?Att\_Lc1\ ?Att\_Lc2) \\
& \qquad (=_{Ext}\ ?Att\_Gc\ (\text{Union}_{Ext} = ?Att\_Lc1 \\
& \qquad\qquad ?Att\_Lc2)))).
\end{aligned}
$$

The local attributes ?Att_Lc1 and ?Att_Lc2 can be combined to a global attribute ?Att_Gc only if ?Att_Lc1, ?Att_Lc2, ?Att_Gc are of type attribute, and both ?Att_Lc1 and ?Att_Lc2 are intentionally equivalent. Also, the extension of global attributes ?Att_Gc equals the union of the extension of their local counterparts ?Att_Lc1 and ?Att_Lc2.

The use of MGKM in the integration process in grid-settings allows us to use the descriptions associated with local classes to validate the user decisions (assertions) about concepts to be integrated. In general, the descriptions of the data elements involved in the execution of an integration operator can either match the integration rules associated with this operator or not. Accordingly, three different situations are possible: an *error* message is signalled, a *warning* is given, or the integration is *granted*.

Depending on both the situation and the kind of correspondence (see Table 5), the system notifies the integration expert with one of the following messages:

Table 5 Cases in which the system can detect anomalies

	$A \equiv B$	$A \subseteq B$	$A \cap B \neq \emptyset$	$A \cap B = \emptyset$
Import	Ignored	Ignored	Ignored	Ignored
Generalise	Error	Warning	Granted	Error
Specialise	Error	Grant	Warning	Error
Merge	Granted	Warning	Error	Error

- Error Message, the interschema correspondence between the concepts description and the predefined semantics of the integration operator are disjoint. For example, if the database schemas Pnem-Patient (*Stores information about patients suffering from a pnemonia infection*), and Lung-Tumor-Patient (*Stores information about patients suffering from lung tumor*) are said to be *merged*, then an error is detected because their interschema corresponding definition indicates that both schema concepts are intentionally related (*both schemas are storing information about patients, but these patients are suffering from different types of diseases*). The *merge* operator group can only be applied if both database concepts to be integrated are *at least* intentionally equivalent.
- Warning Message, the interschema correspondence between the concepts descriptions and the predefined semantics of the integration operator can be matched, but the system believes that a more accurate integration operator can be applied. For example, if the schema concepts Cancer-Patient[4] and Lung-Tumor-Patient[5] are said to be merged by the person responsible for integration using the merger operator, then a warning is detected because the description of Cancer-Patient schema subsumes that of the Lung-Tumor-Patient and therefore it seems that applying the generalisation operator would be a better choice.
- Granted, the definition of the correspondence and the chosen integration operator match perfectly, and the system cannot find a better operator to be used.
- Ignored, the definition of the correspondence does not make any new contribution. For example, if the *Importation operator* is chosen then the definition of interschema correspondences are ignored.

The integration process is semi-automatic and the integrator input is required to solve conflicts. In case of an *Error* or a *Warning* message the enriched schema models are exploited to assist the user (integration expert) during the integration process by automatically identifying integratable schema concepts and validating user assertions about the concepts to be integrated. The integration operators provide the integration expert with the necessary tools to assist in designing the global schema and to resolve conflicts when defining the mappings from the local ISs schemas to the global integrated schema.

4 Generation of the mediator global schema

In this section, we present an integration scenario to show how our integration language is used to define a mediator global schema, that is a mediator view of data stored in local ISs. In the scenario we examine the case when the integrator wishes to *integrate* two databases: Lung-Infection-DB and Cancer-DB, see Fig. 6.

The process of generating global classes is interactive with the integrator (Papakonstantinou et al. 1996; Sheth and Larson Levy et al. 1990; 1996). We assume that both schemas are enriched with semantic knowledge extracted from the MGKM. Lets suppose as a first step the integrator wishes to generalise the Pnem-Patient and the Patient classes of the Lung–Infection-DB and Cancer-DB respectively, and to create a new Hospital-Patient class as a superclass of the two classes. All three classes are added to the view. The attributes belonging to the Hospital-Patient class corresponds to the *union* of the attributes of the classes belonging to Pnem-Patient and Patient, where semantically equivalent attributes

[4]Stores information about patients suffering from Cancer.

[5]Stores information about patients suffering from lung tumor.

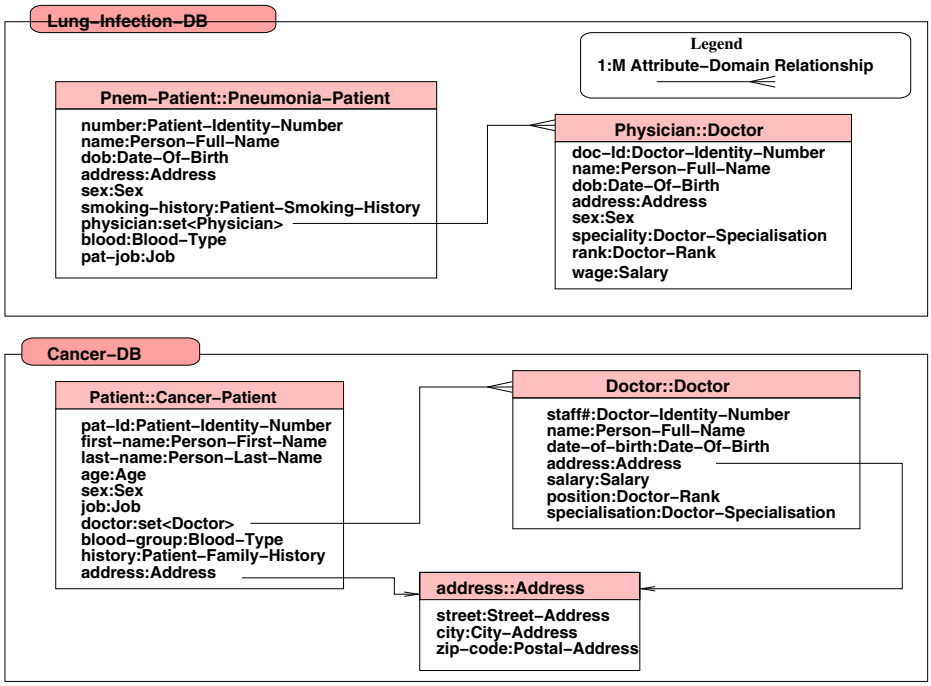


Fig. 6 Enriched ISs schemas of Lung-Infection-DB and Cancer-DB

are automatically unified into unique global attributes in the Hospital-Patient class. Before the integration process is initiated and based on the associated semantic domains with each of the classes to be generalised the system infers (suggests) the following RDF/OWL-based relationships:[6]

```
<owl:Class rdf:ID="#Pnem-Patient.number">
  <mgkm:Is-Equivalent-With rdf:resource="#Patient.pat-Id" />
</owl:Class>

<owl:Class rdf:ID="#Pnem-Patient.name">
  <mgkm:Is-Aggregation-Of>
     <owl:unionOf rdf:parseType="Collection">
       <modl:LocalAttribute rdf:about="#Patient.first-name" />
       <modl:LocalAttribute rdf:about="#patient.last-name />
     </owl:unionOf>
  <mgkm:Is-Aggregation-Of>
</owl:Class>

<owl:Class rdf:ID="#Pnem-Patient.dob">
<mgkm:Has-Data-Representation-Conflict rdf:resource="#Patient
```

[6]The integrator is asked to accept/modify or reject the generated types of correspondences for the interschema relationships among the local integration units

```
.age"/>
  <owl:FunctionalProperty rdf:resourse="Date-Of-Birth-To-Age" />
</owl:Class>

<owl:Class rdf:ID="#Pnem-Patient.physician">
  <mgkm:Is-Equivalent-With rdf:resource="#Patient.doctor" />
</owl:Class>

<owl:Class rdf:ID="#Pnem-Patient.smoking-history">
  <mgkm:Is-Kind-Of rdf:resource="#Patient.history" />
</owl:Class>
.....
```

In the above interschema correspondence assertions, an *Is-Equivalent-With* assertion is assigned when both schema elements are representing the same real-world concepts, *Is-Kind-Of* assertion is assigned when two schema elements exists at different levels of abstraction in two different databases, *Is-Aggregation-of* is assigned when the domain of one schema element maps to a group, or collection of domains in another. The output of this unification process for the Hospital-Patient class of Fig. 6, is the following set of global attributes (g-name,g-dob,g-address,g-sex,g-blood,g-job,g-doctor), the associated internal assertions include:

```
<opt:UnionClass rdf:ID="#g-hospital">
  <owl:unionOf rdf:parseType="Collection">
    <modl:LocalClass rdf:about="#Pnem-Patient" />
    <modl:LocalClass rdf:about="#patient" />
  </owl:unionOf>
</opt:UnionClass>

<opt:UnionAttribute rdf:ID="#g-hospital.name">
  <owl:unionOf rdf:parseType="Collection">
    <modl:LocalAttribute rdf:about="#Pnem-Patient.name" />
    <modl:LocalAttribute>
       <owl:unionOf rdf:parseType="Collection">
         <modl:LocalAttribute rdf:about="#Patient.first-name" />
         <modl:LocalAttribute rdf:about="#Patient.last-name" />
    </modl:LocalAttribute>
  </owl:unionOf>
</opt:UnionAttribute>

....
```

Of course additional information has to be provided for completing the grid level global schema definition. In particular, MetaMed asks the designer to interactively supply information regarding:

1. The *global class* name: the Global Schema Generator (GSG) of our system proposes a set of candidate names for the global class, by exploiting terms in the MGKM ontology that subsumes both classes to be integrated. Based on these names, the integrator can decide the most suitable name for the generated global class or can even use a

completely different term. In our example, the designer decides to assign the name g-Hospital-Patient to the global class generated from integrating g-Pnem-Patient and g-Patient.

2. Mappings between the global attributes of the global class, and the corresponding attributes of the local one. In particular, if a global attribute is obtained from integrating more than one local attribute, mapping information will be generated automatically depending on the integration operator used, the resulting generated classes and their internal representation:

   - and mappings: this specifies that the value of a global attribute corresponds to the union of the attributes of a given class. For example the global attribute name of g-Hospital-Patient correspond to both first-name and last-name attributes of Pnem-Patient. By specifying this mapping relationship for g-name, the integrator states that the values of first-name, and last-name have to be considered when g-Hospital-Patient.name is retrieved from the *Pnem-Patient* database.
   - or mappings: this specifies that the value of a global attribute corresponds to the value of at most one local attribute of a given class. g-Hospital-Patient.name corresponds to the name attribute of the patient class.

3. default values to be assigned to global attributes in correspondence of a given class. A default value is always verified for the global attribute when evaluated on the class. For example, the integrator knows that the global attribute g-rank of g-Hospital-employees

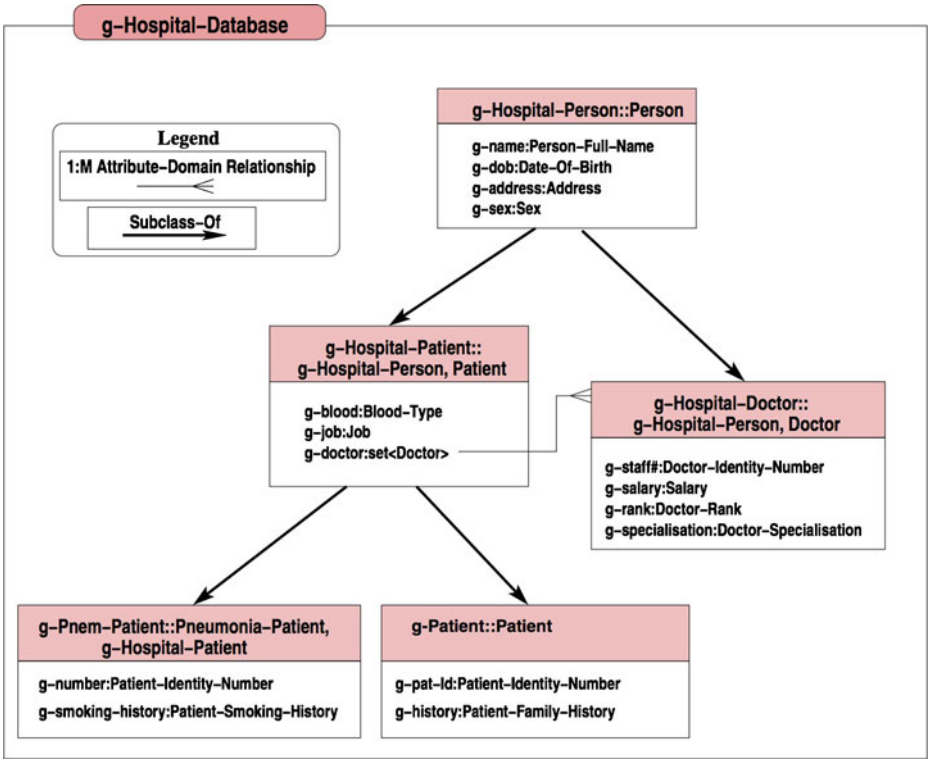


Fig. 7 Mediator global schema of lung-infection-database and cancer-database

has values ‘Doctor’ or ‘Consultant’ when evaluated on the g-Doctor class, even if the global g-rank attribute is not declared in any of the local ISs.

4. new attribute for the global class. The integration operator suggests a name assignment that is constructed from the local integrated schema (which usually not very readable) and provides the integrator with the option to manually change it.

The Doctor and Physician classes are integrated using the Merger operator to create the g-Hospital-Doctor class, see Fig. 7. Furthermore, previously integrated views can participate in further integration procedures. Accordingly, g-Hospital-Patient and g-Hospital-Doctor can be furthur integrated using the same *Merger* operator to generate the g-Hospital-Person class which groups information about all persons working in and/or treated at the hospital, again refer to Fig. 7.

Global classes are specified in terms of the MetaMed ODL$_{I^3}$ knowledge representation language[7] and the information on attribute mappings and default values is represented in the form of *RDF/OWL-assertions*. An example of MODL$_{I^3}$ specification for the global class Hospital-Person is shown bellow:

```
<owl:Class rdf:ID="g-Hospital-Person" >
  <rdf:type rdf:resource="&modl;Interface" />
  <rdfs:subClassOf rdf:resource="&mgkm;Person" />
  <modl:Has-Class-Extent>
     <owl:oneOf rdf:parseType="Collection">
       <modl:Interface rdf:about="#Pnem-Patient"/>
       <modl:Interface rdf:about="#Patient"/>
       <modl:Interface rdf:about="#Physician"/>
       <modl:Interface rdf:about="#Doctor"/>
     </owl:oneOf>
  </modl:Has-Class-Extent>
  <modl:isIdentifiedBy>
     <modl:PrimaryKey>
       <modl:hasAttribute rdf:resource="#g-Hospital-Person.g-
       name"/>
     </modl:PrimaryKey>
  </modl:isIdentifiedBy>
  ...
</owl:Class>

<owl:DatatypeProperty rdf:ID="g-Hospital-Person.g-name">
  <rdf:type rdf:resource="&modl;Attribute" />
  <rdfs:domain rdf:resource="#g-Hospital-Person" />
  <rdfs:range rdf:resource="&mgkm;Person-Full-Name" />
  <rdfs:subPropertyOf rdf:resource="&mgkm;Has-Person-Name"/>
</owl:DatatypeProperty>

<opt:Maps-Class rdf:ID="g-Hospital-Person">
  <modl:Interface rdf:resource="#g-Hospital-Patient" />
```

[7] MODL$_{I^3}$ (Tawil et al. 2008) is a declarative specification of the ODMG-93 Object Definition Language (ODL) (Cattel 1996) that is based on the RDF/OWL knowledge representational language.

```
  <modl:Interface rdf:resource="#g-Hospital-Doctor" />
</opt:Maps-Class>

<opt:Maps-Attribute rdf:ID="g-Hospital-Person.name">
  <owl:oneOf rdf:parseType="Collection">
    <owl:unionOf rdf:parseType="Collection">
      <modl:Attribute rdf:resource="#Cancer-DB.Patient.first-
      name" />
      <modl:Attribute rdf:resource="#Cancer-DB.Patient.last-
      name" />
    </owl:unionOf>
    <modl:Attribute rdf:resource="#Lung-Infection-DB.Pnem-
    Patient.name" />
    <modl:Attribute rdf:resource="#Cancer-DB.Doctor.full-name"
    />
    <modl:Attribute rdf:resource="#Lung-Infection-DB.Physician.
    Name />
  </owl:oneOf>
</opt:Maps-Attribute>
...
```

As we can see from the specification, for each attribute, in addition to its declaration, mapping rules are defined, specifying both information on how to map the attribute on the corresponding attributes of the local classes and on possible default values defined for it on local classes. For example, for the global attribute name, the mapping rule specifies the attributes that have to be considered in each of the classes to which it corresponds. Mapping rules are automatically generated by scanning the mapping relationships associated with the operator history.

The global grid-level schema of the mediator is composed of the global classes generated/defined by the user using the provided integration operators.

4.1 External schema specification module

Part of the customizability philosophy adopted in our design is to enable end users of our system to view the global integrated schema in their preferred DDL. Once the internal mediator schema view is defined, the user can decide on how s/he is interested in viewing that schema. The External Schema Specification Module enable end users to specify templates that facilitate the interpretation of the integrated view internal representation into a target DDL. Each template contains parameters which refer to the necessary information for expressing the syntax of the output DDL. Typical templates for an OO DDL language include Java methods that direct the generation of *class names*, *keys*, *superclasses* and *properties*. These templates are stored in a file called External Schema Specification File (ESSF) for the output language. The following extract of a java program depicts a template that will transform a virtual class from its RDF/OWL internal representation into standard ODL representation:[8]

[8]Other templates can be used to generate for example a Java Data Objects (JDO) specification (Marr 2005)

```
 public createStandardODLTemplate(Interface InterfaceName,
                 Key KeysName,
                 Extent ExtentsName,
                 SuperclassOf Superclasses,
                 Attribute PropertyNames,
                 ValueType PropertyType)
{
   File standardOQL = new File(standardOQL);
   FileWriter fwOQL = new FileWriter(fwStandardOQL);
   write(fwOQL, 'Interface');
   write(fwOQL, InterfaceName);
   fwOQL.write(':');
   write_inheritance(fwOQL, Superclasses);
   fwOQL.write('\n');
   fwOQL.write('(');
   writeStringToFile(fwOQL, 'extent');
   write-extent(fwOQL, ExtentsName);
   fwOQL.write('\n');
   writeStringToFile(fwOQL, 'key');
   fwOQL.write(')'); fwOQL.write('\n');
   fwOQL.write('{'); fwOQL.write('\n');
   writeProperties(fwOQL, PropertyNames, PropertyTypes);
   fwOQL.write('\n');
   writeStringToFile(fwOQL, ('};');
}
void writeStringToFile(FileWriter fw, String s) throws
     IOException
  {
   for(int i=0; i<s.length(); i++)
     fw.write(s.charAt(i));
   fw.write('\n');
  }

void write-inheritance(FileWriter fw, SuperclassOf
      SuperclassesName) throws IOException
  {
   // Enumerates over the inheritance list
     Enumerator eSuperclasses = SuperclassesName.enumerate_list
     (); while(eSuperclasses.has_more_p()){
      writeStringToFile(fw, eSuperclasses.next().toString());
      fw.write(':')
      }
  }
....
..
```

This specification phase is performed once per target DDL. The result of applying the above user defined template to the MODL$_{I^3}$ schema of Fig. 7 we get the following ODMG-93 ODL schema definition:

```
 Interface g-Hospital-Person : Person
(Key g-name)
{
 attribute Person-Name g-name;
 attribute Date-Of-Birth: g-dob;
 attribute Address g-Address;
 attribute Sex g-Sex;
}

Interface g-Hospital-Patient : g-Hospital-Person, patient
( )
{
 attribute Patient-Blood-Type g-blood;
 attribute Job g-job;
 attribute set<Doctor> g-doctor;
}
...
..
```

Queries posed by the user using his/her preferred DDL are first translated to the MODL$_{I^3}$ internal representation before being processed by the QM. Clearly it is much easier to create DDL templates when the target DDL is at least as expressive as MODL$_{I^3}$. When the target DDL is less expressive than the internal representation model, issues of limited expressiveness of the target DDL need to be considered.

5 Query reformulation

In this section we will give a few examples of how a query processor can make use of the integration knowledge in extracting data from several ISs. We describe how it is possible to exploit the information on the grid schema for global processing. When a grid user submits a query on the global schema, MetaMed takes the query and produces a set of subqueries that will be sent to the single information sources. According to other semantic approaches (Comito et al. 2005; Levy et al. 1996; Arens et al. 1993), this process consists of two main phases:

- semantic optimisation;
- query plan formulation;

5.1 Semantic optimisation

In this phase, the MetaMed mediator operates on the query by exploiting the semantic optimistation techniques in order to reduce the query access plan cost. The query is replaced by a new one that incorporates any possible restriction which is not present in the original query but is logically implied by the query on the global schema. The transformation is

based on logical inference from content knowledge (in particular on the integrity constraint rules) of the mediator schema, shared among the local classes from which the global class is generated.

Let us the consider the the request: "*Retrieve the doctors whose annual salary is greater than £*55000":

```
select name
from g-Doctor
where g-salary is > 55000;
```

Let us suppose that in both integrated databases exists a relationship between the salary of the doctor and his rank, stating that all doctors with a salary greater than £55000 are ranked as consultants. This constraint rule can be defined in the following way:

```
(Defining-Axiom R1
      (=> (and (Interface g-Doctor)
           (> g-Doctor.g-salary 55000)
           (Monetary-Units g-Doctor.g-salary pounds))
        (= g-Doctor.rank Consultant))).
```

The mediator executes the semantic expansion of the input query by applying rule R1. The resulting query is the following:

```
select name
from g-Doctor
where g-Doctor.g-salary > 55000
and g-Doctor.rank = 'Consultant';
```

Semantic expansion is performed in order to add predicates in the *where clause*: this process makes query plan formulation more expensive (because a heavier query has to be translated for each interesting source) but single source's query processing overhead can be lighter if secondary indexes into predicates added exist in the servers involved.

5.2 Query plan reformulation

Once the mediator has produced the optimized query at the grid level, a set of subqueries for the local ISs will be formulated. For each IS, by using the mapping rules associated with each global class, the mediator has to express the optimized query in terms of local schemas. In order to produce the translated query, the mediator checks and translates every predicate (*boolean factor*) in the where clause of the global query to determine the local query to be generated. Starting from the local classes associated with the global class(es) queried by the user.

Referring to our query example, the step 1 of the reformulation algorithm will consider the querying of two local ISs' classes (tables) Pnem-Patient and Patient, so that we derive the following queries for servers Lung-Infection-DB and Cancer-DB respectively:

```
select name
from Lung-Infection-DB.Physician
where Physician.wage > 55000
and  Physician.rank = 'Consultant';
```

```
select fname, lname
from Cancer-DB.Doctor
where Doctor.salary > 55000
and  Doctor.position =
'Consultant';
```

If integrity constraints are provided also for a local server, it is possible to perform semantic query optimization with ODB-Tools on the corresponding local queries too, before sending the query to the source. For example, if the following rule is provided for the Doctor class of the Cancer-DB:

```
(Defining-Axiom R2
        (=> (and (Interface Cancer-DB.Doctor)
                 (= Doctor.Position Consultant))
            (> Doctor.staff# 100))).
```

R2 can be used to optimize the local query S2, giving raise to the following query:

```
select fname, lname
from Cancer-DB.Doctor
where Doctor.salary > 55000
and  Doctor.position = 'Consultant'
and  Doctor.staff# > 100;
```

This transformation could be useful if, for example, an index is available for the staff# attribute.

5.3 Query optimisation using extentional knowledge

Another type of knowledge which could be useful for query optimisation is interschema knowledge of extensional type. It is defined in terms of *interschema properties*, specifying mutual relationships between class instances and corresponding databases.

Example of interschema properties that can be specified for two classes c and c' at the extensional level are (*equivalence, containment, overlapping, disjointness*).

Referring to our example, let us suppose that the designer is able to state this extensional property: Ext(Pnem-Patient) $\subset$ Ext(Patient), stating that the patients stored in the Pnemonia class of the Lung-Infection-DB are subset of those stored in the Patient class of the Cancer-DB. The mediator may exploit this knowledge during the query processing phase to minimize the data processing required. In particular, if the user is looking for the name of the patients whose blood type is AB, the following query will be submitted:

```
select name
from g-Hospital-Patient
where g-blood = 'AB';
```

By exploiting the introduced extensional property, the mediator knows that all patient instances of the Pnem-Patient class are also stored (and thus retrieved) in the Patient class. Consequently only the following query is considered and will be sent to the Patient class of the cancer database:

```
select name
from g-patient
where g-blood = 'AB';
```

While no query is sent to the Pnem-Patient table.

6 Conclusions

In this paper we have described the design and implementation of a multiple views integration language that facilitates the efficient production of multiple integration views from multiple heterogeneous ISs in a Grid Environment. The language is based on a small set of integration operators that are capable of providing a consistent and non-redundant representation of distributed participating ISs. The operators are customizable to satisfy the needs of different user groups or participating programs. Users decisions on concepts to be integrated are validated by the system.

The MetaMed system offers an integration language of operator services for grid end-users to work with in terms of integrated and customized mediator views. Global queries are executed by means of mapping requests against the mediator schema to requests against the underlying ISs schemas. The mapping of queries is facilitated by a set of reformulation rules that accompany each view. Finally, we believe that the availability of a rich set of integration operators and the declarative nature of our language substantially increases the degree to which service-oriented mediator functionalities can be integrated in a grid setting.

References

Al-Mourad, M.B., Gray, W.A., Fiddian, N.J. (2003). Multiple views with multiple behaviours for interoperable object-oriented database systems. In *Proceeding of 14th international conference on database and expert systems applications, Czech Republic, lecture notes in computer science (LNCS2736)*. Springer-Verlag.

Arens, Y., Chee, C.Y., Hsu, C., Knoblock, C.A. (1993). Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, *2*(2), 127–158.

Brezany, P., Woehrer, A., Tjoa, A.M. (2004). Novel mediator architectures for grid information systems. *Journal for Future Generation Computer Systems - Grid Computing: Theory, Methods and Applications*, *21*(1).

Cattel, R.G. (1996). *ODMG-93 The object database standard release 1.2*. San Francisco: Morgan Kaufmann. DBLP, http://dblp.uni-trier.de.

Chen, H., Wu, Z., Zheng, G., Mao, Y. (2004). RDF-based schema mediation for database grid. In *Fifth IEEE/ACM international workshop on grid computing (GRID04)* (pp. 456-460).

Comito, C., Talia, D., Gounaris, A., Sakellariou, R. (2005). Data integration and query reformalution in service-based grids: Architecture and roadmap. Technical report, CoreGRID technical report number TR-0013. Institute on knowledge and data management.

Duwairi, R.M. (1997). *Views for interoperability in a heterogeneous object-oriented multidatabase system*. PhD thesis. University of Wales College of Cardiff: Department of Computer Science.

Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I. (2003). Piazza: Data management infrastructure for seman-tic web applications. In *Proceedings of the 12th international world wide web conference* (pp. 556–567).

Halevy, A.Y., Ives, Z.G., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I. (2004). The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, *16*(7), 787–798.

Hillairet, G., Bertrand, F., Lafaye, J. (2009). Rewriting queries by means of model transformations from sparql to oql and vice-versa. In R. Paige (Ed.), *Theory and practice of model transformations, lecture notes in computer science* (Vol. 5563, pp. 116–131). Berlin/Heidelberg: Springer.

Horridge, M., & Bechhofer, S. (2011). The owl api: a java api for owl ontologies. *Semantic Web*, *2*, 11–21.

Jackson, M.J., Antonioletti, M., Dobrzelecki, B., Hong, N.C. (2011). Distributed data management with ogsadai. In S. Fiore & G. Aloisio (Eds.), *Grid and cloud database management*, (pp. 63–86). Berlin/Heidelberg: Springer.

Kaoudi, Z., Dalamagas, T., Sellis, T. (2005). RDFSculpt: managing RDF schemas under set-Like semantics, 123–137.

Koehler, M., & Benkner, S. (2009). A service oriented approach for distributed data mediation on the grid. In *Proceedings of the 2009 8th international conference on grid and cooperative computing* (pp. 401–408).

Kokkinidis, G., & Christophides, V. (2004). Semantic query routing and processing in P2P database systems: the ics-forth sqpeer middleware. In *Proceedings of the 2004 international conference on current trends in database technology, EDBT'04* (pp. 486–495). Berlin/Heidelberg: Springer-Verlag.

Kollia, I., Glimm, B., Horrocks, I. (2011). SPARQL query answering over OWL ontologies. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer & J. Pan (Eds.), *The semantic web research and applications, lecture notes in computer science* (Vol. 6643, pp. 382–396). Berlin/Heidelberg: Springer. doi:10.1007/978-3-642-21034-1.

Levy, A.Y., Rajaraman, A., Ordille, J.J. (1996). Querying heterogeneous information sources using source descriptions. In *Proceedings of 22nd VLDB conference. Mumbai*.

Li, A., Miao, J., Jia, Y. (2010). Research on broken mappings detecting method based on fuzzy aggregation operators in deep web integration environment. In *ICEE* (pp. 125–128). IEEE.

Li, C., & Ling, T.W. (2004). Owl-based semantic conflicts detection and resolution for data interoperability. In *ER (Workshops)* (pp. 266–277).

Lu, H., Ooi, B.-C., Goh, C.H. (1992). On global multidatabase query optimisation. *ACM SIGMOD Record*, *20*(4), 6–11.

Ludascher, B., Gupta, A., Martone, M.E. (2001). Model-based mediation with domain maps. In *Proceedings of 17th international conference on data engineering (ICDE), Heidelberg, Germany IEEE Computer Society*.

Manolescu, I., Florescu, D., Kossmann, D. (2001). Answering XML queries on heterogeneous data sources. In *VLDB* (pp. 241–250).

Manuali, C., & Grif, A.L. (2011). A new collaborative framework for a web service approach to grid empowered calculations. *Future Generation Comparative Systematics*, *27*(3), 315–318.

Marr, S. (2005). The java data objects persistence model. Seminar system modeling 2005, Hasso-Plattner-institute for software systems engineering (pp. 1–8). http://stefan-marr.de/downloads/The-JDO-Persistence-Model.paper.pdf.

Mukherjee, A., & Watson, P. (2012). Case for dynamic deployment in a grid-based distributed query processor. *Future Generation Comparative Systematics*, *28*(1), 171–183.

Muppavarapu, V., Pereira, A., Chung, S. (2010). Role-based access control for a grid system using ogsa-dai and shibboleth. *The Journal of Supercomputing*, *54*, 154–179.

Nejdl, W., & Wolf, B. (2002). EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of 11th WWW conference. Honolulu*.

Orfali, R., & Harkey, D. (1996). Client/Server programming with JAVA and CORBA.

Papakonstantinou, Y., Garcia-Molina, H., Ullman, J. (1995). A query translation scheme for rapid implementation of wrappers. In *Technical report*. Stanford University: Department of Computer Science.

Papakonstantinou, Y., Garcia-Molina, H., Ullman, J. (1996). MedMaker: A mediation system based on declarative specifications. In *Proceedings of international conference on data engineering, New Orleans,* (pp. 132–141).

Qu, C., & Nejdl, W. (2004). Interacting the Edutella/JXTA Peer-to-Peer Network with Web Services. In *SAINT*, (pp. 67–73): IEEE Computer Society.

Ricci, A., Denti, E., Piunti, M. (2010). A platform for developing SOA/WS applications as open and heterogeneous multi-agent systems. *Multiagent and Grid Systems*, *6*(2), 105–132.

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R. (2011a). A reference architecture for building semantic-web mediators. In *CAiSE workshops* (pp. 330–341).

Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R. (2011b). A reference architecture for building semantic-web mediators. In C. Salinesi, O. Pastor, W. Aalst, J. Mylopoulos, M. Rosemann, M.J. Shaw, & C. Szyperski (Eds.), *Advanced information systems engineering workshops, lecture notes in business information processing* (Vol. 83, pp. 330–341). Berlin Heidelberg: Springer.

Sheth, A.P., & Larson, J.A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, *22*(3), 183–236.

Smith, J., Gounaris, A., Watson, P., Paton, N.W., Fernandes, A.A.A., Sakellariou, R. (2002). Distributed query processing on the grid. In M. Parashar (Ed.), *Grid computing GRID 2002, lecture notes in computer science* (Vol. 2536, pp. 279–290). Berlin Heidelberg: Springer.

Tawil, A.-R.H., Gray, W.A., Fiddian, N.J. (1998). Ontological commitments for multiple view cooperation in a distributed heterogeneous environment. In *BNCOD* (pp. 179–180).

Tawil, A.-R.H., Fiddian, N.J., Gray, W.A. (2002). Rich semantic-based schema integration in a federated multiple information server environment. *International Journal of Computers and Applications*, *9*(4), 205–226.

Tawil, A.R., Montebello, M., Bahsoon, R., Gray, W.A., Fiddian, N.J. (2008). Interschema correspondence establishment in a cooperative owl-based multi-information server grid environment. *Information Science*, *178*(4), 1011–1031.

Tian, J., Liu, J., Pan, W., Vincent, M., Liu, C. (2008). Performance analysis and improvement for transformation operators in XML data integration. In Y. Zhang, G. Yu, E. Bertino, & G. Xu, (Eds.), Progress in WWW research and development, lecture notes in computer science (Vol. 4976, pp. 214–226). Berlin/Heidelberg: Springer. doi:10.1007/978-3-540-78849-2.

Vidal, V.M.P., Fernandes de Macêdo, J.A., Pinheiro, J.C., Casanova, M.A., Porto, F. (2011). Query processing in a mediator based framework for linked data integration. *IJBDCN*, *7*(2), 29–47.

Young-Jin, K., & Thottan, M. (2011). SGTP: smart grid transport protocol for secure reliable delivery of periodic real time data. *Bell Labs Technical Journal*, *16*(3), 83–99.