

The ρ -Operator: Enabling Querying for Semantic Associations on the Semantic Web.

Kemafor Anyanwu
LSDIS lab
Department of Computer Science
University of Georgia, Athens, GA 30602
01-706-542-4772
anyanwu@cs.uga.edu

Amit Sheth
LSDIS lab
Department of Computer Science
University of Georgia, Athens, GA 30602
01-706-542-2310
amit@cs.uga.edu

ABSTRACT

Semantic Associations are a class of complex relationships between entities that capture a connectivity of entities or a pattern of entities and relationships between them based on a specific notion of an isomorphism called ρ -isomorphism. In an RDF graph data model, they may be represented as sequences (i.e. paths) between entities, networks of sequences, or a subgraph containing ρ -isomorphic sequences. Capturing such relationships based on their structural properties, allows us to define them as types that may be returned as results of a query. Such a capability, while lacking in most RDF query languages, is essential in supporting many of the tasks found in analytical domains such as national security and business intelligence. In these domains, investigative tasks are often focused on detecting such complex associations that may be buried deep in the data.

This paper presents a formalization of Semantic Associations for the RDF data model. It also shows how querying for such relationships may be enabled on the Semantic Web, through the use of an operator ρ . Finally, it discusses two approaches for implementing the ρ operator. One of the approaches is based on building upon current RDF query languages, allowing the reuse of existing infrastructure.

Categories and Subject Descriptors

H.2.3 [Information Systems]: Database Management—Query Languages

General Terms

Languages, Theory, Management

Keywords

Semantic Web Querying, Semantic Associations, RDF, Complex Data Relationships, graph traversals,

Submitted for review to WWW2003.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '00, Month 1-2, 2000, City, State.

Copyright 2000 ACM 1-58113-000-0/00/0000...\$5.00.

1. INTRODUCTION

The Semantic Web [12] proposes to explicate the meaning of Web resources by annotating them with metadata that have been described in an ontology. Consequently, machine processible representations of ontologies have been the focus of much debate amongst the researchers in the Semantic Web community, which initiated some standardization efforts by the W3C. Some of the standards include, the eXtensible Markup Language (XML) [14], a standard for data representation and exchange on the Web, the Resource Description Framework (RDF) [34], and its companion specification, RDF Schema (RDFS) [20], which together provide a uniform format for the description and exchange of the semantics of web content. An important related issue is the support for ontology-driven content annotation of web content. Recently work in both the academic and commercial communities, has led to the availability of tools that provide automatic [41] semantic (ontology-driven and/or domain-specific) metadata extraction and annotation.

With the progress towards realizing the Semantic Web, researchers are now focused on the development semantic query capabilities for the Web, that will exploit the semantics of web content to provide superior results than present-day techniques that rely mostly on the syntactic representation of content like traditional search engines, and document structure e.g. XQuery [21]. These research activities have led to some RDF query language proposals, including RQL [32], SquishQL [38], TRIPLE [42]. Many of these languages offer most of the essential features for semantic querying like, the ability to query using ontological concepts, inferencing as part of query answering, and the ability to specify non-explicit queries like the use of path expressions. One key advantage of this last feature is that users do not need to have in-depth knowledge of schema and are not required to specify the exact paths that qualify the desired resource entities. However, even with such expressive capabilities, many of these languages do not adequately support a query paradigm that enables the discovery of complex relationships between resources. The paradigm offered by these class of languages as well as many of the earlier ones, is one in which queries are of the form: “Which entities are related to ResourceA via relationship R?” where R is typically specified as possibly a join condition or path expression, etc. That is queries are used to find other resources that satisfy some kind of a criterion. On the other hand, queries of the form: “How is Resource A related to Resource B?”, are supported only in a very limited manner. The query paradigm used to support the first kind of query, requires that the nature of

distinguishing relationship between entities must be, albeit abstract, specified as part of the query, and then the qualifying entities are returned as the result. The use of path expressions does reduce some of the burden of such a requirement. However, the requirement of using an expression still means the user specifies some information about the structure of the relationship as part of the query. To be able to support the second kind of query, the relationships between resources should be returned as the result of the query. Such a paradigm supports the investigative tasks in analytical domains such as national/homeland security and business intelligence, where many tasks are focused on finding different kinds of relationships between entities, e.g. the relationship between terrorist acts and terrorist organizations.

One major challenge in dealing with queries about relationships is that it is often not clear exactly what notion of a relationship is required in the query. For example, in the context of assessing flight security, the fact that two passengers on the same flight are nationals of country with known terrorist groups and that they have both recently acquired some flight training, may indicate a similarity that associates the two passengers. On the other hand, the fact that a passenger placed a phone call or someone in another country that is known to have links to terrorist organizations and activities indicates some kind of an association. In this case though, the association is more of a connectivity of entities. Some of the RDF query languages [32] allow querying for the properties that exist between entities by using property variables, but such a notion of relationships as a single binary relation is limited. In these languages, it is also possible to query for a path by using a sequence of property variables, but this approach requires that you know the length of the sequence in order to specify the required number of property variables. For another notion of a relationship, we look cite a recent news article titled “What is the link between the Beltway Sniper, The Arizona Shooter and the Oklahoma City Bomber?”, nicknames that refer to individuals that committed multiple acts of murders in the named region. This question was posed to try and analyze what, if any, relationship exist between such individuals that commit violent fatal acts against victims picked randomly, because they did not seem to fit the profiles of serial killers who tend to exhibit a pattern in their actions. The relationship that was discovered in this situation, was that all three individuals were post war veterans, that is were members of a particular group of people.

All these examples suggest various kinds of relationships, Therefore, we need a more flexible approach to querying about relationships, and in addition, we need to support various notions of complex relationships such as the ones described earlier. We call such complex relationships, *Semantic Associations*. A preliminary discussion of these so-called *Semantic Associations* has been made in [10].

Our main contributions in this paper are the following: the classification of *Semantic Associations* into four main classes based on their structural properties, allowing us to reason about them in a domain-independent manner, the formalization of these *Semantic Associations* for the RDF data model, the specification of what we call ρ -Queries, which are queries about *Semantic Associations* using an operator, ρ , and a discussion of two implementation strategies for the ρ operator. The first involves the use of a main-memory resident RDF model base and linear graph path algorithms such as [43]. The other is based on a layered approach over existing RDF data stores so that some of the

computation is done by a translation to their corresponding query languages. The rest of the paper is organized as follows: Section 2 discusses some background and motivates our work with the help of an example. Section 3 presents the formal framework for *Semantic Associations*, section 4 discusses implementation strategies for the ρ operator, section 5 reviews some related work, and section 6 concludes the paper.

2. BACKGROUND & MOTIVATION

2.1 RDF

RDF is a standard for describing and exchanging semantics of web resources. It provides a simple data model for describing relationships between resources in terms of named properties and their values. The rationale for the model is that by describing what relationships an entity has with other entities in the domain, we somehow capture the meaning of the entity. Relationships in RDF, or *Properties* as they are called, are binary relationships between two resources, or between a resource and a literal value. An *RDF Statement*, which is a triple of the form (Subject, Property, Object), asserts that a resource, the *Subject*, has a *Property* whose value is the *Object* (i.e. another resource or a literal). This model can be represented as a labeled directed graph, where nodes represent the resources (ovals) or literals (rectangles) and arcs representing properties whose source is the subject and target is the object, and are labeled with the name of the property. For example, in the bottom part of Figure 1, we can see a node $\&r1$ connected by a *paints* arc to the node $\&r2$, which reflects the fact that $\&r1$ (a painter with first name Pablo, and last name Picasso) painted another resource $\&r2$ (a painting). The meaning of the nodes and arcs is derived from the connection of these nodes and arcs to a vocabulary (the top part of the figure). The vocabulary, called an *RDF Schema*, uses the companion specification to RDF called the *RDF Schema*, to describes classes of resources and types of properties for a domain. For example in Figure 1, classes like *Painter*, *Museum* and properties such as *Paints*, are defined. Then resources are connected to classes using an *rdf:property* property indicating an instantiation relationship. Classes and *Properties* in a schema may also organized in a hierarchy using the *rdf:subclassOf* and *rdf:subpropertyOf* properties respectively.

2.2 MOTIVATING EXAMPLE

We will now illustrate *Semantic Associations* by way of a simple example shown in Figure 1. For brevity and pedagogical reasons we have chosen to use a modified version of the example from [32]. However, many of our evaluations are based on datasets in the domain of National Security. The figure shows an RDF model base containing information to be used in the development of a cultural portal, given from two perspectives, reflected in two different schemas (the top part of the figure). The top left section of the picture is a schema that reflects a museum specialist’s perspective of the domains using concepts like *Museum*, *Artist*, *Artifact*, etc. The top right section is a schema that reflects a Portal administrator’s perspective of the domains using administrative metadata concepts like *file-size*, *mime-type*, etc. to describe resources. The lower part of the figure is the model base or description base in the linguo of [32], that has descriptions about some web resources, e.g., museum websites ($\&r3$, $\&r8$), images of artifacts ($\&r2$, $\&r5$,

&r7) and for resources that are not directly present on the Web, e.g., people, nodes representing electronic surrogates are created (&r1, &r4, &r6 for the artists Pablo Picasso, Rembrandt, and Rodin August respectively).

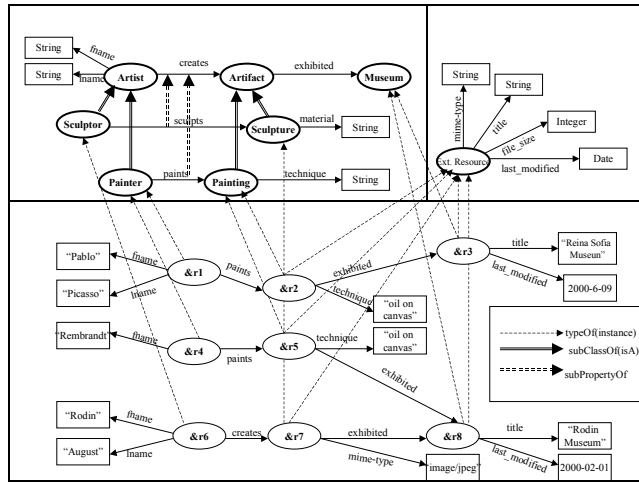


Figure 1: Cultural Portal Information in RDF

Typically, a query language allows you to find all entities that are related by a specific relationship. For example, we may ask a query to retrieve all resources related to resource &r1 via a `paints` relationship, or via a `paints.exhibited` relationship, and get &r2 as a result for the first query and &r3 as the answer for the second query. However, we are unable to ask easily queries such as “How are resources &r1 and &r3 related? Such a query should return for example that “&r1 paints &r2 which is exhibited in &r3”, indicating a path connecting the two entities. With a query such as this one, the user is trying to determine *if* there is a relationship between entities, and *what* the nature of the relationship(s) is(are). It should be possible to ask such a query without any type of specification as to the nature of the relationship, such as using a path expression to give information about the structure of the relationship. This is needed because, in many situations the existence and nature of relationships between entities is not known by user asking the query, and the investigation of the nature of relationship in itself is the objective. Many systems provide tools for browsing schemas to enable users get an idea of what kinds of relationships exist between entities. While this may be a reasonable requirement when querying specific domains with a few schemas involved, on the Semantic Web, many schemas may be involved in a query, and requiring a user to browse them all would be daunting task for users. In fact, in some cases, such information may not be available to all users (e.g., classified information) even though the data may be used indirectly to answer queries. Furthermore, browsing schemas do not always give the complete picture, especially in the case of RDFS schemas, because, entities may belong to different schemas, creating links between entities that are not obvious from just looking at the schemas. For example in Figure 1, the relationship `paints.exhibited.title` connecting &r1 to “Reina Soifa Museum”, is not apparent by just looking at either schema.

So far, we have talked about relationships as paths connecting entities. However, there some more interesting ones that involve

more than just a direct path connecting two entities. Let us take for example, resources &r4 and &r6. Both resources could be said to be related because they have created artifacts (&r5, and &r7) that are exhibited at the *same* museum (&r8). In this case, having some relationship to the *same* museum associates both resources. Another closely related kind of association is class membership. For example, &r1 and &r6 are both Artists, even though of a different kind, and therefore are somewhat associated. Also, &r1 and &r6 could be said to be associated because they both have creations (&r2, and &r7) that are exhibited by a Museum (&r3 and &r8 respectively). In this case, the association is that of a *similarity*. So, in the first three associations the relationships capture some kind of connectivity between entities, while the last association captures a similarity between entities. Note that the notion of similarity used here is not just a structural isomorphism, but a semantic isomorphism of paths, which captures a semantic similarity of both the nodes and arcs along a path. Nodes are considered similar, if they have a common ancestor class. For example in the relationship involving &r1 and &r6, although one case involves a painting and the other a sculpture, we consider them similar because and sculptures and paintings are kinds of Artifacts and sculpting and painting are both kinds of creative activities, i.e. the notion of similarity is extended to properties as well.

We call these kinds of complex relationships, Semantic Associations. The Semantic Associations shown in this example are fairly simple involving only small sequences and are useful only for the purpose of illustration. However, real life situations in analytical domains involve much longer sequences not easily detectable by users in a fast paced environments like at airport security portals where agents may want to determine quickly if a passenger has any kind of link to terrorist organizations or activities.

3. FRAMEWORK

The framework described in this section, provides a formal basis for Semantic Associations. It builds upon the formal model for RDF described in [32], which uses a graph data model and also provides a type system for RDF data. Our major contribution is the extension of this type system to include the notion of a `Property Sequence`, allowing us to capture paths between entities as first class entities. We also define relations on `Property Sequences`, enabling more complex relationships such as the intersecting and isomorphic paths discussed earlier, to be captured as well. Finally, we formalize a basic notion of context required to minimize, for a given query, the search space for Semantic Associations.

3.1 Formal Data Model

To recap, the RDF data model is a labeled directed graph in which (Subject, Property, Object) triples are represented by connecting the Subject and Object nodes with an arc labeled with the Property name. Property types and resource classes are defined in an RDF Schema using its special vocabulary for defining schemas. Using this vocabulary, a `Property` is defined by specifying its domain (the set of classes that it applies to), and its range (either a Literal type e.g. String, Integer, etc or the classes whose entities it may take as values). Classes are defined in terms of their relationship to other classes using the `rdfs:subClassOf` property to place them at the appropriate

location in a class hierarchy, as well as other user specified properties that may include them in their range or domain thereby linking to other classes. Properties may also be organized in a hierarchy using the `rdfs:subproperty` property.

The type system described in [32], forms the basis for a typed RDF query language called RQL. RQL is fairly expressive and allows for a wide variety of queries to be made on RDF model bases. In our discussion of this type system, we will make use of the following abbreviations: \mathcal{C} is the set of all class names in an RDF Schema, \mathcal{P} is the set of property names, and \mathcal{L} is the set of literal type names, e.g. `string`, `integer`. The type system \mathcal{T} is the ? of all possible types that can be constructed from the following types:

$$\tau = \tau_C \mid \tau_P \mid \tau_M \mid \tau_U \mid \tau_L \mid \{\tau\} \mid [1:\tau_1, 2:\tau_2, \dots, n:\tau_n] \mid (1:\tau_1 + 2:\tau_2 + \dots + n:\tau_n)$$

where τ_C indicates a class type, τ_P a property type, τ_M a metaclass type, τ_L a literal type in \mathcal{L} , τ_U is the type for resource URIs. For the RDF multi-valued types we have $\{\}$ is the *Bag* type, $[.]$ is the *Sequence* type, and $(.)$ is the *Alternative* type. The set of values that can be constructed using the resource URIs, literals and class property names is called \mathcal{V} . Then, the interpretation of types in \mathcal{T} is given naturally by the interpretation function $[[\]]$, which is a mapping from τ to the set of values in \mathcal{V} . For example, a class C is interpreted as unary relation of type $\{\tau_U\}$, which is the set of resources (i.e. of type τ_U) that have an `rdf:typeOf` property with range C , and includes the interpretations of C 's subclasses. For a property p , $[[p]]$ is given by

$$\{[v_1, v_2] \mid v_1 \in [[p.domain]], v_2 \in [[p.range]]\} \cup \{[[p']] \mid p' \text{ is a subpropertyOf } p\}$$

Next, we review the definition of the schema and model graphs given in [32].

3.1.1 Definition 1

An RDFS schema is a 5-tuple $RS = (V_S, E_S, \psi, \lambda, H)$ where: V_S is the set of nodes and E_S is the set of edges. ψ is an incidence function $\psi: E_S \rightarrow V_S \times V_S$, and λ is a labeling function that maps class and property names to one of the types in \mathcal{T} , i.e. $\lambda: V_S \cup E_S \rightarrow \mathcal{T}$. H is a well-formed hierarchy of class and property names $H = (N, <, N = C \cup P)$. H is *well-formed* if $<$ is a smallest partial ordering such that: if $p_1, p_2 \in P$ and $p_1 < p_2$, then $p_1.domain \subseteq p_2.domain$ and $p_1.range \subseteq p_2.range$.

3.1.2 Definition 2

An RDF model base, instance of a schema RS , is a 5-tuple $RM = (V_D, E_D, \psi, \lambda, \nu)$, where: V_D is a set of nodes and E_D is a set of edges, ψ is the incidence function $\psi: E_D \rightarrow V_D \times V_D$, ν is a value function that maps the nodes to the values in \mathcal{V} i.e. $\nu: V_D \rightarrow \mathcal{V}$, λ is a labeling function that maps each node either to one of the container type names (`Seq`, `Bag`, `Alt`) or to a subset of class names from the schema, containing those classes whose interpretations contain the value of the node. Formally, $\lambda: V_D \cup E_D \rightarrow 2^{\mathcal{N}} \cup \{\text{Bag, Seq, Alt}\}$. For edges, λ maps an edge $[v_1, v_2]$ to a property name p in the schema, where the interpretation of p contains the pair $[\nu(v_1), \nu(v_2)]$ i.e the values of v_1 and v_2 .

3.2 Data Model Extensions

In the above model, the sequence type $[\tau]$ can be used to capture n -ary relations with heterogeneous member types such as those returned by queries, by defining sequences of type $[\tau_1, \tau_2, \tau_3, \dots, \tau_n]$ for unnamed ordered tuples by $[v_1, v_2, v_3, \dots, v_n]$ where v_i is of type τ_i . Formally, the interpretation of a sequence type $[\tau]$ is given by

$$[[[\tau]] = \{ [1:v_1, 2:v_2, \dots, n:v_n] \mid n > 0, \forall i \in [1..n] v_i \in [[\tau_i]]\}$$

In this approach, it is possible to define a sequence of type $[\tau_{p_1}, \tau_{p_2}, \tau_{p_3}, \dots, \tau_{p_n}]$ to capture a sequence of properties. However, as mentioned before such an approach is not adequate for our purposes because the length of the sequence will have to be known. We therefore extend this with the notion of a Property Sequence, which are first class entities that variables can range over.

3.2.1 Definition 3 (Property Sequence)

A Property Sequence PS is a finite sequence of properties $P_1, P_2, P_3, \dots, P_n$ where each P_i is a property defined in an RDF Schema RS . The interpretation of PS is given by:

$$[[PS] = \{ i: [v_i, v_{i+1}] : 1 \leq i \leq n : (\forall i)_{1 \leq i \leq n} ([v_i, v_{i+1}] \in [[P_i]])\}$$

For $ps \in [[PS]]$, ps is called an *instance* of the Property Sequence PS and is represented in our graph data model as the path $v_1, p_1, v_2, p_2, \dots, v_n$. If such a path exists in a model base RD , then we say that the RD *satisfies* the Property Sequence PS written as $RD \models PS$. The node v_1 is called the *origin* of the sequence and v_n is called the *terminus*. We define a function `NodesOfPS()` which returns the set of nodes of a Property Sequence PS , i.e.

$PS.NodesOfPS() = \{C_1, C_2, C_3, \dots, C_k\}$ where C_i is a class in either the domain or range of some Property P_j in PS , $1 \leq j \leq n$.

For example in Figure 1, for $PS = \text{creates.exhibited.title}$, $PS.NodesOfPS() = \{\text{Artist, Artifact, Museum, Ext. Resource, String}\}$. Next, we define another function `PSNodesSequence` on the instances of Property Sequences i.e.

$ps.PSNodesSequence() = [v_1, v_2, v_3, \dots, v_n]$ where v_i is a node in the model base.

Next, we define a set of binary relations on Property Sequences.

3.2.2 Definition 4 (\bowtie_p -Joined Property Sequences)

$PS_1 \bowtie_p PS_2$ is true if:

$$NodesOfPS(PS_1) \cap NodesOfPS(PS_2) \neq \emptyset$$

The Property Sequences PS_1 and PS_2 are called *joined*, and for $C \in (NodesOfPS(PS_1) \cap NodesOfPS(PS_2))$, C is called a *join node*. For example, in Figure 2, the sequences `creates.exhibited` and `paints.exhibited` are joined because they have a join node `Museum`.

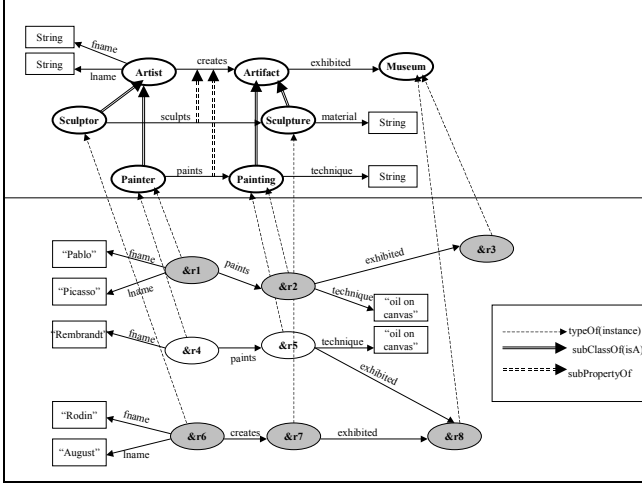


Figure 2 : Isomorphic Property Sequences

3.2.3 Definition 5 (\cong_{ρ} -Isomorphic Property Sequences)

For two property sequences $PS_1 = P_1, P_2, P_3, \dots, P_m$, and $PS_2 = Q_1, Q_2, Q_3, \dots, Q_m$, $PS_1 \cong_{\rho} PS_2$ if

For all $i, 1 \leq i \leq m$:

- 1) $P_i = Q_i$ or $P_i \subseteq Q_i$ or $Q_i \subseteq P_i$ (\subseteq means subpropertyOf)
- 2) $\exists x, y, z$ $x \in \text{domain}(P_i)$ $y \in \text{domain}(Q_i)$ and $z = \text{LeastUpperBound}(P_i, Q_i)$, and $\text{distance}(x, z)$ and $\text{distance}(y, z)$ is minimal.

PS_1 and PS_2 are called ρ -isomorphic. The first definition ensures a similarity of edges while the second ensures a similarity of nodes. For example, in Figure 2, although &r1 is a painter and &r6 is a sculptor both are similar because they are both Artists. Also, because paints is a subproperty of creates, we consider them similar. Therefore the sequences paints.exhibited and creates.exhibited are ρ -isomorphic. With respect to the similarity of nodes, we use the distance function to ensure that the relationship between the nodes is not based on a common parent class that is so far up the hierarchy so that the similarity is not very meaningful.

Note that this example is somewhat misleading because the example shown for Joined Property Sequences also happens to be ρ -Isomorphic. However, the two notions are quite different. In the case of an isomorphism, similarity is maintained along the sequence, while in the case of joined property sequences, the only requirement is that the sequences should meet at some point, similarity of the sequences is not required.

3.3 Semantic Associations

We can now define some binary relations on the domain of entities i.e. resources based on the different types of Property Sequences.

3.3.1 Definition 6 (ρ -pathAssociated)

ρ -pathAssociated (x, y) is true if there exists a Property Sequence with $ps \in [[PS]]$ and, either x and y are the origin and terminus of ps respectively, or vice versa, i.e. y is origin and x is terminus. Then ps is said to satisfy ρ -pathAssociated (x, y) written as $ps \models \rho$ -pathAssociated (x, y).

3.3.2 Definition 8 (ρ -joinAssociated)

Let PS_1 and PS_2 be two Property Sequences such that $PS_1 \bowtie_{\rho} PS_2$ with a join node C , and there exists ps_1 and ps_2 such that $ps_1 \in [[PS_1]]$ and $ps_2 \in [[PS_2]]$ and, $n \in ps_1.PSNodesSequence() \cap ps_2.PSNodesSequence()$, then ρ -joinAssociated (x, y) is true if either of the conditions are satisfied.

- 1) x is the origin of ps_1 and y is the origin of ps_2 or
- 2) x is the terminus of ps_1 and y is the terminus of ps_2 .

This means that either $ps_1.PSNodesSequence = [x, b, c \dots n, \dots, r]$ and $ps_2.PSNodesSequence = [y, \beta, \chi, \pi \dots n, \xi, \psi]$, or $ps_1.PSNodesSequence = [a, b, c \dots n, \dots, r, x]$ and $ps_2.PSNodesSequence = [\alpha, \beta, \chi, \pi \dots n, \xi, y]$ and $n \in [[C]]$.

We say that $(ps_1, ps_2) \models \rho$ -joinAssociated (x, y).

3.3.3 Definition 9 (ρ -cpAssociated)

This is a special case of Definition 5, that captures a subclass or sibling relationship (i.e. common parent) between resources.

ρ -cpAssociated (x, y) is true if there exists two Property Sequences PS_1 and PS_2 such that $PS_1 \bowtie_{\rho} PS_2$ which satisfy ρ -joinAssociated (x, y) and, both PS_1 and PS_2 are of the form: $\text{rdf.typeOf} . (\text{rdfs:subClassOf})^*$. This relation is used to capture the notion that entities are related if they either belong to the same class or to sibling classes. For example, &r1 and &r6 are related because they are both Artists. We say that $(ps_1, ps_2) \models \rho$ -cpAssociated (x, y).

3.3.4 Definition 10 (ρ -IsoAssociated)

ρ -IsoAssociated (x, y) is true if there exists two property sequences PS_1 and PS_2 such that $PS_1 \cong_{\rho} PS_2$, and there exists and there exists ps_1 and ps_2 such that $ps_1 \in [[PS_1]]$ and $ps_2 \in [[PS_2]]$ such that, x is the origin of ps_1 and y is the origin of ps_2 . We say that $(ps_1, ps_2) \models \rho$ -IsoAssociated (x, y).

We say that x and y are *semantically associated* if either ρ -pathAssociated(x, y), ρ -cpAssociated(x, y), ρ -IsoAssociated(x, y), or ρ -joinAssociated(x, y).

3.4 Query Context

When searching for Semantic Associations in response to a query, it would be desirable to limit the search to a specific context. This is especially important in the context of the Semantic Web, where resources may belong to many different ontologies. This means that a resource could be involved in a large number of Semantic Associations. However, not all associations are important in all contexts. So we need to somehow capture the context in which a query is made in order to focus the search on only those associations that are relevant or important in that context. Context is also useful for ranking results. For this reason, we define a basic notion of context which captures the relevant schemas in that context, relevance assignments for particularly important classes or properties in the chosen schemas, some filtering conditions on data, and some user defined preferences e.g. longest paths vs. shortest paths.

We now define this notion of a context.

3.4.1 Definition 11 (Query Context).

A context C is given by a triple (RSc, L, R) , where

$RS_C = \{RS_1, RS_2, RS_3, \dots, RS_n\}$ i.e. the schemas that apply in C ,

$L = \{ \langle name_1, val_1 \rangle, \langle name_2, val_2 \rangle, \dots, \langle name_n, val_n \rangle \}$, where $name_i$ is name of some concept or property in one of the schemas in RS_C , and val_i is a positive integer value indicating its relative precedence level. The values represent a degree of importance for a particular context, and have no meaning outside the context that specifies them. In other words, they are not universal, so that different context may assign different relevance rankings to the classes and properties, or may not rank them at all. A context induces a subgraph on the model base or union of model bases in the context. The subgraph is a weighted subgraph in which the arcs for the ranked properties are assigned as weights, the values given to the properties for that context. R provides filters that may be used to reduce the size of the model base, thereby reducing the search space by specifying value restrictions on qualifying nodes. For example, we may want to restrict the search space to only European Artists and Museums, or 19th century Artists.

3.5 The ρ Operator

A ρ -Query Q is given by $\rho(x, y)$ where x and y are keys, i.e. resource URIs. The result of a ρ -Query is the set of property sequence instances p_i and pairs of property sequence instances (p_i, p_j) such that $p_i \models \rho\text{-pathAssociated}(x, y), \text{ or}$

$(p_i, p_j) \models \rho\text{-cpAssociated}(x, y) \mid \rho\text{-IsoAssociated}(x, y) \mid \rho\text{-joinAssociated}(x, y).$

4. IMPLEMENTATION APPROACHES FOR THE ρ -OPERATOR

Our strategy for implementation involves investigating alternative approaches to implementing the ρ -operator, and evaluate their merits and demerits. We consider two major categories. The first category, involves leveraging existing RDF data storage technologies. Here, a ρ -query processing layer is developed above the RDF data storage layer, which performs some of the computation and, relegates a specific portion of the computation to the data store layer. In the second approach, the implementation involves the use of a memory resident graph representation of the RDF model, along with the use of efficient graph traversal algorithms. We will outline how query processing is done using both approaches.

4.1 Exploiting RDF Data Management Systems

Figure 3 gives an illustration of the first approach (although, this is somewhat of an oversimplification, it adequate for the purposes of this discussion). Here the processing of a ρ -query is broken down to 4 phases. Phases 2 and 4 occur at the data store layer and phases 1 and 3 occur at the ρ -query processing layer. Phase 1 simply captures the resources involved in the query, and the context (though not illustrated) of the query. In the second stage, the resources are classified i.e., the classes that entities belong to, within the given context, are identified. This involves a query to the data store layer, which exploits the `rdf:typeOf` statements to answer the query. Much of the processing is done in the third phase where, *potential* paths involving the entities in the query are derived.

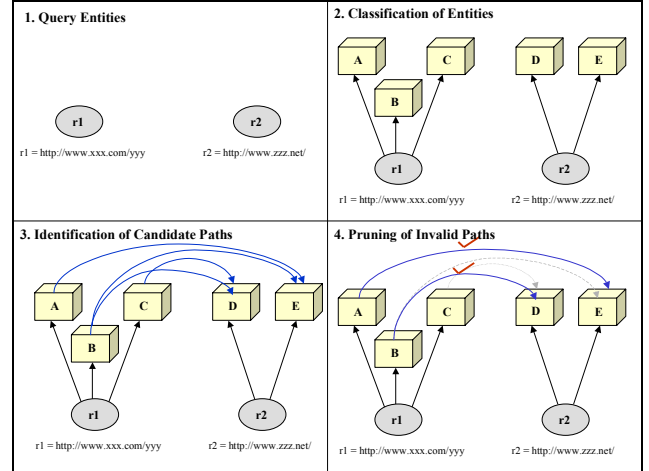


Figure 3: Illustration of ρ -Query Processing

The paths that are found at this phase are called *potential* because they only reflect possibilities and not necessarily concrete paths in the data. Potential paths are found by searching the schema to find paths involving the classes the resources belong. This allows us to get a general idea of what potential paths might exist at the data level. Our approach employs the use of a *Schema Path Index* (SPI), which provides quick access to all possible paths between any two classes for a given schema. This allows us to search a smaller search space, since schemas are relatively small compared their data component. This is done by searching for paths that involve the classes of the resources, in the schemas. Since paths in the schema do not necessarily give information about actual paths that resources participate, actual classes, and hence, *potential* paths between resources in, we need to validate these paths at the data layer to find which one of the paths are actually present in the data. This is done at the fourth phase, which also involves generating queries by creating path expressions that represent the paths found in the schema, and then executing them against the data store for validation. The paths that are valid are those that do not return an empty result.

One issue that arose in this approach, is that schema graphs by themselves do not provide complete information about paths that entities might participate in at the data level. The reason is primarily due to the multiple classification of resources allowed by the RDF data model. Consequently, there may exist paths involving entities at the data layer that will not be found in a schema's SPI. For example in Figure 1, the `paints.exhibited.title` sequence is not a sequence in either the left or right schema, but has an instance in the model base (i.e. between `r1` and the literal node "Reina Sofia Museum"). The reason for this that the node `r3`, because of its membership in both the `Museum` and the `Ext.Resource` classes, can be seen as having created an intermediate class node that collapses `Museum` and the `Ext.Resource` classes, and consequently links the `paints.exhibited` sequence to the `title` sequence.

Our approach for dealing with this situation is to manage the connections between classes created by multiple classification separately. Basically, we migrate links at the data level to the schema by creating artificial nodes that collapse the two class nodes. We do not explicitly create these nodes, instead we store

the information about the schemas that are linked due to multiple classification, in an *InterClass Index* (ISI). These nodes represent candidates for collapsing when searching for paths. So, when a query involves resources that belong to classes that do not have any paths between them or belong to different schemas, we search the ISI to find candidate nodes that if collapsed may result in a path. If no candidate nodes exist, we return an empty set as the result because we are sure we have considered all possibilities.

We are using RDFSuite[8] as the data store layer, and are also investigating the use of SESAME [16] with the hope of gaining some performance advantages because, SESAME stores in schema information in memory and, much our processing involves the use of schema information.

4.2 Using Graph Algorithms

The approach involves the use of a memory-resident graph representation of the RDF model to which graph traversals algorithms can be applied. In the case of connectivity between entities, we search for paths that connect entities or paths that originate or terminate with the entities in the query and intersect at some point. A variation of the work by [43] provides promising fast algorithms for solving path problems. Some modified versions of the algorithms from that work have also been used successfully to support transitive closure computations in large databases and disk based environments [7][5]. One of the more difficult components is checking isomorphism of paths, because the graph isomorphism is known to be NP. However, certain classes of graphs have properties making them amenable to efficient manipulation. For example, [11] describes a polynomial time algorithm for detecting isomorphism in rooted directed path graphs, which includes the exact class of graphs that are required for checking ρ -isomorphism. We are developing an implementation of this promising approach.

5. RELATED WORK

There are interesting issues that arise in the querying of semi-structured data because of their self-describing nature and weak typing mechanisms. Some of the initial insights into these issues came from [2][19]. With respect to RDF data in particular, [32][33] discuss the limitations of other semi-structured query languages e.g. XQUERY [21] LOREL[3], UnQL [17], and logical query languages e.g. TRIPLE [42], [28] for querying RDF data. Some of the reasons include for these languages, the inability to interpret both node and edge labels, lack of support for refinement of properties, and for some languages the requirement for strict typing which is present in RDF due to its support for multiple classification of resources. Another limitation cited was the inability to support seamless querying of both data and schema for schema query languages such as SchemaSQL[35], and some of the other proposed RDF query language that use a triple based model e.g. SquishQL [38], TRIPLE [42].

In general, many semi-structured query languages support navigational queries using path expressions. However, only a few [17][19], allow path variables to be used outside a FROM clause, i.e. to be returned as a result of a query. However, these languages typically assume a rooted directed graph data model which is necessarily required for RDF. Also, [19] provides some functions that range over path variables e.g. difference which returns the difference in the two sets of paths. This provides rich queries to be made about paths. However, some of the operations

that we have discussed here such as the isomorphism operation which takes into account the semantics of the nodes and edges is not supported.

Our work is analogous in its approach to [6] which provides an operator α to extend relational query languages with recursion. The operator works by computing the transitive closure of a binary predicate relation, thereby capturing only hierarchical relationships. Our approach captures a wider range of lateral relationships, because the computation of a path, i.e. involves different relations along the path. Furthermore, [6] disallows computation to be done outside the scope of the α operator in order to conform to the constraints of INF, so that operations such as an intersection of paths like our Joined Property Sequences is not supported.

Another system that bears some similarity to ours conceptually is DISCOVER [30], which provides support for querying for associations between two entities across relations in a relational database. Associations computed are based on primary key to foreign key relationship. However, because of the lack of semantics in relational database schemas, the interpretation of these associations is left to the user. Also, the range of associations that are discussed here cannot be captured by primary key to foreign key relationship.

For logic based querying systems, representing inference rules or queries to capture Semantic Associations, would require capabilities beyond FOL. For example, Kleene closure cannot be expressed in FOL systems.

6. CONCLUSION & FUTURE WORK

Most RDF query systems do not provide adequate querying paradigms to support querying for complex relationships such as Semantic Associations. Support for such querying capabilities is highly desirable in many domains. We have presented a formal framework for these Semantic Associations for the RDF data model, and reviewed some implementation strategies for computing them. Our immediate next steps involve comparison of the two implementations discussed in Section 4 over a testbed consisting of large amount of automatically extracted metadata generated using the SCORE system [41].

7. ACKNOWLEDGMENTS

Our thanks to Drs. Bob Robinson, John Miller, Krys Kochut, and Budak Arpinar for the illuminating discussions and insightful contributions. This work is funded by NSF-ITR-IDM Award # 0219649 titled "Semantic Association Identification and Knowledge Discovery for National Security Applications."

8. REFERENCES

- [1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
- [2] S. Abiteboul. Querying Semi-Structured data. In *Proc. of ICDT*, Jan 1997. <http://citeseer.nj.nec.com/abiteboul97querying.html>
- [3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured

- Data. *International Journal on Digital Libraries*, 1(1):68--88, April 1997.
- [4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [5] [R. Agrawal](#), [H. V. Jagadish](#): Hybrid Transitive Closure Algorithms. [VLDB 1990](#): 326-334
- [6] R. Agrawal. Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries. *IEEE Transactions on Software Engineering*. 14(7):879--885, July 1988
- [7] R. Agrawal, A. Borgida, and H.V. Jagadish. Efficient Management of Transitive Relationships in Large Data Bases. In *SIGMOD'89*, pages 253--262, Portland, Oregon, USA, 1989.
- [8] S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *2nd International Workshop on the Semantic Web*, pages 1--13, Hong Kong, 2001.
- [9] S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. On Storing Voluminous RDF descriptions: The case of Web Portal Catalogs. In *4th International Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, 2001. Available at <http://139.91.183.30:9090/RDF/publications/webdb2001.pdf>.
- [10] K. Anyanwu, A. Sheth, The ρ Operator: Discovering and Ranking Associations on the Semantic Web. *SIGMOD Record* ([Special issue on Amicalola Workshop](#)), December 2002
- [11] L. Babel, I. Ponomarenko, G. Tinhofer. The Isomorphism Problem for Directed Paths and For Rooted Directed Path Graphs. *Journal of Algorithms*, 21:542-564, 1996.
- [12] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
- [13] A. Branstadt, V. B. Le, J. P. Spinrad. Graph Classes: A Survey. *SIAM* 1999
- [14] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. [Extensible Markup Language \(XML\) 1.0. W3C Recommendation](#), February 1998.
- [15] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. 2000.
- [16] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Semantics for the WWW*. MIT Press, 2001. <http://citeseer.nj.nec.com/broekstra01sesame.html>
- [17] P. Buneman, M. Fernandez, D. Suciu. UnQL: A Query Language and Algebra for Semistructured *Data Based on Structural Recursion*. *VLDB Journal*, 9(1):76--110, 2000.
- [18] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *Proc. of ACM SIGMOD*, pp.413--422, 1996.
- [19] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 313--324, Minneapolis, Minnesota, May 1994
- [20] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. 2000.
- [21] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A Query Language for XML. Working draft, World Wide Web Consortium, June 2001. <http://www.w3.org/TR/xquery/>
- [22] DQL: DAML Query Language. <http://www.daml.org/2002/08/dql/>
- [23] S. Handschuh, S. Staab and F. Ciravegna, "S-CREAM - Semi-automatic CREATION of Metadata", <http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/ekaw2002cream-sub.pdf>
- [24] R. Fikes. DAML+OIL query language proposal, August 2001. <http://www.daml.org/listarchive/joint-committee/0572.html>.
- [25] R. H. Gutting. GraphDB: Modeling and querying graphs in databases. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 297--308, 1994
- [26] Y. E. Ioannidis, [R. Ramakrishnan](#), [L.Winger](#): Transitive Closure Algorithms Based on Graph Traversal. [TODS 18](#)(3): 512-576 (1993)
- [27] P. Hayes. RDF Model Theory. W3C Working Draft, September 2001.
- [28] I. Horrocks, S. Tessaris. The Semantics of DQL. <http://www.cs.man.ac.uk/~horrocks/Private/DAML/DQL-semantics.pdf>
- [29] I. Horrocks and S. Tessaris. A Conjunctive Query Language for Description Logic ABoxes. In *Proc. of AAAI-00*, 2000
- [30] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. VLDB*, Aug. 2002.

- [31] ICS-FORTH. The ICS-FORTH RDFSuite web site. Available at <http://139.91.183.30:9090/RDF> , March 2002.
- [32] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, RQL: A Declarative Query Language for RDF, *WWW2002*, May 7-11, 2002, Honolulu, Hawaii, USA (<http://139.91.183.30:9090/RDF/publications/www2002/www2002.html>)
- [33] G. Karvounarakis, V. Christophides, D. Plexousakis, Querying Semistructured (Meta) Data and Schemas on the Web: The case of RDF & RDFS, Technical Report FORTH-ICS/TR-269, February 2000.
- [34] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. 1999.
- [35] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL -- a language for querying and restructuring multidatabase systems. In *Proceedings of 22nd International Conference on Very Large Data Bases*, September 1996, Bombay, India.
- [36] M. Mannino, L. [Shapiro](#), L. Extensions to Query Languages for Graph Traversal Problems. *TKDE* 2(3): 353--363, 1990
- [37] A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.*, 24(6):1235-1258, 1995.
- [38] L. Miller., A. Seaborne, A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *Proc. of 1st International Semantic Web Conference*. ISWC2002. June9-12, 2002, Sardinia, Italy.
- [39] Inkling: RDF query using SquishQL, 2001. <http://swordfish.rdfweb.org/rdfquery/>.
- [40] A. Seaborne. RDQL: A Data Oriented Query Language for RDF Models. (<http://www-uk.hpl.hp.com/people/afs/RDQL/>) 2001.
- [41] A. Sheth, C. Bertram, D. Avant, B. Hammond, K. Kochut, Y. Warke, Semantic Content Management for Enterprises and the Web, IEEE Internet Computing, July/August 2002, pp. 80-87.
- [42] M. Sintek and S. Decker. TRIPLE---A Query, Inference, and Transformation Language for the Semantic Web. [International Semantic Web Conference \(ISWC\)](#), Sardinia, June 2002. (<http://www.dfki.uni-kl.de/frodo/triple/>)
- [43] Tarjan, R. Fast Algorithms for Solving Path Problems. *J. ACM Vol. 28, No. 3*, July 1981, pp.594-614