2009

PrOM: A Semantic Web Framework for Provenance Management in Science

Satya S. Sahoo
Wright State University - Main Campus

Roger Barga

Amit P. Sheth
Wright State University - Main Campus, amit@sc.edu

Krishnaprasad Thirunarayan
Wright State University - Main Campus, t.k.prasad@wright.edu

Pascal Hitzler
pascal.hitzler@wright.edu

Repository Citation
Sahoo, S. S., Barga, R., Sheth, A. P., Thirunarayan, K., & Hitzler, P. (2009). PrOM: A Semantic Web Framework for Provenance Management in Science. *Kno.e.sis Center Technical Report*.
https://corescholar.libraries.wright.edu/knoesis/445

Kno.e.sis Center, Wright State University Technical Report knoesis-TR-2009, 2009

PrOM: A Semantic Web Framework for Provenance Management in Science

Satya S. Sahoo[1], Roger Barga[2], Amit Sheth[1], Krishnaprasad Thirunarayan[1], Pascal Hitzler[1]

[1]Kno.e.sis Center
Computer Science and Engineering Department,
Wright State University,
Dayton, OH, USA 45435

{sahoo.2, amit.sheth, t.k.prasad,
pascal.hitzler}@wright.edu

[2]Microsoft Research
One Microsoft Way,
Redmond, WA 98052, USA

barga@microsoft.com

ABSTRACT

The eScience paradigm is enabling researchers to collaborate over the Web in virtual laboratories and conduct experiments on an industrial scale. But, the inherent variability in the quality and trust associated with eScience resources necessitates the use of provenance information describing the origin of an entity. Existing systems often model provenance using ambiguous terminology, have poor *domain semantics* and include modeling inconsistencies that hinders interoperability. Further, mere collection of provenance information is of little value without a well-defined and scalable query mechanism.

In this paper, we present "PrOM", a framework that addresses both the modeling and querying issues in eScience provenance management. The theoretical underpinning for PrOM consists of, (a) a novel foundational ontology for provenance representation called "Provenir", and (b) the first set of query operators to be defined for provenance query and analysis. The PrOM framework also includes a scalable provenance query engine that supports complex queries (high "expression complexity") over a very large real world dataset with 308 million RDF triples. The query engine uses a new class of materialized views for query optimization that confers significant advantages (up to three orders of magnitude) in query performance.

Categories and Subject Descriptors

H.2.m **[Miscellaneous],** H.2.8 **[Database Applications],** H.2.1 **[Data Models]**

Keywords

Provenance framework, eScience, Provenir foundational ontology, Query operator, Materialized Provenance Views, scalability

1. INTRODUCTION

Scientists, in multiple domains, are leveraging distributed data and computing resources over the Web to achieve their objectives faster, more efficiently and on an industrial scale. This eScience paradigm includes domains such as biology [1], oceanography [2], and astronomy [3], involving the use of a variety of resources such as sensors, Web-based computational tools, and data repositories. In eScience, the lineage of a resource is important metadata not only for researchers, but also for many analytical tools in data mining, data integration, and knowledge discovery. Provenance metadata, from the French word *provenir* meaning "to come from," represents the lineage or history of an entity.

Provenance has been studied from multiple perspectives in computer science, such as database provenance [4], [5] [6], and scientific workflow provenance [7] [8], but there are many open research issues in provenance management. An example is the need for a common representation model for provenance to facilitate provenance interoperability, provenance integration across different projects, and enforce consistent use of terminology. A common provenance model should also closely reflect domain-specific details (*domain semantics*) that are essential to answer end user queries in real world applications. Ontologies are considered a suitable modeling solution for these requirements and in addition they also support reasoning to discover implicit knowledge over large datasets [9] [10].

Ontologies are used in many scientific domains [11] and are gaining rapid community acceptance, for example the National Center for Biomedical Ontologies (NCBO)[1] lists 166 ontologies in the life science domain. Provenance ontologies not only reduce terminological heterogeneity to facilitate interoperability, but also support discovery of implicit knowledge over large datasets using reasoning tools. Unfortunately, provenance information varies across different domains and the creation of a single, all encompassing provenance ontology is impractical. To address this challenge, this paper proposes a modular, multi-ontology approach centered on a foundational ontology for provenance, called Provenir. The Provenir ontology ensures (a) a common modeling basis, (b) conceptual clarity of provenance terms, and (c) use of ontology design patterns for consistent modeling [12]. The Provenir ontology can be extended to create interoperable domain-specific provenance ontologies and this is demonstrated in the paper through the creation of an oceanography domain-specific provenance ontology called Trident.

In addition to provenance representation, a well-defined and scalable query infrastructure is essential to enable real world applications to utilize provenance information. Existing provenance systems often use a generic query mechanism, such as SQL, or ad hoc implementations that satisfy the requirements of the given application. In this paper, we argue for a dedicated provenance query infrastructure that can support provenance queries efficiently and scale with large datasets. The provenance literature features a variety of queries often reflecting the requirements of a specific application [13]. But, to date there has been no systematic study of provenance query characteristics. We introduce a classification scheme for provenance queries not only to categorize provenance queries, but also use the classification scheme to define a set of specialized query operators. The provenance query operators are defined in terms of the Provenir ontology, which enables the query operators to be used with any domain-specific provenance ontology that extends the Provenir ontology.

WWW 2010, April 24-30, 2010, Raleigh, North Carolina.

[1] http://www.bioontology.org/

The Provenir ontology together with the query operators constitutes the theoretical underpinning of the PrOvenance Management (PrOM) framework for eScience. Further, the PrOM framework includes the implementation of a scalable provenance query engine over a RDF data store that uses the SPARQL RDF query language [14] to query provenance information. Provenance queries from the Neptune oceanography project [2] are used to evaluate the performance of the provenance query engine implementation. In the next section, we give an overview of the Neptune project and the oceanography scenario used as the running example in this paper.

1.1 Motivating Scenario in Oceanography

The Neptune project [2], led by the University of Washington, is an ongoing initiative to create a network of instruments widely distributed across, above, and below the seafloor in the northeast Pacific Ocean. We consider a simulated scenario, illustrated in Figure 1, involving data collection by ocean buoys (containing a temperature sensor and an ocean current sensor), and data processing by a scientific workflow that creates visualization charts as output.

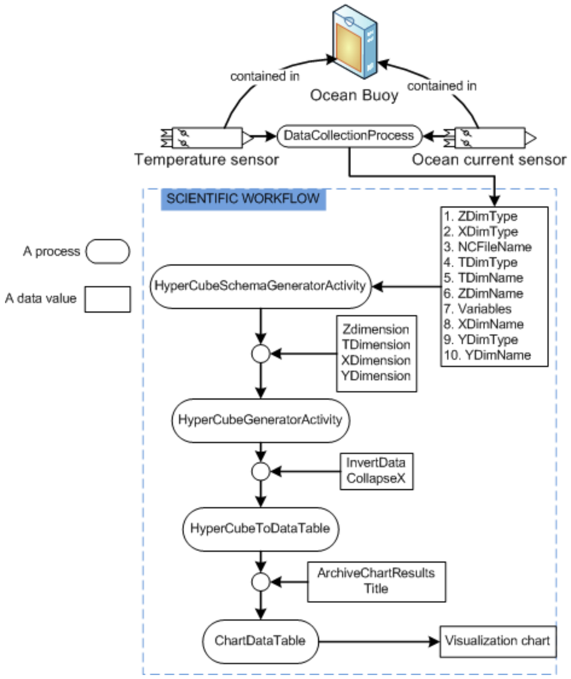

Figure 1: A simulated oceanography scenario from the Neptune oceanography project

We consider two scenarios that require the use of provenance information for analyzing and managing the data in this project:

1. One of the ocean buoys is found to be damaged due to a recent storm. Hence, all visualization charts created using data from this ocean buoy, after it was damaged, need to be discarded and new charts should be created using correct sensor data. This is a typical provenance related query and can be addressed using two approaches. In the first approach, the provenance information of each visualization chart can be analyzed to locate the relevant set of possibly inaccurate visualization charts. In the second approach, a set of constraints can be defined on the provenance information to retrieve the data entities that satisfy these constraints, for example "given the identifier of the damaged ocean buoy, locate all visualization charts that used data from this sensor" (generated after the ocean buoy was damaged).

2. Another typical scenario involves the comparison of provenance information associated with given results. For example, oceanography researchers can query for two visualization charts that were generated from sensor data collected under equivalent weather conditions. The user can define the context for comparison of the provenance information in terms of wind speed, temperature, and precipitation.

We use these two scenarios as the running example to motivate the modeling decisions in the Provenir ontology and also to define the functional semantics of the provenance query operators. We also use the data and the associated provenance information from this scenario to evaluate the performance of the provenance query engine.

The rest of the paper is organized as follows: In section 2, we describe the motivation, constituent components, and use of the Provenir ontology as a foundational model for provenance. In Section 3, we introduce a classification scheme for provenance queries and use this classification scheme to define a set of provenance query operators. In section 4, we describe the implementation and evaluation results of the provenance query engine for complex queries over very large real world datasets. We describe related provenance work in Section 5 and finally conclude in Section 6.

2. PROVENANCE MODELING

Traditionally in scientific experiments, provenance has been represented as hand-written notes to keep track of experiment conditions, results, and project details. ProPreO was one of the first provenance ontologies developed to model provenance in high-throughput proteomics experiments [15]. Similar efforts include the Microarray Gene Expression Data (MGED) ontology that was created to track provenance in microarray experiments [16], while the Functional Genomics Investigation Ontology (FuGO) was created to model provenance in functional genomics. These multiple projects in provenance management highlight the need for interoperability of provenance information, which can be facilitated if the provenance ontologies share a common modeling basis and uniform use of terms. Foundational upper-level ontologies have traditionally been used to enforce these consistent design principles and to facilitate domain ontology interoperability and integration [12].

In this section, we describe the creation of the provenance upper-level ontology called Provenir. We also demonstrate that the Provenir ontology is a well-engineered foundational ontology that can easily be extended by creating a domain-specific provenance ontology for the Neptune oceanography project called Trident.

2.1 The Provenir ontology: A Foundational Model of Provenance

The primary challenge in defining the structure of the Provenir ontology is to strike a balance between the abstract upper-level ontologies, such as the Suggested Upper Merged Ontology (SUMO), and the detailed domain-specific provenance ontologies. The Provenir ontology represents the set of provenance terms that are common across domains and can easily be extended by developers of domain-specific provenance ontologies according to their requirements. This not only significantly reduces the workload of ontology developers, but also ensures modeling consistency.

The top-level classes in the Provenir ontology are derived from two primitive concepts of "occurrent" and "continuant" defined in philosophical ontology [17]. Continuant represents "*… entities which endure, or continue to exist, through time while undergoing different sorts of changes, including changes of place*" [17] and occurrent represents *"…entities that unfold themselves in successive temporal phases*" [17]. It is easy to see that occurrent corresponds to the "process" entities used in science, such as experiment processes and data processing. Other entities can be categorized into two sub-types of continuants namely, "agents" such as temperature sensor and "data" such as visualization charts (examples from the Neptune project described in Section 1.1).

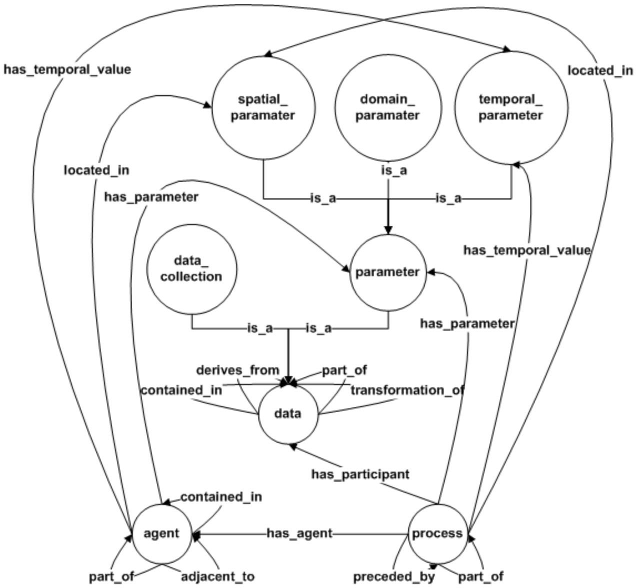

Figure 2: The provenir ontology schema

The three concepts of `process`[2], `data`, and `agent` form the top-level classes in the Provenir ontology (Figure 2). Further, the explicit modeling of relations as first class entities is an important characteristic of Semantic Web modeling languages. For the Provenir ontology we adapted the properties defined in the Relation ontology (RO) [17] to link provenance terms. The RO was created by the Open Biomedical Ontologies (OBO) Foundry and defines a set of ten primitive properties with well-defined "domain" and "range" values. These properties were mapped to appropriate Provenir classes with restrictions on their domain and range values as required for provenance modeling.

For example, the class `data` is linked to a `process` as either input or output value and this is modeled by the relation "`has_participant`" (*domain*: `process`, *range*: `data`). Similarly, the notion that an `agent` initiates, modifies or terminates a `process` is captured by the relation "`has_agent`" (*domain*: `process`, *range*: `agent`). We note that the class `data` is linked to `agent` only through an intermediary `process`, for example a list of peptides (`data`) is linked to a mass spectrometer (`agent`) through the protein analysis `process`. The Provenir ontology also differentiates between the `data` values that undergo change in a process, for example conversion of sensor data to create visualization charts, and `parameter` values that influence the behavior of a `process` (Figure 2).

The Provenir ontology further classifies parameters along the well-defined spatial (`spatial_parameter`), temporal (`temporal_parameter`), and thematic (`domain_parameter`) dimensions. The geographical location of a sensor (`agent`) is an example of `spatial_parameter`. The `parameter` entities are linked to the `data` and `agent` classes by the relation `has_parameter`. A detailed description of the Provenir ontology classes and properties appears in [18]. The Provenir ontology is represented in OWL DL, the decidable profile of the W3C Web Ontology Language (OWL) with a description logic expressivity of $\mathcal{ALCH}$. This ontology is under active consideration at the OBO Foundry for listing as a foundational model of provenance.

The Provenir ontology has been extended to create three domain-specific provenance ontologies; the ProPreO ontology (described at the start of this section) has been re-structured to extend the Provenir ontology. The Parasite Experiment ontology [19] is the second domain-specific ontology created by extending the Provenir to model provenance in gene knockout and parasite strain creation experiments, both ProPreO and the Parasite Experiment ontology are listed at NCBO. In the next section, we describe the third domain-specific ontology called Trident that extends the Provenir ontology to model the provenance information for the Neptune oceanography project.

2.2 Trident ontology: Modeling Oceanography Provenance

The provenance information for the Neptune project can be divided into two categories of (a) workflow provenance corresponding to the components of the scientific workflow, and (b) domain-specific provenance capturing the domain semantics such as details of the sensors and ocean buoy. Existing provenance systems have primarily focused on only workflow provenance [20] while partially or completely ignoring *domain semantic*s. For example, in the Neptune oceanography scenario details describing the sensors and location of ocean buoys are critical provenance information.

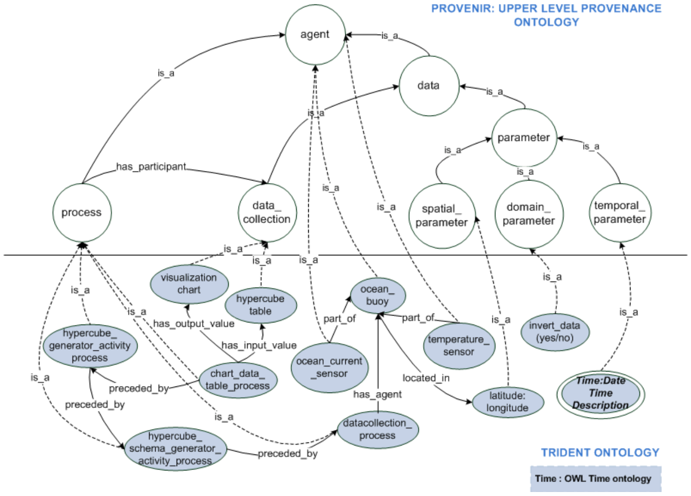

Figure 3: Trident ontology schema extending the Provenir ontology

The Provenir ontology and its extension in form of the Trident ontology incorporate both the domain semantics and the

[2] Ontology classes and relations are denoted using `courier new` font.

workflow-specific provenance. The Trident ontology extends the Provenir agent class to model the temperature and ocean current sensors as well as the ocean buoy. Further, the sensors are linked to specific ocean buoys using the contained_in relation that enables tracking of sensor data from the damaged ocean buoys. Figure 3 gives an overview of the Trident ontology schema and maps its classes to the Provenir ontology.

The formal OWL-based representation of provenance information in the Trident ontology enables the use of the rich set of Semantic Web reasoning mechanism [10] for provenance query and analysis. In the next section, we introduce a dedicated query infrastructure for provenance.

3. PROVENANCE QUERY and ANALYSIS

The provenance literature discusses a variety of queries that are often executed using generic or ad-hoc query mechanisms. Provenance queries in workflow systems focus on execution of computational process and their input/output values (for example, the queries featured as part of the Provenance Challenge [13]). Provenance queries in relational databases trace the history of a tuple or data entity [21]. In contrast, scientists formulate provenance queries that incorporate complex domain semantics using application-specific terminology [22]. But, without a systematic study to identify and understand provenance query characteristics, it is difficult to create a dedicated and well-defined provenance query infrastructure to support this large variety of queries across multiple domains and projects. In the next section we define the first classification scheme for provenance queries.

3.1 Provenance Query Classification

We classify provenance queries into three broad categories using the input parameters and the query results as the classification metrics.

***Category I*: Retrieve provenance information**: This category of queries, given a data entity, returns the complete provenance information that influenced the current state of the data entity. The retrieval of provenance information of a *"HyperCube*" object with identifier *"HyperCube85357162234026*" is an example of this category of queries from the Neptune oceanography scenario. These queries are the most common category of provenance queries.

***Category II*: Retrieve data entities that satisfy provenance constraints**: An exact opposite perspective to the first category of query is to retrieve a set of data entities that satisfy a given set of constraints defined over the provenance information. A query to retrieve "*chart visualization*" files created using data from the damaged ocean buoy with identifier "*oceanBuoy7044*" located at geographical coordinates "*475111N:1222118W*" between "*April 21, 2003 to May 2, 2003*" is an example of this category of queries. This class of queries does not feature frequently in the provenance literature but is important to identify data entities with similar provenance characteristics as demonstrated by the example query to retrieve data from the damaged ocean buoy.

***Category III*: Operations on provenance information**: This category of queries involves modification or comparison of provenance information. For example, the accurate comparison of two sets of ocean temperature observations requires the comparison of the associated provenance information to verify that they were created under equivalent experimental conditions and generated by the same type of sensors.

These three categories of provenance queries have different sets of input/output values and execution semantics. The creation of a standard query mechanism to address each category of provenance query will not only help end-users, but also facilitate the creation of a practical provenance query infrastructure. In the next section, we use this provenance query classification scheme to define a first set of provenance query operators.

3.2 Provenance Query Operators

Our query operators are defined in terms of the Provenir ontology schema that allows them to be used to query any domain-specific provenance ontology (that extends the Provenir ontology). Further, the input, intermediate, and output values of the query operators are strictly typed using the Provenir ontology classes. This section describes both functional semantics of the query operators using the Neptune oceanography example scenario and the execution semantics using set-theoretic representation.

3.2.1 Retrieval of Provenance Information

1. ***provenance ( )*** - The first query operator is defined to support ***Category I*** provenance queries (Section 3.1) and is a closure operation on the provenance information. The operator takes as input any given data entity and returns the complete provenance information associated with the data entity.

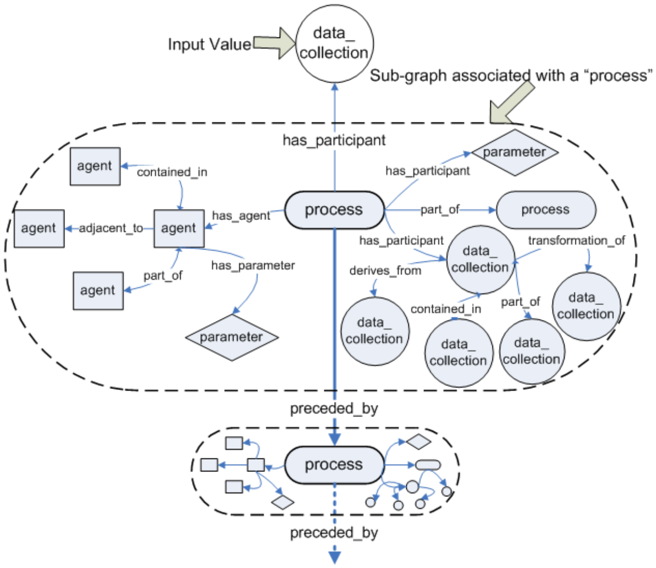

Figure 4: The result graph from the *provenance ( )* query operator

The *provenance ()* query operator defines a pattern constructed in two steps, namely (a) Initialization phase, and (b) Recursive phase. In the initialization phase, the individuals of the class process linked to the input value by property has_participant (Figure 4) are added. In the recursive phase, these process individual are used to locate all individuals of the class process, linked by the property preceded_by, which is implemented as a transitive closure function for the <process, preceded_by) class-property pair of entities. Further, all individuals of the agent, parameter, and data_collection classes linked to process individuals are added to the query result (Figure 4). The query operator is defined formally using set-theoretic representation, where the notation <*s*, *p*, *o*> stands for a subject, predicate, object triple as used in the Provenir ontology.

Definition 1: *provenance ()*

SS: Search Space (set of all available triples)
Proc = {p | (p, rdf:type, process) ∈ SS}
Agent = {a | (a, rdf:type, agent) ∈ SS}
DataC = {d | (d, rdf:type, data) ∈ SS}
Input: dc

IN = {t ∈ SS | t = (s, has_participant, dc)}

P is smallest such that,
P = ({p | ∃ x, y. (P, x, y) ∈ IN} ∪ {p | ∃ x ∈ P. (x, preceded_by, p)}) ∩ Proc
A = {a | (p, has_agent, a) ∈ SS ⋀ p ∈ P} ∩ Agent
D = {d | (p, has_participant, d) ∈ SS ⋀ p ∈ P} ∩ DataC
OUT = IN ∪ {(p_1, preceded_by, p_2) ∈ SS | p_1, p_2 ∈ P} ∪
{(p, has_agent, a) ∈ SS | p ∈ P ⋀ a ∈ A} ∪
{(p, has_participant, d) ∈ SS | p ∈ P ⋀ d ∈ D} ∪
{(p_1, part_of, p_2) ∈ SS | p_1, p_2 ∈ P} ∪
{(a, has_parameter, pa) ∈ SS | a ∈ A ⋀ pa ∈ D} ∪
{(a_1, adjacent_to, a_2) ∈ SS | a_1, a_2 ∈A} ∪
{(a_1, part_of, a_2) ∈ SS | a_1, a_2 ∈A} ∪
{(a_1, contained_in, a_2) ∈ SS | a_1, a_2 ∈A} ∪
{(d_1, part_of, d_2) ∈ SS | d_1, d_2 ∈D} ∪
{(d_1, contained_in, d_2) ∈ SS | d_1, d_2 ∈D} ∪
{(d_1, transformation_of, d_2) ∈ SS | d_1, d_2 ∈D} ∪
{(d_1, derives_from, d_2) ∈ SS | d_1, d_2 ∈D}
Note: *The definition of the set P is recursive and it is checked that the set P is well-defined.*

In the next section, we introduce the second provenance query operator to retrieve data entities that share similar provenance characteristics.

3.2.2 Provenance Context: Retrieval of Data Entities

provenance_context() - This query operator supports the provenance queries in C***ategory II*** (Section 3.1). The query operator takes as input provenance values as constraints and returns data entities that satisfy the provenance constraints. In effect, the query input values define a formal "provenance context" composed of constraint values defined over all available provenance information. For example, the provenance details such as ocean buoy identifier-"*oceanBuoy7044*", geographical coordinates- "*475111N:1222118W*", and temporal constraints- "*April 21, 2003 to May 2, 2003*" form a very specific contextual structure that is used to identify data entities that satisfy these provenance constraints. The data entities in the query result have similar or equivalent provenance and hence can be interpreted with equal level of trust.

The query operator can be formally defined using the *provenance ()* query operator (described earlier in Section 3.2.1).

Definition 2: *provenance_context ()*

SS: Search Space (set of all available triples)
DataC = {d | (d, rdf:type, data) ∈ SS}
Input: set of triples pc = pc_g ∪ pc_v,
where pc_g describes provenance values directly connected to `data` individuals and pc_v includes variables in input triples.

Output = {dc | (pc_g ⊆ provenance (dc)} ∩
{dc | for all (v, x, y) ∈ pc_v we have (dc, x, y) ∈ SS} ∩
{dc | for all (x, y, v) ∈ pc_v we have (x, y, dc) ∈ SS} ∩
{dc | (dc, rdf:type, DataC) ∈ SS}

3.2.3 Compare and Merge Provenance

The third category of query operators is defined to compare and merge provenance information. To focus on the semantics of the query operators, the RDF graph representing provenance metadata is assumed to be a *ground* RDF graph [23], that is, it does not contain any blank nodes. The use of ground RDF graph corresponds to the modeling approach exemplified by the Provenir ontology that represents both provenance and data as "first class citizens" and does not require use of RDF reification or named graphs to model provenance information.

We use a quadruple consisting of vertices, edges, mapping function to label vertices and edges, and a mapping function to map vertices to Provenir classes, to describe a provenance graph.

G = (V, E, l, m)
E ⊆ V^2
l: V ∪ E → L (L is set of Uniform Resource Identifiers)
m: V → Nm (Nm is set of Provenir ontology classes)

3. ***provenance_compare ( )***: Accurate comparison of scientific results requires the comparison of the associated provenance information. For example, two ocean visualization charts are said to be *comparable* if the associated provenance information (type of sensors, parameters used in the scientific workflow) are identical. We use the RDF graph equivalence definition [24] with the added functionality of "coloring" the nodes and labeling the edges using the Provenir ontology schema to define equivalence between two provenance graphs. The formal definition of the query operator is as follows:

Definition 3: *provenance_compare ()*

Input: G_1, G_2
G_1 = (V_1, E_1, l_1, m_1)
G_2 = (V_2, E_2, l_2, m_2)
The two provenance graphs G_1 and G_2 are equivalent if there are bijections,
i = V_1 → V_2 and
j = E_1 → E_2 such that
∀ (e_1, e_2) ∈ E_1 we have (i(e_1), i(e_2)) ∈ j(e_1, e_2) and
∀ x ∈ V_1 : m_1 (x) = m_2 (i(x)) holds.

The next query operator enables the merging of provenance information, for example provenance from different stages of an experiment protocol can be merged to construct an unified view of the experiment process.

4. ***provenance_merge ( )***: The query operator takes as input two provenance graphs and gives as output a single, merged provenance graph. The merged graph does not include any duplicates individuals. A formal definition of the merge operator is described below.

Definition 4: *provenance_merge ()*

Input: G_1, G_2
G_1 = (V_1, E_1, l_1, m_1)
G_2 = (V_2, E_2, l_2, m_2)
Output: (V_1 ∪ V_2, E_1 ∪ E_2, l, m), where
l(x) = { l_1(x) if x ∈ V_1 ∪ E_1
l_2(x) if x ∈ V_2 ∪ E_2 and where
m: V_1 ∪ V_2 → Nc : m(x) = { m_1(x) if x ∈ V_1, m_2(x) if x ∈ V_2

These query operators enable provenance management systems to offer a well-defined mechanism to query provenance information using the Provenir ontology as a foundational model. In the next section, we describe the implementation of a provenance query engine that supports the query operators over an RDF data store.

4. PROVENANCE QUERY ENGINE: IMPLEMENTATION and EVALUATION

The provenance query engine is designed as a Java-based Application Programming Interface (API) to support the provenance query operators over a RDF data store. The query engine described in this section is integrated with an Oracle 10g (release 10.2.0.3.0) data store, but we note that the query engine can be used with any RDF data store that supports SPARQL [14] and inference rules. The Oracle 10g RDF data store uses a SQL table function (RDF_MATCH) to efficiently query RDF data [25]. The default Oracle 10g query interface does not support all SPARQL functions, hence we used the Oracle's Jena [26] based plug-in, which supports the full SPARQL specification.

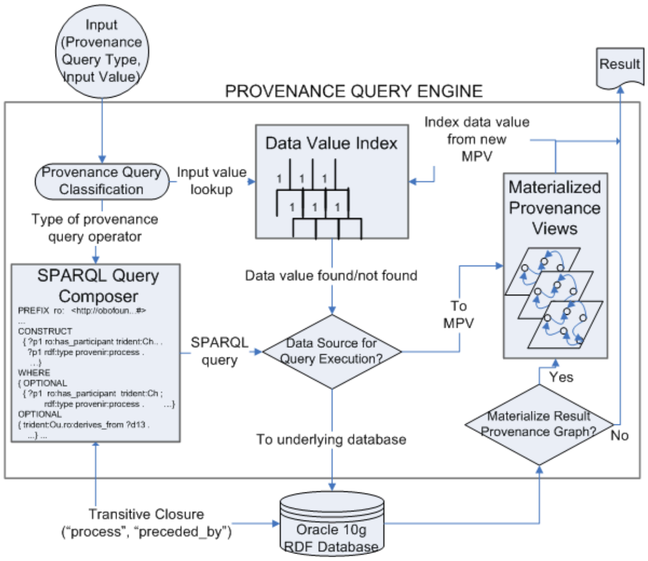
Input (Provenance Query Type, Input Value)
Result
PROVENANCE QUERY ENGINE
Index data value from new MPV
Data Value Index
Provenance Query Classification
Input value lookup
Materialized Provenance Views
Type of provenance query operator
SPARQL Query Composer
PREFIX ro: <http://obofoun...#>
...
CONSTRUCT
{ ?p1 ro:has_participant trident:Ch...
?p1 rdf:type provenir:process .
...}
WHERE
{ OPTIONAL
{ ?p1 ro:has_participant trident:Ch ;
rdf:type provenir:process . ...}
OPTIONAL
{ trident:Ou.ro:derives_from ?d13 .
...} ...
Data value found/not found
To MPV
SPARQL query
Data Source for Query Execution?
Yes
Materialize Result Provenance Graph?
No
To underlying database
Transitive Closure ("process", "preceded_by")
Oracle 10g RDF Database

Figure 5: Architecture of Provenance Query Engine

The provenance query engine consists of three functional components (Figure 5):

a) **A Query Composer**: The query composer maps the provenance query operators to SPARQL syntax according to denotational semantics of the query operators (defined in Section 3.2).

b) **A Function to Compute Transitive Closure over RDF**: SPARQL query language does not support transitive closure for an RDF <*node*, *edge*> combination. Hence we have implemented a function to efficiently compute transitive closure using the SPARQL ASK function. The result of this function is used by the query composer.

c) **Query Optimizer using Materialized Provenance Views**: Using a new class of materialized views based on the Provenir ontology schema called Materialized Provenance Views (MPV) a query optimizer has been implemented that enables the query engine to scale with very large RDF data sets.

In the next sub-section, we describe the SPARQL query composer and transitive closure function.

4.1 SPARQL Query Composition

In this section, we briefly describe the SPARQL language structure and the approach used by the query composer to map the provenance query operator to SPARQL. A SPARQL query is composed of a set of triples (each triple is of the form, <*Subject*, *Property*, *Object*>) to form a query pattern called basic graph pattern (BGP), where any one of the three constituents may be a variable. For example, <*?x*, *is_president_of*, *UnitedStateofAmerica*> is a triple pattern with the variable '*?x*'. The BGP is used to find a sub-graph from the RDF store where the values in the sub-graph may be substituted for the variables in the BGP resulting in a RDF graph equivalent to the sub-graph [14]. But, if suitable instantiation of variables in BGP are not found in the RDF store, no results are returned [14].

The provenance query operators represent the complete result set by defining exhaustive set of dependencies among data, process, and agent. However, in real world scenarios the provenance information available can be incomplete due to application-specific or cost-based limitations. Hence, a straightforward mapping of provenance query operators to SPARQL as a BGP is not desirable. Such a BGP-based query expression pattern may not return a result in the presence of incomplete provenance information. The OPTIONAL function in SPARQL can be used to specify query expression patterns that can succeed with partial instantiation, yielding maximal "best match" result graph. Hence, the query composer uses this OPTIONAL function to create a query expression pattern.

Further, as discussed in Section 3.2, the query composer also needs to compute the transitive closure over the `<process, preceded_by>` combination to retrieve all individuals of the `process` class linked to the input value. But, unlike many graph database query languages such as Lorel or GraphLog, [27], SPARQL does not provide an explicit function for transitive closure to answer reachability queries.[3] We now describe the transitive closure function implemented in the provenance query engine.

4.1.1 *Transitive Closure Function*

We had two options in implementing the transitive closure function, namely a function that is tightly couple to the RDF store or a generic function. We chose a generic implementation using the SPARQL ASK function that allows the provenance query engine to be used over multiple RDF stores. The SPARQL ASK function allows "application to test whether or not a query pattern has a solution," [14] without returning a result set or graph. The transitive closure functions starts with the process instance (p1) linked to the input value and then recursively expands the SPARQL query expression using the ASK function till a *false* value is returned. The SPARQL ASK function, in contrast to the SELECT and CONSTRUCT functions, does not bind the results of the query to variables in the query pattern. Hence, it is a low-overhead function for computing transitive closure. The results of a comparative evaluation of the SPARQL functions along with the performance evaluation of a straightforward implementation of the provenance query engine are presented in the next section.

4.2 Evaluation of a Straightforward Provenance Query Engine Implementation

We use the two standard complexity measures of (a) **Query or Expression Complexity** (varying the syntax of the query with fixed data size), and (b) **Data Complexity** (vary the data size with a fixed query expression) [28], to characterize the performance of the provenance query engine. The evaluation results presented in this section are for the ***provenance ()*** query operator, which represents the majority of provenance queries.

The expression complexity of the SPARQL graph pattern using the OPTIONAL function is PSPACE-complete [29]. As discussed earlier in this section (Section 3.2), the SPARQL query pattern for the *provenance ()* query operator requires the use of OPTIONAL

[3] The W3C DAWG postponed a decision on transitive closure in SPARQL

function with multiple levels of nesting. The other components that affect the evaluation of a SPARQL query are the total number of variables and the number of triples defined in the query pattern. Hence, to evaluate the expression complexity of the *provenance ()* query operator, we use five queries (listed in **Table 1**) with increasing number of variables, triple patterns, and nesting levels using the OPTIONAL function over a fixed dataset size. The five queries, from the Neptune oceanography project, involve retrieval of provenance information associated with different data entities.

Table 1. SPARQL query details to evaluate expression complexity using the *provenance ( )* query operator

Query: Retrieve Provenance of given Input Value	Number of Variables	Number of Triples	Nesting using OPTIONAL
Q1. codar_mnty_908294932772185.nc	31	86	4 levels
Q2. NetCDFReader 90829493474170	45	126	5 levels
Q3. HyperCubeSchema 90829493462995	45	126	5 levels
Q4. HyperCube 90829493567757	59	166	6 levels
Q5. ChartDataTable 90829493849637	73	206	7 levels

The data complexity of SPARQL for a fixed graph pattern expression is LOGSPACE [29]. Using a fixed SPARQL query, we evaluate the *provenance ()* query operator over 5 different datasets (listed in Table 2).

4.2.1 *Experiment Setup and Dataset*

The experiments were conducted using Oracle10g (Release 10.2.0.3.0) DBMS on a Sun Fire V490 server running 64-bit Solaris 9 with four 1.8 GHz Ultra Sparc IV processors and 8GB of main memory. The database used an 8 KB block size and was configured with a 512 MB buffer cache. The timings reported are mean result from five runs with warm cache.

The dataset for the evaluations was generated from the oceanography scenario described in Section 1.1. The scientific workflow was executed using the Trident workflow workbench [30]. The Trident log file with additional details such as temperature sensors, ocean current sensors, and ocean buoys, was used as a template to generate the RDF data.

Table 2. SPARQL query details to evaluate expression complexity using the *provenance ( )* query operator

Dataset Id	Number of Experiment Cycles	Number of Inferred RDF Triples	Total Number of RDF Triples
DS1	100	23,620	32,182
DS2	1000	226,671	309,459
DS3	10,000	2,257,096	3,082,184
DS4	100,000	22,560,776	30,808,427
DS5	1,000,000	225,596, 929	308, 069,953

Five datasets corresponding to 100, 1000, 10000, 100000, and 1 million experiment cycles are used (Table 2). The largest dataset corresponding to 1 million experiment cycles contains about 308 million RDF triples. A rule index [25] was defined for inferencing over the datasets using standard RDFS rules [31] and a user-defined rule to infer that, "If the input value of a `process` (p1) is the same as the output value of another `process` (p2), then p1 is linked to p2 by the property `preceded_by`". Table 2 lists the total number of new RDF triples inferred using the RDFS and user-defined rules for each of the five datasets.

4.2.2 *Transitive Closure Computation using SPARQL Functions*

This experiment compares the performance of transitive closure computation using the SPARQL "SELECT", "CONSTRUCT", and "ASK" functions. The transitive closure for the five queries, Q1 to Q5 (in Table 1), is computed using the largest dataset DS5 (in Table 2) of 308 million RDF triples.

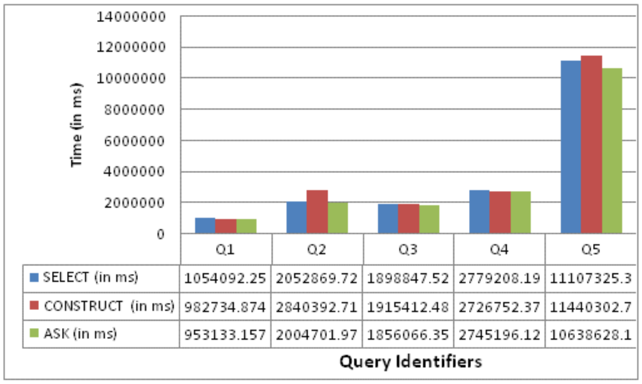

Figure 6: Comparison of SPARQL query functions to compute transitive closure for RDF <node, edge>

The results in Figure 6 show that the ASK function consistently performs better than both the SELECT and the CONSTRUCT functions. This is because both SELECT and CONSTRUCT involve a binding of query results to query pattern variables and graph pattern respectively. In contrast, the ASK function returns a Boolean value indicating if a graph corresponding to the query pattern exists in the RDF store. Hence, the ASK-function based transitive closure function implemented in the query engine is a low-overhead solution.

4.2.3 *Query Expression Complexity*

This experiment characterizes the expression complexity of the *provenance ( )* query operator in SPARQL syntax. The results are for the total query time, including time for transitive closure, query composition, and query execution, for the five provenance queries, Q1 to Q5 (in Table 1). The queries are executed against a fixed size dataset, DS5 (in Table 2), with about 308 million RDF triples representing 1 million experiment cycles.

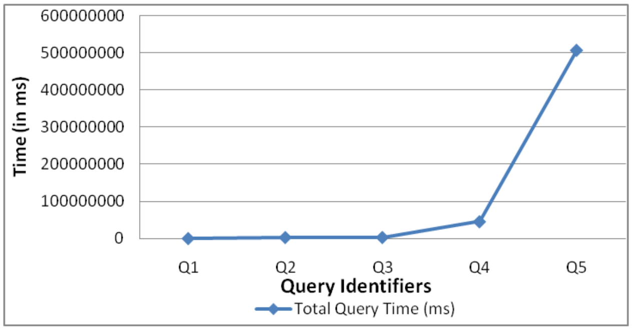

Figure 7: Query expression complexity results with fixed dataset size of DS5

The results (Figure 7) demonstrate that the query time increases with increasing expression complexity of the queries. Thus a straightforward implementation of the query engine is unusable for provenance management systems in eScience projects.

4.2.4 Data Complexity

This experiment characterizes the data complexity of the *provenance ( )* query operator in SPARQL syntax. The query Q5 (in Table 1) with maximum expression complexity is used as the fixed query for evaluation over varying sizes of RDF datasets (in Table 2).

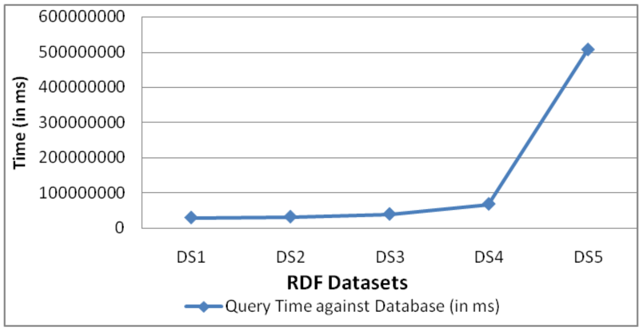

Figure 8: Data complexity results with fixed SPARQL query expression pattern of Q5

Figure 8 illustrates that the data complexity of the *provenance ()* query operator is also large and an effective optimization technique is necessary for use of the provenance query engine in a practical provenance management system.

Overall, the evaluation results (Section 4.2.3 and Section 4.2.4) clearly demonstrate that a straightforward implementation of the provenance query engine cannot scale with both increasing complexity of provenance query expressions and size of provenance data. In the next section, we introduce a new class of materialized views for query optimization to address both these issues.

4.3 Materialized Provenance View

The provenance queries are graph traversal operations for path computations. Path computation over graphs is an expensive operation especially in the case of provenance queries that require computation of fixed paths, recursive pattern-based paths and neighborhood retrieval. The results of our evaluation show that even industrial strength RDF database face severe limitations in terms of response time for complex provenance queries over large scientific datasets. To address this we define a new class of materialized views called materialized provenance views (MPV) that materializes provenance sub-graphs for selected classes of input values. We consider two constraints to decide what data should be materialized, (a) the cost of maintaining the materialized views, that is, if a materialized view needs to be recalculated when new RDF triples are added to the database, and (b) the number and complexity of provenance queries that can be satisfied by a MPV.

Provenance metadata by definition describes past events and therefore they are not subject to frequent updates, except in the presence of errors. Therefore materialized views are a suitable approach for provenance query optimization. For the second constraint, we use the Provenir model to identify one logical unit of provenance information that can be used to satisfy not one but multiple provenance queries. The provenance graph of the final output of an experiment cycle, the "Chart Visualization" file (Figure 9) represents a logical unit of provenance information for the Neptune oceanography project scenario. This provenance query engine computes the unit of provenance information using the Provenir ontology.

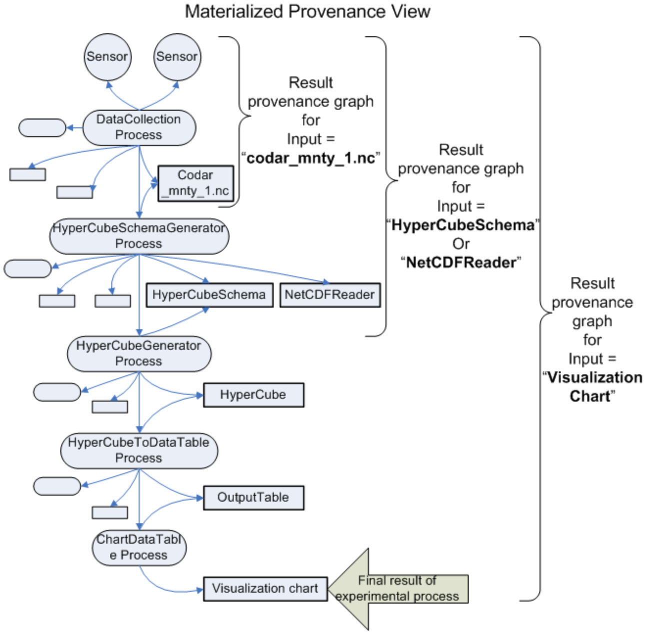
Materialized Provenance View
Sensor
Sensor
DataCollection Process
Codar _mnty_1.nc
HyperCubeSchemaGenerator Process
HyperCubeSchema
NetCDFReader
HyperCubeGenerator Process
HyperCube
HyperCubeToDataTable Process
OutputTable
ChartDataTabl e Process
Visualization chart
Final result of experimental process
Result provenance graph for Input = "codar_mnty_1.nc"
Result provenance graph for Input = "HyperCubeSchema" Or "NetCDFReader"
Result provenance graph for Input = "Visualization Chart"

Figure 9: A MPV corresponds to one experiment cycle

The MPV (Figure 9) can not only satisfy the provenance query for "*Chart Visualization*" files, but also provenance queries for all entities occurring in that one experiment cycle. A B-tree index is used to index all instances of the `data_collection` class that occur in that experiment cycle. Given an input value, the query optimizer looks-up the B-tree index to decide if the query can be satisfied by a MPV or by the underlying database. MPVs are created using a memoization approach instead of an eager strategy, since the total sum of MPVs created for all final output values equals the database itself. In the next section we discuss the evaluation results using MPV for query optimization.

4.4 Evaluation Results using MPV for Query Optimization

We use the straightforward implementation (discussed earlier in Section 4.2.3 and Section 4.2.4) as benchmark to discuss the performance of the provenance query engine using MPV.

4.4.1 Query Optimization using MPV

In the first experiment, queries from Section 4.2.3 are evaluated against a MPV.

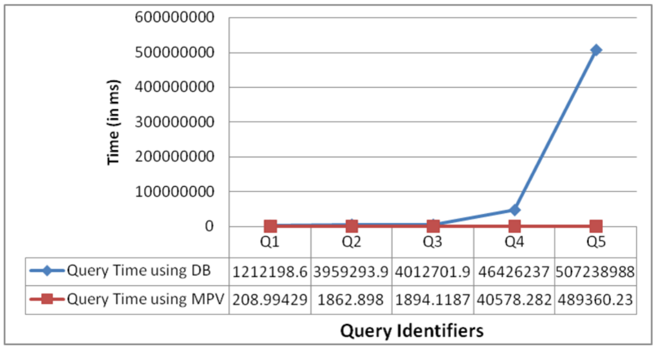

Figure 10: Comparing expression complexity results with and without using MPV

The MPV used in this experiment corresponds to one experiment cycle with final output "*ChartDataTable90829493849637*"; the MPV contained 86 RDF triples and occupied 12KB. Figure 10 clearly demonstrates the significant reduction in total query time through the use of MPV. Note that a single MPV is used to satisfy all provenance queries from Table 1. The percent gain in performance with MPV as compared to query execution against the database is listed in Table 3.

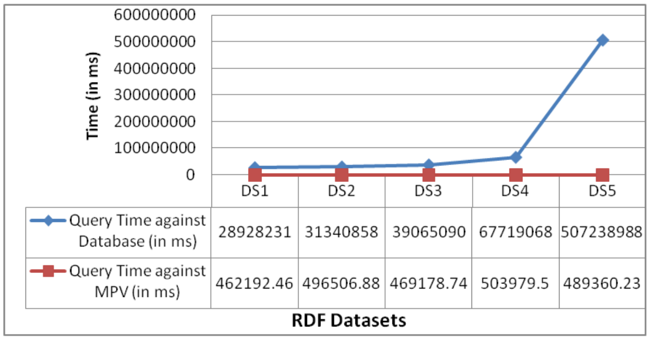

Figure 11: Comparing data complexity results with and without using MPV

Similarly, the results in Figure 11 show significant speed-up in query time over varying sizes of RDF datasets using the fixed query, Q5 (Table 1). Table 3 lists the performance improvement for data complexity using MPV for query optimization.

Table 3: Performance gain for *provenance ( )* query operator using MPV

For Expression Complexity (gain in %)		For Data Complexity (gain in %)	
Q1	99.98	DS1	98.40
Q2	99.95	DS2	98.42
Q3	99.95	DS3	98.80
Q4	99.91	DS4	99.26
Q5	99.90	DS5	99.90

5. RELATED WORK

Provenance has been studied from multiple perspectives across a number of domains. In this section, we correlate the provenance query operators introduced in this paper to existing work in database and workflow provenance as illustrated in Figure 12.

5.1 Database Provenance

Database provenance or data provenance, often termed as "fine-grained" provenance [32], has been extensively studied in the database community. Early work includes the use of annotations to associate "data source" and "intermediate source" with data (polygen model) in a federated database environment to resolve conflicts by Wang et al. [33], and use of "attribution" for data extracted from Web pages by Lee et al. [34]. More recent work has defined database provenance in terms of "Why provenance", "Where provenance" by Buneman et al. [5], and "How provenance" by Green et al. [6].

A restricted view of the "Where provenance" identifies each piece of input data that contributes to a given element of the result set returned by each database query. "Why provenance" was first described in [4], and in this paper we use the syntactic definition of "Why provenance" by Buneman et al. [5] that defines a "proof" for a data entity. The proof consists of a query, representing a set of constraints, over a data source with "witness" values that result in a particular data output. The semantics of the *provenance ()* query operator closely relates to both "Where provenance" and "Why provenance" [5]. To address the limitation of "Why provenance" that includes "…set of all contributing input tuples" leading to ambiguous provenance, Green et al. [6] introduced semiring-based "How provenance." The *provenance ()* query operator over a "weighted" provenance model, which reflects the individual contribution of each component (for example process loops or repeated use of single source data), is comparable to "How provenance."

The Trio project [21] considers three aspects of lineage information of a given tuple, namely how was a tuple in the database derived along with a time value (when) and the data sources used. A subset of queries in Trio, "lineage queries", discussed in [21], can be mapped both as *provenance ()* and as *provenance_context ()* query operators depending on the input value. The SPIDER system [35] built on top of Clio [36] uses provenance information modeled as "routes" (schema mappings) between source and target data to capture aspects of both "Why provenance" and "How provenance". Hence, it closely relates to the semantics of the *provenance ()* query operator.

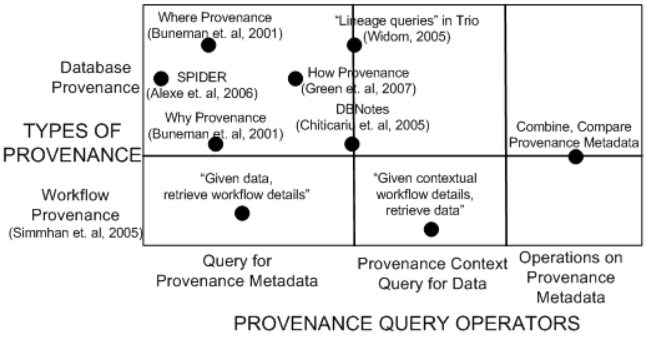
Where Provenance (Buneman et. al, 2001)
"Lineage queries" in Trio (Widom, 2005)
Database Provenance
SPIDER (Alexe et. al, 2006)
How Provenance (Green et. al, 2007)
DBNotes (Chiticariu et. al, 2005)
Why Provenance (Buneman et. al, 2001)
TYPES OF PROVENANCE
Combine, Compare Provenance Metadata
Workflow Provenance (Simmhan et. al, 2005)
"Given data, retrieve workflow details"
"Given contextual workflow details, retrieve data"
Query for Provenance Metadata
Provenance Context Query for Data
Operations on Provenance Metadata
PROVENANCE QUERY OPERATORS

Figure 12: Mapping provenance query operators with existing database and workflow provenance

5.2 Provenance in Scientific Workflows

The rapid adoption of scientific workflows to automate scientific processes has catalyzed a large body of work in recording provenance information for the generated results. Simmhan et al. [7] survey different approaches for collection, representation and management of workflow provenance. The participants in the provenance challenge [8] collected provenance at different levels of granularities such as comprehensive workflow system traces in PASS [37], use of semantic annotations of services by Taverna [38], and recording of data value details and service invocations in Karma [20]. Recent work has also recognized the need for inclusion of domain semantics in the form of domain-specific provenance metadata [22] along with workflow provenance. The semantics of these projects can be mapped to the *provenance ()* query operator.

The Open Provenance Model (OPM) [39] is a generic graph model for provenance representation. In contrast to the Provenir ontology, OPM does not model named relationships as first class entities and requires use of tags on edges to define roles. Further, OPM models only causal properties, while the Provenir ontology models ten fundamental relations, for example `located_in` specifies geographical location. The rules for inferencing proposed in OPM are easily contradicted [40] due to its generic graph model, in contrast the Provenir ontology is supported by the well-defined and extensive reasoning features of the Semantic Web.

6. CONCLUSIONS

This paper introduces PRoM, a Semantic Web provenance management framework for science, consisting of the Provenir ontology, a novel provenance query classification scheme, and provenance query operators. The paper defines a modular approach to interoperable provenance modeling based on the Provenir foundational ontology. The Provenir ontology defined in OWL DL supports modeling of complex domain semantics and also use of reasoning to query provenance. The provenance query operators, defined for the first time, support a variety of queries and enable provenance management systems to offer a dedicated and well-defined query mechanism.

The paper also discusses the implementation of a provenance query engine over an RDF database to support the provenance query operators. The evaluation of a straightforward implementation of the query engine highlights the need for an effective query optimization strategy, which is addressed by a new class of materialized views called MPV. The MPVs are shown to be highly effective for complex provenance queries over very large scientific datasets. Specifically, MPV reduce the query times by approximately three orders of magnitude that enables the use of the provenance query engine as a practical tool for provenance management in scientific applications.

7. ACKNOWLEDGMENTS

This work was funded by NIH Grant# 1R01HL087795-01A1.

8. REFERENCES

[1] "MyGrid." http://www.mygrid.org.uk/, retrieved Oct 29, 2009

[2] "Project Neptune." http://www.neptune.washington.edu/, retrieved Oct 29, 2009

[3]"The Panoramic Survey Telescope and Rapid Response System (Pan-STARRS)." http://pan-starrs.ifa.hawaii.edu/ retrieved Oct 29, 2009

[4] Y. Cui, et al, "Practical Lineage Tracing in Data Warehouses," *ICDE*, San Diego, California, pp. 367 2000.

[5] P. Buneman, et al., "Why and Where: A Characterization of Data Provenance," *ICDT*, 2001, pp. 316 - 330

[6] T. J. Green, et al., "Provenance Semirings," in *Symposium on Principles of database systems (PODS)*, 2007, pp. 675–686.

[7] Y. L. Simmhan, et al., "A survey of data provenance in e-science" *SIGMOD Rec.*, vol. 34, pp. 31 - 36 September 2005.

[8] "IPAW 2008." http://www.sci.utah.edu/ipaw2008/, retrieved Oct 29, 2009

[9]"OWL Web Ontology Language Overview," D. L. McGuinness, van Harmelen, F., (Eds) *W3C Recommendation*, 2004.

[10] J. Arnold, et al., "Metabolomics," in *Handbook of Industrial Mycology* NY: Marcel Dekker, 2004, pp. 597-633.

[11] D. Georgakopoulos, et al., "An Overview of Workflow Management: From Process Modeling to Infrastructure for Automation," *Distributed and Parallel Databases (DAPD),* 3(2), pp. 119-153, 1995.

[12] D. Oberle, et al., "DOLCE ergo SUMO: On Foundational and Domain Models in SWIntO (SmartWeb Integrated Ontology)," *Journal of Web Semantics: Science, Services and Agents on the World Wide Web,* 5 (3), pp. 156-174, 2007.

[13] "Provenance Challenge ". http://twiki.ipaw.info/bin/view/Challenge/ retrieved Oct 29, 2009

[14] "SPARQL Query Language for RDF," E. Prud'hommeaux, Seaborne, A.,(Eds), *W3C Recommendation*, 2006.

[15] S. S. Sahoo, et al., "Knowledge modeling and its application in life sciences: a tale of two ontologies," *WWW*, Scotland, 2006, pp. 317-326.

[16] P. L. Whetzel, et. al, "Development of FuGO: An Ontology for Functional Genomics Investigations," *OMICS: A Journal of Integrative Biology* vol. 10, pp. 199-204, 2006.

[17] B. Smith, et al., "Relations in biomedical ontologies," *Genome Biol,* vol. 6, p. R46, 2005.

[18] S. S. Sahoo, et al., "Where did you come from...Where did you go?" An Algebra and RDF Query Engine for Provenance ", Wright State University Technical Report, 2009.

[19] S. S. Sahoo, et al., "Ontology-driven Provenance Management in eScience: An Application in Parasite Research," *ODBASE 09,* Vilamoura, Algarve-Portugal, 2009 (*to appear*).

[20] Y. L. Simmhan, et al., " Karma2: Provenance management for data driven workflows," *International Journal of Web Services Research*, vol. 5, Issue 2 pp. 1-22, 2008.

[21] J. Widom, "Trio: A System for Integrated Management of Data, Accuracy, and Lineage," in *Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, Pacific Grove, California, 2005.

[22] S. S. Sahoo, et al., "Semantic Provenance for eScience: Managing the Deluge of Scientific Data," *IEEE Internet Computing,* vol. 12, pp. 46-54, 2008.

[23] P. Hayes, "RDF Semantics," B. McBride, Ed., *W3C Recommendation* 2004.

[24] G. Klyne, et al., "Resource Description Framework (RDF): Concepts and Abstract Syntax," *W3C Recommendation*, 2004.

[25] E. I. Chong, et al., "An efficient SQL-based RDF querying scheme," *VLDB*, Trondheim, Norway, 2005, pp. 1216-1227.

[26] B. McBride, "Jena: A Semantic Web Toolkit," *IEEE Internet Computing,* vol. 6, pp. 55-59, Nov. 2002.

[27] R. Angles, et al., "Survey of graph database models," *ACM Comput. Surveys,* vol. 40, pp. 1-39, Feb. 2008.

[28] M. Vardi, "The Complexity of Relational Query Languages," in *14th Ann. ACM Symp. Theory of Computing (STOC '82)*, 1982, pp. 137-146.

[29] J. Pérez, et al., "Semantics and Complexity of SPARQL," in *Int'l Semantic Web Conf. (ISWC '06)*, Athens, GA, 2006, pp. 30-43.

[30] "Trident Workflow Workbench." http://www.microsoft.com/mscorp/tc/trident.mspx, retrieved Oct 29, 2009

[31] "RDF Vocabulary Description Language 1.0: RDF Schema", Brickley, D., Guha, R.V., (Eds.) *W3C Recommendation*

[32] W. C. Tan, "Provenance in Databases: Past, Current, and Future," *IEEE Data Eng. Bull.,* vol. 30, pp. 3 -12 2007.

[33] Y. R. Wang, et al., "A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective," *VLDB*, 1990, pp. 519–538.

[34] T. Lee, et al., "Multimodal Integration of Disparate Information Sources with Attribution," in *Entity Relationship Workshop on Information Retrieval and Conceptual Modeling*, 1997.

[35] B. Alexe, et al., "SPIDER: a Schema mapPIng DEbuggeR. ," *VLDB*, 2006, pp. 1179–1182.

[36] L. M. Haas, et al., "Clio Grows Up: From Research Prototype to Industrial Tool," *SIGMOD*, Baltimore, MD, 2005, pp. 805–810.

[37] D. A. Holland, et al., "PASSing the provenance challenge," *Concurrency and Control: Practice and Experience,* vol. 20, pp. 531-540, 2008.

[38] J. Zhao, et al., "Mining taverna's semantic web of provenance," *Journal of Concurrency and Computation:Practice and Experience, , 20 (5), pp. 463 - 472* 2007.

[39] "Open Provenance Model." http://twiki.ipaw.info/bin/view/Challenge/OPM, retrieved Oct 29, 2009

[40] Y. L. Simmhan, "FeedbackonOPM," http://twiki.ipaw.info/pub/Challenge/OpenProvenanceModelWorkshop/FeedbackonOPM.pptx, retrieved Oct 29, 2009.