2008 20th IEEE International Conference on Tools with Artificial Intelligence

Enabling Cross Constraint Satisfaction in RDF-based Heterogeneous Database Integration

Luis Martín
Polytechnic University of Madrid, Campus de Montegancedo, Boadilla del Monte, CP 28660 Spain
email: lmartin@infomed.dia.fi.upm.es

Alberto Anguita
Polytechnic University of Madrid, Campus de Montegancedo, Boadilla del Monte, CP 28660 Spain
email: aanguita@infomed.dia.fi.upm.es

Ana Jiménez
Polytechnic University of Madrid, Campus de Montegancedo, Boadilla del Monte, CP 28660 Spain
email: ajimenez@infomed.dia.fi.upm.es

José Crespo
Polytechnic University of Madrid, Campus de Montegancedo, Boadilla del Monte, CP 28660 Spain
email: jcrespo@infomed.dia.fi.upm.es

Abstract

The problem of database integration has been widely tackled through different approaches. While data transformation based systems, such as Data Warehouses, reached the acceptation of the industry during the 80's, in the last decade query translation based approaches have gained popularity given their adequacy to dynamic domains. While the former are based on gathering actual data in central repositories, the latter allow data to remain in the original databases. There still exist several issues to be tackled in query translation, mainly if no ad-hoc schema is described, such as problems with scalability and query processing. In this paper we describe a complete database integration and semantic mediation approach, borrowing techniques from both data transformation and local as view data translation methods, and addressing the cross referencing problem. This work has been carried out in the framework of ACGT (Advancing Clinico-Genomic Trials on Cancer) project, supported by the European Commission.

1. Introduction

Dynamic data environments, such as those related with many research activities, require database integration approaches that leverage the complexity in terms of data scalability and performance, as well as a means to maintain access to up to date information in the databases. For this reason, query translation based approaches have gained popularity in the last decade, even when some very important problems have not been already solved. There exist two main different ways to tackle query translation, namely Global as View [1] and Local as View [2]. In Global as View, an ad-hoc schema is created representing the complete integrated database set. By contrast, in Local as View standalone descriptions of the sources are built by means of a global model. When dealing with unstable database integration environments—i.e. where database schemas can change, or new databases can arrive into the system— Local as View based approaches behave better, since there is no need of updating the global schema. However, this type of approach leads to several problems that need to be taken into consideration. The main issue is scalability: in the worst case, all the local views need to be inspected to find the translation of each element in the original query. Another problem is related to expressiveness: the global model is a description of the domain, not and ad-hoc description of the integrated repository (as in Global as View). This can lead to conflicts when a user formulates a query and what is queried is not contained in the underlying databases. Traditionally, Local as View approaches describe the way to translate the query, leaving aside the actual integration of results. In previous works, the data integration process has been defined as the union of the result sets retrieved from the produced queries. This implies that the simple union of the tables is equivalent to the results expected in the original query. Given the fact that most common query translation algorithms produce queries containing the variables of interest of the original query, this approach should work properly. However, there exist two situations where this is not so simple: a) there are variables in the original query translated to more than one variable in more than one database, 2) the original query contains constraints implying values from different databases (cross-reference, aggregation…). It is not possible to satisfy such kind of constraints by simply using a union operation on the retrieved results. Some systems partially cover these cases by supporting the *join* operation of semantically equivalent fields. In our system, we address these issues by creating a projection of the intermediate results retrieved from the original databases, comprising an RDF repository. Final

1082-3409/08 $25.00 © 2008 IEEE
DOI 10.1109/ICTAI.2008.67

results are retrieved using a query produced during the query translation process together with the dedicated queries for the underlying depositories. The results of this query are equivalent to the data expected to be retrieved from the original one.

Our approach is based on the utilization of an ontology, acting as global model. The global schema is extracted from this ontology, using its underlying RDF Schema. This global schema has two main roles in the mediation process: 1) acting as semantic framework for the mapping process, and 2) describing the complete universe of queries for the mediated database set. The use of ontologies for database integration has been widely studied and implemented in projects such as Ontofusion—carried out in our research lab [3] —, TSIMMIS [4] and KAON [5] among others.

The main issue to tackle when trying to implement tools based on a Local as View approach is performance. It is known that query translation in the latter has NP-Hard complexity. This leads to a scalability problem difficult to cope with. In our tool, we restrict the set of queries by creating user profiles. These profiles are based in previously gathered user requirements, which are used during the mapping process to create customized integrated database sets. Given that the mapping process in Local as View is the less complex one among all query translation approach, updates are feasible if profiles need to be changed. Although not all the queries are available, the profile should allow a user to formulate all the different queries he/she may need in his/her work. Another problem that users frequently find when dealing with a Local as View based mediation system is the complexity of the global schema. In the ideal case, this schema is nothing but a description of the domain of data—e.g. an ontology or the combination of several ones—. The schema exposed by our tool is indeed restricted to the allowed queries, being much simpler to understand and navigate.

This paper is organized as follows. Section 2 presents a state of the art review in the field. Section 3 introduces the methods used in this system. Section 4 describes the tools developed. Section 5 shows a set of experiments illustrating the systems behavior. Finally, section 6 points out our conclusions.

2. Background

There exists a plethora of papers dedicated to the subject of heterogeneous database integration. Most systems designed for this purpose follow a mediation-based approach, where a global schema is used to define the space of possible queries that the system is able to cope with. In addition, mappings between the global schema and the schemas of the integrated sources are stored. These mappings contain the necessary information to translate a query in terms of the global schema—i.e. global query—to a set of queries in terms of local schemas. Global as View and Local as View are the two existing approaches to define the mappings. The final result of a global query is obtained by merging the results of the generated queries. Most papers in this area cover the problem of producing queries for the physical databases, but make little to none mention about how actual data are merged to produce a unique result set.

Ullman [6] reviews the theoretical concepts that surround the query translation process in view-based integration systems. He considers global queries as a conjunction of predicates—no mention of additional constrains that bound the solutions to such predicates is made. The process of answering a query is described as a search of combinations of views contained in the original conjunction of predicates. Each of these combinations provides a partial solution to the original query. The procedure to obtain the final solution is stated in the paper as performing "*the union of all these partial solutions*", but no further details on this topic are given. Xu [7] describes his own approach for a data integration system using a mixture of LaV and GaV approaches for query reformulation, however no mention can be found about how the final result is computed. Saw [8] presents his work for integrating heterogeneous XML data sources, focusing on coping with the structural differences of the schemas to integrate. No mention is made about how final results are produced.

Some works briefly mention the process of obtaining a result to the original query by performing either a union of partial results—at most, a join operation is proposed. Halevi [9] proposes his bucket algorithm for performing query decomposition, stating that the result of this process is *"the union of two conjunctive queries"*. Pottinger performs a similar statement when describing her Minicon algorithm for query translation [10]. The same idea can be found in [11], where Cali describes his system for performing semantic integration of heterogeneous sources. Xiao, in his description of his own approach for integration of heterogeneous XML sources [12], states that the partial results are *"integrated (by using union) to produce the answer to q"*. Lehti, in his paper about integration of XML sources using OWL as global schema [13], mentions the necessity of defining join conditions when merging data from different sources. Camillo [14] focuses on the translation mechanism for producing a set of queries from a global query, mentioning also the necessity of including a mechanism *"to unify query results coming from several XML sources into a single query result in accordance to the global schema"*.

Other systems do take into account global integrity constraints during the process of query processing and result merging—to cope with the problem of inconsistencies between sources. Proper join operations are performed when partial results share semantically equivalent fields. Lenzerini [15] mentions the inconsistent sources issue in integration environments, but makes no mention of cross-constrains solving. In [16], a method for merging partial results using integrity constrains is presented, however only cases where all integrated sources share the same schemas are described. In [17], a method for performing query processing under primary and foreign keys restrictions over the global schema is described. Amann et al. [18] present an approach for integrating heterogeneous XML sources. In it, global and local schemas are populated with key values. This allows

deciding, when possible, whether instances from different sources refer to the same entity, and thus require a join operation when merging the results. Jian et al. [19] use union and join operations to generate the final result from a set of partial results obtained from accessing the underlying databases.

Some systems generate query processing plans which include relational algebra operators to merge the data obtained from local sources. In DISCO [20], scan, project and join operators over partial results are used in order to compose the final answers to user queries. Mena et al. [21] propose a system which performs data integration by storing partial results in an auxiliary SQL database, which is later queried using extended relational algebra operators in order to perform proper union of data. This system is however limited to relational databases which significantly limits its scope.

Nevertheless, none of the mentioned approaches feature full cross-reference constrains in queries over the global schema. They consider semantic equivalence of fields only when these represent keys in the global schema, in order to avoid inconsistencies between sources in the final results.

3. Methods

Database integration can be divided in a set of different sub problems, tackled by different approaches. Among the most important sub problems we have identified schema level heterogeneity, instance level heterogeneity, performance in query translation and results retrieval, complexity of the mapping process and complex query constraints satisfaction. In our group we have developed different models and tools to deal with many of these issues. In this section we present the query translation and data integration complete method we use to cope with schema level heterogeneities and query constraints satisfaction. The other problems mentioned above are also tackled by our system (see section IV), but the specific methods are beyond the scope of this paper.

This section illustrates the method used to obtain a set of results R from a given query over the global schema Q. The goal is to integrate all partial results from the local sources giving all possible information, while satisfying the constraints contained in Q. The process involves: i) the generation of a set of queries in terms of the local databases integrated in the system, ii) the storage of the partial results that these queries generate into an auxiliary database, and iii) the extraction of the final results from this database. The procedure of translating a global query into a set of local queries has been covered in many previous publications [6, 7, 9, 10, 11, 12, 14, 15]. The approach exposed in this work does not differ from them in the essence of query translation. However, complete support for cross constraints is provided by using an auxiliary database to store partial results. In [21], an auxiliary database is employed to store partial results, but only relational databases are supported, and only join constraints are applied over the auxiliary database. By contrast, our system integrates RDF-based databases—which encloses the relational model—and supports any kind of constraint included in the global query, even when it involves data from different sources.

Figure 1 depicts the schema of the complete process of answering global queries.

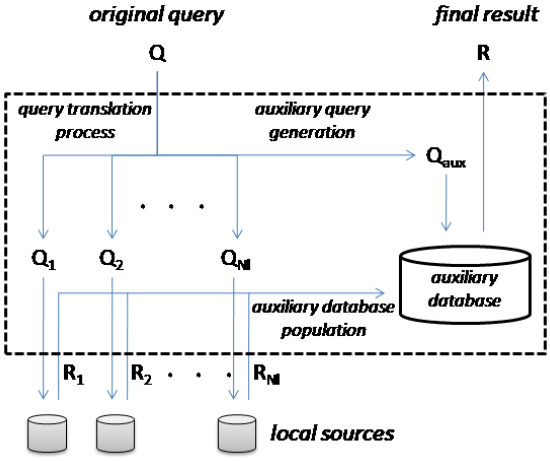

Fig 1. General schema of the database integration system.

The next subsections describe the process to obtain the set of local subqueries and the auxiliary query from a given global query, the generation of the auxiliary database for retrieving the final result, and the general algorithm followed by the system.

3.1. Translation of global queries

This subsection contains definitions which are later used to describe the general algorithm of the integration system.

Definition 3.1 A *query* Q is a triple $\{S, V, C\}$, where S is the set of symbols representing the queried variables—in the *SELECT* statement—, V is the set of global views composing the query, and C is the set of constraints contained in the query.

Definition 3.2 A *view combination function* α is a function that generates all possible combinations of views from a given set of views and a set of joining constraints: $\alpha(V, C) \rightarrow \{v_i | \exists vi_1, vi_2 \in V, \exists (var_1 = var_2) \in C, v_i = join\ (vi_1, vi_2)_{(var_1 = var_2)}\}$.

Definition 3.3 A *view translation function* β is the function that translates global views into semantically equivalent local views—as defined in a given local description: $\beta(ld, vi) \rightarrow v | (v \approx vi) \in ld$, meaning $(v \approx vi)$ that v and vi are mapped views in ld

Definition 3.4 A *query translation function* γ is the function that generates all local queries and the query for the auxiliary database from a given global query and a set of local descriptions LD: $\gamma(S, V, C, LD) \rightarrow \{LQ, Q_{aux}\}$. The generated elements are LQ—the set of local queries—and Q_{aux}—the query for the auxiliary database.

LQ is itself composed by three sets of elements: $LQ = \{S'_i, V'_i, C'_i | 1 \leq i \leq N_l\}$, being N_l is the number of local descriptions—mappings with local sources. S'_i represents the set of queried variables in local query i, V'_i is the set of views in local query i and finally C'_i is the set of constraints in local query i. We define these as follows: V'_i is the translation of the views in V with ld_i. $V'_i = \{v'_j | \exists vi \in V, \beta(ld_i, vi) = v'_j\}$. C'_i is formed by constraints contained in C whose complete set of variables exist in at least one of the views of the local query: $C'_i = \{c'_j | c'_j \in C, \forall var \in c'_j \; \exists v'_k \mid var \in v'_k\}$. These are the necessary constraints for correctly performing query i. Finally, the queried variables in the local query S'_i are in first place the intersection of the queried variables in the global query with the variables contained in the set of local views $S \cap varsInV'_i$ —of course global variables are previously translated according the local descriptions. In addition, variables from cross constraints—those that affect variables from more than one local source—must be queried too, since those constraints are treated a posteriori $\{var_j | \exists c \in C, \exists var_k, \exists V'_l, var_j, var_k \in c, \; var_j \in V'_i, var_k \in V'_l\}$.

The other product of function γ is Q_{aux} which, again, is composed by a set of queried variables, a set of views, and a set of constraints: $Q_{aux} = \{S, V_{aux}, C_{aux}\}$. The queried variables are the same as in the original query, as this query must provide the result for that query. The views in Q_{aux} are the projection of the variables in S in S'_i: $V = \{\Pi_{S'_i}(W) | 1 \leq i \leq N_l\}$, where W is the table containing all the variables from the original query. Finally, C_{aux} is the set of cross constraints, that is, constrains which affect views from different local queries: $C_{aux} = \{c_i | c_i \in C, \exists var_1, \exists var_2, \exists V_1 \in V'_j, \exists V_2 \in V'_k, var_1, var_2 \in c_i, var_1 \in V_1, var_2 \in V_2\}$.

Therefore, given a global query Q posed by the user, the process of translating Q to generate a set of subqueries expressed in terms of the underlying databases is first applying function a—to generate all combinations of global views—an function c—to generate the set of local queries. This process also produces the auxiliary query, used to retrieve the final results from the auxiliary database.

3.2. Generation of the auxiliary database

The RDF-based auxiliary database is populated with the results retrieved from the local sources. The purpose is to obtain the result to the original query from it. The data stored in it is subsequently retrieved with an auxiliary query—which details were given in the previous subsection—, enabling the use of cross constraints.

The population of the auxiliary database implies two steps: i) a sequential input of each of the partial results obtained from the underlying databases, and ii) the generation of the RDF database representing the integration of such results. During the first step, an internal structure storing the contents of the included partial results is maintained. Each time a new result set is added, its rows are added—or merged if necessary—to the rows of previous results stored in this structure. The procedure to decide whether a new row must be merged with an existing row is as follows: the intersecting values of both rows are examined—i.e. the fields containing values in both rows—, and if in every field the values are equivalent, then the rows are merged in a single row resulting from the union of both rows. This process provides the simplest form of data integration, as the task of detecting inconsistencies or errors in the integrated data is handled by other tools in the system—see section IV. Once all partial results have been added, the second step is executed. An RDF database is created from the contents of the internal structure. This RDF contains a unique class *ResultSet*. For each of the fields in the integrated results, a *datatype* property is added to this class. Finally, each row in the internal structure produces an instance of the *ResultSet* class, filling its *datatype* values with the values in such row. The resulting RDF database effectively holds the integration of the partial results acquired from the local databases, and thus can be queried to obtain the result to the original query posed in the system. Figure 2 represents the process of integrating the individual resultsd and generating the RDF auxiliary database.

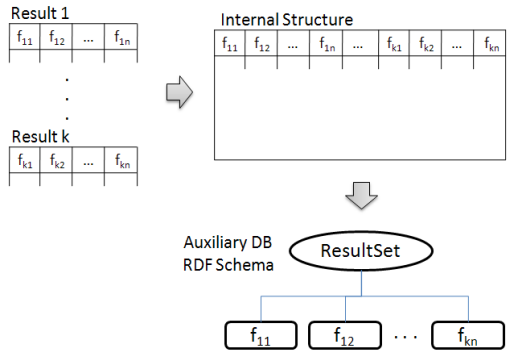

Fig 2. Generating the RDF Schema of the auxiliary database from the result of several local sources

Special care is taken in order to preserve the types of the extracted data in the generated auxiliary database. The *datatype* elements associated to the underlying databases fields are typed accordingly to such fields. At the end of the process, the auxiliary database acts like a temporary global database, storing the collected and integrated data for a single query.

3.3. General algorithm

Having described all elements and functions that take part in the process of answering queries posed in the system, we can now define the general algorithm followed by the system to obtain a result from a global query.

*Given **Q**: a global query posed in the system, composed by **S**—the set of queried variables—, **V**—the set of views composing **Q**—and, **C**—a set of constraints over variables in **V**.*

*Given **LD**: the set of local descriptions containing the mappings between the global schema and the schemas of the local sources.*

1. Apply function α to **V** and **C** in order to obtain all possible combinations of views. Use the newly generated set of views in **Q** instead of **V**.
2. Apply function γ to **Q** and **LD** in order to obtain the set of local queries **LQ** and the auxiliary query Q_{aux}.
3. For each local query Q_i in **LQ**:
    a. Launch Q_i against the local source, obtaining the result R_i.
    b. Generate the part of the RDF Schema of the auxiliary database corresponding to R_i.
    c. Populate the auxiliary database with the data rows contained in R_i.
4. Perform query Q_{aux} over the auxiliary database, obtaining the results **R**.
5. Return **R**.

4. Tools

The ACGT Semantic Mediation layer is comprised by several tools designed to answer and optimize queries. Database integration can be divided in a range of sub problems that mostly need to be solved using different approaches (see section III). In our system we have developed a range of collaborating services supporting the different aspects of the general task. These services interact with the query and data to obtain consistent results. The architecture of the ACGT Semantic Mediation Layer is shown in figure 3.

As can be seen, the Semantic Mediator acts as the core of this system, enabling interactions with the rest of tools. OntoQueryClean [22] and OntoDataClean [23], two systems developed in our group and reported before, are both devoted to solve instance level heterogeneities. Their services are invoked by the mediator when necessary, using the defined interfaces. Although it is not within the query translation process itself, we have decided to include the Mapping Tool as part of the system, given the importance of the mapping process in mediation. The mapping tool aids in the construction of mappings between the global schema and the data repositories. By using these mappings, the mediator is able to automatically produce the required local views and restricted global schemas, necessary for the given database integration approach.

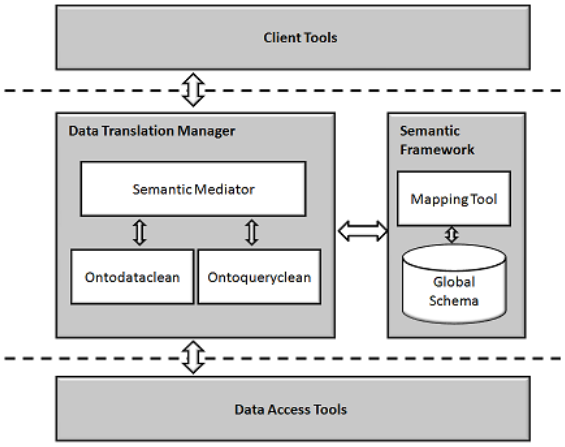

Fig 3: Architecture of the Semantic Mediation Layer

The interfaces used for communicating the services, both internally and externally, are based in web service technologies. An entity acts as a client of a service if it needs something or as a supplier when it is needed. The main internal client of the ACGT Semantic Mediation Layer is the Semantic Mediator, given that it coordinates the work of all the other components. The Semantic Mediator requests, for example, to OntoDataClean to homogenize a set of retrieved data before proceeding with its integration in the general result set.

Data representation in the ACGT Semantic Mediation Layer is based on RDF. We use an RDF Schema (RDFS) to represent the data structure. This global schema represents the possible queries that a client can formulate. The user then gets the perception of querying a unique RDF repository. Regarding the query language, SPARQL [24] was chosen due to its intermediate level of expressiveness and suitability for general purpose applications, as the integration of heterogeneous data sources. A query in SPARQL contains a description of a subset of the global schema in terms of views. Each one of these views has a meaning that is mapped to the corresponding view in one or more underlying databases. The SPARQL query contains a description of the structure of the results and a set of constraints.

This software was built using java technologies, and is exposed as an OGSA-DAI service. The Semantic Mediator is the core tool of this layer. It has two main services: 1) to launch a query, and 2) to browse the schema. The former offers the possibility of sending a query formulated in terms of the global schema for a given integrated set of databases. The Semantic Mediator returns the results in a selected format, together with a metadata file, containing semantic annotations for these data. The second service shows a restricted version of the global schema, representing only the queries that are possible in the integrated database set

Requests are sent to the Semantic Mediator in form of an OGSA-DAI request document [25, 26]. This document contains a very simple workflow, comprised by OGSA-DAI activities. In the case of the Semantic Mediator, this document contains a chain of activities invocations

specifying things like the query, in which form the client wants results to be retrieved or where they have to be delivered.

5. Experiments and results

In this section, we address the issue of proving the feasibility and fitness of the selected approach, together with the performance of the presented tools. We have tested our system with a range of heterogeneous databases of the biomedical domain. Previous experiments, including image (DICOM) and relational sources, with preliminary versions of the system were performed and documented [27]. To achieve this goal, we have classified the different types of conflicts that can be present at different levels.

We identify four conflict categories: a) instance level conflicts, b) schema level conflicts, c) instance-schema conflicts and d) cross constraint conflicts. Instance level conflicts can be present in the retrieved data or in the original query. These conflicts are treated by OntoDataClean and OntoQueryClean tools. Experiments in this area have been documented in previous works [23, 24]. Schema level and instance-schema level conflicts are registered in the mapping files and solved by the mediation algorithm. A schema level conflict is a semantic heterogeneity among two or more databases related to elements in their schemas. When a schema element in the global schema needs to be mapped to instance level knowledge in an underlying database, we say that we have an instance-schema conflict. These conflicts are solved at the mapping level as well. Finally, we find a cross constraint conflict when the original query contains a restriction that involves information from different databases.

In our experiments, we have selected as global schema the semantic core of the ACGT platform: the ACGT Master Ontology on Cancer (MO) [28]. This ontology covers the domain of clinical trials on cancer, so we built ad-hoc databases in this field presenting the different cases of heterogeneity included on purpose. This way, we knew the expected results for every query, being able to evaluate the behavior of the system.

5.1. Case Study

This subsection presents in detail the execution of one of the tested queries, explaining how the system acts in each of the steps and what products are obtained from it. Figure 4 shows the global query performed in the system.

```
SELECT ?id ?age ?regDate ?diagDate
WHERE {
 ?patient hasRegistration ?reg .
 ?reg hasProcessEnd ?endOfReg .
 ?endOfReg hasDate ?regDate .
 ?patient hasDiagnosis ?diag .
 ?diag hasProcessEnd ?endOfDiag .
 ?endOfDiag hasDate ?diagDate .
 ?patient hasAge ?age .
 ?patient hasIdentifier ?id .

 FILTER (?age > 35)
 FILTER (?regDate > ?diagDate)
}
```

Fig 4: Global query involving data from two repositories

This query retrieves data contained in two different repositories. Namely, the patient's personal data, including age, and the date of registration are stored separately from the date of diagnosis. Data from these sources is related by a patient identifier field. From the two constraints contained in the query, the one involving only the age of the patients can be dealt directly in the sources. The second constraint however involves data from more than one source, thus must be tackled a posteriori—it will be appropriately included in the auxiliary query.

The system identifies the elements contained in the query and applies the mapping information in order to generate the queries for the local repositories. The query for the first repository retrieves the patient's id, , age and registration date, properly restricting for values of age greater than 35. The query for the second repository gathers patient identifiers together with dates of diagnosis. The results obtained from these two queries are partially shown in figure 5.

id	*age*	*reg_date*
1	45	12-05-2005
2	74	23-03-2006
3	36	04-07-2005
4	67	11-03-2005
5	51	14-12-2005
…	…	…

id	*diag_date*
1	01-02-2005
2	29-12-2005
3	29-06-2005
4	02-04-2005
5	20-05-2005
…	…

Fig 5: Part of the result sets obtained with the generated subqueries

These results sets are integrated as described in section III in order to generate the auxiliary database. In this case, the only intersecting field is the patient id, thus two different rows will be merged when this value is equivalent in both. The result will be rows with four fields: the shared id, the age and registration date from the first result set, and the diagnosis date from the second result set. Figure 6 depicts the contents of the internal structure after having added both results.

id	age	reg_date	diag_date
1	45	12-05-2005	01-02-2005
2	74	23-03-2006	29-12-2005
3	36	04-07-2005	29-06-2005
4	67	11-03-2005	02-04-2005
5	51	14-12-2005	20-05-2005
…	…	…	…

Fig 6: the previous step to the auxiliary database generation produces an internal structure storing the integration of all results sets

The next step is producing the actual auxiliary RDF database from the contents of the internal structure. This involves generating one class with four datatypes—one for each of the previous fields. The *id* and *age* datatypes are of integer type, while *reg_date* and *diag_date* store date values. The schema of this database is shown in figure 7.

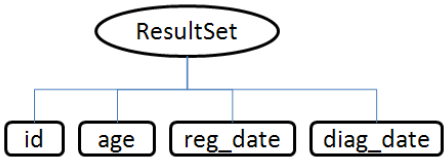

Fig 7: auxiliary database RDF Schema for the presented example

For each row of integrated results an instance of the *ResultSet* class is generated. Thus, for example, the first row in figure XXX produces an instance whose datatype values are respectively instantiated to '1', '45', '12-05-2005' and '01-02-2005'. Parallel to the population of this database, the system automatically generates the auxiliary query, devoted to extracting the final results from the auxiliary database. Figure 8 shows the auxiliary query generated for this experiment.

```
SELECT DISTINCT ?id ?age ?regDate ?diagDate
WHERE {
 {
  ?row id       ?id .
  ?row age      ?age .
  ?row regDate  ?regDate .
  ?row diagDate ?diagDate .
 }

FILTER (?regDate > ?diagDate)

}
```

Fig 8: auxiliary query generated by the system

This query includes the second constraint contained in the original query, since it could not be treated by the individual sources. This way, we ensure that the results obtained from this query represent indeed the results corresponding to the global query posed by the user. Fig 9 shows the results retrieved from the auxiliary database.

id	age	reg_date	diag_date
1	45	12-05-2005	01-02-2005
2	74	23-03-2006	29-12-2005
3	36	04-07-2005	29-06-2005
5	51	14-12-2005	20-05-2005
…	…	…	…

Fig 9: final results retrieved from the auxiliary query and returned by the system

As can be seen, the results contain the product of a proper merging between the partial results of the two involved local repositories. The identifier field was used to perform the join of similar data. The cross constraint was successfully processed, and no unexpected data was included in the final results.

5.2. Results

We built two integrated repositories presenting different types of heterogeneities using the databases mentioned above. A set of queries were designed to test the behavior of the system in different scenarios. These queries tested not only isolated heterogeneities, but also cases of combination. The system was running in a single Intel Core 2 Duo 3Gb+, 8Gb RAM computer. A set of 20 queries were built to test the different cases of heterogeneity present in both repositories.

In all cases, heterogeneities were solved and constraints were satisfied. The automatic construction and storage of the temporal repository did not lead to complexity problems. However, no formal tests regarding performance were done. We understand that the creation of the temporal repository is not more complex than unifying separate results directly. We plan to study performance issues in future studies.

6. Conclusion

Classical Local as View database integration approaches are based on union operations to join the individual results and build an integrated data set. In this work we present a method uses a temporal RDF data repository to allow more complex constraints in the original query, involving data from different data sources. In future releases of our system, we plan to include features such as aggregation operators. Although aggregation is not yet supported by SPARQL, this type of specifications could be easily added to the query system, and applied directly to the temporal repository.

ACKNOWLEDGMENT

We would like to thank all partners in the ACGT project for their technical advisory. We want to thank also Prof. Victor Maojo for his comments on this work.

7. References

[1] J. D. Ullman, "Information integration using logical views", in Proceedings of the International Conference on Database Theory (Delphi, Greece), 1997, pp 19–40.

[2] A.Y. Levy, A. Rajaraman, and J.J. Ordille, "Querying heterogeneous information sources using source descriptions", in Proceedings of the Twenty-second International Conference on Very Large Data Bases (VLDB'96), Mumbai (Bombay), India, September 1996, pp. 251–262.

[3] D. Pérez-Rey, V. Maojo, M. García-Remesal, R. Alonso-Calvo, H. Billhardt, F. Martin-Sánchez and A. Sousa, "ONTOFUSION: Ontology-based integration of genomic and clinical databases", in Computers in Biology and Medicine, In Press, Corrected Proof, Available online 6 September 2005

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom, "The TSIMMIS project: Integration of heterogeneous information sources", in Proceedings of the 10th Meeting of the Information Processing Society of Japan, , Tokyo, Japan, October 1994, pp. 7–18.

[5] Bozsak E. et al. "KAON - Towards a Large Scale Semantic Web". In K. Bauknecht, A. Min Tjoa, and G. Quirchmayr, editors, EC-Web 2002, volume 2455 of Lecture Notes in Computer Science, Springer, September 2002, pp. 304–313.

[6] [Ullman00] J.D. Ullman, "Information integration using logical views", Proc. of the 6th Int. Conf. on Database Theory (ICDT'97), volume 1186 of Lecture Notes in Computer Science, Springer, 1997, pp. 19-40.

[7] L. Xu, and D.W. Embley, "Combining the Best of Global-as-View and Local-as-View for Data Integration", Proceedings of ISTA 2004: 3rd International Conference on Information Systems Technology and its Applications, Salt Lake City, USA, 2004, pp. 123-136.

[8] N.T.H. Saw, and K.H.S. Hla, "Semantic Interoperating and Accessing Heterogeneous and Autonomous XML Sources", Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'06), Hong Kong, 2006, pp. 216-219.

[9] A.Y. Halevy, "Answering queries using views: A survey", VLDB Journal, 10:4, 2001, pp. 270-294.

[10] R. Pottinger, and A. Halevy, "Minicon: a scalable algorithm for answering queries using views", VLDB J. 10(2), 2001, pp. 182-198.

[11] A. Calì, D. Calvanese, G. De Giacomo, M. Lenzerini, P. Naggar, and F. Vernacotola, "IBIS: Semantic data integration at work", Proc. of the 15th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2003), volume 2681 of Lecture Notes in Computer Science, Springer, 2003, pp. 79-94.

[12] H. Xiao, and I.F. Cruz, "Integrating and Exchanging XML Data Using Ontologies", LNCS Journal on Data Semantics, Springer Verlag, 2006, pp. 67-89.

[13] P. Lehti, and P. Fankhauser, "XML Data Integration with OWL: Experiences and Challenges", 2004 Symposium on Applications and the Internet (SAINT 2004), 2004, pp. 160-170.

[14] S.D. Camillo, C.A. Heuser, and R. dos Santos Mello, "Querying Heterogeneous XML Sources through a Conceptual Schema", Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003), 2003, pp. 186-199.

[15] M. Lenzerini, "Data integration: A theoretical perspective", Proceedings of the Symposium on Principles of Database Systems (PODS), 2002, pp. 233-246.

[16] J. Lin, and A.O. Mendelzon, "Merging databases under constraints", Int. J. of Cooperative Information Systems, 7(1), 1998, pp. 55-76.

[17] D. Lembo, M. Lenzerini, and R. Rosati, "Source inconsistency and incompleteness in data integration", Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002), 2002.

[18] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl, "Ontology-Based Integration of XML Web Resources", Proceedings of the 1st International Semantic Web Conference (ISWC 2002), 2002, pp. 117-131.

[19] L. Jian, and J. Beihong, "Query Division and Reformulation in Ontology-Based Heterogeneous Information Integration", 15th International Conference on Computing (CIC '06), Mexico City, Nov. 2006, pp. 186-196.

[20] A. Tomasic, L. Raschid, and Patrick Valduriez, "Scaling Access to Heterogeneous Data Sources with DISCO", IEEE Transactions on Knowledge and Data Engineering, v.10 n.5, September 1998, pp.808-823.

[21] E. Mena, V. Kashyap, A.P. Sheth, and A. Illarramendi, "OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies", Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (CoopIS 1996), 1996, pp. 14-25.

[22] A. Anguita, L. Martín, J. Crespo, and M. Tsiknakis, "An Ontology Based Method to Solve Query Identifier Heterogeneity in Post-Genomic Clinical Trials", Proceedings of the 21st International Congress of the European Federation for Medical Informatics (MIE2008), Göteborg (Sweden), May 25-28, 2008, pp. 3-8.

[23] D. Perez-Rey, A. Anguita, and J. Crespo, "OntoDataClean: Ontology-based Integration and Preprocessing of Distributed Data", Lec. notes in Computer Science 4345, 2006, pp. 262-272.

[24] [SPARQL] SPARQL Query Language for RDF. Available at: http://www.w3.org/TR/rdf-sparql-query/

[25] The OGSA-DAI Project. Available at: http://www.ogsadai.org.uk/

[26] M. Antonioletti, M.P. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. "The Design and Implementation of Grid Database Services in OGSA-DAI". Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, February 2005, pp. 357-376.

[27] L. Martín, E. Bonsma, A. Anguita, J. Vrijnsen, M. García-Remesal, J. Crespo, M. Tsiknakis, V. Maojo, "Data Access and Management in ACGT: Tools to Solve Syntactic and Semantic Heterogeneities Between Clinical and Image Databases", in Advances in Conceptual Modeling – Foundations and Applications, Lecture Notes in Computer Science, 2007, pp. 24-33.

[28] M. Brochhausen, G. Weiler, C. Cocos, H. Stenzhorn, N. Graf, M. Doerr, M. Tsiknakis, "The ACGT Master Ontology on Cancer - a New Terminology Source for Oncological Practice", in IEEE CBMS 2008: 21st IEEE International Symposium on Computer-Based Medical Systems, Jyväskylä, Finland, June 17-19, 2008.