ρ-Queries: 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

This paper presents the notion of Semantic Associations as complex relationships between resource entities. These relationships capture both a connectivity of entities as well as similarity of entities based on a specific notion of similarity called ρ-isomorphism. It formalizes these notions for the RDF data model, by introducing a notion of a Property Sequence as a type. In the context of a graph model such as that for RDF, Semantic Associations amount to specific certain graph signatures. Specifically, they refer to sequences (i.e. directed paths) here called Property Sequences, between entities, networks of Property Sequences (i.e. undirected paths), or subgraphs of ρ-isomorphic Property Sequences.

The ability to query about the existence of such relationships is fundamental to tasks in analytical domains such as national security and business intelligence, where tasks often focus on finding complex yet meaningful and obscured relationships between entities. However, support for such queries is lacking in contemporary query systems, including those for RDF.

This paper discusses how querying for Semantic Associations might be enabled on the Semantic Web, through the use of an operator ρ. It also discusses two approaches for processing ρ-queries on available persistent RDF stores and memory resident RDF data graphs, thereby building on current RDF query languages.

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.

Copyright is held by the author/owner(s).
WWW2003, May 20-24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

1. INTRODUCTION

The Semantic Web [13] proposes to explicate the meaning of Web resources by annotating them with metadata that have been described in an ontology. This will enable machines to "understand" the meaning of resources on the Web, thereby unleashing the potential for software agents to perform tasks on behalf of humans. Consequently, significant effort in the Semantic Web research community is devoted to the development of machine processible ontology representation formalisms. Some success has been realized in this area in the form of W3C standards such as the eXtensible Markup Language (XML) [16] which is a standard for data representation and exchange on the Web, and the Resource Description Framework (RDF) [42], along with its companion specification, RDF Schema (RDFS) [17], which together provide a uniform format for the description and exchange of the semantics of web content. Other noteworthy efforts include OWL [25], Topic Maps [53], DAML+OIL [31]. There are also related efforts in both the academic and commercial communities, which are making available tools for semi-automatic [30] and automatic [49][29] semantic (ontology-driven and/or domain-specific) metadata extraction and annotation.

With the progress towards realizing the Semantic Web, the development of semantic query capabilities has become a pertinent research problem. Semantic querying techniques will exploit the semantics of web content to provide superior results than present-day techniques which rely mostly on lexical (e.g. search engines) and structural properties (e.g. XQuery [24]) of a document. There are now a number of proposals for querying RDF data including RQL [40], SquishQL [45], TRIPLE [49], RDQL [48]. These languages offer most of the essential features for semantic querying such as the ability to query using ontological concepts, inferencing as part of query answering, and some allow the ability to specify incomplete queries through 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 pervasive querying paradigm offered by these languages is one in which queries are of the form: "Get all entities that are related to resource A via a relationship R" where R is typically specified as possibly a join condition or path expression, etc. In this approach, a query is a

specification of which *entities* (i.e. resources) should be returned in the result. Sometimes the specification describes a relationship that the qualifying entities should have with other entities, e.g. a join expression or a path expression indicating a structural relationship. However, the requirement that such a relationship be specified as part of the query is prohibitive in domains with analytical or investigative tasks such as national/homeland security [11] and business intelligence, where the focus is on trying to uncover obscured relationships or associations between entities and very limited information about the existence and nature of any such relationship is known to the user. In fact, in this scenario the relationship between entities is the subject of the user's query and should being returned as the result of the query as opposed to be specified as part of the query. That is, queries would be of the form "*How* is Resource A *related* to Resource B?". For example, a security agent may want to find any relationship between a terrorist act and a terrorist organization or a country known to support such activities.

One major challenge in dealing with queries of this nature 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 a country with known terrorist groups and that they have both recently acquired some flight training, may indicate an association due to a similarity. On the other hand, the fact that a passenger placed a phone call to someone in another country that is known to have links to terrorist organizations and activities may indicate another type of association characterized by connectivity. Therefore, various notions of "relatedness" should be supported.

This paper intends to make two main contributions. First, we formalize a set of complex relationships for the RDF data model, which we call `Semantic Associations`. Second, we outline two possible approaches for processing queries about Semantic Associations through the use of an operator ρ (ρ-Queries). One of the two approaches is based on processing ρ-queries on persistent RDF data systems such as RDFSuite [8], while the other is based on processing these queries on a main memory based representation of an RDF model such as JENA [56].

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

Although there are various knowledge modeling languages that may be used on the Semantic Web such as Topic Maps [55], UML [47], DAML+OIL [31], OWL [25], etc., in this paper we have chosen to formalize Semantic Associations for the RDF data model. It should be clear that we are not suggesting that the notion of Semantic Associations only applies to RDF. On the contrary, the notion is very general and is applicable to any data model that can be represented as a graph. The choice of RDF for formalization does not confer serious problems however. In the first place, some of these other models e.g. DAML+OIL build upon RDF. Secondly, there is work on mappings from other formalisms to RDF [20][41].

Next, we will briefly summarize the RDF data model and then motivate our work with an example.

2.1 RDF

RDF [42] 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, 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` (which can be either 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 contains describes types of entities i.e. classes (e.g. Museum) and types of properties (e.g. creates) for the domain. The vocabulary description and is done using the companion specification to RDF called the RDF Schema specification [17]. For example in Figure 1, classes like `Painter`, `Museum` and properties such as `Paints`, are defined. Resources are connected to classes using an rdf:`typeof` property indicating an instantiation relationship.

2.2 MOTIVATING EXAMPLE

Although the focus of our current evaluations involves scenarios in the National Security domain, for brevity and pedagogical reasons, for this paper we have chosen to use a modified version of the example from [40]. We will now illustrate Semantic Associations by way of a simple example shown in Figure 1. 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 [40]), 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).

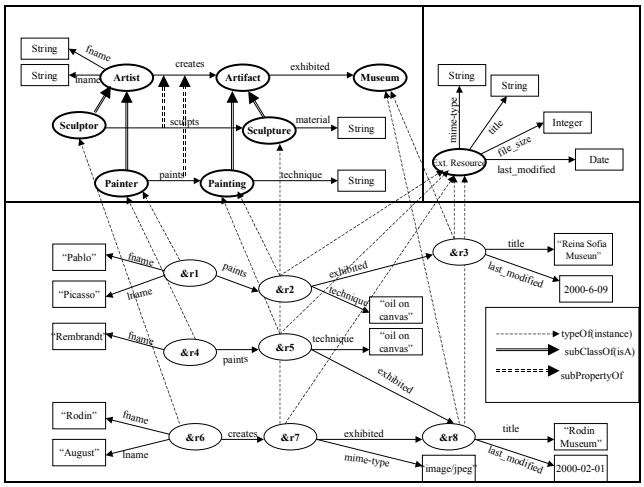

Figure 1: Cultural Portal Information in RDF

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 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. For example, the following example RQL query

```
select * from
{;Artist}@P{X}.{;Sculpture}@Q{Y}.@R{Z}
```

finds all data paths that traverse the class hierarchies `Artist` and `Sculpture`, containing three schema properties, one for each property variable (@variable). However, we notice that the query requires that a property variable be added for every edge in the required path. That is, the user is required to have some idea of at least the structure e.g. length, of the relationship. One approach that some of these systems offer to alleviate this problem is that they provide mechanisms for browsing or querying schemas to allow users to get the information they need. 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 a daunting task for the user. 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 in terms of a directed path connecting two entities. However, there are some other interesting types of relationships. Let us take for example, resources &r4 and &r6. Both resources could be said to be related because they have both 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. This kind of connectivity is an undirected path between the entities. 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 similarity, but a semantic similarity of paths (nodes and edges) that the entities are involved in. 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 sculptures and paintings are kinds of Artifacts and sculpting and painting are both kinds of creative activities (the notion of similarity is extended to properties as well).

The Semantic Associations shown in this example are fairly simple involving only short paths and are useful only for the purpose of illustration. However, in environments that support information analytics and knowledge discovery involve longer paths, especially undirected paths, which are not easily detectable by users in fast-paced environments. For example at airport security portals, agents may want to quickly determine 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 on the formalization for the RDF data model given in [40], by including a notion of a `Property Sequence`. A Property Sequence allows us to capture paths in the RDF model and forms the basis for formalizing Semantic Associations as binary relations on Property Sequences. Secondly, we some complex queries called ρ-queries for querying about Semantic Associations.

3.1 Formal Data Model

In section 2.1, we describe the RDF data model informally as a labeled directed graph. To recap, the RDF Schema specification [17] provides a special vocabulary for describing classes and properties in a domain. 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:sublassOf` 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 them to other classes. Properties may also be organized in a hierarchy using the `rdfs:subPropertyOf` property.

The formalization in [40] defines a graph data model along with a type system that connects the RDF Model & Syntax specification with the RDFS schema specification using an interpretation mechanism. It forms the basis for a typed RDF query language called RQL [40]. RQL is fairly expressive and supports a broad range of queries. Its type system T is the set of all possible types that can be constructed from the following types:

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

where τ_C indicates a class type, τ_P a property type, τ_M a metaclass type, τ_L a literal type in the set L of literal type names (string, integer, etc.), and τ_U is the type for resource *URI*s. For the RDF multi-valued types we have {.} as the *Bag* type, [.] is the *Seq*uence type, and (.) is the *Alt*ernative type. The set of values that can be constructed using the resource URIs, literals and class property names is called V. Then, the interpretation of types in T is given naturally by the interpretation function [[ ]], which is a mapping from τ to the set of values in 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 the subclasses of C. 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']] | p' is a `subPropertyOf` p$\}$

It defines an RDF schema as a 5-tuple RS = (V_S, E_S, ψ, λ, H) where: V_S is the set of nodes and E_S is the set of edges. ψ is an incidence function ψ: $E_S \rightarrow V_S \times V_S$, and λ is a labeling function that maps class and property names to one of the types in T, i.e. λ: $V_S \cup E_S \rightarrow T$. H = (N, <), where N = C $\cup$ P, C and P are the set of class and property names in RS, respectively. H is a *well-formed* hierarchy, i.e., < is a smallest partial ordering such that: if $p_1, p_2 \in P$ and $p_1 < p_2$, then p_1.domain $\leq p_2$.domain and p_1.range $\leq p_2$.range. It also formalizes an instance of an RDFS schema called a `description base` which contains all the asserted instances of classes and properties of an RDF schema.

We generalize these definitions to `sets` of RDF schemas and description bases as basic notion of context for a ρ−query.

3.1.1 Definition 1

The RDFS *schema set* of RDFS Schemas RSS = $\{RS_i: 1 \leq i \leq n\}$.

Let $\mathcal{C} = C_{S1} \cup C_{S2} \cup \ldots \cup C_{S2}$ where C_{Si} is the set of class names in schema RS_i and $\mathcal{P} = P_{S1} \cup P_{S2} \cup \ldots \cup P_{Sn}$, where P_{Si} is the set of property names in RS_i then $\mathcal{N} = \mathcal{C} \cup \mathcal{P}$.

[40] defines a description base RD which is an instance of an RDFS schema RS containing all the asserted instances of the classes and properties in RS. We generalize that definition here to the union of instances of the set of schemas in an RDFS schema set.

3.1.2 Definition 2

An instance of an RDF schema set RSS = $\{RS_1, RS_2, .. RS_n\}$, is a description base RDS defined as a 5-tuple = (V_{DS}, E_{DS}, ψ, ν, λ), where $V_{DS} = V_{D1} \cup V_{D2} \cup \ldots \cup V_{Dn}$ and V_{Di} is the set of nodes in the description base of the schema RS_i, and E_{DS} is defined similarly. ψ is the incidence function ψ: $E_{DS} \rightarrow V_{DS} \times V_{DS}$, ν is a value function that maps the nodes to the values in V i.e. ν: $V_{DS} \rightarrow V$, λ is a labeling function that maps each node either to one of the container type names (`Seq`, `Bag`, `Alt`) or to a set of class names from the schema set RSS whose interpretations contain the value of the node, and each edge e = $[v_1, v_2]$ to a property name p in RSS, where the interpretation of p contains the pair $[\nu(v_1),\nu(v_2)]$, i.e., the values of v_1 and v_2. Formally, λ: $V_D \cup E_D \rightarrow 2^{\mathcal{N}} \cup \{Bag, Seq, Alt\}$ in the following manner:

i. For a node n in RDS, $\lambda(n) = \{c \mid c \in \mathcal{C}$ and $\nu(n) \in [[c]]\}$

ii. For an edge e from node n_1 to n_2, $\lambda(e) = p \in \mathcal{P}$ and the values of n_1 to n_2 belong in the interpretation of p: $[\nu(n_1), \nu(n_2)] \in [[p]]$.

In order capture paths in the RDF graph model, we define a notion of a Property Sequence, represented in the graph as a sequence of edges (i.e. properties). There is a choice to be made in the method for realizing such a notion in a query language such as RQL. One option is to add paths or property sequences as types in a query language making them first class citizens. The second option is to realize them as complex operations such as Cartesian product, on property types. We choose the later approach because attempting to make paths as first class citizens brings up additional issues such as defining path subsumption and so on. We will now define the notion of a Property Sequence.

3.1.3 Definition 3 (Property Sequence)

A `Property Sequence` PS is a finite sequence of properties $[P_1, P_2, P_3, \ldots P_n]$ where P_i is a property defined in an RDF Schema RS_j of a schema set RSS. The interpretation of PS is given by:

$[[PS]] \subseteq \bigtimes_{i=1}^{n} [[P_i]]$ where for ps $\in$ [[PS]], called an `instance` of PS, ps[i] $\in$ $[[P_i]]$ for $1 \leq i \leq n$ and ps[i][1] = ps[i+1][0]).

ps[i][1] refers to the second element of the i^{th} ordered pair and ps[i+1][0] refers to the first element of the $i+1^{th}$ ordered pair. We define a function `NodesOfPS()` which returns the set of nodes of a Property Sequence PS, i.e.

`PS.NodesOfPS()` = $\{C_1, C_2, C_3, \ldots C_k\}$ 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 = `Creates.exhibited.title`, `PS.NodesOfPS` () = {Artist, Artifact, Museum, Ext. Resource, String}.

Let PS = $[P_1, P_2, P_3, \ldots P_n]$, a description base RDS is said to `satisfy` or be a `model` of PS (RDS |= PS) if there exists a sequence of edges $e_1, e_2, e_3, \ldots e_n$ in the description base RDS such that for all i, $\lambda(e_i) = P_i$, $\psi(e_i) = (v_i, v_{i+1})$ and $\bigtimes_{i=1}^{n} (v_i, v_{i+1})$ = ps for some ps $\in$ [[PS]].

We define a function `PSNodesSequence`() on Property Sequence instances that returns its sequence of nodes, i.e. `ps.PSNodesSequence()` = $[v_1, v_2, v_3, \ldots v_{n+1}]$. The node v_1 is called the `origin` of the sequence and v_{n+1} is called the `terminus`.

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

3.1.4 Definition 4 ($\bowtie_\rho$ Joined Property Sequences)

$PS_1 \bowtie_\rho PS_2$ is true if:

$\texttt{NodesOfPS}(PS_1) \cap \texttt{NodesOfPS}(PS_2) \neq 0$.

The Property Sequences PS_1 and PS_2 are called `joined`, and for $C \in (\texttt{NodesOfPS}(PS_1) \cap \texttt{NodesOfPS}(PS_2))$, 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`.

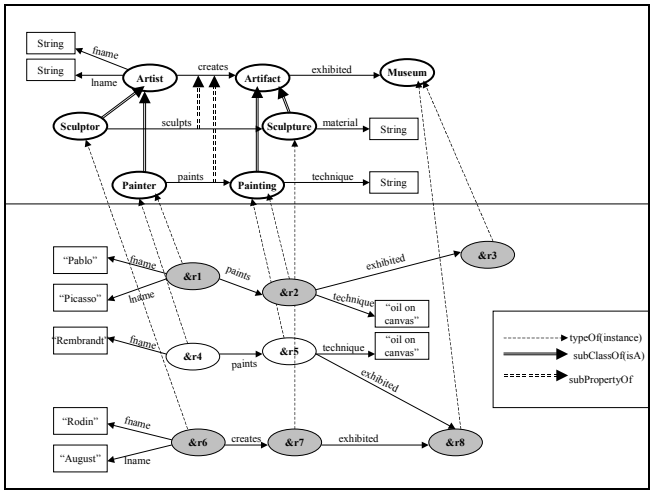

Figure 2 : Isomorphic Property Sequences

3.1.5 Definition 5 ($\cong_\rho$ ρ-Isomorphic Property Sequences)

Two property sequences $PS_1 = P_1, P_2, P_3, \ldots P_m$, and $PS_2 = Q_1, Q_2, Q_3, \ldots Q_m$, are called `ρ-isomorphic` ($PS_1 \cong_\rho PS_2$), if

for all i, $1 \leq i \leq m$: $P_i = Q_i$ or $P_i \subseteq Q_i$ or $Q_i \subseteq P_i$ ($\subseteq$ means `subPropertyOf`)

For example in Figure 2, the sequences `paints.exhibited` and `creates.exhibited` are `ρ-isomorphic` because `paints` is considered to be similar to `creates`, since `paints` is a subproperty of `creates.` Note that the example that we use here is somewhat misleading because the example shown for Joined Property Sequences also happens to be ρ-Isomorphic. However, the two notions are quite different because Joined Property Sequences are not required to be similar.

3.1.6 Definition 6 (Length)

The `length` of a Property Sequence is equal to the number of properties in the Property Sequence. In the case of a Joined Property Sequence its length is the sum of all the properties in its constituent Property Sequences, i.e. the length of the `undirected` path from origin of one Property Sequence to the origin of the other Property Sequence. For example, in Figure 2, the Joined Property Sequences [`creates.exhibited`, `paints.exhibited`] has a length of 4.

3.2 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.

3.2.1 Definition 7 (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 $ps \models$ `ρ-pathAssociated (x, y)`.

3.2.2 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, $n \in ps1.\texttt{PSNodesSequence}() \cap ps2.\texttt{PSNodesSequence}()$, 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 $ps_1.\texttt{PSNodesSequence} = [\ \mathbf{x}, b, c \ldots \mathbf{n}, .,., r\ ]$ and $ps_2.\texttt{PSNodesSequence} = [\ \mathbf{y}, \beta, \chi, \pi. . \mathbf{n}, \xi, \psi]$, or $ps_1.\texttt{PSNodesSequence} = [\ a, b, c \ldots \mathbf{n}, .,., r\ ,\mathbf{x}]\ ]$ and $ps_2.\texttt{PSNodesSequence} = [\ \alpha, \beta, \chi, \pi. . \mathbf{n}, \xi, \mathbf{y}]$ and $\mathbf{n} \in [[C]]$.

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

3.2.3 Definition 9 (ρ-cpAssociated)

This is a special case of Definition 5 that captures an inclusion 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: `rdf.typeOf.(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) $\models$ `ρ-cpAssociated (x, y)`. However, in order to reduce the possibility of meaningless associations e.g. both x and y belong to `rdfs:Resource`, we make further restrictions. We say that `ρ-cpAssociated (x, y)` is `strong` if

1) For the join node `j` of the Joined Property Sequence (inducing the association (i.e. the common parent of `x` and `y`), $j \subseteq \lceil\ \rceil$, where $\lceil\ \rceil$ called the `ceiling`, refers to the most general class in the hierarchy that is to be considered, which is usually user-specified.
2) the length of the Joined Property Sequence inducing the association is minimal. By minimal we mean that it is less than a specific value indicated by context or specified by user.

The first restriction ensures that we do go to far up the hierarchy looking for a common parent, while the second ensures that the relationship is not too distant to be meaningful in the user's context.

3.2.4 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 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) $\models$ `ρ-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).

3.3 ρ-Queries

A ρ-Query Q is defined as a set of operations that map from a pair of keys (e.g. resource URIs) to the set of Property Sequences $\mathcal{PS}$ in the following manner:

1. $\rho: \tau_U^{(2)} \rightarrow 2^{\mathcal{PS}}$
2. $\rho^{\bowtie\rho}: \tau_U^{(2)} \rightarrow 2^{\mathcal{PS}^{(2)}}$
3. $\rho^{\cong\rho}: \tau_U^{(2)} \rightarrow 2^{\mathcal{PS}^{(2)}}$

$\tau_U^{(2)} = \{ \{x, y\} : x, y \in \tau_U \text{ and } x \neq y \}$. Similarly, $\mathcal{PS}^{(2)}$ is the set of pairs of Property Sequences. In 1., we map from a pair of keys `x` and `y` to a set of Property Sequences that induces a ρ-pathAssociation of `x` and `y`. In 2., we map from (`x`, `y`) to a set of binary tuples of Property Sequences that induces either a ρ-joinAssociation or a strong ρ-cpAssociation of `x` and `y` and in 3., we map from (`x`, `y`) to a set of binary tuples of Property Sequences that induces a ρ-isoAssociation.

4. STRATEGIES FOR PROCESSING ρ-QUERIES

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, which we have developed a partial implementation for, involves leveraging existing RDF persistent 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

In this approach, we leverage existing RDF data storage technologies such as RDFSuite [8] and SESAME [18] and develop a ρ-query processing layer which performs some of the computation and, relegates a specific portion of the computation to the data store layer. Figure 3 gives an illustration of this 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 captures the query, i.e. the resources and context (i.e. schema set). 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 discussed by querying a PathGuide (a combination of index structures that stores information about paths that exist between resources classes). There are two kinds of paths that are kept in the PathGuide. The first kind of path is that which is obvious from the schema. The second kind is those paths that exist at the data level but are not evident at the schema level. This is because of the fact that the RDF data model allows multiple classifications of entities. Consequently, every instance of a multiple classification induces a connection between two classes that does not exist at the schema level, and thus is not general to all entities of those classes. Therefore, a query to the PathGuide yields *potential* property sequences paths between entities, since some of the paths are not general to entire classes but specific to the entities that are multiply classified. For example in Figure 1, the `paints.exhibited.title` sequence is not a sequence in either the left or right schema, but is present in the description base (i.e. between &r1 and the literal node "`Reina Sofia Museum`"). The reason for this is `&r3`'s membership in both the `Museum` and the `Ext.Resource` classes, this 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` property.

The fourth stage of query processing involves the validation of the paths found in the PathGuide for the specific entities in the query, by executing queries on the underlying data store. The output of this stage selects only those paths whose queries do not return an empty result.

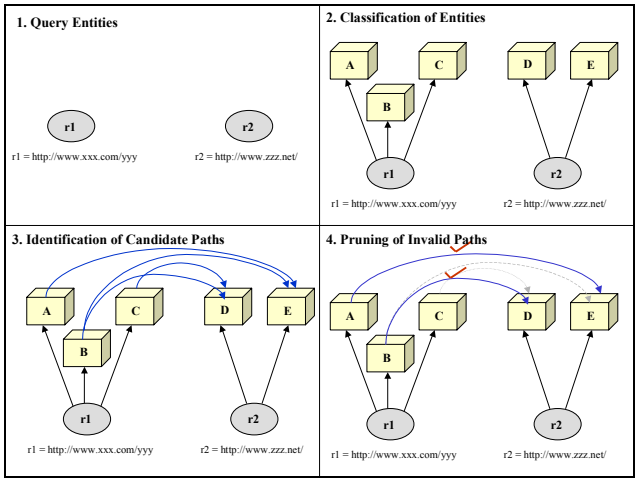

Figure 3: Illustration of ρ-Query Processing

4.1.1 Issues

Two challenges arise from storing all potential paths between classes in the PathGuide indexes. The first is that it causes the size of indexes to be quite large. Second, the potential paths found in the PathGuide in response to a query, could generate a large number of RQL queries that need to be validated at the data store layer, which slows down processing time significantly. However, heuristics could be employed to minimize these problems. For example, to reduce the size of the indices, we could choose to avoid adding every single potential path between classes in the index, but include only those whose *support value* is at least as large as a user supplied *threshold value,* where the *support value* represents the percentage of resources that are involved in

multiple classification for any given pair of classes. This means that if very few resources induce a connection between two otherwise unconnected schema classes because of a multiple classification, then we do not include in the indexes, those additional paths created due to the multiple classification, thereby reducing the size of the indices. The rationale for this is that the probability of those paths being involved in the result of a query is low, therefore the additional cost of storing the paths in the indices may not be worth it. A second heuristic is to try to prune the number of paths that need to be validated at the data storage layer. This could be done by assigning weights to Semantic Associations based on the contextual relevance and then validating only those associations with a high relevance weight. Our work in this area is still in progress.

An additional problem with processing ρ-queries on existing RDF storage systems is that some of these systems represent each property as a separate relation in a relational database model. Therefore, the computation of a Property Sequence results in a large number of joins which has a negative impact of the speed of query processing. Currently, we do not see any easy solution to this problem.

4.2 Using Graph Algorithms

This approach involves the computation of Semantic Associations on a memory-resident graph representation of the RDF model such as that provided by JENA [56], or the memory representation of the schema set as in SESAME [18], to which graph traversals algorithms can be applied. In the case of ρ-pathAssociation we can search for paths between entities, and in the case of a ρ-joinAssociation we check if the two entities belong in the same connected component. One issue with this approach is that that trying to find all paths between entities could easily lead to an exponential time algorithm. However, [52] provides promising fast algorithms for solving path problems which may be employed for such computations. In particular, it offers near-linear time algorithms for computing a path expression representing the set of all paths between nodes in a graph. Such a representation may then be processed using contextual information to prune paths that are not very relevant in the given context. In addition, other heuristics may be added. For example, a user may be asked to characterize the desired result set, e.g. shortest paths or longest paths, which will also help to reduce the result set. Similar heuristics to those discussed in the first approach that use context to prune paths based on degree of relevance can also be used here. In that case, the complexity of the problem can be analyzed along the number of semantic paths retrieved

$$\text{Complexity} = \Sigma_{(n-1)}^{(l=1)} (\#\text{ paths of length } l)\ (\text{probability of keeping path of length } l).$$

Another issue is the complexity of graph isomorphism problem which is known to be NP-complete. However, certain classes of graphs have properties making them amenable to efficient manipulation. For example, [12] 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 currently working on a prototype implementation of this approach.

5. RELATED WORK

There is some relationship between our work and that on querying object-oriented and semi-structured data using path expressions [2][3][19][22][23][24][34]. Although, these systems provide powerful and expressive capability, allowing users to query for data without having in-depth schema knowledge, most of them work on the premise that the goal of a query is to find data entities but not complex relationships such as Semantic Associations. Some of these systems [19][22] support paths as first class entities and allow for path variables to be used outside of the FROM clause, i.e. to be returned as a result of a query which suggests that queries for ρ-pathAssociations could be supported. However, they typically assume a simpler data model which is a rooted directed graph without the nuances of RDF such as multiple classification and property hierarchies. Furthermore, the more complex Semantic Associations such as the ρ-joinAssociation and ρ-Isomorphism are not supported, even in systems like [22] which provide some functions that range over path variables, e.g., the difference function which returns the difference in the set of paths that originate from two nodes.

With respect to RDF, the current generation of RDF query languages RQL [40], SquishQL [45], RDQL [48], do not support path variables as first class entities and so cannot even be used for querying for path relationships. In the case of the logic-based RDF query languages such as TRIPLE [51], the inference rules required to reason about the full range of the Semantic Associations discussed here, would require functionality beyond FOL.

The DISCOVER system [38] provides a keyword proximity search capability over relational databases, which return associations called `Joining Sequences`. Joining Sequences represent the paths connecting keywords in the query, obtained by traversing foreign key links. However, the semantics associated with these associations is not explicit, but is implicit in the database schema. Thus, the interpretation of the meaning and usefulness of the associations must be done by users. Furthermore, other more complex Semantic Associations such as the ρ-Isomorphism are not captured.

There is a common intuition underlying our work and some of the tasks related to data mining, in that they both involve discovering relationships. However, there are significant differences in the goals, methods and results produced by the both kinds of systems. The first difference is articulated in a statement made in [32], where data mining is said to be opportunistic while information access techniques (such as ours) are goal-driven. Traditional data mining [21][26] focuses on discovering patterns and relationships in data that can be used to develop models of the data. In association analysis [7], rules that associate attribute-value pairs are learned from *patterns* of co-occurrences of attribute values in *data*, which capture co-occurrence *relationships between attributes*. On the contrary, we do not try to learn patterns from data rather, we provide specific rules for inferring *relationships between entities* by looking at property value dependencies, and focus on providing methods for verifying whether these kinds of associations exist between entities. That is, we identify meaningful *sequences* of binary predicates while data mining association rules involve *sets* of attribute value pairs. Therefore, we view data mining as a complimentary technology. For example, the association rules learnt from patterns in data can

provide knowledge that can be used to guide the search for Semantic Associations or to rank resulting Semantic Associations based on how close the follow the patterns.

An initial discussion on Semantic Associations is made in [10].

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. There are many open research issues that we plan to focus on in the near future. First, it may be necessary to develop data organization techniques for data that will adequately support the kinds of queries discussed here. Such techniques should eliminate the need for an excessive number of expensive computations such as joins during query processing. Secondly, we plan to develop techniques for dealing with the space complexity problem of the indices used in the PathGuide. For example we may use encoding schemes that compress path information, or heuristics for managing the size of the indices. Another top priority is the development of context-sensitive ranking algorithms that assign higher weights to results that are most relevant in the query context. Finally, we will perform a comparative study of the two implementation strategies 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, and to Boanerges Aleman-Meza on his revision comments. We are also indebted to Dr. Vassilis Christophides whose comments and suggestions were invaluable in preparing the final version of the paper.
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. Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries. IEEE Transactions on Software Engineering. 14(7):879-- 885, July 1988.

[6] 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.

[7] R. Agrawal, T. Imielienski and A. Swami. Mining Assocation Rules between Sets of Items in Large Databases. Proc. Conf. On Management of Data. Washington, DC, USA, 207--216. ACM Press, New York, NY USA 1993.

[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] D. Avant, M. Baum, C. Bertram, M. Fisher, A. Sheth, Y. Warke, "Semantic Technology Applications for Homeland Security," Proc. of the 11th Intl Conf. on Information and Knowledge Management (CIKM 2002), McLean, VA, November 4-9, 2002, pp. 611--613.

[12] 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.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, May 2001.

[14] B. Berendt, A. Hotho, G. Stumme. Towards Semantic Web Mining. In Proceedings of the International Semantic Web Conference, pp. 264--278, Sardinia, Italy. June 2002.

[15] A. Branstadt, V. B. Le, J. P. Spinrad. Graph Classes: A Survey. SIAM 1999.

[16] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998.

[17] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. 2000.

[18] 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.

[19] 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.

[20] W. W. Chang, A Discussion of the Relationship Between RDF-Schema and UML. A W3C Note, NOTE-rdf-uml-19980804.

[21] M. Chen, J. Han and P. Yu. Data Mining: An Overview from the Database Perspective. IEEE Trans. On Knowledge and Data Engineering. Vol. 8. No. 6. December 1996.

[22] 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, pp. 313--324, Minneapolis, Minnesota, May 1994.

[23] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In Proc. of ACM SIGMOD, pp. 413--422, 1996.

[24] 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/

[25] M. Dean, D. Connolly, F. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider, and L. Stein. OWL Web Ontology Language 1.0 Reference, W3C Working Draft 29 July 2002. http://www.w3.org/TR/owl-ref/.

[26] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusany. Advances in Knowledge Discovery and Data Mining.. AAAI/MIT Press 1996.

[27] R. Fikes. DAML+OIL query language proposal, August 2001. http://www.daml.org/listarchive/joint-committee/0572.html.

[28] R. H. Guting. GraphDB: Modeling and querying graphs in databases. In Proceedings of the International Conference on Very Large Data Bases, pp. 297--308, 1994.

[29] B. Hammond, A. Sheth, and K. Kochut, Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content, in Real World Semantic Web Applications, V. Kashyap and L. Shklar, Eds., IOS Press, December 2002, pp. 29--49.

[30] S. Handschuh and S. Staab. Authoring and annotation of web pages in CREAM. In The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, 7-11, May, 2002

[31] F. Harmelen, P. F. Patel-Schneider, I. Horrocks, eds. Reference Description of the DAML+OIL (March 2001) ontology markup language.

[32] M. Hearst. Distinguishing between Web Data Mining and Information Access. Position statement for Web Data Mining KDD 97.

[33] Y. E. Ioannidis, R. Ramakrishnan, L. Winger: Transitive Closure Algorithms Based on Graph Traversal. TODS 18(3): 512--576 (1993).

[34] Y. E. Ioannidis, Y. Lashkari. Incomplete Path Expressions and their Disambiguation, In Proc. of the 1994 ACM SIGMOD, International Conference on Management of Data. p.138-149, May 24-27, 1994, Minneapolis, Minnesota, United States.

[35] P. Hayes. RDF Model Theory. W3C Working Draft, September 2001.

[36] I. Horrocks, S. Tessaris. The Semantics of DQL. http://www.cs.man.ac.uk/~horrocks/Private/DAML/DQL-semantics.pdf

[37] I. Horrocks and S. Tessaris. A Conjunctive Query Language for Description Logic Aboxes. In Proc. of AAAI-00, 2000.

[38] V. Hristidis and Y. Papakonstanti-nou. DISCOVER: Keyword search in relational databases. In Procs. VLDB, Aug. 2002.

[39] ICS-FORTH. The ICS-FORTH RDFSuite web site. Available at http://139.91.183.30:9090/RDF, March 2002.

[40] 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.

[41] M. S. Lacher and S. Decker. On the Integration of Topic Maps and RDF Data. In Proc. of Semantic Web Working Symposium. Palo Alto. California. August 2001.

[42] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. 1999.

[43] M. Mannino, L. Shapiro, L. Extensions to Query Languages for Graph Traversal Problems. TKDE 2(3): 353--363,1990.

[44] A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. SIAM J. Comput., 24(6):1235--1258, 1995.

[45] L. Miller, A. Seaborne, A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In Proc. of 1st International Semantic Web Conference. ISWC2002. June 9-12, 2002, Sardinia, Italy.

[46] A. Nanopoulos. Y. Manolopoulos. "Mining Patterns from Graph Traversals", Data and Knowledge Engineering, Vol.37, No.3, pp.243-266, 2001.

[47] J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[48] A. Seaborne. RDQL: A Data Oriented Query Language for RDF Models. 2001. http://www.hpl.hp.com/semweb/rdql-grammar.html

[49] 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.

[50] A. Sheth, S. Thacker and S. Patel. Complex Relationship and Knowledge Discovery Support in the InfoQuilt System. VLDB Journal. September 25, 2002.

[51] 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/

[52] Tarjan, R. Fast Algorithms for Solving Path Problems. J. ACM Vol. 28, No. 3, July 1891, pp. 594—614.

[53] DQL: DAML Query Language. http://www.daml.org/2002/08/dql/

[54] Inkling: RDF query using SquishQL, 2001. http://swordfish.rdfweb.org/rdfquery/.

[55] ISO/IEC 13250: 2000 Topic Maps, Jan, 2000. http://www.topicmaps.org/

[56] JENA – A Java API for RDF.

[57] Whitepaper on National Security and Intelligence, Semagix Inc. 2002. http://www.semagix.com/pdf/national_security.pdf