Querying RDF Data from a Graph Database Perspective

Renzo Angles and Claudio Gutierrez

Department of Computer Science, Universidad de Chile
{rangles, cgutierr}@dcc.uchile.cl

Abstract. This paper studies the RDF model from a database perspective. From this point of view it is compared with other database models, particularly with *graph database models*, which are very close in motivations and use cases to RDF. We concentrate on query languages, analyze current RDF trends, and propose the incorporation to RDF query languages of primitives which are not present today, based on the experience and techniques of graph database research.

1 Introduction

The *Resource Description Framework* (RDF) can be viewed from at least two perspectives: (1) From a logical perspective, as a minimal fragment of logic that includes all relevant features needed as representation language for metadata, or as the W3C recommendation [1] says: RDF is an assertional language intended to be used to express propositions using precise formal vocabularies; and (2) From a database perspective, as an extension of data models used in the database community, in particular graph database models. The former point of view has been an active area of research. This does not come as surprise knowing that RDF emerged as a language to represent metadata on the Web, distilling the experience of the community of knowledge representation and Web researchers and developers [2]. The latter point of view has received less attention and will be the focus of this paper. We will consider RDF as a data model in the database tradition.

The term *data model* has been used in the database community with different meanings and in diverse contexts. In this paper we will use it in two senses. In a broad or abstract sense, a data model is a collection of conceptual tools for describing the real-world entities to be modeled in the database and the relationships among these entities [3]. In a strict or concrete sense, a *data model*, as defined by Codd [4], is as a combination of three components: (a) a collection of data structure types; (b) a collection of transformation operators and query language and, (c) a collection of general integrity rules.

In the broad sense, RDF can be considered a data model: a collection of conceptual tools for describing real-world entities, namely metadata on the Web. But also in the strict sense of the term, RDF qualifies as well: Point (a) has been more or less addressed at a basic level. One of the documents of the RDF suite [5]

A. Gómez-Pérez and J. Euzenat (Eds.): ESWC 2005, LNCS 3532, pp. 346–360, 2005.

speaks of *graph data model* meaning by this concept the data structure implicitly defined by sets of triples. Although one can discuss the precise meaning of this concept [6], the graph-like nature of RDF data is clear. Point (c), namely the integrity constraints, is an open issue for RDF, especially when considering the duality of RDF as an open world specification of distributed resources on the Web versus RDF as data model for large single-source repositories with all the issues of a standard database system. The topic of constraints is outside the scope of this paper.

This paper concentrates on point (b), namely query languages. This is an active area from a development and implementation point of view, and there is a W3C Working Group addressing the issue of RDF data access, which has a proposal of RDF query language directed mainly to access data on distributed sources [7]. There are also works addressing foundational issues, e.g. [8, 9, 10]. Nevertheless, the discussion about RDF as a full fledged strict database model and the design and primitives of a query language for such a model is a topic less developed, and probably one of the most needed if we want to take advantage of all the potentialities of the RDF data model (e.g. query optimization, query rewriting, views, update).

The consideration of RDF as database model puts forward the issue of developing coherently all its database features. In particular its query language should address the kind of queries and problems of the application domains the abstract data model is intended to represent. One of the motivations of this paper are those application domains where interconnection at large scale and navigation of a network is the main modeling theme. Examples of this are biology [11], police-like applications [12], navigation in bibliographic databases, etc. To give a flavor of the type of problems, consider queries like: "are suspects A and B related?", submitted to a police database, or "what is the Erdős number of author X", submitted to (an RDF version of) DBLP. The first asks for "relevant" paths connecting these resources in the (RDF) police database, and the second asks simply for the length of the shortest path between the nodes representing Erdős and X. Current proposals of RDF query languages [13, 14, 7] do not support[1] queries like these. To address these type of problems, the notions and techniques of *graph databases* can be very valuable. Graph databases are systems designed to support storing and querying information in the form of graphs. They were important, together with object-oriented databases, in the database research of the nineties, and lost part of their appeal after the irruption of semi-structured data models and XML. We claim that graph database models can be a sound support for the design of an RDF database model, particularly for RDF query languages.

Contributions. In this paper we study the RDF model from a database perspective, compare it with other abstract database models, focusing on query languages and graph databases. We restrict in this paper to the logical level,

[1] A language is said to *support* a feature if it provides facilities that make it convenient (reasonable easy, safe and efficient) to use that feature [15].

i.e., avoid –when is possible– physical, implementation and indexing considerations. In particular we:
- Compare the RDF model with classical abstract database models putting particular emphasis in graph database models.
- Study current RDF query languages with respect to their capabilities to support graph-like queries and conclude that they give little or no support for them.
- Survey the notions, techniques and systems developed in the area of graph database query languages, and its applicability to the RDF model.
- Propose primitives for RDF query languages based on the graph database experience.

Outline of the paper. Section 2 compares the RDF model with other database models. Section 3 surveys graph database models and their query languages. Section 4 presents a brief review of current RDF query languages and investigates the support they give for querying graph-like data. Finally, in Section 5 we propose a set of primitives to be incorporated into RDF query languages. Each of them is carefully reviewed against experience of query language development in graph databases.

2 Comparison of RDF with Other Abstract Database Models

Beginning in the seventies numerous data models have been proposed, each of them with their own concepts and terminology. Surveys and taxonomies of data models are as manifold as data models themselves (see e.g. [3, 16, 17], [18]). Several of these data models have features relevant for the RDF model. In this section we compare the RDF model with the most important of them. A summary is presented in Table 1.

Physical Models. They were the first ones to offer the possibility to organize large collections of data. Among the most important ones are the hierarchical [19] and network [20] models. These models lack good abstraction level and are very close to physical implementations. The data-structuring is not flexible and not apt to model non-traditional applications. For our discussion they do not much have relevance.

Relational Data Model. was introduced by Codd [21] to highlight the concept of level of abstraction by introducing a clean separation between physical and logical levels. Due to its simplicity of modeling it gained wide popularity among developers and business applications. It is based on the simple mathematical notion of relation, which together with its associated algebra and logic, made the relational model a primary model for database research. In particular, its standard query and transformation language, SQL, became a paradigmatic language for querying.

Although an RDF specification can be logically viewed as a set of binary relations, the differences with the relational model are manifold. Among the

Table 1. Summary of comparison among different database models. The parameters are: abstraction level, complexity of the data items modeled, degree of connectivity among the data and support to get this information, and finally, flexibility to store different types of data

MODEL	LEVEL	DATA COMPLEX.	CONNECTIVITY	TYPE of DATA
Network	physical	simple	high	homogeneous
Relational	logical	simple	low	homogeneous
Semantic	user	simple/medium	high	homogeneous
Object-O	logical/physical	complex	medium	heterogeneous
XML	logical	medium	medium	heterogeneous
RDF	logical	medium	high	heterogeneous

most relevant ones are: the relational model was directed to simple record-type data with a structure known in advance (airline reservations, accounting, etc.). The schema is fixed and extensibility is a difficult task. Integration of different schemas is not easy nor automatizable. The query language does not support paths, neighborhoods and queries that address connectivity (an exception is transitivity). There are no objects identifiers, but values.

Semantic Models. ([22]) have their origin in the necessity to provide more expressiveness and incorporate a richer set of semantics into the database from the user point of view. They allow database designers to represent objects and their relations in a natural and clear manner (similar to the way the user view an application) by using high-level abstraction concepts such as aggregation, classification and instantiation, sub- and super-classing, attribute inheritance and hierarchies [16]. A well-known example is the entity-relationship model [23]. It has become a basis for the early stages of database design, but due to lack of preciseness cannot replace models like relational or O-O.

For RDF database research, semantic models are relevant because they are based on a graph-like structure which highlights the relations between the entities to be modeled.

Object Oriented Data Models. ([24]) are based on the object-oriented programming paradigm. Their objective is representing data as a collection of objects that are organized in classes and have complex values and methods associated with them. They are intended to model non-conventional database applications consisting of complex objects systems with many semantically interrelated components as in CAD/CAM, computer graphics or information retrieval.

Object-oriented database models have been related to Graph database ones because the explicit or implicit graph structure in their definitions [25], [26], [27]. Nevertheless, there remain important differences rooted in the form that each of them models the world. O-O models view the world as a set of objects having certain state (data) and interacting among them by methods. On the contrary, graph database models, and RDF in particular, model the world as a network

of relations. The emphasis in RDF is on the interconnection of the data, the network of relations among the data and the properties of these relations. The emphasis of O-O is on the objects, their values and methods. However, there are proposals to apply O-O concepts to RDF [28].

Semistructured Data Models. ([29, 30, 31]) are oriented to model semi-structured data. Of all the most visible models in the literature, the semi-structured data model is one of the closest in several points to RDF. Semi-structured models deal with data whose structure is irregular, implicit and partial, and whose schema is usually very large, contained within the data itself, and rapidly evolving [31]. One of the best representative is OEM [32]. It is a model based on objects, which have unique identifiers, and values that can be simple types or object references. There is a natural graph representation: objects are nodes, and values are labeled arcs. The main differences with RDF are: the lightweight inferencing available, the existence of blank nodes, the stronger typing system and the fact that labels are also nodes in RDF.

Another representative is the XML model [33]. There are substantial differences between XML and RDF. First, RDF has a higher abstraction level; in fact RDF is an application of XML to represent metadata. Structurally XML has a ordered-tree-like structure against the graph structure of RDF. At the semantic level, in XML the information about the data is part of the data (in other words XML is self-describing); in contrast, RDF expresses explicitly the information about the data using relations between entities. An important advantage of RDF is its extensibility in both schema and instance level. See [34, 35] for a major comparison of these models.

3 Graph Database Models and Their Query Languages

3.1 Graph Database Models

Graph database models appeared with the objective of modeling information whose logical structure is a graph. In this sense, they are the closest to the RDF model by the data type used. Among the first ones, we have the the Logical Data Model [36, 37] and the Functional Data Model [38], which define an implicit structure of labeled graphs. The Logical data model introduces basic, composition, and collection nodes, all of which can be modeled in RDF. On the other hand, in many semantic and object oriented data models the conceptual representation of data is transparently graph-based. For example O_2 [39] defines basic, tuple-structured, and set-structured types (the first type is similar to an RDF blank node and the remainder two can be modeled as relations in RDF); GOOD [27] is oriented primarily to graphical user interfaces; OEM [32] addresses the information exchange problem, and is oriented to express resources and relations in a standard way (in agreement to the RDF philosophy); GDM [40] defines instances and schema graphs with features similar to RDF (e.g. domain and range of relations, typeOf properties). Models like G-BASE [41], Gram [42],

GraphDB [43] and GRAS [44] propose explicit graph data models.[2]. Besides these models based on graphs, there are other approaches which use as formalization generalizations of the notion of graph, such as hypergraphs (e.g. see GROOVI [25], the hypernode model [45, 46]) and hygraphs (e.g. see Hy+ [47]). Note that strictly speaking, RDF graphs are ordered hypergraphs [6].

3.2 Graph Query Languages

There are several proposals of query languages for models that represent information with a explicit or implicit graph structure. In this context, from now on we assume that a graph database has n nodes and e edges.

Cruz et al. [48] propose the graphical query language G for querying data represented as a labeled graph. It introduces the concept of graphical query, which is based on a pattern graph that use regular expressions to represent recursive queries. G evolved into a more powerful language called G+ [49] where a query graph is the basic building block. Query graph nodes may be labeled with variables and edges labeled with regular expressions. A simple query has two elements, a query graph that specifies the class of patterns to search and a summary graph that represent how to restructure the answer obtained by the query graph.

GraphLog [50] is a query language for hypertext. It presents a extension of G+ by adding negation and unifying the concept of a query graph. A query is now only one graph pattern containing one distinguished edge (which corresponds to the restructured edge of the summary graph in G+). The effect of the query is to find all instances of the pattern that occur in the database graph and for each one of them define a virtual link represented by the distinguished edge.

Gram [42] presents a query algebra where regular expressions over data types are used to select walks (paths) in a graph. It uses a data model where walks are the basic objects. A walk expression is a regular expression without union, whose language contains only alternating sequences of node and edge types, starting and ending with a node type. The query language is based on a hyperwalk algebra with operations closed under the set of hyperwalks.

Gemis and Paredaens [51] present PaMal, a graphical model for describing schemes and instances of object-databases and a graphical data manipulation language based on pattern matching.

Güting [43] proposed an object-oriented data model and query language for graph databases called GraphDB. A database in GraphDB is a collection of object classes divided in: simple classes (simple objects that represent nodes), link classes (links between nodes that represent edges) and path classes (representing several paths in the database). A query consists of several steps. Each step computes operations that specify argument subgraphs in the form of regular ex-

[2] Note that a direct applicability of a graph model to RDF is not possible due to the particular RDF graph property where resources possibly can occur as edge labels as as well as node labels. To solve this problem an intermediate model (e.g bipartite graphs [6]) can be defined.

pressions over link class names that extend or restrict dynamically the database graph.

Lorel [30] is a query language for semistructured data designed for the Object Exchange Model (OEM) [32]. Lorel is a extension of OQL [52], extending its characteristics to handling semistructured data.

Oriented to search the Web, Flesca and Grego [53] show to how use partially ordered languages to define path queries to search databases and present results on their computational complexity. In addition, a query language based on the previous ideas was proposed in [54].

4 Current RDF Query Languages and Their Graph Support

4.1 Brief Overview of RDF Query Languages

Several languages for querying RDF data have been proposed and implemented, some in the lines of traditional database query languages (e.g. SQL, OQL), others based on logic and rule languages. Some of them are: *RQL* [9] is a typed language for querying RDF repositories; *SquishQL*[3] is a SQL-style query language that permits simple graph navigation in RDF sources; *RDQL* [55] is an implementation of *SquishQL*; *RDFQL*[4] is a statement-based query language with a SQL-style to perform queries, inference operations, and construction of views on RDF structured data; *TRIPLE* [56] is a language that allow rule definition, inference and transformation of RDF models; *Notation 3* (N3) [57] provides a text-based syntax for RDF; *Versa*[5] is a graph-based language with some support for rules; *SeRQL* combinescharacteristics of languages like *RQL, RDQL, N-Triple, N3* plus some new features; *RXPath*[6] is a query language based on XPath; Good surveys are [13, 14].

W3C members that conform the RDF DAWG presented a Working Draft in October 2004, which specifies a set of use cases, requirements, and objectives for an RDF query language and data access protocol [58]. *SPARQL* [7] is an RDF query language designed to meet such requirements and design objectives mentioned previously. It defines a query language with a SQL-like style, where a simple query is based on query patterns, and query processing consists of binding of variables to generate pattern solutions (graph pattern matching). *SPARQL* is still a work in progress.

4.2 Graph Properties in Current RDF Query Languages

To illustrate the problems of current RDF query languages in querying graph-like properties, we chose seven query languages and seven graph properties one

[3] `http://ilrt.org/discovery/2001/02/squish/`
[4] `http://www.intellidimension.com/`
[5] `http://4suite.org/`
[6] `http://rx4rdf.liminalzone.org/`

Table 2. Support of some current RDF query languages for some example graph properties ("±" indicates partial support and "×" no support)

PROPERTY	RQL	SeRQL	RDQL	Triple	N3	Versa	RxPath
Adjacent nodes	±	±	±	±	±	±	×
Adjacent edges	±	±	±	±	×	×	×
Degree of a node	±	×	×	×	×	×	×
Path	×	×	×	×	×	×	±
Fixed-length Path	±	±	±	±	±	×	±
Distance between two nodes	×	×	×	×	×	×	×
Diameter	×	×	×	×	×	×	×

would like to retrieve (see [59]). The summary of the results, presented in Table 2, are as follows. An RDF graph can be considered a directed graph. This direction produces problems in languages that do not have a union operator when retrieving neighborhoods, e.g. "all statements involving a given resource". Some query results violate the query language property of closure [14] by returning results which are not in RDF format. There are two main problems concerning paths: (a) most languages support only querying for patterns of paths which are limited in length and form (the issue of edge direction blows up the size of the query exponentially); (b) RxPath is able to retrieve only paths starting from a fixed node and with some other restrictions. Aggregated functions like COUNT, MIN, MAX applied to paths could be used to answer queries as for the degree of a node, the distance between nodes, and the diameter of a graph. None of these functions is systematically supported, even though, for example, the original version of RQL has a COUNT function on the number of triples.

5 Graph Primitives for RDF Query Languages

In this section we present desirable graph primitives of a query language for the RDF data model, based on the experience of the graph database query languages discussed in previous sections. We stress the graph-like features that in our opinion are missing in today's RDF query languages.

Before discussing the primitives in detail, let us enumerate desirables features for an RDF query language. They are very much inspired by a similar wish-list stated by Abiteboul [31] for semi-structured data. They are: Standard database-style query primitives; Navigation in the style of semi-structured data or Web-style browsing; Searching for patterns in an information-retrieval style; Temporal queries, including versioning; Querying both the data and the schema in the same query; Incorporating transparently the lightweight inferencing of RDF Schema and relevant polynomial-time extensions; Sound theoretical foundation;

The following groups of primitives comprise features of graph query languages (see Sec. 3), graph properties presented in section 4.2 and those found in the DAWG Draft. We think they constitute a starting point of graph properties

Table 3. Support of some graph database query languages for the example graph properties of Table 2 ("√" indicates support, "±" partial support, "×" no support, and "?" indicates there is no information available)

PROPERTY	G	G+	GraphLog	Gram	GraphDB	Lorel	F-G
Adjacent nodes	±	√	√	√	±	√	±
Adjacent edges	±	√	√	√	±	√	±
Degree of a node	×	√	√	×	?	×	×
Path	√	√	√	√	√	√	√
Fixed-length Path	√	√	√	√	√	√	√
Distance between two nodes	×	√	√	×	?	×	×
Diameter	×	√	√	×	?	×	×

that should be supported by an RDF query language. In each case we survey the support that graph database languages gives them. As motivation, Table 3 shows the support graph query languages give to the properties in Table 2.

Paths and Connectedness. One of the most fundamental graph problems is to compute reachability information (use case 2.5 in DAWG Draft [58]). In fact, many of the recursive queries that arise in relational databases and, more generally in data with graph structure, are in practice graph traversals characterized by path problems. The importance of such queries is studied in several works [60, 61, 62, 63]. One of the challenges to incorporate such notion into a query language is its computational complexity. Finding simple paths with desired properties in direct graphs is very difficult, and essentially every nontrivial property gives rise to an NP-complete problem [64]. Yannakakis [65] surveyed a set of paths problems relevant to the database area including computing transitive closures, recursive queries and the complexity of path searching. Extension of query languages for solve graph traversal problems are surveyed in [66].

In what follows, we describe the support that the query languages of the database models described in Section 3.1 give to path problems.

A initial implementation of G translate the graphical queries into C-Prolog programs. Simple paths are traversed using certain non-Horn clause constructs available in Prolog. Although, it does not support cycles or finding the shortest path, it was a good approximation to a graph query language.

The evaluation of path queries in G+ is a two-stage process consisting of a depth-first search of the graph database and use of a nondeterministic finite state automaton to control the search. In addition path queries are a subset of the class of linear chain queries and hence can be evaluated rapidly in parallel. The evaluation algorithm can be shown to compute the identity query in $O(e)$ time and the transitive closure in $O(ne)$ time. G+ was implemented in the HyperG system providing primitive operators like depth-first search, shortest path, transitive closure and connected components.

Motivated by the implementation of G+, Mendelzon and Wood [67] studied the problem of finding all pair of nodes connected by a simple path such that

the concatenation of the labels along the path satisfies a regular expression. Although the regular simple path problem is in general NP-complete, the paper presents an algorithm that runs in polynomial time in the size of the graph when some conditions fulfilled: the graph is acyclic, the regular expression is restricted (according to the definition in the paper), or the graph complies with a cycle constraint compatible with the regular expression. The evaluation algorithm uses a deterministic finite automaton to traverse paths in the graph. They also prove the intractability of certain types of simple paths in a particular class of direct graphs and characterize a class of queries about regular simple paths which can be evaluated in polynomial time. The analysis and implementation in this paper, assume that the graph can be entirely stored in main memory.

The expressive power of GraphLog is characterized by establishing the equivalence between GraphLog, stratified linear Datalog (a language of function-free Horn clauses), non deterministic logarithmic space, and transitive closure. The queries expressible in the language are exactly those that can be computed in space logarithm in the size of the database.

To implement graph operations in GraphDB, efficient graph algorithms are used. Shortest path and cycle both were implemented using the A* algorithm. Moreover, nodes, paths and subgraphs are indexed using path classes and index structures like B-Tree and LSD-Tree.

Lorel presents a SQL-style query language that support two types of path expressions, simple path expressions, which allow to obtain the set of objects reachable by following a sequence of labels starting from a named object in the OEM graph and a more powerful syntax for path expressions, called general path expressions based on wildcards and regular expressions. To outperform query execution, the Lore DBMS [68] implements the query language Lore and uses two kinds of indexes, a link (edge) index called Lindex, and a value index called Vindex. A Lindex takes an object identifier and a label, and returns the object identifiers of all parents via the specified label. A Vindex takes a label, operator, and a value, and returns all atomic objects having an incoming edge with the specific label and a value satisfying the specific operator and value. Vindexes and Lindexes are implemented using B+ trees and linear hashing respectively.

In graph databases where the number of nodes is very large (e.g. the Web) it is useful to subdivide the domain of evaluation by selecting subsets of the domain on the base of some criteria. With this objective, Flesca and Greco [53] introduce partially ordered regular languages based on some order on the nodes. Such languages are an extension of regular languages where strings are partially ordered, for example, two strings s_1 and s_2, such that $s_1 > s_2$, denote two paths in the graph with the constraint that the path s_1 should be preferred to the path s_2. In later work [54], they present an algebra for partially ordered relations, an algorithm for the computation of path queries and show that computing an instance of a graph query can be done in polynomial time. Also, they present a SQL-like language that consider general paths and extended regular expressions, and show how extended regular expressions can be used to search the Web. With similar motivations, and in the context of RDF, Anyanwu and Sheth [69] intro-

duced a path operator ρ to address relevant relationships between entities called semantic associations. Semantic associations are represented in a RDF graph as sequences (i.e. edges, paths) between entities or more complex structures of sequences, and a notion of similarity between them is defined. The implementation of the ρ-operator is evaluated on two strategies, first implementing a processing layer in existing RDF data storage technologies and, second the use of a memory resident graph representation of the RDF model along with the use of efficient graph traversal algorithms (e.g. transitive closure and isomorphism of paths).

Pattern Matching. consists in determining if there exists a mapping (or isomorphism) between a graph pattern and a subgraph of a database graph (use cases 2.1, 2.12 and 2.13 in DAWG Draft [58]). Pattern matching deal with two problems, the graph isomorphism problem that has a unknown computational complexity, and the subgraph isomorphism problem which is NP-complete. Pattern matching has attracted a great deal of attention specially on data mining (see [70] for a survey), update [51, 71], querying [48, 72, 50] and visualization [26]. Sasha *et al.* [64] present a survey of pattern-matching based algorithms for fast searching in trees and graphs.

PaMal use graph patterns to describes the part of the database instance that are affected by a operation (addition and deletion of nodes and edges). In the case of GraphDB, the subgraph problem is solved moving the conditions into subsequent graph operations or other database access.

Aggregate Functions. are operations non related to the data model that permit to summarize or operate on the query results (use cases 2.3, 2.4, 2.6, 2.8, 2.10, 2.11, 2.14 and 2.15 in DAWG Draft [58]). Such functions are oriented to deal directly with the structure of the underlying graph, such as the degree of a node, the diameter of the graph (or a set of nodes), the distance between nodes, etc.

With the purpose of performing computations on retrieved subgraphs product of a query operation, G+ defines two types of summary operators: path operators which summarize on the values of the attributes along paths and set operators which summarize on the values of the attributes on a set of paths. The set of such operators include sum, products, maximum and count.

GraphLog becomes more expressive that relational algebra and calculus with aggregates, adding aggregate operators (e.g. MAX, SUM, etc.) and path summarization. The implementation of GraphLog use algorithms discussed in [67].

Gram, consistent with its SQL-like syntax, defines two types of algebraic operations: unary (projection, selection, renaming) and binary (join, concatenation, set operations) which are closed under the set of hyperwalks. PaMal provides a reduce-operation to work with a special group of instances called reduced instances and programming constructs (loop, procedure and program). Finally, GraphDB query language support further operations, e.g. for sorting, grouping, and aggregate functions (e.g. Sum).

Neighborhoods. The notion of neighborhood is relevant for information having a graph-like nature (use case 2.7 in DAWG Draft [58]). In these models, in-

formation (represented by nodes) closed (in the graph) is usually semantically related . The primary notion is *adjacency.* Both node and edge adjacency in RDF are important in various contexts. A more advanced notion of adjacency, like *the k-neighborhood of a node*, is necessary in several contexts. The need of 1-neighborhood retrieval in an RDF Graph is argued in [73] and [74]. In the RDF context, inference of new triples is relevant in Vertex and Edge adjacency queries. To the best of our knowledge, the notion of neighborhood as primitive for query languages has not been studied systematically in the database literature.

6 Conclusions

We considered RDF from the perspective of graph database modeling. We compared it with other database models. We surveyed graph database models and query languages in order to argue the convenience that the RDF community incorporate database experience and technologies into further development of the RDF model and query language design. In concrete, we propose that RDF query language should incorporate graph database query language primitives. Further work includes developing use cases, formalizing requirements and building benchmarks for queries using the graph-like structure of the model.

Acknowledgments. This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Chile. R. Angles was supported by Mecesup project No. UCH0109. C. Gutierrez was partially supported by FONDECYT No. 1030810.

References

1. Hayes, P.: RDF Semantics. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ (2004)
2. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ (1999)
3. Silberschatz, A., Korth, H.F., Sudarshan, S.: Data models. ACM Computing Surveys 28 (1996) 105-108
4. Codd, E.F.: Data Models in Database Management. In: Proc. of the workshop on Data abstraction, databases and conceptual modeling, ACM Press (1980) 112-114
5. Klyne, G., Carroll, J.: Resource Description Framework (RDF) Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (2004)
6. Hayes, J., Gutierrez, C.: Bipartite Graphs as Intermediate Model for RDF. In: Proc. of the 3th ISWC Conference. Number 3298 in LNCS, Springer-Verlag (2004) 47-61
7. Prudhommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/ (2005)
8. Horrocks, I., Tessaris, S.: Querying the Semantic Web: A Formal Approach. In: Proc. of the 13th ISWC. Number 2342 in LNCS, Springer-Verlag (2002) 177-191

9. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proc. of the 11th WWW conference, ACM Press (2002) 592–603
10. Gutierrez, C., Hurtado, C., Mendelzon, O.: Foundations of Semantic Web Databases. In: Proc. of the 23th ACM PODS. (2004)
11. Olken, F.: Tutorial on Graph Data Management for Biology. IEEE Computer Society Bioinformatics Conference (CSB) (2003)
12. Sheth, A., Aleman-Meza, B., Arpinar, I.B., Halaschek-Wiener, C., Ramakrishnan, C., Bertram, C., Warke, Y., Avant, D., Arpinar, F.S., Anyanwu, K., Kochut, K.: Semantic Association Identification and Knowledge Discovery for National Security Applications. Journal of Database Management 16 (2005) 33–53
13. Magkanaraki, A., Karvounarakis, G., Anh, T.T., Christophides, V., Plexousakis, D.: Ontology Storage and Querying. Tech. Report 308, ICS-FORTH - Hellas (2002)
14. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: Proc. of the 3th ISWC conference. Number 3298 in LNCS, Springer-Verlag (2004) 502
15. Stroustrup, B.: What Is Object-Oriented Programming? IEEE Softw. 5 (1988) 10–20
16. Navathe, S.B.: Evolution of data modeling for databases. Communications of the ACM 35 (1992) 112–123
17. Beeri, C.: Data Models and Languages for Databases. In: Proc. of the 2nd ICDT. Volume 326 of LNCS., Springer-Verlag (1988) 19–40
18. Kerschberg, L., Klug, A.C., Tsichritzis, D.: A Taxonomy of Data Models. In: Systems for Large Data Bases, North Holland and IFIP (1976) 43–64
19. Tsichritzis, D.C., Lochovsky, F.H.: Hierarchical Data-Base Management: A Survey. ACM Comput. Surv. 8 (1976) 105–123
20. Taylor, R.W., Frank, R.L.: CODASYL Data-Base Management Systems. ACM Comput. Surv. 8 (1976) 67–103
21. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 26 (1983) 64–69
22. Peckham, J., Maryanski, F.J.: Semantic Data Models. ACM Computing Surveys 20 (1988) 153–189
23. Chen, P.P.: The Entity-relationship Model-toward a Unified View of Data. ACM TODS 1 (1976) 9–36
24. Kim, W.: Object-Oriented Databases: Definition and Research Directions. IEEE TKDE 2 (1990) 327–341
25. Levene, M., Poulovanssilis, A.: An Object-oriented Data Model Formalised through Hypergraphs. DKE 6 (1991) 205–224
26. Andries, M., Gemis, M., Paredaens, J., Thyssens, I., Bussche, J.: Concepts for Graph-Oriented Object Manipulation. In: 3rd EDBT Conference. Volume 580 of LNCS., Springer-Verlag (1992) 21–38
27. Gyssens, M., Paredaens, J., Bussche, J., Gucht, D.: A Graph-Oriented Object Database Model. IEEE TKDE 6 (1994) 572–586
28. Bassiliades, N., Vlahavas, I.P.: R-DEVICE: A Deductive RDF Rule Language. In: Proc. of the 3th RuleML. (2004) 65–80
29. Buneman, P.: Semistructured Data. In: Proc. of the 16th PODS, ACM Press (1997) 117–121
30. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.: The Lorel Query Language for Semistructured Data. Int. Journal on Digital Libraries 1 (1997) 68–88

31. Abiteboul, S.: Querying Semi-Structured Data. In: Proc. of the 6th Int. Conference on Database Theory. Volume 1186 of LNCS., Springer-Verlag (1997) 1–18
32. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object Exchange across Heterogeneous Information Source. In: Proc. of the 11th ICDE, Taipei, Taiwan, IEEE (1995) 251–260
33. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998. (http://www.w3.org/TR/1998/REC-xml-19980210)
34. Gil, Y., Ratnakar, V.: A Comparison of (Semantic) Markup Languages. In: Proc. of the 15th FLAIRS Conference. (2002)
35. Arroyo, S., Ding, Y., Lara, R., Stollberg, M., Fensel, D.: Semantic Web Languages. Strengths and Weakness. In: International Conference in Applied computing. (2004)
36. Kuper, G.M., Vardi, M.Y.: A New Approach to Database Logic. In: Proc. of the 3th ACM PODS, ACM Press (1984) 86–96
37. Kuper, G.M., Vardi, M.Y.: The Logical Data Model. ACM TODS 18 (1993) 379–413
38. Shipman, D.W.: The Functional Data Model and the Data Language DAPLEX. ACM TODS 6 (1981) 140–173
39. Lécluse, C., Richard, P., Vélez, F.: O2, an Object-Oriented Data Model. In: Proc. of the 1988 ACM SIGMOD Intl. Conference on Management of Data, ACM Press (1988) 424–433
40. Hidders, J.: Typing Graph-Manipulation Operations. In: Proc. of the 9th ICDT, Springer-Verlag (2002) 394–409
41. Kunii, H.S.: DBMS with Graph Data Model for Knowledge Handling. In: Proc. of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow, IEEE (1987) 138–142
42. Amann, B., Scholl, M.: Gram: A Graph Data Model and Query Language. In: European Conference on Hypertext Technology, ACM Press (1992) 201–211
43. Güting, R.H.: GraphDB: Modeling and Querying Graphs in Databases. In: Proc. of 20th VLDB Conference, Morgan Kaufmann (1994) 297–308
44. Kiesel, N., Schurr, A., Westfechtel, B.: GRAS: A Graph-Oriented Software Engineering Database System. In: IPSEN Book. (1996) 397–425
45. Levene, M., Poulovassilis, A.: The Hypernode Model and its Associated Query Language. In: Proc. of the 5th Jerusalem IT Conference, IEEE (1990) 520–530
46. Poulovassilis, A., Levene, M.: A Nested-graph Model for the Representation and Manipulation of Complex Objects. ACM Transactions on Information Systems 12 (1994) 35–68
47. Consens, M., Mendelzon, A.: Hy+: A Hygraph-based Query and Visualization System. SIGMOD Rec. 22 (1993) 511–516
48. Cruz, I.F., Mendelzon, A.O.,Wood, P.T.: A Graphical Query Language Supporting Recursion. SIGMOD Rec. 16 (1987) 323–330
49. Balmin, A., Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D., Wang, T.: A System for Keyword Proximity Search on XML Databases. In: Proc. of 29th VLDB Conference. (2003) 1069–1072
50. Consens, M.P., Mendelzon, A.O.: Expressing Structural Hypertext Queries in Graphlog. In: Proc. of the 2th ACM Conf. on Hypertext, ACM Press (1989) 269–292
51. Gemis, M., Paredaens, J.: An Object-Oriented Pattern Matching Language. In: Proc. 1th ISOTAS, Springer-Verlag (1993) 339–355

52. Alashqur, A.M., Su, S.Y.W., Lam, H.: OQL: A Query Language for Manipulating Object-oriented Databases. In: Proc. of the 15th VLDB Conference, Morgan Kaufmann (1989) 433–442
53. Flesca, S., Greco, S.: Partially Ordered Regular Languages for Graph Queries. In: Proceedings of the 26th ICALP. Volume 1644 of LNCS., Springer-Verlag (1999)
54. Flesca, S., Greco, S.: Querying Graph Databases. In: Proceedings of the 7th EDBT Conference. Volume 1777 of LNCS., Springer-Verlag (2000) 510–524
55. Seaborne, A.: RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
56. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. Proc. of the 1th ISWC (2002)
57. Berners-Lee, T.: Notation 3 - An RDF Language for the Semantic Web. http://www.w3.org/DesignIssues/Notation3 (2001)
58. Clark, K.G.: RDF Data Access Use Cases and Requirements, W3C Working Draft. http://www.w3.org/TR/rdf-dawg-uc/ (2004)
59. Angles, R., Gutierrez, C., Hayes, J.: RDF Query Languages Need Support for Graph Properties. Technical Report TR/DCC-2004-3, Department of Computer Science, University of Chile (2004)
60. Agrawal, R., Jagadish, H.V.: Algorithms for Searching Massive Graphs. IEEE TKDE 6 (1994) 225–238
61. Agrawal, R., Jagadish, H.V.: Materialization and Incremental Update of Path Information. In: Proc. of the 5th ICDE, IEEE Computer Society (1989) 374–383
62. Agrawal, R., Jagadish, H.V.: Efficient Search in Very Large Databases. In: Proc. of the 14th VLDB Conference. (1988) 407–418
63. Guha, R.V., Lassila, O., Miller, E., Brickley, D.: Enabling Inferencing. The Query Languages Workshop (1998)
64. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and Applications of Tree and Graph Searching. In: Proc. of the 21th ACM PODS, ACM Press (2002) 39–52
65. Yannakakis, M.: Graph-theoretic Methods in Database Theory. In: Proc. of the 9th ACM PODS, ACM Press (1990) 230–242
66. Mannino, M.V., Shapiro, L.D.: Extensions to Query Languages for Graph Traversal Problems. IEEE TKDE 2 (1990) 353–363
67. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. In: Proc. of the 15th VDLB Conference, Morgan Kaufmann (1989) 185–193
68. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record 26 (1997) 54–66
69. Anyanwu, K., Sheth, A.: The ρ-operator: Enabling Querying for Semantic Associations on the Semantic Web. In: The 12th WWW Conference. (2003)
70. Washio, T., Motoda, H.: State of the Art of Graph-based Data Mining. SIGKDD Explor. Newsl. 5 (2003) 59–68
71. Hidders, J., Paredaens, J.: GOAL, A Graph-Based Object and Association Language. CISM - Advances in Database Systems 1993 (1993) 247–265
72. Cruz, I.F., Mendelzon, A.O., Wood, P.T.: G+: Recursive Queries without Recursion. In: Proc. of the 2th International Conference on Expert Database Systems, Addison-Wesley (1989) 645–666
73. Sayers, C.: Node-centric RDF Graph Visualization. Technical Report HPL-2004-60, HP Laboratories (2004)
74. Guha, R., McCool, R., Miller, E.: Semantic search. In: Proc. of the 12th WWW conference, ACM Press (2003) 700–709