An Abstract Data Model for the Tabulator Data Browser

RENZO ANGLES and CLAUDIO GUTIERREZ
Computer Science Department, Universidad de Chile

1. INTRODUCTION

One of the main features of the RDF data model is its ability for modeling highly interconnected data in an extensible way. In this sense, given the graph-like nature of RDF, browsing RDF data arises as a natural requirement. At the moment, there are many applications for browsing RDF, for example Palm-DAML, RDF Author, IsaViz, CS-Aktive, mSpace, BrownSauce, DAM viewer. These applications, commonly named Semantic Web Data Browsers, use different visualization models (e.g. circle-and-arrow diagrams), each one having its advantages and disadvantages.

The Tabulator [2] is a semantic data browser which provides the user with different views (e.g. outlines and tables) for visualizing, querying and browsing of RDF data. The central component of the Tabulator is the *outliner view*, a tree-like environment that provides a powerful interface for presenting RDF data as a nested structure of resources. Moreover, the user is able to express queries through the definition of graphical patterns.

The objective of this paper is to show that underlying the outliner view, there is a powerful data model which needs to be studied formally to take advantage of its properties. In this sense, we present and analyse the outliner view. We define a general framework based on the notion of *nested graphs*, a concept that was studied in the area of databases by the hypernode model [9; 7; 12]. An hypernode extends the plain structure of a graph to a nested structure, allowing a simple and flexible representation of nested complex objects.

We study nested graphs in the context of *complex objects* in databases, in particular the notion of equivalence. In this direction, the information capacity of a nested graph is defined in terms of its plain representation called a *graphset*. We treat *data relativism* [5] when analyzing the issue of transforming nested graphs into graphsets and viceversa. Relevant properties for visualization of nested graphs are defined. Finally, we present an algebraic query language oriented to capture the basic queries supported by the outliner-view. This language consists in two basic operators for *selection* and *projection* over a set of nested graphs. Both operators are based on matching of nested graph patterns and are closed under sets of nested graphs.

The contribution of this paper is to define formally an abstract data model which allows us to study properties and characteristics of the Tabulator. We show that *nested graphs*, *graphsets* and *RDF graphs* are three equivalent abstract representations for RDF data. Using this, we propose that the nested graph model can be considered an abstract representation for the outliner view of the tabulator RDF browser.

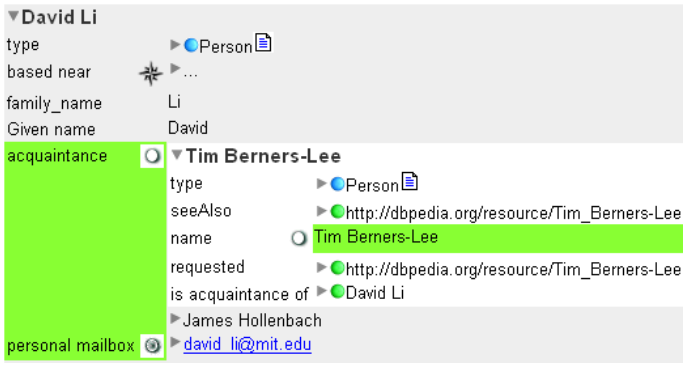

Fig. 1. Outliner View

The paper is organized as follows. In section 2 we study The Tabulator by describing and discussing its characteristics; Section 3 presents formally the notion of nested graphs and we show that it can be considered an abstract representation for the Tabulator; In section 4 we present a query language for the model; Finally, Section 5 presents some conclusions.

2. THE TABULATOR

Many RDF visualizers (e.g. IsaViz) present RDF data as circle-and-arrow diagrams. This interface is very intuitive and useful when trying to understand the structure of data (a graph in the case of RDF), however it is not an appropriate way to look at data with many nodes or for comparing objects of the same class. Moreover, they are not proper for typical enterprise users who are accustomed to seeing and handling data through tables or matrices.

The Tabulator [2] is a generic semantic data browser whose design was motivated by the lack of a straightforward generic data browser with an appropriate user interface metaphor. The workspace of the Tabulator[1] is divided in different views and forms. The *outliner view* (see Figure 1) is a powerful component where the user is able to: visualize RDF data as a nested structure of resources (in the form of a tree); browse RDF data by expanding nodes and navigating across its tree structure; and query RDF data by defining graphically a graph pattern. For complex queries, a *SPARQL view* (see Figure 2) allows the user to introduce SPARQL queries [13] through a web form. The result of a query is displayed in other modular views, e.g, tables. All these views, work together to provide a rich environment for visualizing, exploring and querying of RDF data. Following, we describe the main features presented by the Tabulator, which are relevant toward it formalization.

2.1 Exploring vs. Analyzing

The Tabulator studies RDF querying in the context of exploring and analyzing linked data. In the *exploration mode*, the user is unaware of what data is available, then the user should be able to recognize and follow links. In the *analysis mode*,

[1] http://dig.csail.mit.edu/2005/ajar/release/tabulator/0.8/tab.html

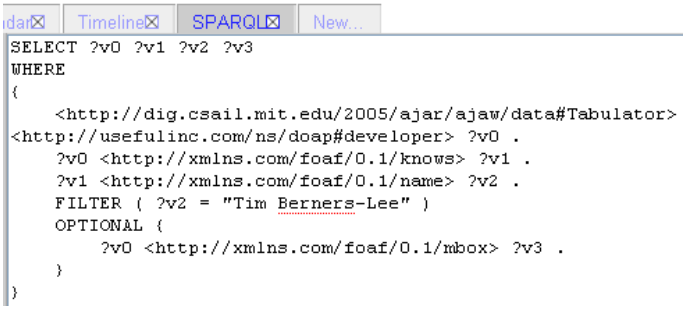

Fig. 2. SPARQL View

the user can select certain resources by using a query language. According to these criteria, the exploration of data in the Tabulator is carried out in the *outliner view*, where an RDF graph is visualized as a tree structure. Exploration begins in a root node (a resource that act as the container) and navigation is achieved by expanding nodes (the properties of the resource) to get more information about them. In this sense, the outliner view models a nesting of resources.

In the analysis mode, the Tabulator provides query facilities for basic users (who have not any experience or detailed knowledge of RDF) and experienced user (who are able to query using SPARQL). In this sense, the outliner view support simple queries represented by graphical patterns, and the *SPARQL view* to introduce complex queries expressed in the SPARQL query language [13].

Figure 1 shows a screenshot of the outliner view characterized by its tree structure. Here, a resource is represented as a node whose properties and values can be viewed by expanding the node through its icon ▶. If a resource is expanded, it is possible to see a new bunch of expandable resources, which corresponds to values for its properties. For example, the node `David Li` looks expanded and shows its properties, where the value of the property `acquaintance` occurs also expanded, showing the description for `Tim Berners-Lee`.

2.2 Querying in The Tabulator

In the outliner view, the user can define a *graphical pattern* by selecting certain fields (arcs or predicates) in a "query-by-example style", and ask the Tabulator to find all examples of that pattern. In the SPARQL view, a graphical pattern can be viewed as a SPARQL expression (with filter and optional), or it can be edited to construct more complex queries. The result of a query can be displayed in a number of modular views, including tables, projections of time and space axes onto a calendar and map.

Now, in the the outliner-view we can also query data by means of a graphical pattern which is constructed by highlighting fields. For example, in Figure 1, the graphical pattern, looks for developers having an acquaintance named "Tim Berners-Lee" and optionally having a personal mailbox. Note that the optional field is specified by selecting its radio button. The graphical pattern can be edited as a SPARQL expression in the SPARQL view and the result of the query can be viewed in the Table view (Figure 3).

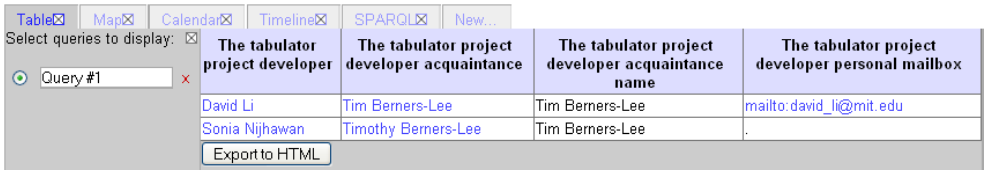

Fig. 3. Table View

2.3 Discussion

Before presenting an abstract data model for the Tabulator, consider the following characteristics of the Tabulator:

—The outliner view allows encapsulation of information because a non-expanded resource hides its properties.

—There is a nesting of resources which is not necessarily hierarchical and cyclic references are possible. For example, in Figure 1, the property `is acquaintance of` introduces a cyclic reference to the predecessor resource `David Li`.

—The model allows to see outgoing and incoming properties for a resource (an incoming property comes from a statement where the resource occurs as the object). It follows the independence of direction for properties in RDF, where the direction one choses for a given property is arbitrary (it does not matter whether one defines “parent” or “child”). The outline is therefore redundant, in that whenever an object is expanded more than one deep, for outer link will also be found as a ”dual” inner link in the opposite direction (e.g. `acquaintance` and `is acquaintance of`).

—A table is a more compact and natural form for the user, but it does not reflect connectivity between resources.

Moreover, consider the following questions about future work on the Tabulator[2]:

—In the case of selecting a set of objects, what could we do with such sets? Restrict a view to only things in that set? Start a new outline view with that set?

—How to deal with large numbers of items as result of a query?

—How to display multiple queries on a single view in order to compare them?

There are other two RDF browsers with similar characteristics. The BrownSauce RDF Browser [3] uses a web interface to show a resource its properties and the resources that reference it. The navigation is possible, but the data of a unique resource is visible in each moment. The DAML Viewer [4] provides a user interface for navigation of nested objects called node view. It shows outgoing and incoming properties, and in case of selecting a node the result, is other node view.

[2]http://dig.csail.mit.edu/2005/ajar/ajaw/ToDo.html
[3]http://brownsauce.sourceforge.net/
[4]http://www.daml.org/viewer/

3. ABSTRACT DATA MODEL FOR THE OUTLINER VIEW

The abstract data model for the outliner view to be defined in this section is based on the notion of *nested graphs*, a concept that was introduced in the area of graph database modeling by the Hypernode Model [9; 7; 12]. The single data structure of this model is the *hypernode*, a graph whose nodes can themselves be graphs.

Hypernodes are simple and flexible data structures that: extends the plain structure of a graph to a nested structure; supports the representation of arbitrarily complex objects; and presents inherent ability to encapsulate information. Other models based on hypernodes are Simatic-XT [10] and GGL [4].

Assume the following domains: an infinite set of URI references U; an infinite set of blank nodes $B = \{N_j : j \in \mathbf{N}\}$; and an infinite set of RDF Literals L. We will use the following abbreviations: UBL for the set $U \cup B \cup L$ and UB for the set $U \cup B$.

3.1 Nested Graphs

We present a formal framework for nested graphs. The basic concepts are complemented with fundamental operators to extract relevant data modeled by a nested graph. Finally, we present properties for nested graphs, when defining *empty*, *cyclic*, *hierarchical*, and *encapsulated* nested graphs.

Definition 1. (Nested Graph) A *Nested Graph* is defined recursively as a pair (n, E_n), where $n \in UB$ is the name of the nested graph and E_n is a finite set of triples (v_1, v_2, v_3) satisfying that:
(i) $v_1 \in U$ or v_1 is a nested graph;
(ii) $v_2 \in \{\text{“}+\text{”}, \text{“}-\text{”}\}$; and
(iii) $v_3 \in UBL$ or v_3 is a nested graph.
We denote by NG the set of nested graphs.

Graphically we represent a triple (v_1, v_2, v_3) by $v_1 \xrightarrow{v_2} v_3$. A triple is called an edge, v_1 and v_3 are called nodes of the edge and v_2 its label. A node v is called a *primitive node* if $v \in UBL$, otherwise if v is a nested graph it is called a *complex node*.

Definition 2. (Basic Operators) Let $N = (n, E_n)$ be a nested graph. The *basic operators name*, $nodes^k$ and $graphs^k$ are defined as follows:

—*name* returns the name of a node. For a complex node $N \in NG$, $name(N) = n$. For a primitive node $v \in UBL$, $name(v) = v$.
—$nodes^k(N)$ extracts nodes from N by examining recursively each nested graph in N until it reaches the k_{th} level of nesting. Let $k > 1$.
$nodes^1(N) = \bigcup_{(v_1,v_2,v_3)\in E_n} \{v_1, v_3\}$
$nodes^k(N) = nodes^1(N) \cup \bigcup_{N' \in nodes^1(N) \cap NG} nodes^{k-1}(N')$
$nodes^*(N) = \bigcup_{k \geq 1} nodes^k(N)$
—$graphs^k(N)$ extracts nested graphs from $nodes^k(N)$.
$graphs^k(N) = \{v \mid v \in nodes^k(N) \cap NG\}$
$graphs^*(N) = \bigcup_{k \geq 1} graphs^k(N)$

Finally, we have the following definitions: if $nodes^1(N) = \varnothing$ then N is an *empty nested graph*; N is *cyclic* if $N \in nodes^*(N)$ or if there exist $N' \in graphs^*(N)$ such that N' is cyclic; A nested graph is *hierarchical* if it is not cyclic; A node v (simple or complex) is *encapsulated* in N if $v \in nodes^*(N)$. Two nested graphs N_1 and N_2 are *compatible* if $name(N_1) = name(N_2)$.

Next, we study the notion of equivalence between nested graphs in the context of *complex objects* in databases, specifically the issue of comparing the *information capacity* of complex object types. The intuitively approach usually taken in the conceptual literature is to say that one object type is *absolutely dominated* by another object type if we can construct at least as many objects of the second type as of the first type. Moreover, one object is *query dominated* by another object type if any query over a collection of objects of the first object type can be translated into a query over a collection of objects of the second object type. The information capacity of two object types is absolutely equivalent (query equivalent) if both object types absolutely dominate (query dominate) each other [6; 8].

Following the above approach, we introduce the notion of graphsets as an equivalent plain representation for nested graphs. This approach will allow us to study other properties of nested graphs, for example comparing their information capacity. The activity of structuring the same data in different ways is known in the area of databases as "data relativism". Consider that it is crucial to understand data relativism at a fundamental level, so that systems can effectively translate between alternative data representations [5].

3.2 Graphsets

In this section, we present a plain (or unnested) representation for a nested graph, called a *graphset*. The complexity of working with a nested structure is usually reduced by transforming it as a plain structure. This result is shown when we define additional operators and properties for nested graphs in terms of graphsets.

Definition 3. (Plain Graph) A *Plain Graph* is a nested graph G satisfying that $graphs^1(G) = \varnothing$, *i.e.*, a plain graph has no nested graphs as nodes.

Given two compatible plain graphs $G_1 = (n, E_1)$ and $G_2 = (n, E_2)$, we define the union, $G_1 \cup G_2 = (n, E_1 \cup E_2)$; intersection, $G_1 \cap G_2 = (n, E_1 \cap E_2)$; and difference, $G_1 - G_2 = (n, E_1 - E_2)$ between G_1 and G_2.

Definition 4. (Graphset) A *Graphset* S is a set of plain graphs satisfying that, for each two plain graphs $G_1, G_2 \in S$ then G_1 and G_2 are not compatible, *i.e.* a name $n \in UB$ identifies at most one plain graph.

Given two graphsets S_1, S_2, we say that S_1 is a *sub-graphset* of S_2, denoted $S_1 \subseteq S_2$, iff $\forall (n, E_1) \in S_1$ there exists $(n, E_2) \in S_2$ satisfying that $E_1 \subseteq E_2$. Two graphsets are *equivalent*, denoted $S_1 \equiv S_2$, if $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$. Additionally, we define the following operations between S_1 and S_2:

Intersection: $S_1 \cap S_2 = \{G \cap G' | G \in S_1, G' \in S_2 \text{ are compatible}\}$

Difference: $S_1 - S_2 = \{G | G \in S_1 \text{ is not compatible with all } G' \in S_2\}$
$\cup \{G - G' | G \in S_1, G' \in S_2 \text{ are compatible}\}$

Maximal-intersection: $S_1 \sqcap S_2 = \{G \cup G' | G \in S_1, G' \in S_2 \text{ are compatible}\}$

Union: $S_1 \cup S_2 = (S_1 - S_2) \cup (S_2 - S_1) \cup (S_1 \sqcap S_2)$

Let $N = (n, E_n)$ be a nested graph. We introduce the transformation of nested graphs into graphsets by defining the operators $flat$ and $flat^*$ as follows:

—$flat(N) = (n, E'_n)$ where $E'_n = \{(name(v_1), v_2, name(v_3)) \mid (v_1, v_2, v_3) \in E_n\}$
—$flat^*(N) = flat(N) \cup \{flat(N') \mid N' \in graphs^*(N)\}$

Then, $flat(N)$ transform N into a plain graph by flattening its first level of nesting. $flat^*(N)$ flattens N and each nested graph encapsulated in N.

The following lemma defines the equivalence - in terms of representation - between nested graphs and graphsets.

Lemma 1. Let NG the set of nested graphs and GS the set of graphsets. There are functions $f : NG \to GS$ and $g : GS \to \mathcal{P}(NG)$ such that: (i) for each $N \in NG$, $N \in g(f(N))$; and (ii) for each $S \in GS$, $f(g(S)) = S$.

Proof. .

(1) Given a nested graph N, just consider $f(N) = flat^*(N)$.
(2) Define $g : GS \to \mathcal{P}(NG)$ as follows: a graphset S can be transformed into a set of nested graphs by expanding the structure of each plain graph in S, i.e, replacing simple nodes by complex nodes. The function g is defined by the following algorithm: given a graphset S, for each plain graph $G \in S$, for each primitive node $v \in UB$ occurring in G, we replace v by a complex node $(n, E_n) \in S$ if $v = n$.

□

Following the notion that the information capacity of a representation is given by the set of objects modeled by the representation, the *information capacity of a nested graph* N is given by N plus the set of plain graphs encapsulated by N, that is the graphset $flat^*(N)$. Additionally, we say that the equivalence of two nested graphs is given by the equivalence in their information capacities, i.e. they model exactly the same set of objects.

In [1], it was shown that two complex object types are absolutely equivalent if and only if they can both be reduced to a normal form complex object type, which is based on some natural restructuring operators. Based on this idea, we define the equivalence of nested graphs as follows:

Definition 5. (Equivalence of nested graphs) Two nested graphs N_1 and N_2 are *equivalent*, denoted $N_1 \equiv N_2$, if they can be reduced to the same graphset, i.e. $flat^*(N_1) = flat^*(N_2)$.

Additionally, we say that N_1 is a *subgraph* of N_2, denoted $N_2 \subseteq N_1$, iff N_1 is compatible with N_2 and $flat^*(N_1) \subseteq flat^*(N_2)$

3.3 RDF Graphs and Graphsets

In this section we present the RDF model following the W3C documents. We use the abstract definition of RDF to show that we can represent an RDF graph as a graphset and viceversa.

An *RDF triple* is a triple (s, p, o) such that $s \in UB$ is called the *subject*, $p \in U$ the *predicate* and, $o \in UBL$ the *object*. An *RDF Graph* G_{rdf} [11] is a set of RDF triples. The *vocabulary* of G_{rdf}, is the set of elements of UB that occur in the triples of G_{rdf}.

Lemma 2. An RDF Graph can be represented as a graphset.

PROOF. An RDF graph G_{rdf} is represented by a graphset S, if for each resource or blank node u in G_{rdf}, there exist a plain graph G in S, which encapsulates the set of RDF triples where u occurs as subject plus the triples where u occurs as object.

The function $\mathcal{G}set(G_{rdf})$ returns the graphset S by using the following algorithm: for each $u \in vocabulary(G_{rdf})$, there is a plain graph (u, E_u) where
$E_u = \{(p, +, o)|(u, p, o) \in G_{rdf}\} \cup \{(p, -, s)|(s, p, u) \in G_{rdf}\}$.

In this context, a plain graph (u, E_u) interprets a resource identified u with the set of properties E_u. A triple $(p, +, o)$ in E_u is called an *outgoing property* and a triple $(p, -, s)$ is called an *incoming property*. □

Lemma 3. A graphset can be represented as an RDF Graph.

PROOF. A graphset S can be transformed into an RDF graph by retrieving the triples modeled by each plain graph G in S. The function $\mathcal{G}rdf(S)$ returns an RDF graph from a graphset S as follows:

$$\mathcal{G}rdf(S) = \{(u, p, o)|(p, +, o) \in G, G \in S\} \cup \{(s, p, u)|(p, -, s) \in G, G \in S\}$$ □

Note. A plain graph can be considered a special type of *Named Graph* [3] (n, g) where the triples of g have the same subject n. Additionally, a graphset can be considered a special type of *RDF Dataset* [13] D where there is not default graph in D and the named graphs in D are special.

3.4 Nested graphs in the Tabulator

Now, we present the main result of the paper.

Proposition 1. The nested graph model is an abstract representation for the outliner view.

PROOF. (Note that the outliner view has no formal definition to compare if formally to own model). Consider the outliner view presented in Figure 1. An abstract representation for the expanded node `David Li` can be the nested graph presented in Figure 4 . If we compare both figures, we can see that an expanded node in the outliner view can be represented by a nested graph, and outgoing and incoming properties are represented by edges labeled "+" and "-" respectively. Note that, the redundant link `is acquaintance of` → `David Li` of the outliner view is represented by the edge `acquaintance` $\bar{\rightarrow}$ `David Li`. □

Now, consider the process of visualize an RDF graph G_{rdf} in the outliner view. For Lemma 2 we represent G_{rdf} as a graphset S. For Lemma 1, we transform S in a set of nested graphs M. Finally, the outliner view of the Tabulator is a graphic representation of M.

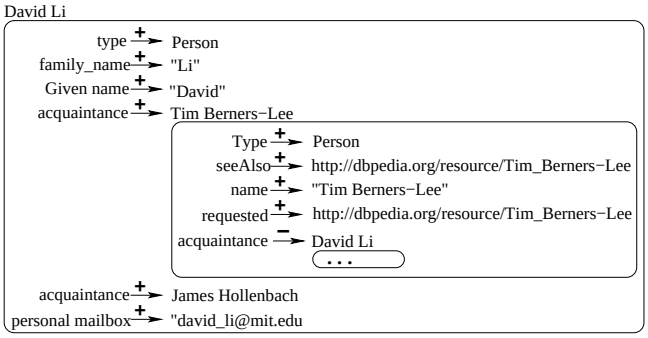

Fig. 4. A nested graph that represent the outliner view of Figure 1

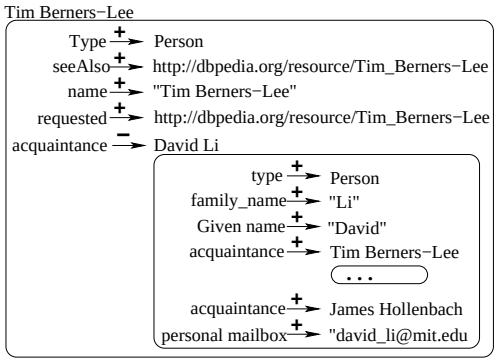

Fig. 5. A nested graph equivalent to the nested graph presented in Figure 4

If we construct a set of nested graphs M from a graphset S, we will have a nested graph in M for each plain graph in S, but in some cases it is possible that a subset of M is enough for modeling all the data modeled by the graphset S. We define this subset as the core of a set of nested graphs.

Definition 6. (Core of a set of nested graphs) Let M be a set of nested graphs. A *Core* of M is a minimal subset M' of M satisfying that $flat^*(M') = flat^*(M)$

For example, consider the graphset S obtained by transforming the nested graph of Figure 4. We will have two plain graphs in S, one for `David Lee` and other for `Tim Berners-Lee`. Now, if we obtain the set of nested graphs M from the graphset S (by using Lemma 1), we will have the nested graphs of Figure 4 and Figure 5. Note that, these nested graphs are equivalent (such that they can be reduced to the same graphset S) and both are a core of the graphset S, because any of them contains all the data modeled by S.

Now, if the core of a set of nested graphs M is a single nested graph N, then N is called the *single-source* of M. In the context of graphsets, the above property introduce the notion of a *single-source graphset*.

Definition 7. (Single-source graphset) A graphset S is called a *single-source graphset* if there exists a nested graph N such that $flat^*(N) = S$.

The notion of a core is useful for visualization of nested graphs. For example consider the problem of selecting a good start point for navigation. If we use the core as a minimal set of navigation, we can reduce the number of visible or active nested graphs without losing data, such that all the data could be accessed by navigating through the nested graphs of the core. Clearly this problem could have a direct solution if we have a single-source graphset, the all the data can be accessed from a single nested graph.

4. ALGEBRAIC QUERY LANGUAGE

In this section we sketch an algebraic query language that simulates the graphic queries supported by the outliner view. The query language consists of two operators (projection and selection) which are based in the use of nested graph patterns.

Definition 8. (Nested Graph Pattern) Let V an infinite set of variables (disjoint from UBL). A *Nested Graph Pattern* $\mathcal{P}$ is a nested graph where some URI's, labels or blank nodes are replaced by variables from V. We denote by $var(\mathcal{P})$, the set of variables occurring in $\mathcal{P}$.

Definition 9. (Plain Graph Pattern and Graphset Pattern) A *Plain Graph Pattern* is a nested graph pattern which is plain. A *Graphset Pattern* is a graphset of plain graph patterns. We denote by $var(\mathcal{SP})$, the set of variables occurring in a graphset pattern $\mathcal{SP}$.

A *matching* is a function $\psi : UBLV \rightarrow UBL$ defined as follows:
(a) for $u \in UL$, $\psi(u) = u$.
(b) for $x \in BV$, $\psi(x) = t$ where $t \in UBL$.[5]
(c) for a plain graph pattern $\rho = (p, E_p)$, $\psi(\rho) = (n, E_n)$ where $n = \psi(p)$, $E_n = \{(\psi(v_1), v_2, \psi(v_3)) | (v_1, v_2, v_3) \in E_p\}$.
A *matching* ψ is consistent with a graph pattern $\rho = (p, E_p)$ if $\psi(\rho)$ is a plain graph, i.e., $\psi(p) \in UB$ and $\forall (v_1, v_2, v_3) \in E_p$, $\psi(v_1) \in U$ and $\psi(v_3) \in UBL$.

Definition 10. (Matching of plain graph patterns) Let ρ be a plain graph pattern, and G a plain graph. We say that ρ *matches* G, denoted $\rho \prec G$, if there exist a matching ψ such that $\psi(\rho)$ is a subgraph of G. Additionally, if $\psi(\rho)$ is equal to G we say that ρ *completely matches* G, denoted $\rho \preceq G$. For a particular matching ψ we will use the notations $\rho \prec_\psi G$ and $\rho \preceq_\psi G$.

Definition 11. (Matching of graphset patterns) A graphset pattern $\mathcal{SP}$ matches a graphset S, denoted $\mathcal{SP} \prec S$, if there exist a matching ψ such that for each plain graph pattern $\rho \in \mathcal{SP}$ there is a plain graph $G \in S$ satisfying that $\rho \prec_\psi G$. For a particular matching ψ we will use the notation $\mathcal{SP} \prec_\psi S$.

Definition 12. (Matching of nested graph patterns) A nested graph pattern $\mathcal{P}$ matches a nested graph N, denoted $\mathcal{P} \prec N$, if there exist a matching ψ such that $flat^*(\mathcal{P}) \prec_\psi flat^*(S)$.

Definition 13. (Selection and Projection operators) Let $\mathcal{S}$ be a set of nested graphs and $\mathcal{P}$ a nested graph pattern where some edges are marked as *OPTIONAL*.

[5]A blank node can appear in a query pattern. It behaves as a variable; a blank node in a query pattern may match any RDF term UBL (SPARQL 2.7)

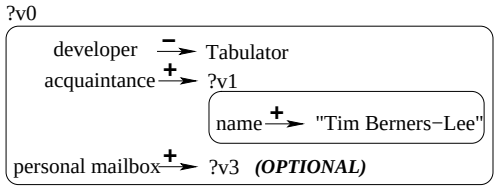

Fig. 6. A nested graph pattern equivalent to the graphical pattern of Figure 1

The operators Selection and Projection are defined as follows:

• The *Selection* operator, denoted $\sigma_{\mathcal{P}}(\mathcal{S})$, returns the nested graphs of S which satisfy the nested pattern $\mathcal{P}$ considering some optional edges. It means that, a nested graph N in S, is part of the result set, if and only if there exist $\mathcal{P}'$ which is a sub-nested graph pattern of $\mathcal{P}$ where some optional edges are omitted, satisfying that $\mathcal{P}' \prec N$.

• The *Projection* operator, denoted $\Pi_{\mathcal{P}}(\mathcal{S})$, projects the nested graphs of $\sigma_{\mathcal{P}}(\mathcal{S})$ according to $\mathcal{P}$. It means that, a nested graph N is part of the result set, if and only if there exist N' that belong to $\sigma_{\mathcal{P}}(\mathcal{S})$ and N is a subgraph of N' whose edges are projected according to $\mathcal{P}$ (or $\mathcal{P} \preceq N$).

A *Query Expression* is defined as the recursively application of the selection and projection operators. Consider the graphical pattern presented in Figure 1 and its equivalent SPARQL expression in Figure 2. We express its query in our query language through a selection $\sigma_{\mathcal{P}}$ where $\mathcal{P}$ is the nested graph pattern showed in Figure 6.

In Figure 3 we can note that the Tabulator follows the SPARQL approach where solutions to queries are given in the form of result sets (result sets are sets of mappings from the variables occurring within the query to nodes of the queried data). In contrast, the result of a query expression in our query language is a set of nested graphs, i.e, our query language is closed under sets of nested graphs.

5. CONCLUSIONS

In this paper, we have analysed the semantic web browser Tabulator, principally its underlying data structure called the outliner view. We have proposed an abstract data model for the outliner view based on the notion of nested graphs, together with their plain representation called graphsets. We showed that an RDF graph can be modeled in terms of graphsets and consequently as nested graphs. When studying formal properties of RDF graphs viewed as nested graphs (or graphsets), we observed that its is possible to optimize some visualization parameters. Finally, we proposed a algebraic query language closed under nested graphs. Currently, we are analyzing the properties of nested graphs and graphsets (e.g. what is the most appropriate core to start a navigation?) and we are improving the expressiveness of the query language by adding other operators.

REFERENCES

S. Abiteboul and R. Hull. Restructuring hierarchical database objects. *Theoretical Computer Science*, 62(1-2):3–38, 1988.

T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06)*, 2006.

J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference World Wide Web (WWW '05)*, pages 613–622. ACM Press, 2005.

M. Graves, E. R. Bergeman, and C. B. Lawrence. A Graph-Theoretic Data Model for Genome Mapping Databases. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, page 32. IEEE Computer Society, 1995.

R. Hull. Relative information capacity of simple relational database schemata. In *Proceedings of the 3rd ACM Symposium on Principles of Database Systems (PODS)*, pages 97–109. ACM Press, 1984.

R. Hull. A survey of theoretical research on typed complex database objects. pages 193–261, 1987.

M. Levene and G. Loizou. A Graph-Based Data Model and its Ramifications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 7(5):809–823, 1995.

M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springe-Verlag, 1999.

M. Levene and A. Poulovassilis. The Hypernode Model and its Associated Query Language. In *Proceedings of the 5th Jerusalem Conference on Information technology*, pages 520–530. IEEE Computer Society Press, 1990.

M. Mainguenaud. Simatic XT: A Data Model to Deal with Multi-scaled Networks. *Computer, Environment and Urban Systems*, 16:281–288, 1992.

F. Manola and E. Miller. RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, Feb 2004.

A. Poulovassilis and M. Levene. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. *ACM Transactions on Information Systems (TOIS)*, 12(1):35–68, 1994.

E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/, June 2007.