Ontocloud – a Clinical Information Ontology Based Data Integration System

Diogo F.C. Patrão[1], **Helena Brentani**[2], **Marcelo Finger**[3], **Renata Wassermann**[3]

[1] A.C. Camargo Cancer Center

[2]Psychiatry Dept, Univ. of São Paulo

[3]Computer Science Dept, Univ. of São Paulo

djogo@cipe.accamargo.org.br, helena.brentani@gmail.com,

{mfinger,renata}@ime.usp.br

Abstract. *Relevant biomedical research relies on finding enough subjects matching inclusion criteria. Researchers struggle to find eligible patients due to: information scattered in many different databases, incompatible data representation, and the technical knowledge required to work directly with databases. We identified the required features of a clinical data search system and used it to design and evaluate Ontocloud, a prototype based on open source software and open standards of a dynamic ontology based database integration system with inference capabilities. A comparison between Ontocloud and three other database integration system showed that our prototype fulfilled its purpose and can be improved to be used in production.*

1. Introduction

The technology to quickly retrieve patient information from the Electronic Health Record is crucial to biomedical research. Traditional term based search techniques have been failing to bring accurate and precise results, due to the high complexity of this knowledge domain[Chard et al. 2011]. Database integration [Lenzerini 2002][Halevy 2001][Haas et al. 2002] provides techniques to consolidate information on several source databases through a set of mappings, into a single global database, which is then queried by the user. The most established database integration tools are based on relational databases, which are not tailored to deal with different conceptualizations of the source databases[Sujansky 2002].

Data collection for cancer research in a large hospital such as A.C. Camargo Cancer Center is hindered by a series of factors, the most important being: (1) Data is stored in many different databases in diverse ways, constantly changing and evolving; (2) Data is represented in a computer friendly format, hard to understand by physicians and scientists; (3) Collecting data manually is a time-consuming task, and clinical research projects need speed and accuracy on the recruit phase, (4) the same information may be present in different levels of detail, and (5) certain information is not explicitly asserted, but may be inferred based on indirect data.

In this work, we designed, implemented and evaluated a prototype of a database integration system called Ontocloud, based on open source software and standards. It addresses the issues (1)-(5), by providing some key features: dynamic access to data on source databases; ontologies as the medium for data integration; and inference of concepts, harmonizing the detail level of similar information (the semantic mismatch issue), independence of source databases and data annotation. We describe how we implemented Ontocloud to solve a use case of integrating medical document metadata, and compare its characteristics against three other database integration architectures.

2. Background

2.1. Database integration

A database integration problem is described as taking several sources of complementary data and providing a single view for those sources[Halevy 2001]. In a theoretical perspective[Lenzerini 2002], we can represent a data integration system $\mathcal{I}$ as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the global view, $\mathcal{S}$ is the set of source databases and $\mathcal{M}$ is the set of mapping functions from $\mathcal{S}$ to $\mathcal{G}$.

There are two methods for providing the global view $\mathcal{G}$: *dynamic* or *static*. Dynamic methods translate a query on global view $\mathcal{G}$ to queries on the relevant source databases $\mathcal{S}$ and translate back the answers using the mappings $\mathcal{M}$. Static methods (or data warehouse methods) create a materialized global view, by translating and copying all data from the sources $\mathcal{S}$ into a new database $\mathcal{G}$.

Both methods have their benefits and drawbacks. Dynamic methods rely on *query rewriting* or *query answering*, which are hard computational problems and therefore imply on slower performance. As they directly query the source databases, results are always up to date. Static methods are easier to set up and faster to query, however there is the need to translate all data on the sources and construct a new database before any queries can be answered. This procedure may require a higher level of access on the source databases and may take a great deal of time and disk space. Also, results are mostly always outdated, and the global database needs to be refreshed periodically[Halevy 2001].

Regarding the mappings, database integration systems can be classified as global as view (GAV) or local as view (LAV). Mappings on GAV systems transforms the source database into the global view, and queries are answered by several different algorithms[Halevy 2001]. Mappings on LAV systems maps the global view into the source, and in order to answer a query presented to the global view $\mathcal{G}$ the system should apply query answering (to infer results on $\mathcal{G}$ based on results on $\mathcal{S}$) or query rewriting (which translates the mappings from LAV to GAV). GAV mappings are easier for a developer to create than LAV mappings, however the former requires that all source databases are joined in one statement, being thus harder to add and remove sources than LAV. The query answering or rewriting step in a LAV system, depending on the complexity of mappgins, may demand a great deal of computation to be solved, if solvable at all; GAV systems relies on faster algorithms.

2.2. Ontologies, inference and database integration

An ontology represents knowledge in a formal framework, as concepts and relationships between pairs of concepts. Ontologies have been considered in heterogeneous database integration due to their ability to perform inferences and potential to deal correctly with the semantic mismatch problem [Wache et al. 2001] [Cruz and Xiao 2005].

Semantic mismatch is a problem that is intrinsic to data integration that usually leads to *loss of specificity* [Sujansky 2002] [Hull 1997]. It occurs when two sources of information have fields with similar but incompatible meanings. Usually, when it is necessary to join the two sources, the lowest level of detail should be adopted. In some cases of concept overlap it can be impossible to join sources. Ontologies, in data integration, mitigate information loss for some types of semantic mismatch. Figure 1 presents an example of such mismatch for information on patient smoking.

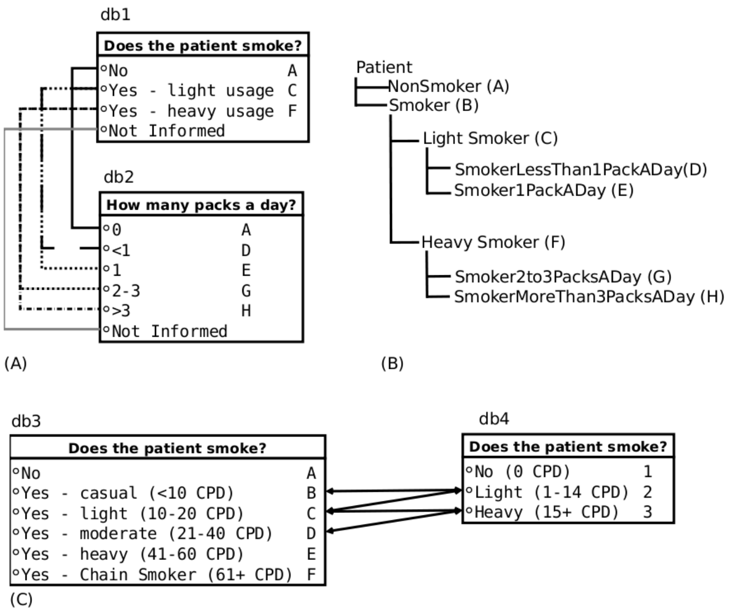


Figure 1. Semantic mismatch. (A) An example of a similar field in two different database $db1$ and $db2$; when the $db2$ field has value `0`, it is equivalent to a value of `N` on $db1$, <1 and `1` is equivalent to `Yes - light usage` and `2-3` and >3 to `Yes - heavy usage`. A reverse mapping would not be possible without loss of information specificity, because the options on db1 regarding light and heavy smokers might mean more than an option on db2. (B) An ontology that classifies all concepts involved on the source databases db1 and db2 from (A). Specific concepts such as $Smoker2to3PacksADay$ are classified under more general concepts, in this case, $HeavySmoker$ and $Smoker$. Instances of a specific class are considered also as belonging to its parent classes. (C) db3 and db4 contains an example of a semantic mismatch that is impossible to solve: note how the concept `Yes - light (10-20)` on $db3$ can be mapped to both `light (1-14 CPD)` and `heavy (15+ CPD)` on $db4$, at the same time that those two concepts on $db4$ maps each to two concepts on $db3$ (CPD - Cigarretes per day).

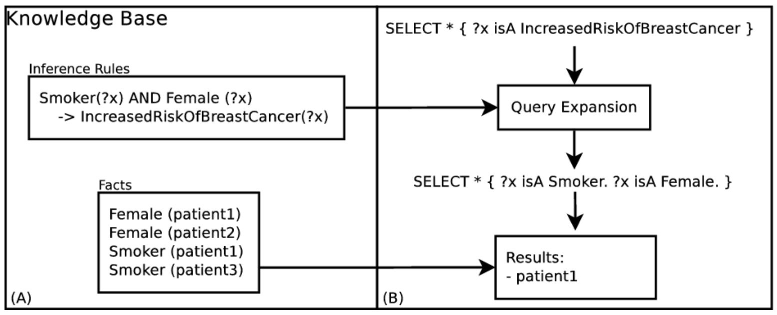
Ontologies can be represented in RDF/XML[1] format or in triplestores, which can be thought of as an equivalent of a database for ontologies. SPARQL[2] is the query language defined for querying data in an ontology. The SPARQL 1.1 specification allows for joining remote endpoints and thus integrating different datasets.

Inference is the process by means of which new information is derived from existing data from an ontology. Given abstract concepts, general rules can be added to a

[1]http://www.w3.org/TR/PR-rdf-syntax/
[2]http://www.w3.org/TR/rdf-sparql-query/

knowledge base to allow for new facts to be inferred[Russell and Norvig 2003]. An inference rule is divided in two parts, the head and the body. If the statements on the body is true, then the head statement will also be true. See Figure 2 for an example of an inference rule.

Query expansion [Bhogal et al. 2007] achieves inference by applying the rules over the query statements, instead of the facts of the knowledge base. A query q_G that specifies concepts presents on the head part of some inference rule may have this statement substituted by the body part of the rule (Figure 2).



Knowledge Base
Inference Rules
Smoker(?x) AND Female (?x) -> IncreasedRiskOfBreastCancer(?x)
Facts
Female (patient1)
Female (patient2)
Smoker (patient1)
Smoker (patient3)
(A)
SELECT * { ?x isA IncreasedRiskOfBreastCancer }
Query Expansion
SELECT * { ?x isA Smoker. ?x isA Female. }
Results:
- patient1
(B)

Figure 2. Inference by means of query expansion. (A) The inference rule states that if there is a patient $?x$ which belongs to class `Smoker` and `Female` (rule body), then this patient also belongs to class `IncreasedRiskOfBreastCancer` (head). (B) The term specified on the query is not stated as a fact on the Knowledge base, however a inference rule allows the terms to be substituted and the query can be answered.

3. Related work

Calvanese [Calvanese et al. 2007] describes Mastro-I, a data integration management system designed in order to maintain data complexity within reasonable bounds. It relies on the IBM product Infosphere Federation Server[3] to access source databases. In other work[Calvanese et al. 2011], the same group describes a database integration case using Mastro-I, in which five different data models were used, including XML-based and relational databases. The integration was made in two steps: first the different data models were combined using the InfoSphere Federation Server; then the Mastro-I system was used to map those entities into concepts, thus achieving data integration. In this architecture, there are two layers of heterogeneity solving: first, all relational data is mapped at the Federation Server, and then mapped into DL concepts, where integration is actually achieved.

DBOM [Cure and Bensaid 2008] is a GAV data integration system that uses decidable fragments of OWL language, OWL-DL and OWL-DL lite, to map results from queries over a relational database to an ontology. Several different relational sources can be used at once. It is able to deal with different degrees of confidence on each source, by configuring parameters on the mappings. It is implemented as a Protégé[4] plug-in, however, it is not cited whether this plugin is available, nor it has been found on the internet for download. As a use case the author presents the integration of two drug databases.

[3]http://www-01.ibm.com/software/data/infosphere/federation-server/

[4]http://protege.stanford.edu

The Query Integrator System (QIS) [Iller and Adkarni 2004] is a layer-based architecture that uses ontologies to represent and annotate metadata about the source databases; each change detected on the schemas generates annotations that can be reviewed later. It focuses on a dynamic environment where the source database schemas are constantly changing. Queries are composed by means of a visual tool that presents the annotation about the source databases, and translates these queries into SQL in the source databases.

Min et al [Min et al. 2009] integrated two sources of prostate cancer clinical data: one mantained by the Radiation Oncology department and the other from the Tumor Registry. The first contained data about radiotherapy treatment and the other demographic data. Both databases were integrated into one ontology by using a single D2R-Server instance. Integration was done by mapping concepts to two different databases in one single server. The integration was horizontal, as each database contained complementary data about one patient, except for one field, the TNM status, which was present in both.

Analyzing the available tools, none of them has features allowing to solve all of the clinical database integration issues we verified, except for Mastro-I and DBOM. However, the first relies on non-free software and it requires that relational sources are integrated first on a relational layer (the Infosphere Federation Server), and then on the ontology layer (Mastro). DBOM seems to be an interesting take on the subject, however it is not available anywhere for download. QIS has very interesting features but is based on obsolete standards and software.

4. Ontocloud design

Ontocloud was designed to provide dynamic access to a consolidated database global view of several database sources, using ontologies to consolidate heterogeneous data. Given a set of source databases $\mathcal{S}_{1..n}$, a set of source endpoint $\mathcal{E}_{1..n}$ should be provided. Each source publishes its objects of interest concepts of the global view $\mathcal{G}$ by means of a SPARQL endpoint. In order to get answers to a query $q_{\mathcal{G}}$ over the global database $\mathcal{G}$, the query must go through two transformation steps: the query expansion step accounts for inference, substituting terms not directly defined on the source endpoints; then the query federator step provides the query with SERVICE clauses that indicate in which source endpoint each concept is to be found (Figure 3).

Ontocloud uses four ontologies. The global ontology lists the classes and properties in which the global database will be represented, as well as annotations. The federation ontology specifies the source databases and which classes and properties of the global ontology they implement. The mapping ontology relates tables and columns from a source database to basic concepts on the global ontology. The inference ontology maps derived concepts to basic concepts through an ontology alignment file.

The global ontology should be the starting point when designing an ontology based database integration system, as the queries to be issued will refer to this ontology. It should be well anotated and descriptive, and should comprise the high-level concepts that will be queried as well as the ones actually on the source databases. Those are called *base concepts*, because they are directly related to a database object. The others are called *derived concepts* and should be related to base concepts by rules on the inference ontology.

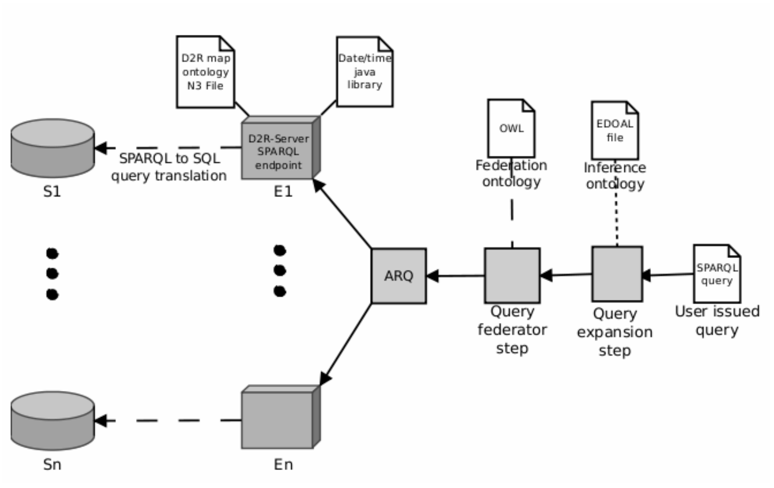


Figure 3. Ontocloud system architecture.

Each relational source database is required to have its own mapping file, which will translate access to the partial RDF graph of the global database $\mathcal{G}$ to SQL queries on the actual database objects.

The federation ontology lists all source databases and which concepts from the global ontology they provide. It is used by the federator step to translate a query $q_{\mathcal{G}}$ to a query $q_{\mathcal{E}}$ over the source endpoints.

Those characteristics make Ontocloud an adequate solution to integrating clinical databases, as stated in the introduction: (1) Integrated sources are independent, so adding, modifying or removing sources does not interfere with other sources; (2) The usage of ontologies allows for annotation of concepts, making it easier for a non-technically trained user to understand it; (3) Data is accessed directly from the sources, yielding always up-to-date results; (4) Ontologies provides tools for dealing with semantic mismatch; and (5) Inference of higher level concepts based on raw data, making all assumptions about data explicit and easy to audit.

4.1. Implementation

Ontocloud implementation was based on open standards and open source software. Its implementation is described in this section (illustrated in Figure 3). To map source databases as a SPARQL endpoint, we used D2R-server[Bizer and Seaborne 2004] with custom mapping N3 files. The query execution engine was ARQ, and custom software was implemented to perform the query expansion (to accomplish inference) and query federator (to indicate what are the databases to be looked into) steps.

D2RQ [Bizer and Seaborne 2004] is an OBDA[5] open source software. It is a Jena library that translates access to an RDF ontology specification by means of SQL queries, according to a mapping file. It includes D2R-Server, a server that provides a SPARQL endpoint over the mapped database, and dump-rdf[6], that converts the entire mapped database to a RDF file. Jena is a “Java framework for semantic web applica-

[5]Ontology Based Data Access
[6]http://d2rq.org/dump-rdf

tions”[7], providing an API for handling RDF, OWL, inference, triple storage and a query engine. JDBC[8] is a Java library that provides an unified API to access several different databases. D2R-Server did not provide any function for date and time operations, so we wrote custom Java classes and used it in SPARQL queries.

The Query Expansion Step used the inference ontology to translate queries using derived concepts into base concepts. We used the Mediation[9] library, which translates queries on an ontology A to an ontology B by means of an EDOAL [David et al. 2011] ontology alignment file. However, instead of mapping between two different ontologies, we mapped between concepts of the same ontology, avoiding circular references.

The Query Federator Step used the federation ontology to translate a query over the global ontology to the source endpoints. For each triple specified in the SPARQL query, it checks in which sources the concepts involved are present, and surrounds the triple with a SERVICE clause. If a concept is present in more than one source, it replaces the triple with a UNION of all SERVICE clauses. The software was written in Java using Jena library.

4.2. Use case

We selected as use case the problem of integrating clinical documents metadata from four information systems used at A.C. Camargo Cancer Center: **EHR**, which contains most data from clinic services; **Pathology**, that contains reports from anatomic pathology tests (visual inspection of sample tissues); **Image**, that contains reports from imaging tests; and **Prescriptions**, that contains both inpatient evolution (texts describing the patient’s day-to-day evolution) and prescriptions of drugs and procedures.

We retrospectively consulted the Medical Informatics Laboratory ticket system, in which all query request made by doctors, managers and researchers are registered. Based on it, we compiled 17 queries of varying complexity to benchmark our integration system [10]. The Ethics Committee of A. C. Camargo Cancer Center, where this research was conducted, granted a waiver on informed consent. To answer those queries, we designed the global schema layout as depicted on Figure 4 and created the mappings accordingly.

We looked into the source databases for tables and columns that contained the needed information required by the defined global schema. Most databases contained all fields needed for the desired integration, except for the type of document on Pathology, Image and Prescription databases and the brazilian person registry number (CPF) for physicians on the Prescription database. We inquired physicians and discovered that documents on Pathology and Image databases are always reports and the documents on Prescription database can be a evolution or a prescription, depending whether a field is blank or not; the CPF number could be found for physicians which were linked to another database table, but not all of them (in this case, we simply created a new record without the CPF number).

To account for missing data, we created simple rules of inference based on knowledge provided by physicians. For Pathology and Image, all documents were stated to have

[7]http://jena.apache.org/

[8]http://www.oracle.com/technetwork/java/overview-141217.html

[9]https://github.com/correndo/mediation

[10]The queries are available at http://diogopatrao.com/ob/ as Supplementary Table 1.

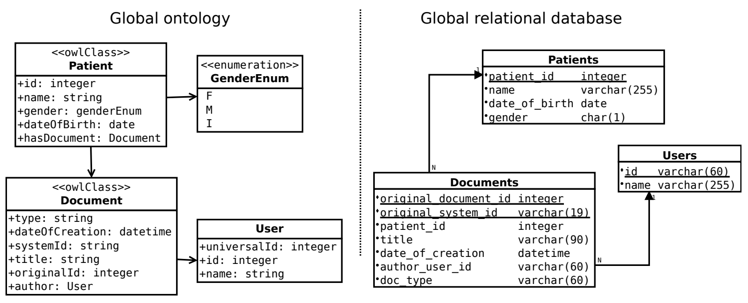


Figure 4. Global schema of our use case, for ontology and relational database integration architectures.

"PATHOLOGY REPORT" and "IMAGING EXAM REPORT" type. For Prescription, a conditional rule (based on whether a text field has data or not) was used to determine if a document belonged to "PRESCRIPTION" or "INPATIENT EVOLUTION" type. These rules were embedded on the mapping files.

It was possible to infer patient class based on the presence of certain types of documents on the patient's EHR; we implemented inference rules on the query expansion step using EDOAL ontology alignment file format. Examples of those rules can be found on Suplementary Table 2[11].

We replicated the original databases, by retrieving pertinent tables and columns and storing them into a single MySQL server. To extract the original sources into the MySQL database we used Pentaho Data Integration Community Edition[Golfarelli 2009]. It is a software suite to design and perform ETL (Extract, Transform, Load - a static database integration method). It allows one to graphically design scripts to extract data from several types of database, transform, mix and store them in a different database table or file.

5. Experimental setup

In order to assess performance and accuracy of Ontocloud we have set up three other database integration systems, which exhausts all combinations of the main database integration architecture characteristics: dynamic or static data acess, and relational database or ontology data representation. We evaluated accuracy in a qualitative way, by making sure that all 17 queries yielded equivalent results on all database integration systems evaluated. Query performance was evaluated as the total clock time a query took for completion on a integration system.

5.1. Source to global mapping

After replicating the source databases, we proceeded to set up all four database integration systems. The Supplementary Figure 1 depicts the experimental setup, and Supplementary Table 3 the database size and extraction times.

The tools used to set up the other integration systems are as follows:

[11]http://diogopatrao.com/ob/

- **Triplestore:** Openlink Virtuoso Universal Server Open Source Edition provides, among many other things, an RDF triple store and a SPARQL endpoint. We chose Virtuoso to implement Triplestore, the static access ontology based integration method. For each of the four source databases, we used D2R to dump data into an N3 file. Those files were imported using Virtuoso Bulk Loader[12].
- **Federation:** Teiid[13] is an open-source, dynamic relational database integrator system; it allows the creation of views over database resources published on a JBoss[14] server, and it is accessible as a JDBC resource. Federation, the dynamic database integration architecture, was designed as a Teiid instance. For each table in the global schema, we wrote a consolidated view, composed of queries over each source database joined by UNION clauses. Those queries did all necessary mapping to provide the required information, even if it was spread in different tables on the source database. The missing document type of Pathology, Image and Prescription databases was inferred directly on the view statement as a SQL constant or expression.
- **Replication:** The Replication architecture was created by materializing the Federation queries (translated to MySQL dialect) into tables. As in the source databases, every column in each database was indexed.

5.2. Experiments

The 17 queries were transcribed to each integration system language (SPARQL for ontology based systems and SQL for the others) and dialect (function names and namespaces were slightly different between MySQL and Teiid, and between Virtuoso and ARQ). There is no SERVICE specific optimizations on Jena, and we have not implemented it for Ontocloud. In contrast, Teiid, the software we chose for implementing Federation, was highly optimized for this type of queries. To account for this difference, we implemented two sets of queries for Ontocloud: one using both query expansion and query federator step, and other querying directly the sources with queries tuned by hand. This way, we get the actual running time for current software and an estimation of what the timing would be if there was an optimization step. We ran one single round of all 17 queries in all systems, without time limit and saving the results. To avoid server resource competition, only one one query on a single integration system was executed at a given time.

The computer server in which the experimental setup was created and tests were performed had 4 cores with 3.00GHz, 64bits, and 8GB of RAM, running CentOS 5. The database software installed was MySQL server version 5.0.95. We also used Pentaho Data Integration Community Edition version 4.0.1, ARQ-2.8.8, PHP 5.2.5, Virtuoso Open Source Edition 6.1.4.3127, D2R-Server 0.8, Java 1.6.0.23, Teiid 7.7 and JBoss 5.1.0 GA.

6. Results

A functional comparison between all systems can be seen on Table 1. All four integration systems were successfully configured and deployed. Except for queries 14 and 17 on Ontocloud Optimized, and queries 10-17 on Ontocloud Unoptimized, which were not

[12]http://www.openlinksw.com/dataspace/dav/wiki/Main/VirtBulkRDFLoader
[13]http://www.jboss.org/teiid/
[14]http://www.jboss.org/

completed due to lack of memory, all other queries on all evaluated systems completed successfully and yielded the same results. Ontocloud Optimized performed better than Federation on 7 queries out of 17, and was 15% faster than Ontoclound Raw (without optimizations). Replication was the fastest method of all, followed by Triplestore which performed better than Federation and Ontocloud on 13 queries. Time measurements for all database integration systems can be seen on Figure 5 and Supplementary Table 4.

Integration system	Data access strategy	Data heterogeneity solving method	Missing data	Annotation	Query expansion
Ontocloud	Dynamic	By ontology	Mapping	Yes	Yes
Federation	Dynamic	Least detailed	Mapping	No	No
Triplestore	Static	By ontology	Materialized	Yes	No
Replication	Static	Least detailed	Materialized	No	No

Table 1. Data integration architecture features.

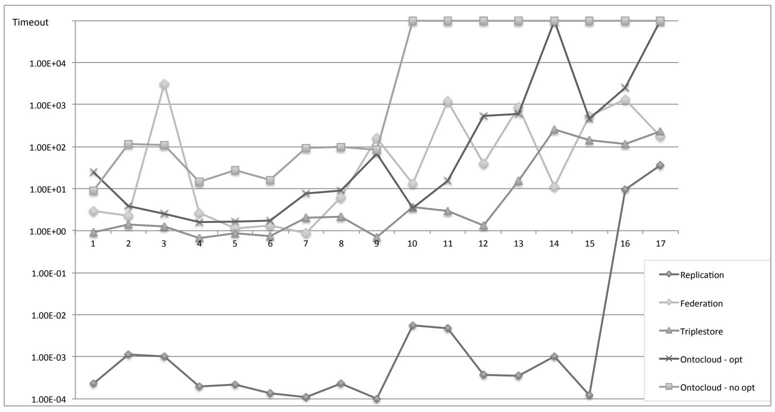


Figure 5. Time that each method took for running the 17 queries. The vertical axis unit is log_{10} seconds, and in the horizontal axis we display the query number. Query failures were plotted on the "Timeout" line, above all other measures.

7. Discussion

The implementation of Ontocloud and the use case experiments showed that it is an adequate database integration system for clinical data, as it accomplish the five objectives: (1) The configuration of source databases was completely independent, except for the Federation Ontology, which lists the URL of each endpoint and the concepts each implements; (2) The global ontology contained human-readable descriptions, so data would be easily understandable by non-technical personnel; (3) Data is accessed directly from the sources, yielding always up-to-date results; (4) Mappings provideded missing data in a way that is transparent to the end user and (5) Higher level concepts like TratedPatient and InPatient are easily understood by physicians and managers, while being translated by the query expansion step to its definition on raw data, allowing the query to be performed.

As we set up the integration systems, fundamental differences between Federation and Ontocloud arised. Federation requires that the developer explicitly join all sources in a single database view. That makes adding a new source to it a difficult and risky task, as it is required to work on a SQL statement that involves several different source databases and any mistake may compromise the whole integration system. Each Ontocloud source is configured without the need to take other sources in consideration. Instead, it relies on the Query Federator step, which adds to the original query clauses indicating in which source endpoint each triple will be resolved. Therefore, by keeping the mapping files separated, Ontocloud facilitates the maintenance of source databases.

Inference on Ontocloud was based on the Mediation library, which allowed us to implement rules by expanding each query term. The inference rules are detached from the database integration itself, and can be maintened independently of the sources. Also, as those rules are represented on an ontology language, it is more suitable for domain experts to maintain it than on the relational methods, in which rules should be implemented on SQL language. It also improves the information management of such a system, as it keeps the raw data (on the mapping ontologies) apart from the higher level concepts (on the inference ontology).

Ontocloud performance suffered on queries with aggregation or that dealt with date operations. This occurs because SPARQL aggregation keywords and date manipulation functions are not translated directly to SQL, instead all results are retrieved and transformations are performed in memory. That both hindered performance and required a lot of memory. Also the queries generated by Query Federator step contained a lot of SERVICE keywords, each containing only one triple. An important optimization would be to join triples on the same SERVICE pattern, minimizing the access to source endpoints. Also, the order of triples and filters on the SPARQL query is crucial to determine the performance. Those optimizations are beyond the scope of this work, but would certainly put Ontocloud on a par with the other methods. For the purpose stated in this work, the speed of Ontocloud seems a fair tradeoff for the ability of yielding up-to-date results at any time and performing inference.

8. Conclusion

We have successfully designed and implemented Ontocloud to perform ontology-based database integration. It implements important features in an clinical data integration system: The sources are loosely coupled, favoring distributed and dynamic management of sources; uses ontologies to integrate data, which is prone to reuse and more human readable; has dynamic access to sources, always yielding up-to-date results; and allows inference. We believe that this system architecture can be extended and improved, as indicated in the discussion, to become a production level tool very useful in the medical informatics context.

References

Bhogal, J., Macfarlane, A., and Smith, P. (2007). A review of ontology based query expansion. *Information Processing & Management*, 43(4):866–886.

Bizer, C. and Seaborne, A. (2004). D2RQ-treating non-RDF databases as virtual RDF graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2011). The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., and Rosati, R. (2007). Mastro-i: Efficient integration of relational data through dl ontologies. In *Proc. of the 20th Int. Workshop on Description Logics (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/*, pages 227–234.

Chard, K., Russell, M., Lussier, Y. A., Mendonça, E. A., and Silverstein, J. C. (2011). A cloud-based approach to medical NLP. *AMIA - Annual Symposium proceedings / AMIA Symposium*, 2011:207–16.

Cruz, I. F. and Xiao, H. (2005). The role of ontologies in data integration. *Journal of engineering intelligent systems*, 13(4):854–863.

Cure, O. and Bensaid, J.-D. (2008). Integration of relational databases into OWL knowledge bases: demonstration of the DBOM system. In *2008 IEEE 24th International Conference on Data Engineering Workshop*, pages 230–233. IEEE.

David, J., Euzenat, J., Scharffe, F., and dos Santos, C. T. (2011). The alignment API 4.0. *Semantic web*.

Golfarelli, M. (2009). Open Source BI Platforms: A Functional and Architectural Comparison. In Pedersen, T., Mohania, M., and Tjoa, A., editors, *Data Warehousing and Knowledge Discovery*, volume 5691 of *Lecture Notes in Computer Science*, pages 287–297. Springer-Verlag.

Haas, L. M., Lin, E. T., and Roth, M. A. (2002). Data integration through database federation. *IBM Systems Journal*, 41(4):578–596.

Halevy, A. Y. (2001). Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294.

Hull, R. (1997). Managing semantic heterogeneity in databases. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '97*, pages 51–61, New York, New York, USA. ACM Press.

Iller, P. E. L. M. and Adkarni, P. R. N. (2004). QIS : A Framework for Biomedical Database Federation. *Journal of the American Medical Informatics Association*, 11(6):523–534.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, page 246. ACM.

Min, H., Manion, F. J., Goralczyk, E., Wong, Y.-N., Ross, E., and Beck, J. R. (2009). Integration of prostate cancer clinical data using an ontology. *Journal of biomedical informatics*.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 3rd edition.

Sujansky, W. (2002). Heterogeneous Database Integration in Biomedicine. *Journal of Biomedical Informatics*, 34(2001):285–298.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. (2001). Ontology-Based Integration of Information A Survey of Existing Approaches. In *IJCAI-01 Workshop: Ontologies and Information Sharing*, volume 2001, pages 108–117.