Graph Management to Improve Querying of Health and Social Data

María Constanza Pabón[1,2], Claudia Roncancio[3] and Martha Millán[1]
[1]*Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Cali, Colombia*
[2]*Departamento de Electrónica y Ciencias de la Computación, Universidad Javeriana-Cali, Cali, Colombia*
[3]*University of Grenoble, Grenoble Isere, France*

Keywords: Graph Querying, Data Integration, Data Exploration.

Abstract: Large amount of data related to health care are stored in heterogeneous data sources. Independently, social media provides information about people's environment and activities, such as family relationships or patient's habits and social interaction. This information could be used to complement patients medical profiles to improve patient's care. Providing expert users with mechanisms to integrate and query such sources becomes crucial to retrieve information allowing to improve the analysis of patient's situations. This work contributes to facilitating visualization and querying of data coming from such sources. We adopt a graph data model at the conceptual level as it facilitates the integration of structured and semi-structured data. Our purpose is to go a step forward by providing a conceptual query language intended to allow end users, medical domain experts, to retrieve data from heterogeneous data sources by ad hoc queries. In this paper we introduce a set of operators to query data by transforming a graph and we analyze how they fulfill some design features of the conceptual language. These operators allow successive graph transformation to generate subgraphs with filtered data and to derive new relations representing information that is implicit or that is sparse in the data.

1 INTRODUCTION

The amount of data collected by health information systems has increased exponentially last years. Despite the efforts to provide technologies to integrate and to improve the access to this information, patient's health care data is still distributed in heterogeneous data sources. On the other hand, the amount of available social media data is growing every day. Social media could provide important information about people's environment and activities, such as family relationships or patient's habits and social interaction. This information could be used to complement patients medical profiles, in order to improve patient's care. Therefore, a significant potential benefit could be obtained combining data sources from medical care systems and social media.

In order to facilitate physicians and medical specialists access to aforementioned information, we work on a data integration system and a conceptual query language [1]. We adopt virtual data integration based on mediation. At the mediator level, a global data model provides a view of data available. This view is used to formulate queries. Their evaluation leads to a distributed execution on the data sources and final integration by the mediator. In DIG (*Data Integration using Graphs*), our data integration system, we adopt a graph data model for the global schema. Due to its expressivity and flexibility, graphs could represent structured and semi-structured data and allow the mapping of many other data models. Therefore, the set of data sources may include both medical and social data, easily represented by graphs. Besides, semantic and conceptual models are usually based on graphs. Therefore, the mediator's global graph data model is also the conceptual model underlying the conceptual query language.

However, as data and their relations are complex, using a graph query language to express ad hoc queries may be difficult for end-users and domain experts —ad hoc end-users queries can not be determined prior to the moment a query is expressed. To overcome this difficulty and to facilitate data exploration, this work is a step forward to support a user-friendly conceptual query language, with a graphical interface. To support such a query language we require a set of high-level operators. They facilitate query formulation to users who are domain ex-

[1]This work is supported by the French Government through the ANR Innoserv project.

Pabón M., Roncancio C. and Millán M..
Graph Management to Improve Querying of Health and Social Data.
DOI: 10.5220/0004805403430350
In *Proceedings of the International Conference on Health Informatics* (HEALTHINF-2014), pages 343-350
ISBN: 978-989-758-010-9

perts but who have little or no experience with graph query languages. In this paper we introduce a set of graph transformation operators as part of the conceptual query language. Through successive graph transformations users select filtered data and derive new relations based on paths connecting objects in the graph. These relations make explicit information that is implicit or sparse in the data. The operators are part of the query engine in the DIG system. Some benefits derived from the operators are:

• The operators allow users to manipulate objects maintaining its relationships and attributes, and deriving new relationships between objects connected by undirected paths. In this sense, the operators provide a high level of abstraction.

• We use a conceptual schema and take advantage of it, by, first, bringing users a general vision of the data, thus allowing better use of information, and second, guiding users during data exploration. The use of a data schema is possible since the chosen data model is flexible.

• By the use of the proposed operators the graphical interface can avoid unintuitive notions for end users such as the use of variables, the need of distinguish output variables, and the need to construct query patterns from scratch by a drag-and-drop approach.

• The operators allow incremental query formulation. A feedback about how the result will be like can be provided to the user. This helps preventing the execution of queries that have failed to fulfill the needs of the user.

This paper is structured as follows. Section 2 introduces conceptual query language features and the underlying data model. Section 3 introduces the operators and illustrates their use. Section 4 describes operator implementation DIG system. Related work is discussed in Section 5. Conclusions and future work are highlighted in Section 6.

2 DATA MODEL AND QUERY LANGUAGE FEATURES

Several features are required to provide a conceptual query language intended to allow end users to explore and retrieve data from heterogeneous data sources. First of all, a flexible underlying conceptual data model is required. Second, queries should be expressed on a data schema, navigation should be facilitated and incremental query formulation with feedback at each step should be possible to guide the user. Also, as much as possible, variables and specific notions proper to query languages, should be hidden. These features, intended to ease the query expression, are described in Sections 2.1 and 2.2.

2.1 Graph Data Model

We use the notation and definitions of GDM (Hidders, 2002). This graph data model allows to define independent schema and instance graphs, and the relation between them. GDM also includes composite values and n-ary relations. GDM graphs have simple nodes and edges—they do not include hypernodes, hyperedges, nor attributes attached to the nodes and edges. Nodes represent classes in a schema graph and entities in a instance graph, while edges represent attributes in both graphs. GDM provides three kinds of classes: objects, composite values and basic values.

In our approach, the data schema is essential and plays three roles: first, as a conceptual model that uses concepts easily understandable by users belonging to the application domain; second, as a mean to specify the input of the operators; and third, as a mean to guide users in the query formulation process. Besides, the data schema gives more expressivity and allows independence between the definition of the structure of data and the instance. Also, usually, as the schema is concise, it is easier to visualize than the instance.

The graphical representation of GDM graphs is simple. GDM handles two independent graphs representing the database schema and the database instance, and the relationship between them. Additionally, the definition of the graph includes functions to represent the kind of class (object, composite or basic

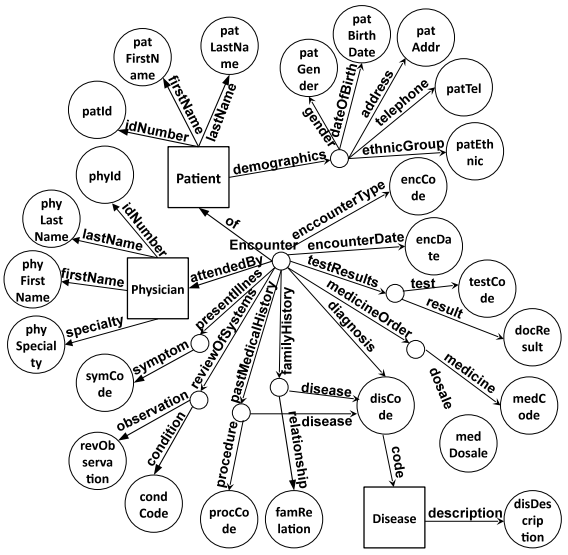


Figure 1: Clinical record basic schema example. Squares represent object nodes, small empty circles represent composite-value nodes, and circles with a basic-type name inside represent basic-value nodes.

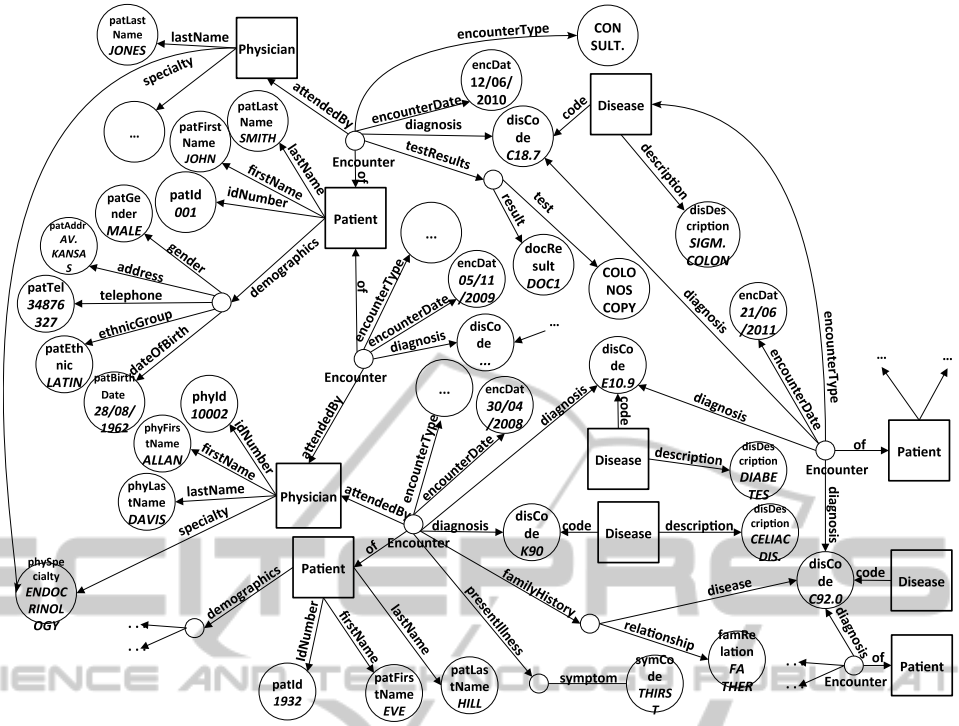


Figure 2: Clinical record basic instance example (for clarity, values are represented as basic-value nodes).

value) and the class of nodes, and the type of edges. Therefore, such metadata do not add nodes nor edges to the graph. Thus, a GDM graph model has less nodes and edges than, for example, a corresponding RDF schema. In this sense, GDM schema graph is easier to navigate and to understand. Besides, the operators handle in a uniform way entities and their attributes, since both abstract objects and basic values are represented with nodes, and neither, nodes nor edges, has attributes in its internal structure. Furthermore, the representation of basic values as nodes lead to connections between instance entities (i.e. patients diagnosed with the same disease are linked to a node representing that disease). The transformation operators allow to take advantage of those connections to find relationships in data.

Figures 1 and 2 illustrate a schema and instance graphs of basic clinical data. A clinical record has the patient's demographic data, such as the date of birth, birth place, race, and gender. Other data related to medical care such as reason of the consultation, description of the illness or diagnoses, are also available.

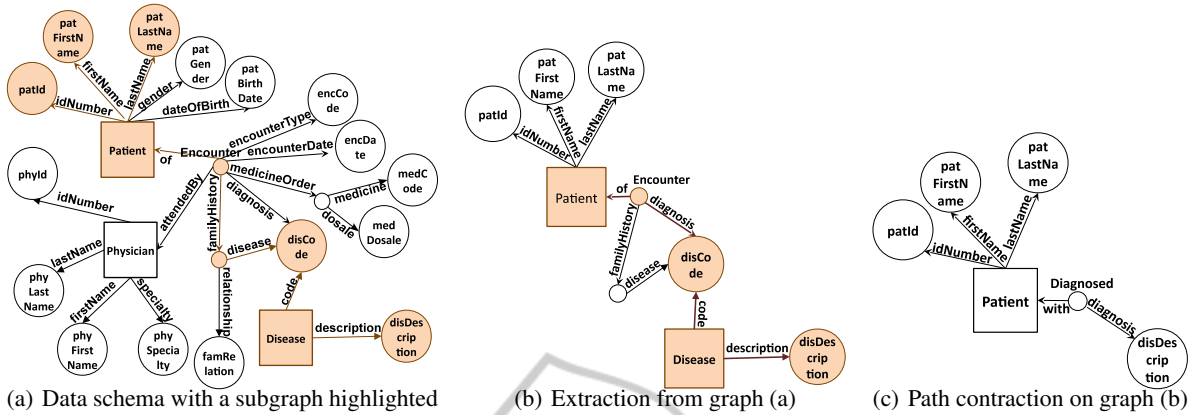
2.2 Graph Schema and Querying

The interface takes advantage of the graph schema to help users in query formulation. The schema graph is displayed and the interface guides users to explore it. This helps finding the nodes that represent classes and attributes required for a query and to apply proper graph transformations. When a transformation is applied, the transformed schema graph is displayed. This is done without the execution of the operator over the data sources. Users can then corroborate the expected query result, and revert or replace the transformations applied at any time during the query formulation process. This prevents executing queries that have failed to fulfill the needs of the user. Thus, based on a graphical interface, the schema graph could be transformed by the user allowing him to select interest entities, as shown in Figure 3.

The visualization of a schema graph, together with the user's knowledge about the domain, allows users to determine, in a natural way, the information to access.

The input of the operators is selected directly over the schema graph. This avoids dragging (from some list) class and attribute names to built a query pattern.

Most graphical query interfaces are based on the construction of search patterns by drawing (or dragging) nodes and edges (Catarci et al., 2003; Gyssens et al., 1990; Consens and Mendelzon, 1990; San Martín et al., 2011). Some of them offer predefined graph patterns (Bhowmick et al., 2013). Others allow to select in the data graph some interesting patterns and to convert its nodes and edges to variables



(a) Data schema with a subgraph highlighted (b) Extraction from graph (a) (c) Path contraction on graph (b)

Figure 3: Schema transformation example.

or predicates. Therefore, in most cases, users have to construct the entire query from scratch, and they need to imagine how to do it mixing the elements of the language or the graphical interface. In contrast, our approach is schema based, therefore we provide users facilities to navigate on the data schema graph and to select interesting entities by transforming it.

Moreover, graphical interfaces that help users to formulate formal textual language queries (Smart et al., 2008; Groppe et al., 2011) usually map the language elements to a graphical representation. Thus the interface depends on the language structure. Therefore, even graphical, the query formulation relies on the language structure. In contrast, our operators are based on the user interaction with the schema graph, by navigating and transforming it.

3 GRAPH TRANSFORMATION OPERATORS

In this section we describe the main graph transformation operators, we provide examples and highlight some of their characteristics.

3.1 Description of the Operators

The operators are high-level, in the sense that they allow to perform transformations that in current proposals would require the execution of several operations. Each of them takes as input a schema and instance directed graphs, a set of interest nodes or edges and other parameters, and returns as output schema and instance directed graphs.

The **subgraph extraction** operator allows selecting all data that corresponds to a given set of schema edges. Given the schema graph shown in Figure 3(a), the subgraph extraction operator can be used to retrieve the subgraph that includes patients' id, first and last names, the diagnoses they have received and the diseases in their family history. See Figure 3(b).

The **value filtering** operator allows selecting the basic-value nodes of some interest classes. These nodes are linked to other basic-value nodes that fulfill a given condition. The value filtering operator returns the selected nodes and the paths between them. It returns transformed schema and instance graphs. This operator can be used to find, for example, the id and names of patients linked by one or several specified path patterns to a hepatitis B node. These patterns could be diagnoses or diseases in their family history.

The **class filtering** operator includes two steps. First, it selects objects that belong to one interest class and fulfill a given condition. And second, it retrieves all data linked to those selected objects by an undirected path that corresponds with a schema path. This includes not only the selected objects' attributes, but also their relationships with other objects or composite values and their attributes, provided that they are covered by an undirected path of the schema. Class filtering does not change the schema graph but selects data associated with subsets of objects of the interest class. It is useful to retrieve, for example, information of female patients born in 2005 who have been diagnosed with diabetes. In this case, the interest class is patient, the operator retrieves each patient that fulfills the given condition. Then, retrieves the data linked to those patients by some undirected path that corresponds with a schema path. In the schema given in Figure 3(a) each patient selected will have the patient's attributes (idNumber, firstName, etc.) plus encounters, physician and diseases related to the patient, and their attributes.

The **path contraction** operator changes both the schema and the instance, to generate graphs that make explicit relations that are implicit in the original graph. The path contraction operator replaces in-

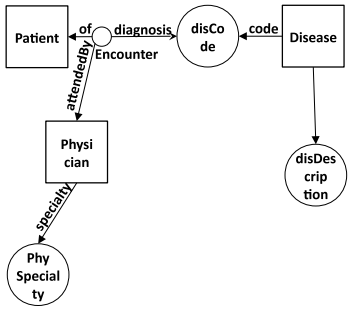


Figure 4: Subgraph extraction resulting schema.

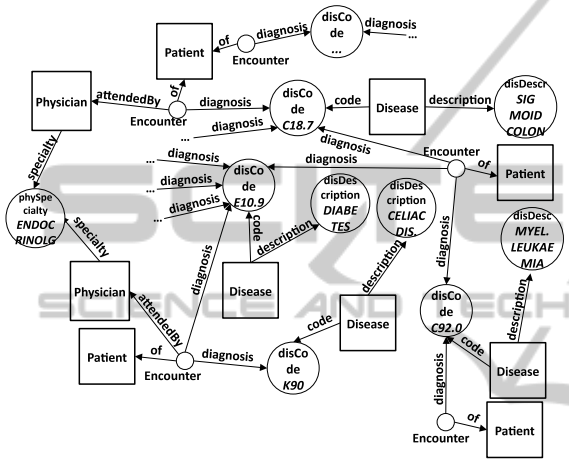


Figure 5: Subgraph extraction resulting instance.

ner nodes and edges of a given schema's path with a new node connected to the path's initial and terminal nodes. It is useful, for example, to establish a more direct relationship between patient and disease description nodes, so an user can easily see diagnoses given to patients. See Figures 3(b) and 3(c).

Finally, to accomplish specific domain query requirements, specific transformations be defined combining several operations. As an example, we define the **co-occurrence** operator. Given two classes *A* and *B*, this operator allows to find pairs of objects belonging to class *A* that are related to the same object of class *B*. The relationship is established by a path that corresponds with a given schema undirected path. The co-occurrence operator materializes a relationship inverse to the given path, then it applies contraction of the path plus its inverse. For example, this operator can be used to identify pairs of diseases that appear on the same patient.

3.2 Examples of Queries

Let us consider other examples of queries using successive transformations on the graph. Based on Figure 1, consider the query *Find pairs of diseases diagnosed by endocrinologist on patients previously diagnosed with diabetes*. It can be calculated as follows: First, extract the subgraph including *Patient*, *disDescription* (disease description) and *phySpecialty* (physician specialty) classes, and the path between them that represents a diagnosis (see Figures 4 and 5). Second, apply a class filter on the subgraph to obtain patients having a "*Diabetes*" diagnosis. The schema is maintained, and Figure 6 shows an example of the answer instance graph. Third, apply a class filter on the previous result to obtain all data linked to physicians with a "endocrinology" specialty. Again, the schema is maintained. Finally, apply a co-occurrence operator to create a direct relationship between pairs of diseases diagnosed to the same patient. The schema and instance result are shown in Figure 7.

Enterprise social media may provide information about employees, their contacts, activities, interest, and participations in work teams, as shown in Figure 8(a). When such a data source is integrated in the mediation system, the global schema includes part of its information (Figure 8(b)). Consider the case where we need ID and last name of patients who work at the same building (same address) and have been diagnosed with "whooping cough" since October, 2013. Based on the given schema, a value filter could be used to obtain these data from patients related to an encounter that took place after "2013/10/01", along with the diagnoses given in those encounters. Then, a

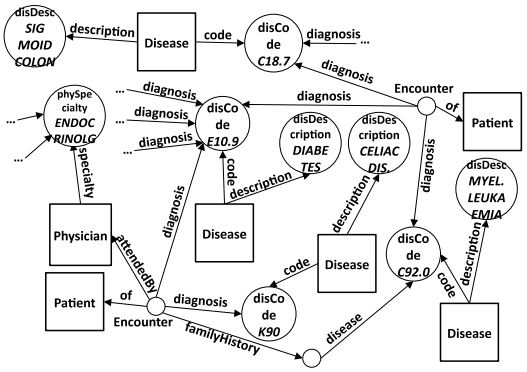
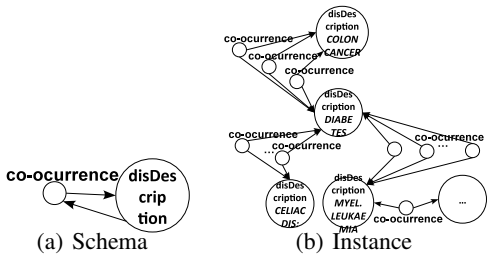
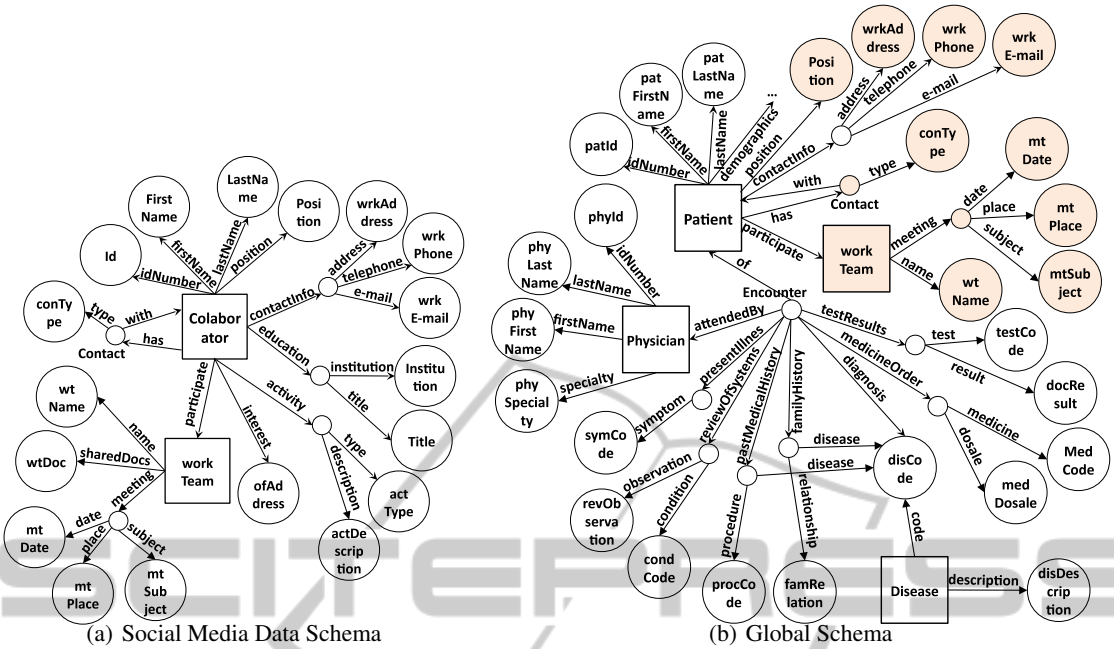


Figure 6: Class filtering result.



(a) Schema (b) Instance

Figure 7: Co-occurrence result.



(a) Social Media Data Schema (b) Global Schema

Figure 8: Schema graph including a social media source.

second value filter is used to extract same data from patients with a "whooping cough" diagnosis. Finally, a path contraction operator allows to show a direct relationship between patients and their work address. The answer includes several subgraphs like the one shown in Figure 9.

3.3 Operator Characteristics

In order to accomplish the conceptual language design features, the proposed graph transformation operators have the following characteristics.

First, the input for the operators are specified by the nodes and edges in the graph. This allows to hide the notion of variable, since users manipulate only objects, its attributes and relationships. This also allows to avoid the construction of query patterns from scratch.

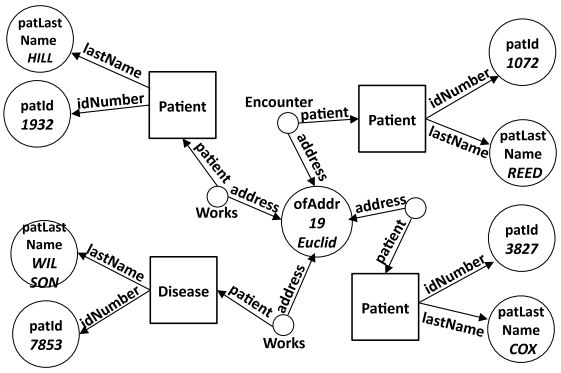


Figure 9: Query result.

Second, the output schema is defined according to the semantics of each operator. Then, the input of the operators does not require the specification of the structure (or pattern) of the output graph. Besides, our operators graphical interface may visualize the schema result before the execution of the query, in order to allow users to better understand the result. The operators are compositional since both the input and the output are graphs.

Third, the operators mix the concept of subgraph and pattern matching. Subgraphs allow to select objects with incomplete information, while pattern matching is used to identify objects that exhibit specific characteristics.

Fourth, languages that offer direct transformations of the graph (Gyssens et al., 1990; Hidders and Paredaens, 1993) usually provide basic operations for adding and deleting nodes and edges, and abstraction (abstraction allows grouping nodes with common characteristics (Gyssens et al., 1990)). In contrast, our operators encapsulate several basic operations. This characteristic facilitates the graphical interface functionality with respect to the incremental query construction and the capacity to go forward and backward during the query formulation. Besides, this characteristic opens optimization opportunities.

Fifth, the operators could be executed generating the corresponding queries in other languages as SPARQL (Harris and Seaborne, 2013) or GraphLog (Consens and Mendelzon, 1990), or can be imple-

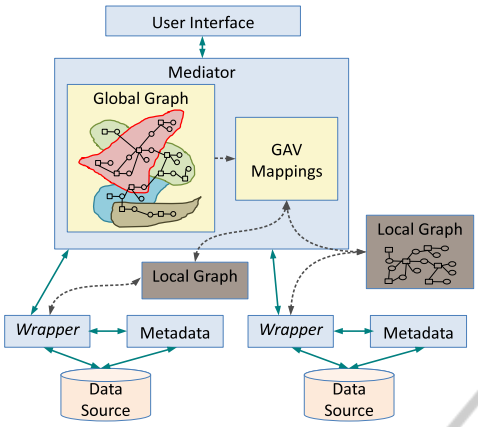


Figure 10: DIG System Architecture.

mented directly in a query engine.

4 DIG SYSTEM

We implemented DIG, Data Integration using Graphs, as a proof of concept for our proposal. DIG follows a mediation approach (Figure 10). The mediator uses a graph data model to represent the data available in several heterogeneous data sources. Each data source has its own data model and query language. In DIG their database schema is also represented by a local graph. DIG uses a GAV (Global As a View) approach to map the global graph with the local ones. The user interface captures the queries and express them by means of the proposed operators. The mediator generates subqueries in SPARQL (Harris and Seaborne, 2013), which are sent to each data source for execution. Data sources which cannot be accessed by a SPARQL query require a wrapper.

A set of metadata associated to wrappers and data sources is a distinguish component of DIG. The inclusion of metadata responds to a feature frequently presented in patients medical data in which is common to find unstructured data. As examples, we find free description texts referring the patient condition, or medical images. In this case, metadata is composed of semantic descriptors (i.e. concepts of one or several ontologies) associated to unstructured attributes.

The DIG prototype has been implemented in Java. The mediator uses Neo4j [2] to manage the graph database schema. A wrapper to access a DICOM [3] repository has also been developed. Metadata is managed in RDF repositories. In particular, DICOM images are annotated with SMITag (López et al., 2012). SMITag allows to select regions of interest (ROI) in the images and annotate them with concepts belonging to multiple ontologies (i.e. ICD-10, RadLex, FMA, and ontologies in the Bioportal repository). These annotations are stored in a Jena triple store. Access to relational databases will be done by the use of a SPARQL endpoint, such as DARQ (Quilitz and Leser, 2008) or fedX (Schwarte et al., 2011).

[2]http://www.neo4j.org/

[3]http://medical.nema.org/standard.html

5 RELATED WORK

In order to facilitate the access to graph based data, several query languages have been proposed (Gyssens et al., 1990; Consens and Mendelzon, 1990; Blau et al., 2002; Chau et al., 2008; San Martín et al., 2011). Also, since RDF and OWL models are naturally represented with graphs, we consider visual query languages and graphical interfaces proposed to retrieve RDF or OWL data, (Catarci et al., 2003; Smart et al., 2008; Groppe et al., 2011). All of them use graph patterns to specify the query and their graphical interface —when it is described— ask users to construct the pattern. Among its distinctive characteristics, we highlight the ones more related to our work: GOOD (Gyssens et al., 1990) makes use of the database schema to define the graph patterns and propose five operators to transform a graph. In GraphLog (Consens and Mendelzon, 1990), NITELIGHT (Smart et al., 2008) and Gruff [4] the nodes are labeled with variables, besides in GraphLog (Consens and Mendelzon, 1990) the edges are labeled with regular expressions. Both notions, variables and regular expressions, are unintuitive. An SNQL (San Martín et al., 2011) query is defined by two patterns, an extraction and a construction pattern. The last one defines the output graph. NITELIGHT (Smart et al., 2008) proposes graphical notation to represent the elements of a triple pattern. Most of these proposals allow to place boolean conditions on the attribute values and global constraints in the edges. Users have to mark out in some way the output variables.

Regarding graph transformations, SPARQL (Harris and Seaborne, 2013), GraphLog (Consens and Mendelzon, 1990), the language for GOOD (Gyssens et al., 1990) and SNQL (San Martín et al., 2011) are languages that allow to have a graph as query output. However, only GMOD and GOOD are based on direct graph transformations, in the sense that they do not require the specification of two patterns, one for selection and the other for output construction. GOOD has operators for abstraction and for the addition and

[4]http://www.franz.com/agraph/gruff/

deletion of nodes and edges. Since these are basic operators, a query is expressed by long and complex patterns. GMOD provides a unique operation that includes three patterns: selection, addition, and deletion patterns. Addition patterns modify the database schema. In contrast, we propose operations that encapsulate several basic operations (addition of nodes and edges), offering a higher level of abstraction.

6 CONCLUSIONS AND FUTURE RESEARCH

This paper reports work to improve data exploration in heterogeneous data sources. We consider that join querying of health related data and social information can be helpful in understanding patient situations. Considering the characteristics of such sources, we use a conceptual data model, GDM, a graph data model, at the mediator level, and propose a set of operators to query data by transforming the graph. The operators are proposed to support a conceptual query language (with a graphical interface) intended to allow end users, medical domain experts, to retrieve data from the heterogeneous sources.

We followed a design approach in which operators emerge from the desired interface features. We exploit a global graph schema to help users in expressing and incrementally refine their queries. We introduced high-level graph transformation operators which allow expressive querying.

A first version of the DIG system uses Neo4J at the mediation level. The mediator process implements subgraph extraction and value filter operators and generates subqueries in SPARQL (Harris and Seaborne, 2013). Class filter and path contraction operators do not generate subqueries, because their processing will be based in a subgraph extraction results. The completion of the prototype and performance evaluation is future work.

Research perspectives mainly concern optimization and visualization issues. Optimization of the distributed execution plan for graph exploration and, visualization to provide adequate representation of queries and data graphs to be well accepted by end-users.

REFERENCES

Bhowmick, S. S., Choi, B., and Zhou, S. (2013). VOGUE: Towards A Visual Interaction-aware Graph Query Processing Framework. In *CIDR*. www.cidrdb.org.

Blau, H., Immerman, N., and Jensen, D. (2002). A Visual Language for Querying and Updating Graphs. Technical report, University of Massachusetts, Amherst.

Catarci, T., Di Mascio, T., Franconi, E., Santucci, G., and Tessaris, S. (2003). An Ontology Based Visual Tool for Query Formulation Support. In *LNCS*, volume 2889, pages 32–33. Springer.

Chau, D. H., Faloutsos, C., Tong, H., Hong, J. I., Gallagher, B., and Eliassi-Rad, T. (2008). GRAPHITE: A Visual Query System for Large Graphs. In *Intl. Conf. on Data Mining Workshops*, pages 963–966. IEEE.

Consens, M. P. and Mendelzon, A. O. (1990). GraphLog: A Visual Formalism for Real Life Recursion. In *Proc. of the 9th Symposium on Principles of Database Systems*, pages 404–416, New York, USA. ACM.

Groppe, J., Groppe, S., and Schleifer, A. (2011). Visual Query System or Analyzing Social Semantic Web. In *Proc. of the 20th Intl. Conf. Companion on World Wide Web*, pages 217–220, New York,USA. ACM.

Gyssens, M., Paredaens, J., and Gucht, D. V. (1990). A Graph-oriented Object Model for Database End-user Interfaces. *SIGMOD Record*, 19(2):24–33.

Harris, S. and Seaborne, A. (2013). *SPARQL 1.1 Query Language*. W3C Recommendation.

Hidders, J. (2002). Typing Graph-Manipulation Operations. In *Proc. of the 9th Intl. Conf. on Database Theory*, ICDT, pages 394–409, London, UK. Springer-Verlag.

Hidders, J. and Paredaens, J. (1993). GOAL, A Graph-based Object and Association Language. *CISM - Advances in Database Systems*, pages 247–265.

López, F., Ceballos, O., and Díaz, N. (2012). SMITAG: Red Social para la Anotación Semántica de Imágenes Médicas. In *XXXVIII Latin American Conf. on Informatics CLEI*, Colombia.

Quilitz, B. and Leser, U. (2008). Querying Distributed RDF Data Sources with SPARQL. In *ESWC'08*, pages 524–538. Springer-Verlag.

San Martín, M., Gutierrez, C., and Wood, P. T. (2011). SNQL: Social Networks Query Language. Technical report, Universidad de Chile.

Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M. (2011). FedX: A Federation Layer for Distributed Query Processing on Linked Open Data. In *Extended Semantic Web Conf. ESWC*.

Smart, P. R., Russell, A., Braines, D., Kalfoglou, Y., Bao, J., and Shadbolt, N. R. (2008). A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. In *Proc. of the 16th Intl. Conf. on Knowledge Engineering: Practice and Patterns*, volume 5268 of *LNCS*, pages 275–291. Springer.