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 $\rho$-isomorphism. In an RDF graph data model, they may represented as sequences (i.e. paths) between entities, networks of sequences, or a subgraph containing $\rho$-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 $\rho$. Finally, it discusses two approaches for implementing the $\rho$ 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,

1. INTRODUCTION
The Semantic Web [12] proposes to explicate the meaning of Web resources by annotating them with metadata that have 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 ResourceA related to ResourceB?”, 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 $\rho$-Queries, which are queries about Semantic Associations using an operator, $\rho$, and a discussion of two implementation strategies for the $\rho$ 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 $\rho$ 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 ($\&r2$, $\&r5$, etc.)
Rodin there is a relationship between entities, 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).

![Diagram](image)

**Figure 1: Cultural Portal Information in RDF**

Typically, a query language allows you to find all entities that are related to 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 “&r1 is related to &r2?” 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 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 Sofia 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: C is the set of all class names in an RDF Schema, P is the set of property names, and L is the set of literal type names, e.g. string, integer. The type system T is the set of all possible types that can be constructed from the following types:

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

where \( \tau_C \) indicates a class type, \( \tau_P \) a property type, \( \tau_M \) a metaclass type, \( \tau_L \) a literal type in L, \( \tau_U \) is the type for resource URIs. For the RDF multi-valued types we have \( [\cdot] \) is the Bag type, \( (\cdot) \) is the Sequence type, and \( (\cdot, \cdot) \) is the Alternative type. The set of values that can be constructed using the resource URIs, literals and class property names is known. We therefore extend this with the notion of a Property Sequence, which are first class entities that variables can range over.

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 \( [\tau_1, \tau_2, \tau_3, \ldots, \tau_n] \) for unnamed ordered tuples by \( [v_1, v_2, v_3, \ldots, v_n] \) where \( v_i \) is of type \( \tau_i \). Formally, the interpretation of a sequence type \( [\tau] \) is given by

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

In this approach, it is possible to define a sequence of type \( [\tau_1, \tau_2, \tau_3, \ldots, \tau_n] \) to capture a sequence of properties. 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 \( P_1, P_2, P_3, \ldots, P_n \) where each \( P_i \) is a property defined in an RDF Schema RS. The interpretation of PS is given by:

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

For \( p \in [PS] \), ps is called an instance of the Property Sequence PS and is represented in our graph data model as the path \( v_1, p_1, v_2, p_2, \ldots, v_n \). 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 \( v_i \) is called the origin of the sequence and \( v_n \) is the terminus. We define a function NodesOfPS() which returns the set of nodes of a Property Sequence PS, i.e.

\[ PS.NodesOfPS() = \{ C_1, C_2, C_3, \ldots, C_k \} \]

where \( C_i \) is a class in either the domain or range of some Property \( P_i \) in PS, \( 1 \leq i \leq n \).

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

\[ ps.PSNodesSequence() = \{ v_1, v_2, v_3, \ldots, v_n \} \]

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

3.2.2 Definition 4 (Joined Property Sequences)

The Property Sequences PS1 and PS2 are called joined, and for C \( \in \) (NodesOfPS(PS1) \( \cap \) NodesOfPS(PS2)) 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.

NodeOfPS(PS1) \( \cap \) NodeOfPS(PS2) \( \neq \) 0.
The similarity of the sequences is not required. In the case of joined property sequences, the only case of an isomorphism, similarity is maintained along the path. However, the two notions are quite different. In the example shown for Joined Property Sequences also happens to be isomorphic. With respect to the similarity of nodes, we use the distance function to ensure that the relationship between the nodes is isomorphic. 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) \( \exists x, y, z \ ( x \in \text{domain}(P_i) \land y \in \text{domain}(Q_i) \land z = \text{LeastUpperBound}(P_i, Q_i)) \), and \( \text{distance}(x, z) \) and \( \text{distance}(y, z) \) is minimal.

\( \rho \)-Isomorphic. The first definition ensures a similarity of edges while the second ensures a similarity of nodes. For example, in Figure 2, although &rl is a painter and &r6 is a sculptor both are similar because they are both Artists. Also, because paints is a subproperty of paints, we consider them similar. Therefore the sequences paints.exhibited and creates.exhibited are \( \rho \)-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 \( \rho \)-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 (\( \rho \)-pathAssociated)**

\( \rho \)-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 \( \rho \)-pathAssociated \((x, y)\) written as \( ps \models \rho \)-pathAssociated \((x, y)\).

**3.3.2 Definition 8 (\( \rho \)-joinAssociated)**

Let \( PS_1 \) and \( PS_2 \) be two Property Sequences such that \( PS_1 \cong \rho \) \( PS_2 \), 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, \( n \in \text{ps1.PSNodesSequence()} \cap \text{ps2.PSNodesSequence()} \), then \( \rho \)-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 \( ps_2 \).
2) \( x \) is the terminus of \( ps_1 \) and \( y \) is the terminus of \( ps_2 \).

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

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

**3.3.3 Definition 9 (\( \rho \)-cpAssociated)**

This is a special case of Definition 5, that captures a subclass or sibling relationship (i.e. common parent) between resources. \( \rho \)-cpAssociated \((x, y)\) is true if there exists two Property Sequences \( PS_1 \) and \( PS_2 \) such that \( PS_1 \cong \rho \) \( PS_2 \) which satisfy \( \rho \)-joinAssociated \((x, y)\) and, both \( PS_1 \) and \( PS_2 \) are of the form: \( \text{rdf.typeOf.(rdfs:subclassOf)}^* \).

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, &rl 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 (\( \rho \)-IsoAssociated)**

\( \rho \)-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 \( \rho \)-pathAssociated \((x, y)\), \( \rho \)-cpAssociated \((x, y)\), \( \rho \)-IsoAssociated \((x, y)\), or \( \rho \)-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 \( (RS, L, R) \), where
\( RS_C = \{ RS_1, RS_2, RS_3, \ldots, RS_n \} \) i.e. the schemas that apply in \( C \), \( L = \{ \langle name_1, val_1 \rangle, \langle name_2, val_2 \rangle, \ldots, \langle name_n, val_n \rangle \} \), 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 \( \rho \)-Operator

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

\[ (p_i, \rho) \models \rho\text{-cpAssociated}(x, y) \mid \rho\text{-IsoAssociated}(x, y) \mid \rho\text{-joinAssociated}(x, y) \]

4. IMPLEMENTATION APPROACHES FOR THE \( \rho \)-OPERATOR

Our strategy for implementation involves investigating alternative approaches to implementing the \( \rho \)-operator, and evaluate their merits and demerits. We consider two major categories. The first category, involves leveraging existing RDF data storage technologies. Here, a \( \rho \)-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 \( \rho \)-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 \( \rho \)-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 \( \text{rdf:typeOf} \) 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 \( \rho \)-Query Processing](image)

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 \text{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 \( r_1 \) and the literal node “Reina Sofia Museum”). The reason for this that the node \( \& r_3 \), 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 \text{paints.exhibited} sequence to the \text{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 p-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 express in FOL systems. 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.

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


