5-13-2008

A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data

Matthew Perry
Wright State University - Main Campus

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

Knoesis Center Technical Report
Department of Computer Science and Engineering
Wright State University

Technical Report: KNOESIS-TR-2008-01

A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data

Matthew Perry and Amit P. Sheth
May 13, 2008

Noname manuscript No.
(will be inserted by the editor)

A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data

Matthew Perry · Amit P. Sheth

Received: date / Accepted: date

Abstract Spatial and temporal data are critical components in many applications. This is especially true in analytical applications ranging from scientific discovery to national security and criminal investigation. The analytical process often requires uncovering and analyzing complex thematic relationships between disparate people, places and events. Fundamentally new query operators based on the graph structure of Semantic Web data models, such as semantic associations, are proving useful for this purpose. However, these analysis mechanisms are primarily intended for thematic relationships. In this paper, we describe a framework built around the RDF data model for analysis of thematic, spatial and temporal relationships between named entities. We present a spatiotemporal modeling approach that uses an upper-level ontology in combination with temporal RDF graphs. A set of query operators that use graph patterns to specify a form of context are formally defined. We also describe an efficient implementation of the framework in Oracle DBMS and demonstrate the scalability of our approach with a performance study using both synthetic and real-world RDF datasets of over 25 million triples.

Keywords Ontology · Semantic Analytics · RDF Querying · Spatial RDF · Temporal RDF

Matthew Perry
Kno.e.sis Center, Department of Computer Science and Engineering, Wright State University, Dayton, OH, USA
E-mail: perry.66@wright.edu

Amit P. Sheth
Kno.e.sis Center, Department of Computer Science and Engineering, Wright State University, Dayton, OH, USA
E-mail: amit.sheth@wright.edu

1 Introduction

Analytical applications are increasingly exploiting complex relationships among named entities as a powerful analytical tool. Such connect-the-dots applications are common in many domains including national security, drug discovery, and medical informatics. Semantic Web Technologies [5] are well suited for this type of analysis. It is often necessary that the analysis process spans across multiple heterogeneous data sources, and ontologies and semantic metadata standards help facilitate aggregation and integration of this content. In addition, standard models for metadata representation on the web, such as Resource Description Framework (RDF) [46], model relationships as first class objects making it very natural to query and analyze entities based on their relationships. Researchers have consequently argued for graph-based querying of RDF [16], and fundamentally new analytical operators based on the graph structure of RDF have emerged (e.g., semantic associations [17] and subgraph discovery [61]). These operators allow querying for complex relationships among named entities where an ontology provides the context or domain semantics. We use the term *semantic analytics* to refer to this process of searching and analyzing semantically meaningful connections among named entities. Semantic analytics has been successfully used in a variety of settings, for example identifying conflict of interest [11], detecting patent infringement [50] and discovering metabolic pathways [47].

So far, semantic analytics tools have primarily focused on thematic relationships, but spatial and temporal data are often critical components in analytical domains. In fact, most entities and events can be described along three dimensions: *thematic*, *spatial* and *temporal*. Consider the following event: *Fred Smith moved into*

the house at 244 Elm Street on November 16, 2007. The thematic dimension describes *what* is occurring (the person Fred Smith moved to a new residence). The spatial dimension describes *where* the event occurs (the new residence is located at 244 Elm Street). The temporal dimension describes *when* the event occurs (the moving event occurred on November 16, 2007). Unfortunately, integrated semantic analytics over all three dimensions is not currently possible because of the following gaps in the state of the art:

- *Current GIS and spatial database technology does not support complex thematic analytics operations.* Traditional data models used for GIS excel at modeling and analyzing spatial and temporal relationships among geospatial entities but tend to model the thematic aspects of a given domain as directly attached attributes of geospatial entities. Thematic entities and their relationships are not explicitly and independently represented, making analysis of these relationships difficult.
- *Current semantic analytics technology does not support analysis of spatial and temporal relationships.* Semantic analytics research has focused on thematic relationships between entities. Thematic relationships can be explicitly stated in RDF graphs, but many important spatial and temporal relationships (e.g., distance and elapsed time) are implicit and require additional computation. Semantic analytics tools depend on explicit relations and must be extended if they are to use implicit spatial and temporal relations.

This paper describes a framework that aims to bridge these gaps. In [55] a modeling approach was presented that tries to overcome the limitations described above by modeling spatial, temporal and thematic (STT) data using ontologies and temporal RDF graphs. A variety of query operators that combine thematic relationships with spatial and temporal relationships are possible with this modeling approach. In [56], initial definitions and a prototype implementation of a core set of query operators were presented. We further develop the ideas presented in these papers and describe a framework to support STT analytics over Semantic Web data.

1.1 Contributions

We propose a framework that extends current semantic analytics technology so that spatial and temporal data is supported in addition to thematic data. We address problems of data modeling, data storage and query operator design and implementation. Specifically, we make the following contributions:

- An ontology-based spatiotemporal modeling approach using temporal RDF.
- A formalization of a set of spatial, temporal and thematic query operators for the proposed modeling approach that builds on a notion of context and supports computation of implicit spatial and temporal relations.
- A SQL-based implementation of the proposed query operators that involves a storage and indexing scheme for spatial and temporal RDF data and an efficient treatment of temporal RDFS inferencing.
- A detailed performance study of the implementation using large synthetic and real-world RDF datasets.

The initial ideas underlying this framework appeared in the proceedings of ACM-GIS 2006 [55] and GeoS 2007 [56]. We have further developed, refined and extended the material presented at these conferences. Our new contributions include (1) a revised formalization of a core set of query operators that generalizes from path templates to graph patterns, (2) a deeper discussion of the RDF serializations used in the framework, the algorithms used for implementation and the relevant related work in the literature and (3) a more complete and extensive evaluation of our implementation that involves not only synthetically-generated RDF datasets but also a real-world RDF dataset of over 25 million triples. Our implementation demonstrates excellent scalability for very large RDF datasets in this evaluation (e.g., execution time of less than 500 milliseconds for a 10-hop graph pattern query over a 28 million triple dataset).

1.2 Outline

The remainder of the paper is organized as follows. Section 2 presents motivating examples. Section 3 discusses related work in spatial and temporal data management and management of RDF data. Our modeling approach is discussed in Section 4, and query operators over this model are formalized in Section 5. Section 6 presents an implementation of this framework in Oracle DBMS. An experimental evaluation of our implementation is presented in Section 7, and Section 8 gives conclusions and discusses future work.

2 Motivating Examples

We will motivate this work with a set of examples from the environmental sciences domain. Suppose a hydrology researcher is investigating the effects of human activities on rainfall-runoff relationships. Through some

initial work using a GIS system, the researcher has noticed an increase in home-owner's insurance claims related to water damage within a certain geographic region. A possible reason for this could be a reduction in ground vegetation in the area due to human activities, as this vegetation helps prevent flash flood events. An interesting search would be *find any factories with manufacturing processes that may adversely affect nearby ground vegetation and only return those factories within the identified zone of houses*. We may pose the following SQL query involving the *spatial_restrict* table function for such a search:

```
SELECT f as factory
FROM TABLE (spatial_restrict('
  (?f uses_manufacturing_process ?m)
  (?m has_by_product ?p)
  (?p negatively_affects <Ground_Vegetation>)
  (?f located_at ?l)',
  <Houses Region>,
  'GeoRelate(mask=inside)');
```

With this query, we are using the *spatial_restrict* operator to specify a thematic connection (*context*) between factories and substances that negatively impact ground vegetation, and we are then using a spatial relationship to limit the results to those factories inside the spatial feature (i.e., polygon) formed from the boundary of the region of homes in question. We also provide a *spatial_extent* operator that allows retrieving the spatial geometry associated with a given thematic entity with respect to a given context, and a *spatial_eval* operator that computes the spatial relationship between two thematic entities with respect to a given context.

We provide analogous *temporal_extent*, *temporal_restrict* and *temporal_eval* operators to query temporal aspects of connections between entities. The *temporal_extent* operator returns the temporal properties of a given relationship and the *temporal_restrict* operator allows optional filtering based on these temporal properties. For example, *find all flood insurance claims occurring after a given factory became operational and return the dates of the claims.*

```
SELECT c as claim, start_date, end_date
FROM TABLE (temporal_restrict('
  (?o files_claim ?c)
  (?c related_to <Flood_Damage>)
  (?c for_policy ?p) (?p type <Homeowners>)',
  'AFTER', '2006-03-02', '2006-03-03',
  'INTERSECT'));
```

In this query, we are specifying a graph pattern that identifies a particular type of insurance claim. We are additionally limiting the results to those that are valid *after* the input time interval. The INTERSECT keyword indicates the type of temporal interval to use for a given result subgraph. In this case, we are interested in the time interval during which each edge (RDF statement) in the subgraph is valid. Our final operator, *temporal_eval*, acts as a temporal join for thematic subgraphs.

Our implementation allows multiple operators to be used in a single SQL query. We can therefore execute spatio-temporal-thematic queries that combine spatial and temporal operators. These possibilities are discussed in Section 6.1. Though we refer to our queries as spatial, temporal or spatiotemporal in the paper, all our queries involve a significant thematic component due to the graph patterns used in the queries.

We use the running scenario of historical analysis of battlefield events of World War II to illustrate concepts in the remainder of the paper. We chose this scenario because it is easy to understand and because we have generated large synthetic datasets corresponding to this scenario that are used in our evaluation.

3 Related Work

We divide related work into two categories: (1) data modeling and (2) query languages and query processing.

3.1 Data Modeling

We first discuss the use of ontologies in Geographic Information Science (GIS) and then cover spatiotemporal modeling approaches.

Ontologies and GIS: There has been significant work regarding the use of geospatial ontologies in GIS. Ontologies in GIS are seen as a vehicle to facilitate interoperability and to limit data integration problems both from different systems and between people and systems [10]. Fonseca et al. [28] present an architecture for an ontology-driven GIS in which ontologies describe the semantics of geographic data and act as a system integrator independent of the data model used (e.g., object vs. field).

On the Web, the use of ontology for better search and integration of geospatial data and applications is embodied in the Geospatial Semantic Web [26]. From a Web context, Kolas et al. [48] outline specific types of geospatial ontologies needed for integration of GIS data and services: base geospatial ontology, feature data

source ontology, geospatial service ontology, and geospatial filter ontology. The base geospatial ontology provides core geospatial knowledge vocabulary while the remaining ontologies are focused on geospatial web services.

Our work is complementary to the work on geo-ontologies. The geo-ontologies above would be mapped to (i.e., subsumed by) the spatial classes in our upper-level ontology (presented in Section 4.2). Our work provides a means to further incorporate non-spatial thematic knowledge and analysis with the geospatial knowledge and analysis provided through geo-ontologies and GIS. That is, we provide a framework that allows analysis of thematic and temporal relationships in addition to spatial relationships.

Spatiotemporal Models: Spatiotemporal data models have received considerable attention in both the GIS and Database communities, and many good surveys exist (e.g., [52][57]). In a recent survey, Pelekis et. al identify 10 distinct spatiotemporal data models [52]. In general, our modeling approach differs through its extensive use of thematic relationships. We not only conceptually separate thematic entities from spatial entities, but we also utilize indirect thematic relationships to link thematic entities to spatial entities in a variety of ways (i.e. different contexts). A review of each distinct model is outside the scope of this paper, but we will review some of the most similar.

Of the models discussed in the literature, the three domain model is conceptually the most similar to our RDF-based approach. The three domain model, introduced by Yuan, is described in [76][77]. This model represents semantics, space and time separately. To represent spatiotemporal information in this model, semantic objects are linked via temporal objects to spatial objects. This provides temporal information about the semantic (thematic) properties of a given spatial region. This is analogous to temporal *located at* and *occurred at* relationships in our upper-level ontology. The three domain model is quite similar to our approach in that it represents thematic entities as first class objects rather than attributes of geospatial objects. The key difference is that the three domain model relies on direct connections from thematic entities to spatial regions whereas our model allows more flexibility through indirect connections composed of sequences of thematic relationships.

Our modeling approach also has similarities with object-oriented approaches. A recent proposal by Worboys and Hornsby [75] combines the object-oriented and event-based modeling approaches to model dynamic geospatial domains. They define an upper-level ontology similar to the one we present in Section 4.2. They model the concept of a setting and a situate function that maps entities and events to settings. Settings can be spatial, temporal, or spatiotemporal. In contrast to our work, the authors focus on geospatial objects and events and model what we would consider a thematic entity (e.g., an airplane) as a geospatial entity. That is, the separation between the thematic and spatial domains is not as strongly emphasized. Our RDF-based modeling approach provides a means to assign spatial properties to those entities not directly connected to a spatial setting and allows deeper analysis of purely thematic relationships.

General modeling approaches and languages have also been extended for spatiotemporal data. Tryfona and Jensen extended the entity-relationship model to create the spatiotemporal entity-relationship model (STER) [70][71]. Price et. al extended the Unified Modeling Language (UML) to create spatiotemporal UML [58]. RDF is similar to these modeling languages in the sense that it is a general purpose ontology language and can model entities and relationships for a given domain. Our approach could therefore be seen as an extension of RDF (i.e. spatial types in combination with temporal triples) to allow for modeling spatial and temporal entities and relationships. RDF is different from these other languages in that it also serves as a model for storing and querying data in the form of RDF triples whereas UML and ER are primarily for conceptual modeling. We can thus query relationships directly as first class objects in RDF graphs, and we utilize this capability to design and implement relationship-based query operators. Furthermore, RDF statements carry well-defined semantics, and corresponding inferencing mechanisms must be supported.

3.2 Query Languages and Query Processing

We first review approaches to querying thematic RDF data and then discuss querying spatial and temporal data on the Semantic Web. This is followed by a review of querying spatial and temporal data using traditional database technology.

Querying RDF: Many RDF query languages have been proposed in the literature. These include SQL-like languages (e.g., SPARQL [59], RDQL [64]), functional languages (e.g., RQL [45]), rule-based languages (e.g., TRIPLE [65]) and graph traversal languages (e.g., RxPath [67]). For a detailed comparison of these languages, see [37][16]. Recently, SPARQL has emerged as a W3C recommendation. As an alternative to defining a new query language, an approach for querying RDF data

directly in SQL has been proposed [25]. This facilitates easy integration with other SQL queries against traditional relational data and saves the overhead of translating data from SQL to the RDF query language data format. Our implementation described in Section 6 follows this approach and introduces new SQL functions for spatial and temporal querying of RDF data.

A variety of systems for management of persistent RDF data have been presented in the literature. These systems usually rely on an underlying relational database representation. Three main types of storage schemes are commonly used [69]: (1) *schema-aware* - one table per RDF(S) class or property (e.g., Sesame using PostgreSQL [24], the vertical partitioning scheme described in [8]), (2) *schema-oblivious* - a single three-column (subject, predicate, object) table storing all statements (e.g., Jena [74], 3Store [39], Sesame using MySQL [24], Oracle Semantic Data Store [3]) and (3) *hybrid* - one table storing class membership information and one table for each group of properties with the same range type such as Resource or integer (e.g., RDFSuite [12]). Efficient evaluation of queries using these systems typically involves transformation into a SQL query against the underlying RDBMS representation, and traditional relational indexes are used to speed up query processing.

Alternate approaches persistently store RDF data using lower-level structures such as Hash Tables (Redland [20]) and B^+-Trees (YARS [40]) and traverse these structures to evaluate queries.

All the previously mentioned techniques index RDF data based on a "collection of triples" conceptualization. The GRIN index proposed by Udrea, et al. [72] exploits the graph structure of the RDF data. A GRIN index is a tree structure where leaf nodes represent a set of triples in the RDF graph and interior nodes are represented by a vertex, radius pair (v, r) that represents all vertices in the RDF graph within r hops of vertex v. Graph pattern queries are evaluated by traversing the tree to find all triples that may contain an answer to the query. A subgraph matching algorithm is then run over the identified portion of the RDF graph. The initial implementation of GRIN used a main-memory representation, which was followed by a disk-based implementation using PostgreSQL [60].

Our approach uses an underlying relational database representation of RDF data that follows the schema-oblivious storage scheme. This storage scheme is augmented with additional structures for more efficient searching over spatial and temporal data. We utilize traditional spatial and temporal indexes in our query processing strategies and use composite B^+-tree indexes for efficient evaluation of graph pattern queries.

Spatial and Temporal Data on the Semantic Web: Work is somewhat limited with regards to incorporating spatial and temporal relationships into queries over Semantic Web data. Examples of querying geospatial RDF data are mostly seen in Web applications and semantic geospatial web services [44][68]. In general, this work mainly focuses on interoperability, and query processing proceeds by translating RDF representations of spatial features into geometric representations on the fly and then performing spatial calculations. In contrast, we look at how the relationship-centric nature of the RDF model can enable new query types and also address issues related to efficient query processing.

The SPIRIT spatial search engine [43] combines an ontology describing the geospatial domain with the searching and indexing capability of Oracle Spatial for the purposes of searching documents based on the spatial features associated with named places mentioned in the document. In contrast, our searching operators are intended for general purpose querying of ontological and spatial relationships.

Querying for temporal data in RDF graphs is less complicated as RDF supports typed literals such as xsd:date, and corresponding query languages support filtering results based on literal values. However, this is far from supporting full temporal RDF as graphs discussed in this paper.

Gutierrez et al. introduced the concept of temporal RDF graphs and formally defined them in [32][33]. In addition, the authors briefly discussed aspects of a query language for temporal RDF graphs, but a through investigation of such a language has not been completed, and no implementation issues were mentioned. To the best of our knowledge, our work in [56] is the first to investigate efficient schemes for storing and querying temporal RDF and implementation of RDFS inferencing that incorporates the concept of valid time for RDF statements. In [60], Pugliese et al. present tGRIN an extension of the GRIN index for temporal RDF data. The tGRIN extension factors in the temporal distance between vertices in addition to the graph distance (number of edges). The authors approach using tGRIN, however, supports a more limited form of temporal RDFS inferencing than we do. Specifically, they only support inferences related to *rdfs:subPropertyOf*. Pugliese et al. also support a different form of temporal RDF queries than we support. Their queries involve temporal conditions on single edges of a graph pattern. In contrast, our queries involve temporal conditions on time intervals derived from multiple edges in a graph pattern (e.g., the intersection of the time intervals of each edge in a graph pattern).

Semantic Web researchers have proposed incorporating past work on qualitative spatial and temporal reasoning into the Semantic Web reasoning framework as an alternative to adding spatial and temporal capabilities to query languages. Hobbs and Pen translated a subset of Allen's interval calculus [14][15] to OWL to create the OWL-Time ontology [42]. In [9], Abdelmonty et al. demonstrated that OWL is insufficient to fully support the spatial reasoning required for a geo-ontology (e.g., it is very hard to define a class of *HousesNearMotorways* made up of individuals of type house that are within a specific distance of motorways). In a follow-on paper, Smart et al. showed how to use additional rules and specialized tools to help overcome the shortcomings of OWL [66]. Our approach differs in that our implementation does not involve reasoning over relative spatial and temporal relations (e.g., $(x \textit{ before } y) \wedge (y \textit{ before } z) \Rightarrow (x \textit{ before } z)$). Instead we support the computation spatial and temporal relations using time values that are grounded to a timeline and spatial features that are grounded to a coordinate system.

Spatial and Temporal Query Processing: Management of spatial and temporal data has long been an area of interest [34][35][51].

Processing temporal queries over relational data is well covered in the literature. Usually temporal information is stored as time intervals. Selection queries generally retrieve all intervals that intersect a given query interval. Various structures have been proposed for efficient execution of such queries [62]. Another important task is interval join queries that join two relations based on overlapping intervals. Many approaches to evaluate these joins exist in the literature [29].

Processing spatial queries is also a well-researched topic. Spatial selection queries return a set of spatial objects that satisfy a spatial predicate [18]. Various types of spatial index structures have been developed for such queries (e.g., the R-Tree [21][36] and quadtree [63]). Also important are spatial join queries, which join sets of spatial objects based on a spatial predicate. A variety of methods for evaluating spatial joins have been proposed [19][23][31].

Work on indexing and querying spatiotemporal data or moving objects is also of interest [35]. Indexing approaches usually optimize queries about future positions of spatiotemporal objects or queries about past states of the spatiotemporal objects [38]. Various approaches to indexing spatiotemporal objects appear in the literature [49].

A key difference of the query types addressed here is our focus on thematic relationships. Rather than querying a set of spatial or temporal objects, we are querying thematic objects associated to spatial objects via a chain of thematic relationships (i.e. in a specific context). For example, the following relationships could represent a battle participation context: (*Soldier, on_crew_of, Vehicle*) (*Vehicle, used_in, Battle*) (*Battle, occurred_at, Spatial_Region*). In other words, the spatial object associated with an entity is determined dynamically at run time. Therefore, we cannot create direct spatial indexes for these thematic entities. Similarly, we compute a temporal interval for a subgraph connecting multiple entities, also dynamically generated at runtime, making it infeasible to directly index the derived intervals. Rather than trying to improve upon existing indexing techniques for traditional queries over spatial and/or temporal objects, we focus on how to incorporate these indexing techniques into our query processing procedures.

4 Modeling Approach

Our ontology-based modeling approach is presented in this section. We give preliminary descriptions of RDF, RDFS and Temporal RDF and present the core ontologies used in our modeling approach.

4.1 Preliminaries

RDF: RDF has been adopted by the W3C as a standard for representing metadata on the Web. The RDF data model is defined as follows. Let U, L and B be pairwise disjoint sets of URIs, literals and blank nodes, respectively. The union of these sets $U \cup B \cup L$ is referred to as the set of RDF Terms RT. An *RDF triple* is a 3-tuple $(s, p, o) \in (U \cup B) \times U \times RT$ where s is the *subject*, p is the *property* and o is the *object*. A set of RDF triples is referred to as an *RDF Graph*, as RDF can be represented as a directed, labeled graph where a directed edge labeled with the property name connects a vertex labeled with the subject name to a vertex labeled with the object name.

RDFS: RDF Schema (RDFS) [22] provides a standard vocabulary for describing the classes and relationships used in RDF graphs and consequently provides the capability to define ontologies. Ontologies serve to formally specify the semantics of RDF data so that a common interpretation of the data can be shared across multiple applications. Classes represent logical groups of resources, and a member of a class is said to be an instance of the class. The RDFS vocabulary offers a set of built-in classes and properties. Two of the most relevant classes are *rdfs:Class* and *rdf:Property*, and some of

the most relevant properties are *rdf:type*, *rdfs:domain, rdfs:range*, *rdfs:subClassOf* and *rdfs:subPropertyOf*. The *rdf:type* property is used to define class and property types (e.g., the triple (*S, rdf:type, rdfs:Class*) asserts that S is a class). *rdf:type* is also used to denote instances of classes (e.g., (*s, rdf:type, S*) asserts that s is an instance of S). *rdfs:domain* and *rdfs:range* allow us to define the domain and range for a given property, and *rdfs:subClassOf* and *rdfs:subPropertyOf* allow us to create class and property hierarchies.

A set of entailment rules are also defined for RDF and RDFS [41]. Conceptually, these rules specify that an additional triple can be added to an RDF graph if the graph contains triples of a specific pattern. Such rules describe, for example, the transitivity of the *rdfs: subClassOf* property (i.e. (*x, rdfs:subClassOf, y*) (*y, rdfs:subClassOf, z*) $\Rightarrow$ (*x, rdfs:subClassOf, z*)).

Temporal RDF: In order to analyze the temporal properties of relationships in RDF graphs, we need a way to record the temporal properties of the statements in those graphs, and we must account for the effects of those temporal properties on RDFS inferencing rules. Gutierrez et. al. introduced the notion of temporal RDF graphs for this purpose [32][33].

Temporal RDF graphs model linear, discrete, absolute time and are defined as follows [33]. Given a set of discrete, linearly ordered time points T, a *temporal triple* is an RDF triple with a temporal label $t \in T$. A statement's temporal label represents its valid time. The notation $(s, p, o) : [t]$ is used to denote a temporal triple. The expression $(s, p, o) : [t_1, t_2]$ is a notation for $\{(s, p, o) : [t] \mid t_1 \leq t \leq t_2\}$. *A temporal RDF graph* is a set of temporal triples. For a temporal RDF graph G_t, $TRIPLES(G_t)$ denotes the set $\{(s, p, o) \mid \exists\, t \in T$ with $(s, p, o) : [t] \in G_t\}$.

The following example illustrates these concepts. Consider a soldier *s1* assigned to the 1st Armored Division (*1stAD*) from April 3, 1942, until June 14, 1943, and then assigned to the 3rd Armored Division (*3rdAD*) from June 15, 1943, until October 18, 1943. This would yield the following triples: (*s1, assigned_to, 1stAD*) : [04:03:1942, 06:14:1943], (*s1, assigned_to, 3rdAD*) : [06:15:1943, 10:18:1943].

We must also account for the effects of temporal labels on RDFS inferencing rules (see Section 6.2.2). To incorporate inferencing into temporal RDF graphs, a basic arithmetic of intervals is needed to derive the temporal label for inferred statements. For example, interval intersection would be needed for *rdfs:subClassOf* (e.g., (*x, rdfs:subClassOf, y*) : $[1, 4] \wedge$ (*y, rdfs:subClassOf, z*) : $[3, 5] \Rightarrow$ (*x, rdfs:subClassOf, z*) : $[3, 4]$).

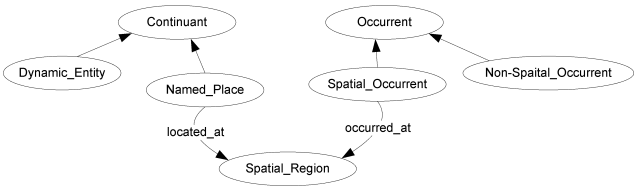


Fig. 1 Upper-level ontology integrating spatial and thematic dimensions

4.2 Ontology-based Model

Here we discuss our ontology-based approach for modeling theme, space and time. We present an upper-level ontology defining a general hierarchy of thematic and spatial entity classes and associated relationships connecting these entity classes (see Figure 1). We intend for application-specific domain ontologies in the thematic dimension to be integrated into the upper-level ontology through subclassing of appropriate classes and relationships. Temporal information is integrated into the ontology by labeling relationship instances with their valid times. A unique aspect of this approach is that we do not require the spatial properties of each thematic entity to be explicitly recorded. Instead, we utilize relationships in the thematic domain to indirectly provide spatial properties. This gives the benefit of greater flexibility in the integration of thematic and spatial information.

Thematic Dimension: Our upper-level thematic ontology consists of a fundamental class hierarchy and a few basic relationships. In developing the class hierarchy, we first follow the approach of Grenon and Smith's Basic Formal Ontology [30] and distinguish between Continuants and Occurrents. *Continuants* are those entities that persist over time and maintain their identity through change. Examples from our historical battlefield analysis scenario could include a soldier, an aircraft or a city. *Occurrents* represent events and processes; they happen and then no longer exist. Examples are the bombing of a target or the execution of a training exercise. A second division of entities concerns spatial properties. Some Occurrents are inherently spatial such as a battle; others are not, such as the assignment of a solider to a division. We therefore explicitly represent *Spatial Occurrents* and *Non-Spatial Occurrents*. Continuants also have varying spatial properties. We distinguish a special type of Continuant that we refer to as a Named Place. *Named Places* are entities that serve as locations for other physical entities and Spatial Occurrents. They have very static spatial behavior over time and are distinguished by a strong association with their spatial location. Examples of Named Places include a

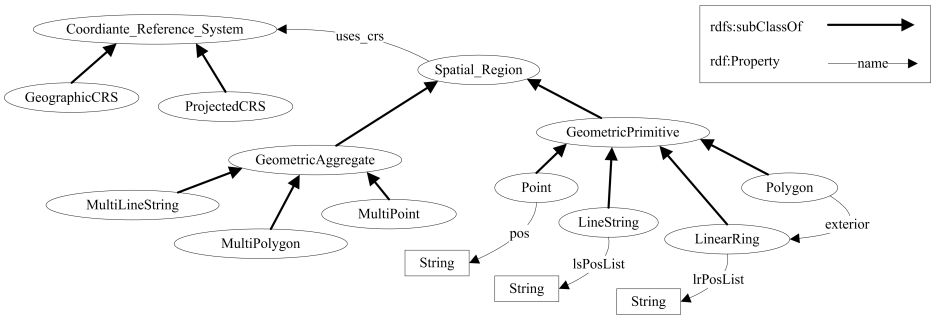


Fig. 2 GeoRSS GML-based ontology modeling basic spatial geometries. Note that Geometric Aggregates contain collections of their respective Geometric Primitives (e.g., MultiPolygon contains a collection of Polygons). These relations and attributes of Coordinate Reference System have been left out of the figure for clarity.

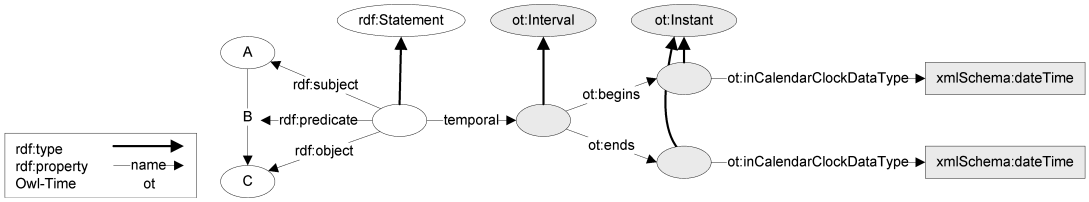


Fig. 3 Temporal reification of the RDF statement (A B C). Constructs from the OwlTime ontology are shown in gray.

city, a zip code, a building, or a lake. In contrast to a Named Place, we distinguish another subclass of Continuant: Dynamic Entity. *Dynamic Entities* are those entities with dynamic spatial behavior whose identities are not as strongly associated with space. Examples include a person or a vehicle. We do not make further philosophical distinctions between these two types of Continuants as the final decision depends upon the domain and application.

Spatial Dimension: The spatial portion of our upper-level ontology consists of a top-level class and two corresponding relations. *Spatial Regions* represents basic spatial geometries (i.e. georeferenced points, lines and polygons). The *occurred at* relation connects Spatial Occurrent to Spatial Region, and *located at* connects Named Place to Spatial Region. These relations allow us to associate a thematic concept, such as the city of Berlin or the Battle of the Bulge, with its geospatial properties. Spatial properties of thematic entities can consequently be derived using the associated Spatial Regions.

The spatial features represented by the Spatial Region class are complex types that need to be fully modeled with a spatial ontology. Fortunately, there is movement towards standard ontologies for spatial geometries, for example work done as part of the Open Geospatial Consortium (OGC) Semantic Web Interoperability Experiment [1] and the W3C geo incubator group [7]. The existing OGC Geographic Markup Language (GML) specification serves as an excellent basis for these ontologies as discussed in [9][48]. We propose a spatial ontology based on the GeoRSS GML specification [**?**]. The ontology models 2-dimensional spatial geometries and associated spatial reference system information. Figure 2 illustrates the RDF representation of this ontology.

Temporal Dimension: We use temporal RDF graphs [33] to incorporate the time dimension into our model. Temporal information is represented by associating time intervals with relationship instances in the ontology. The time interval on the relationship denotes the times at which the relationship is valid. These time intervals are grounded to a discrete, linearly-ordered timeline. RDF reification is used to associate time intervals with RDF statements to realize temporal RDF graphs. We use a portion of the OWL-Time ontology [42] to model the time intervals themselves, and a new property *temporal* asserts that the reified statement is valid during the given time interval. Figure 3 illustrates this approach.

5 Querying Approach

Our approach for querying over this ontology-based model utilizes the graph-centric structure of RDF data. For spatial aspects, we use subgraphs in the RDF graph to connect thematic entities (e.g., Dynamic Entities) to Spatial Regions. A given thematic entity can be connected to various Spatial Regions through a variety of different subgraphs, yielding a many-to-many

mapping. Associated domain ontologies clarify the semantics of these subgraphs, and we refer to a given subgraph as a *context*. That is, a thematic entity has spatial properties with respect to a given context. Using a military ontology, for example, a soldier could be associated with the spatial properties of his residence in one context (*Solider, lives_at, Residence*) (*Residence, located_at, Spatial_Region*) or with the locations of his training facilities using a different context (*Soldier, member_of, Military_Unit*) (*Military_Unit, trains_at, Base*) (*Base, located_at, Spatial_Region*). For temporal aspects, we derive temporal intervals for these subgraphs through computations over the temporal values of the edges (temporal RDF triples) that make up the subgraph.

In this section, we introduce and formalize a set of query operators that follow the basic approach outlined above. We introduce spatial operators that allow (1) retrieving the spatial properties of an entity with respect to a given context (*spatial_extent*), (2) retrieving the set of entities whose associated Spatial Regions satisfy a spatial predicate (*spatial_restrict*) and (3) retrieving pairs of entities whose associated spatial regions satisfy a given spatial relation (*spatial_eval*). We introduce temporal operators that allow (1) deriving a temporal interval for a subgraph in the RDF graph (*temporal_extent*), (2) filtering a set of subgraphs by evaluating a temporal predicate over their derived time intervals (*temporal_restrict*), and (3) retrieving pairs of subgraphs whose time intervals satisfy a given temporal relation (*temporal_eval*).

Our framework differs from traditional approaches to querying RDF data in that computation of implicit relationships are supported. We do not rely on the existence of explicit RDF statements asserting spatial and temporal relationships such as inside and after. Instead, we perform computations at query time to establish the existence of these relationships that are implicit in the RDF dataset.

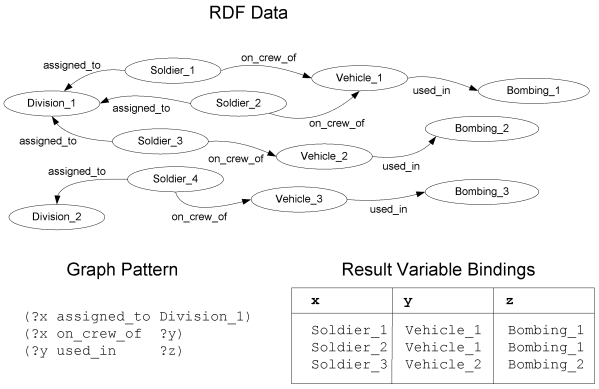


Fig. 4 Example graph pattern from historical analysis of WWII scenario with resulting variable bindings.

5.1 Graph Patterns

Our querying approach relies on specifying a type of connection between resources in an RDF graph. We use SPARQL-like graph patterns to express these connection types. Conceptually, a *graph pattern* is a set of RDF triples where the subjects, properties and/or objects may be replaced with variables. In general, a graph pattern query against an RDF graph G returns a set of mappings between the variables in the graph pattern and terms (URIs, Blank Nodes and Literals) in G such that replacing variables with their corresponding terms results in a set of triples actually present in G. Figure 4 illustrates an example graph pattern query. A formal syntax for SPARQL graph patterns and formal semantics for SPARQL graph pattern queries is given in [53]. We present a fragment of this formalization to define the general concept of a graph pattern, which we use to formally define our proposed query operators.

Let UL denote the union $U \cup L$ (recall that U is the set of URIs and L is the set of Literals) and let VN be a set of variables disjoint from the set of RDF Terms RT.

A *graph pattern* is defined recursively as follows:
Basis: A tuple from $(UL \cup VN) \times (U \cup VN) \times (UL \cup VN)$ is a graph pattern (*triple pattern*).
Recursion: If P_1 and P_2 are graph patterns, then (P_1 AND P_2) is a graph pattern.

The semantics of a graph pattern are defined in terms of a function $[[\cdot]]$, which takes a graph pattern expression and returns a set of mappings where a mapping $\mu : VN \rightarrow RT$ is a function from VN to RT. For a triple pattern tp, we denote the set of variables in tp as $var(tp)$, and we denote the triple obtained by replacing the variables in tp according to the mapping μ as $\mu(tp)$. For a graph pattern GP, we denote the set of triples obtained by replacing the variables in GP according to μ as $\mu(GP)$, and we refer to this set of triples as an *instance* of GP. For a mapping μ, the subset of VN where it is defined is called its domain $dom(\mu)$. Two mappings μ_1 and μ_2 are compatible if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$. In other words the union $\mu_1 \cup \mu_2$ is also a mapping. In addition, for two sets of mappings M_1 and M_2, the *join* is defined as:

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1 \text{ and } \mu_2 \in M_2 \text{ and } \mu_1 \text{ and } \mu_2 \text{ are compatible mappings}\}$$

Let G be an RDF graph, tp a triple pattern and P_1, P_2 graph patterns. The evaluation of a graph pattern over G, denoted $[[\cdot]]_G$, is defined recursively as:

Basis: $[[tp]]_G = \{\mu \mid dom(\mu) = var(tp) \text{ and } \mu(tp) \in G\}$
Recursion: $[[P_1 \text{ AND } P_2]]_G = [[P_1]]_G \bowtie [[P_2]]_G$

5.2 Spatial Operators

We define our spatial operators using what we term a spatial context. Conceptually, a *spatial context* specifies a type of connection between a thematic entity and a spatial entity. Given a temporal RDF graph G_t, a spatial context is defined as a 2-tuple (GP, v) where GP is a graph pattern and $v \in var(GP)$ is a variable in GP identifying a Spatial Region instance. That is, for each mapping $\mu \in [[GP]]_{TRIPLES(G_t)}$ with $\mu(v) = x$, there exists a triple (*x, rdf:type, Spatial_Region*) in $TRIPLES(G_t)$. Note that G in the previous section refers to a plain RDF graph, and here G_t refers to a temporal RDF graph. Also recall that $TRIPLES(G_t)$ denotes the plain RDF graph created by removing the temporal information from G_t. As an example, consider the spatial context below that connects a soldier (*?x*) to a Spatial Region (*?s*).

(‘(*?x assigned_to ?y*) (*?y participates_in ?z*)
(*?z occurred_at ?s*)’, ‘*?s*’)

In the following, for a Spatial Region URI sr, we use $geom(sr)$ to refer to the actual spatial geometry (i.e. point, line, polygon) represented by sr according to the spatial ontology described in Section 4.2. We use S to denote the set of all possible spatial geometries.

The first spatial operator we define, *spatial_extent*, is intended to find the spatial properties of a thematic entity with respect to a given spatial context. The query “*what are the spatial properties of the 101st Airborne Division with respect to battle participation*” (Example 1) illustrates an example search using this operator. We can think of this operator as retrieving the spatial features corresponding to the identified Spatial Region in the result subgraphs of a graph pattern query.

$spatial\_extent((GP, v))_{Gt} \rightarrow \{(\mu, s)\}$
Given:
a spatial context (GP, v), a temporal RDF graph G_t
Find:
$\{(\mu, s) \mid \mu \in [[GP]]_{TRIPLES(G_t)}$ and $s = geom(\mu(v))\}$
Example 1:
ANS ← spatial_extent(
‘(⟨*101st Airborne Division*⟩ *participates_in ?x*)
(*?x occurred_at ?s*)’, ‘*?s*’)$_{Gt}$

The next two spatial operators focus on spatial relationships. As a prerequisite, we define a spatial formula, which is used to express conditions on spatial relationships. Spatial formulas are built from qualitative spatial functions and metric spatial functions. A *qualitative spatial function* is a Boolean function $qsf : S \times S \rightarrow \mathbb{B}$. Any of the following topological spatial relations identified by Egenhofer and Herring [27] may be used as qualitative spatial functions in our formalization: *disjoint, touch, overlap boundary disjoint, overlap boundary intersect, equal, contains, covers, inside, covered by.*

A *metric spatial function* is a function $msf : S \times S \rightarrow \mathbb{R}$. We use one metric spatial function *distance* : $S \times S \rightarrow \mathbb{R}$, which returns the distance between two spatial geometries. Let VS be a set of variables disjoint from VN and RT. We define a *metric spatial expression*, mse, as follows, where $s_1, s_2 \in S \cup VS$ and $r \in \mathbb{R}$.

$$\langle mse \rangle ::= \langle msf(s_1, s_2) \rangle \langle comp \rangle r$$
$$\langle comp \rangle ::= < \mid > \mid \leq \mid \geq \mid =$$

A *spatial formula* sf evaluates to a Boolean value for a given graph and is defined in terms of metric spatial expressions and qualitative spatial functions. A spatial formula takes the following form, where $s_1, s_2 \in S \cup VS$.

$$\langle sf \rangle ::= \langle mse \rangle | \langle qsf(s_1, s_2) \rangle | \langle sf \rangle \text{AND} \langle sf \rangle | \langle sf \rangle \text{OR} \langle sf \rangle$$

The spatial formulas used in our formalization are expressions containing exactly one free variable $\$s$ or exactly two free variables $\$s_1$ and $\$s_2$ and are denoted as $sf(\$s)$ and $sf(\$s_1, \$s_2)$.

The next spatial operator, *spatial_restrict*, is designed to retrieve thematic entities based on their spatial relationships with a given location in a given context. An example of this type of search is “*which military units have spatial extents that are within 20 miles of (48.45 N, 44.30 E) in the context of battle participation*?” Note that the variable $\$s$ used in the spatial formula is different from the variable v in the graph pattern that represents a Spatial Region instance, as v corresponds to a URI and $\$s$ corresponds to a spatial geometry. (Example 2).

$spatial\_restrict((GP, v), sf(\$s))_{Gt} \rightarrow \{(\mu, s)\}$
Given:
a spatial context (GP, v), a spatial formula sf defined over S and a variable $\$s$,
a temporal RDF graph G_t
Find:
$\{(\mu, s) \mid \mu \in [[GP]]_{TRIPLES(G_t)}$ and $s = geom(\mu(v))$

and sf evaluates to *true* for $\$s = s\}$

Example 2:

ANS ← *spatial_restrict*(
'(*?x participates_in ?y*) (*?y occurred_at ?s*)', '*?s*',
distance(*$s*, $(48.45N, 44.30E)) \leq 20$ miles)$_{Gt}$

The final spatial operator, *spatial_eval*, investigates how thematic entities are related in space. We can think of this operator as a spatial join between thematic entities with respect to a given context. As an example, consider the query "*which infantry unit's operational area overlaps the operational area of the 3rd Armored Division*?" (Example 3).

$$spatial\_eval((GP_1, v_1), (GP_2, v_2), sf(\$s_1, \$s_2))_{Gt} \rightarrow \{(\mu_1, s_1, \mu_2, s_2)\}$$

Given:

a spatial context (GP_1, v_1), a spatial context (GP_2, v_2), a spatial formula sf defined over S and variables $\$s_1$, $\$s_2$, a temporal RDF graph G_t

Find:

$\{(\mu_1, s_1, \mu_2, s_2) \mid \mu 1 \in [[GP_1]]_{TRIPLES(G_t)}$, $\mu_2 \in [[GP_2]]_{TRIPLES(G_t)}$ and $s_1 = geom(\mu_1(v_1))$, $s_2 = geom(\mu_2(v_2))$ and sf evaluates to *true* for $\$s_1 = s_1$, $\$s_2 = s_2\}$

Example:

ANS ← *spatial_eval*(
'(*?x*$_1$ *participates_in ?y*$_1$) (*?y*$_1$ *occurred_at ?s*$_1$)',
'*?s*$_1$', '(⟨*3rd Armored Division*⟩ *participates_in ?y*$_2$)
(*?y*$_2$ *occurred_at ?s*$_2$)', '*?s*$_2$',
overlap-boundary-intersect ($\$s_1$, $\$s_2$) = *true*)$_{Gt}$

5.3 Temporal Operators

The basic idea behind our temporal operators is that we derive a time interval for a graph pattern instance using the time intervals associated with the triples in the graph pattern. These derived intervals are used to restrict graph pattern query results and to perform temporal joins between graph pattern instances.

We will first give some initial definitions. Let T be a set of totally ordered time points. Let G_t be a temporal RDF graph defined over T. For each statement $e = (s, p, o) \in TRIPLES(G_t)$, let $temporal(e) = \{t \mid (s, p, o) : [t] \in G_t\}$. For a set of time points $T' \subseteq T$, let $contig\_intervals(T') = \{[t_i, t_j] \mid \forall\ t \in T$: (if $t_i \leq t$ and $t \leq t_j$ then $t \in T'$) and $t_{i-1} \notin T'$ and $t_{j+1} \notin T'\}$.

Consider the following example:

Suppose:

$T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
$T' = \{2, 3, 4, 7, 8\}$

Then:

$contig\_intervals(T') = \{[2, 4], [7, 8]\}$

Given a set of temporal triples $E = \{e_1, e_2, ..., e_n\}$, we define the *interval expansion* of E, $int\_expansion(E)$, as the set

$contig\_intervals(temporal(e_1)) \times$
$contig\_intervals(temporal(e_2)) \times ...$
$contig\_intervals(temporal(e_n))$

Consider the following example:

Suppose:

$E = \{e_1, e_2, e_3\}$,
$contig\_intervals(temporal(e_1)) = \{[2, 4], [7, 8]\}$,
$contig\_intervals(temporal(e_2)) = \{[1, 5], [7, 9]\}$,
$contig\_intervals(temporal(e3)) = \{[4, 5]\}$

Then:

$int\_expansion(E) = \{\{[2, 4], [1, 5], [4, 5]\}$,
$\{[2, 4], [7, 9], [4, 5]\}$, $\{[7, 8], [1, 5], [4, 5]\}$,
$\{[7, 8], [7, 9], [4, 5]\}\}$

Given a set of time intervals $I = \{(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)\}$ defined over T, let $s_{min} = \min_{1 \leq i \leq n} s_i$, $s_{max} = \max_{1 \leq i \leq n} s_i$, $t_{min} = \min_{1 \leq i \leq n} t_i$, and $t_{max} = \max_{1 \leq i \leq n} t_i$. We define two values, *intersect* and *range*, as follows:

$$intersect(I) = \begin{cases} [s_{max}, t_{min}] & \text{if } s_{max} \leq t_{min}, \\ null & \text{if } s_{max} > t_{min} \end{cases}$$

$$range(I) = \begin{cases} [s_{min}, t_{max}] & \text{if } s_{min} \leq t_{max}, \\ null & \text{if } s_{min} > t_{max} \end{cases}$$

Conceptually, $intersect(I)$ is the largest time interval that intersects each interval in I, and $range(I)$ is the smallest interval that contains each interval in I.

The first temporal operator we define, *temporal_extent*, is intended to compute and return the derived time intervals for the results of a graph pattern query. This operator can return one of two time intervals: (1) the *intersect* interval that represents the time interval during which all statements in the graph pattern instance are valid and (2) the *range* interval that represents the time interval during which any statement in the graph pattern instance is valid. As an example consider the query "*find all pairs of soldiers who were members of the 101st Airborne Division at the same time and return the times of the joint membership*" (Example 3).

$temporal\_extent(GP, IT)_{Gt} \rightarrow \{(\mu, i)\}$

Given:

a temporal RDF Graph G_t, a graph pattern GP, an interval type $IT \in \{intersect, range\}$

Find:

$\{(\mu, i) \mid \mu \in [[GP]]_{TRIPLES(G_t)}$ and
$i \in intersect/range(int\_expansion(\mu(GP)))\}$

Example 3:

ANS $\leftarrow$ *temporal_extent* (
'(*?x assigned_to* ⟨*101st Armored Division*⟩)
(*?y assigned_to* ⟨*101st Armored Division*⟩)',
'*intersect*')$_{Gt}$

The remaining temporal operators examine temporal relationships. To specify conditions on these relationships, we define a temporal formula which is constructed from qualitative and metric temporal functions. For a given temporal RDF graph G_t over time domain T, let I denote the set of all time intervals over T. A *qualitative temporal function* is a Boolean function $qtf : I \times I \rightarrow \mathbb{B}$. Any of the thirteen interval relations identified by Allen [13] can be used in qualitative temporal functions in our formalization.

A *metric temporal function* is a function $mtf : I \times I \rightarrow \mathbb{Z}$. We use one metric temporal function *elapsed_time* $: I \times I \rightarrow \mathbb{Z}$, which is defined for two disjoint time intervals as the duration of time between the end of the earliest interval and the start of the latest interval. The function returns zero if the intervals are not disjoint.

Let VT be a set of variables disjoint from VN, RT and VS. We define a metric temporal expression, *mte*, as follows, where $i_1, i_2 \in I \cup VT$ and $z \in \mathbb{Z}$.

$$\langle mte \rangle ::= \langle mtf(i_1, i_2) \rangle \langle comp \rangle z$$
$$\langle comp \rangle ::= < \mid > \mid \leq \mid \geq \mid =$$

A *temporal formula* tf evaluates to a Boolean value for a given graph and is constructed from qualitative temporal functions and metric temporal expressions. It takes the following form, where $i_1, i_2 \in I \cup VT$.

$$\langle tf \rangle ::= \langle mte \rangle \mid \langle qtf(i_1, i_2) \rangle \mid \langle tf \rangle \text{ AND } \langle tf \rangle \mid \langle tf \rangle \text{ OR } \langle tf \rangle$$

The temporal formulas used in our formalization are expressions containing exactly one free variable $\$t$ or exactly two free variables $\$t_1$ and $\$t_2$ and are denoted as $tf(\$t)$ and $tf(\$t_1, \$t_2)$.

The first relationship-based temporal operator, *temporal_restrict*, is concerned with the temporal properties of a single entity. This operator inquires about the properties of an entity at a given time. For example, one may ask "*which members of the 3rd Armored Division participated in battles during September 1944?*" (Example 4). The basic idea behind this operator is that we specify a graph pattern query and then restrict the set of results based on the temporal extents of the graph pattern instances.

$temporal\_restrict(GP, IT, tf(\$t))_{G_t} \rightarrow \{(\mu, i)\}$

Given:

a temporal RDF Graph G_t, a graph pattern GP,
an interval type $IT \in \{intersect, range\}$,
a temporal formula tf defined over I and
a variable $\$t$

Find:

$\{(\mu, i) \mid \mu \in [[GP]]_{TRIPLES(G_t)}$ and
$i \in intersect/range(int\_expansion(\mu(GP)))$ and
tf evaluates to true for $\$t = i)\}$

Example 4:

ANS $\leftarrow$ *temporal_restrict*('
(*?x assigned_to* ⟨*3rd Armored Division*⟩)
(⟨*3rd Armored Division*⟩ *participates_in ?y*)',
'*intersect*', *during*($\$t$,
[09:01:1944, 09:31:1944]) = *true*)$_{Gt}$

The final temporal operator, *temporal_eval*, allows for querying temporal relationships between entities. This operator can be thought of as a temporal join between graph pattern instances. This operator is designed for a query such as "*which speeches by President Roosevelt were given within 1 day of a major battle?*" (Example 5).

$temporal\_eval(GP_1, IT_1, GP_2, IT_2, tf(\$t_1, \$t_2))_{Gt}$
$\rightarrow \{(\mu_1, i_1, \mu_2, i_2)\}$

Given:

a temporal RDF Graph G_t, a graph pattern
GP_1, an interval type $IT_1 \in \{intersect, range\}$,
a graph pattern GP_2, an interval type
$IT_2 \in \{intersect, range\}$, a temporal
formula tf defined over I and variables $\$t_1$, $\$t_2$

Find:

$\{(\mu_1, i_1, \mu_2, i_2) \mid \mu_1 \in [[GP_1]]_{TRIPLES(G_t)}$ and
$i_1 \in intersect/range(int\_expansion(\mu_1(GP_1)))$
and $\mu_2 \in [[GP_2]]_{TRIPLES(G_t)}$ and
$i_2 \in intersect/range(int\_expansion(\mu_2(GP_2)))$
and tf evaluates to *true* for $\$t_1 = i_1$
and $\$t_2 = i_2\}$

Example 5:

ANS $\leftarrow$ *temporal_eval*('
(⟨*President Roosevelt*⟩ *gives ?x*)', '*intersect*',
'(*?y participates_in ?z*)', '*intersect*',
temporal_distance($\$t_1$, $\$t_2$) $\leq$ 1 day)$_{Gt}$

6 Implementation

In this section, we describe the implementation of our spatial and temporal RDF query operators using Oracle's extensibility framework [2]. The implementation builds on Oracle's existing support for RDF storage and inferencing and support for spatial object types and indexes. The existing support for these features is

the main reason we chose Oracle database for our implementation. We create SQL table functions for each of the previously discussed query operators. Additional structures are created to allow for spatial and temporal indexing of the RDF data for efficient execution of the table functions.

Our implementation uses procedural and declarative SQL and the built-in index structures of the DBMS. We do not depend on any lower-level interfaces of the DBMS, and no modifications to the database kernel are required. Our implementation could therefore be extended to another DBMS and is not restricted to Oracle. We will first give definitions of the table functions that correspond to the query operators defined in the previous section. This is followed by a discussion of our storage and indexing scheme and finally our query processing strategies.

6.1 Table Functions

We define four table functions: two spatial and two temporal. The following descriptions use the term *spatial geometry* to refer to an *SDO_GEOMETRY* object that would be stored in Oracle Spatial. We can think of a spatial geometry as the implementation of the class Spatial Region.

The *spatial_extent* table function implements the *spatial_extent* query operator described previously, and optional parameters are used to give the filtering functionality of the *spatial_restrict* operator. The signature for the table function is shown below:

```
spatial_extent (graphPattern VARCHAR,
    spatialVar VARCHAR, ontology RDFModels,
    <geom SDO_GEOMETRY>,
    <spatialRelation VARCHAR>)
returns AnyDataSet;
```

The *graphPattern* and *spatialVar* parameters represent the spatial context for the query, and *ontology* determines the temporal RDF graph to search against. This function returns a table with rows containing one column for each distinct variable in the graph pattern and one column for the spatial geometry. Each row contains the URI bound to each variable and the spatial geometry corresponding to the Spatial Region bound to *spatialVar*. Two optional parameters, a spatial geometry and a spatial relationship, can be used to filter the graph pattern instances. In this case, the table would only contain those graph pattern instances whose associated spatial geometries satisfy the specified spatial relation with the input spatial geometry. Our implementation currently supports the following spatial relationships: *disjoint*, *touch*, *overlap boundary intersect*, *overlap boundary disjoint*, *equal*, *contains*, *covers*, *inside*, *covered by*, *anyinteract* and *within distance*.

The example below shows a SQL query using the *spatial_extent* function that selects all soldiers who were on the crew of a vehicle used in a military event that occurred within 45 miles of a given point.

```
SELECT x
FROM TABLE (spatial_extent(
    '(?x <on_crew_of> ?y) (?y <used_in> ?z)
     (?z <occurred_at> ?l)', 'l',
    SDO_RDF_Models('military'),
    SDO_GEOMETRY(2001, 8265,
      SDO_POINT_TYPE(-71.796531, 44.304772,
      NULL), NULL, NULL),
    'GEO_DISTANCE(distance=45 unit=mile)'));
```

The *spatial_eval* table function implements the *spatial_eval* query operator defined previously. The signature for this table function is shown below:

```
spatial_eval (graphPattern VARCHAR,
    spatialVar VARCHAR, graphPattern2 VARCHAR,
    spatialVar2 VARCHAR, spatialRelation
    VARCHAR, ontology RDFModels)
return AnyDataSet;
```

graphPattern and *spatialVar* specify the first spatial context, and *graphPattern2* and *spatialVar2* specify the second spatial context. *spatialRelation* identifies the spatial relation for joining the two graph pattern instances. This function returns a table containing a column for each variable in *graphPattern* and *graphPattern2* and a column for each associated spatial geometry (s_1 and s_2). For each row in the resulting table, s_1 *spatialRelation* s_2 evaluates to *true*.

The example below shows a SQL query using the *spatial_eval* function that selects those platoons that train within 30 miles of *Platoon_12996*.

```
SELECT b
FROM TABLE (spatial_eval(
    '(<Platoon_12996> <trains_at> ?z)
     (?z <located_at> ?l)', 'l',
    '(?b <trains_at> ?c)
     (?c <located_at> ?d)', 'd',
    'GEO_DISTANCE(distance=30 unit=mile)',
    SDO_RDF_Models('military')));
```

The *temporal_extent* table function implements both the *temporal_extent* and *temporal_restrict* operators discussed previously. Optional parameters are used to perform filtering based on temporal properties. The signature for the table function is shown below.

```
temporal_extent (graphPattern VARCHAR,
```

```
    intervalType VARCHAR, ontology RDFModels,
    <start DATE>, <end DATE>,
    <temporalRel VARCHAR>)
return AnyDataSet;
```

This function takes three parameters as input, specifically a graph pattern, a String value specifying the interval type (*INTERSECT* or *RANGE*), and a parameter specifying the temporal RDF graph to search against. The table returned contains a column for each variable in the graph pattern and two *DATE* columns that specify the start and end of the time interval computed for the graph pattern instance. Three optional parameters, two *DATE* values to identify the boundaries of a time interval and a temporal relationship, can be used to filter the found graph pattern instances. In this case, assuming the *DATE* columns in the returned table are named *stDate* and *endDate*, each row in the result satisfies the condition $[stDate, endDate]$ *temporalRel* $[start, end]$. Our implementation currently supports seven temporal relationships: *before*, *after*, *during*, *overlap*, *during_inv*, *overlap_inv* and *anyinteract*.

The example below shows a SQL query using the *temporal_extent* function that selects all soldiers on the crew of a military vehicle and their corresponding platoons during the time interval [10:04:1942, 09:21:1944].

```
SELECT x, a
FROM TABLE (temporal_extent(
    '(?x <on_crew_of> ?y) (?y <used_in> ?z)
     (?x <assigned_to> ?a)',
    'INTERSECT',
    SDO_RDF_Models('military'),
    to_date('1942-10-04', 'yyyy-mm-dd'),
    to_date('1944-09-21', 'yyyy-mm-dd'),
    'DURING'));
```

The *temporal_eval* table function implements the *temporal_eval* operator described previously. It has the following signature:

```
temporal_eval (graphPattern VARCHAR,
    intervalType VARCHAR, graphPattern2
    VARCHAR, intervalType2 VARCHAR,
    temporalRel VARCHAR, ontology RDFModels)
return AnyDataSet;
```

graphPattern and *intervalType* specify the left hand side of the join operation, while *graphPattern2* and *intervalType2* specify the right hand side. *temporalRel* identifies the join condition. This function returns a table containing a column for each variable in *graphPattern* and *graphPattern2* and four *DATE* columns $(start_1, end_1, start_2, end_2)$ to indicate the derived time interval for each found graph pattern instance. For each row in the resulting table, $[start_1, end_1]$ *temporalRel* $[start_2, end_2]$ evaluates to *true*.

The example below shows a SQL query using the *temporal_eval* function that selects all pairs of soldiers ($s1$ and $s2$) such that $s1$ was leader of a platoon in *Division_2186* and $s2$ was leader of a platoon in *Division_2191* at overlapping times.

```
SELECT s1, s2
FROM TABLE (temporal_eval(
    '(?s1 <leader_of> ?y) (?y <platoon_of> ?z)
     (?z <battalion_of> <Division_2186>)',
    'INTERSECT',
    '(?s2 <leader_of> ?b) (?b <platoon_of> ?c)
     (?c <battalion_of> <Division_2191>)',
    'INTERSECT',
    'OVERLAP',
    SDO_RDF_Models('military')));
```

Multiple functions can be used in a single SQL query. This allows us to join the tables that result from a function execution and thus provides a mechanism for spatio-temporal-thematic queries. For example, the following query selects all soldiers who were on the crew of a vehicle that was used in a military event that occurred within an input bounding box and also returns the times at which this particular spatial relationship holds.

```
SELECT s.x, t.start_date, t.end_date
FROM
  TABLE (spatial_extent(
    '(?x <on_crew_of> ?y) (?y <used_in> ?z)
     (?z <occurred_at> ?l)', 'l',
    SDO_RDF_Models('military'),
    SDO_GEOMETRY(2003, 8265,
     NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3),
     SDO_ORDINATE_ARRAY(-81.970263, 41.061209,
     -80.518693, 41.964041)),
    'GEO_RELATE(mask=inside)')) s,
  TABLE (temporal_extent(
    '(?x <on_crew_of> ?y) (?y <used_in> ?z)
     (?z <occurred_at> ?l)',
    'INTERSECT',
    SDO_RDF_Models('military'))) t
WHERE s.x = t.x AND s.y = t.y AND s.z = t.z
    AND s.l = t.l;
```

6.2 Storage and Indexing Scheme

This section presents our storage and indexing scheme for spatial and temporal RDF data. We will first give an overview of existing Oracle capabilities for storing

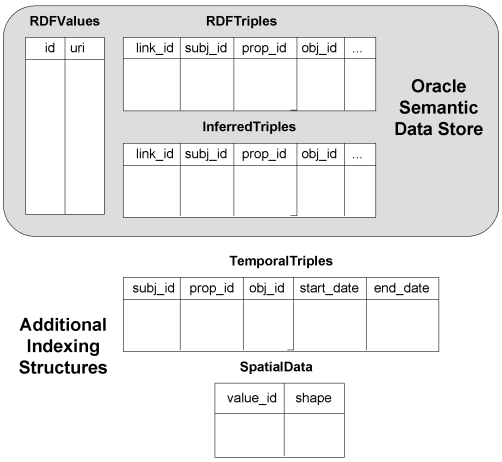


Fig. 5 Storage structures for RDF data. Existing tables of Oracle Semantic Data Store are shown at the top, and our additional tables for efficiently searching spatial and temporal data are shown at the bottom.

spatial geometries and RDF data and then present our spatial and temporal indexing schemes.

6.2.1 Existing Oracle Technologies

Oracle's Semantic Data Store [3] provides the capabilities to store, inference over, and query semantic data, which can be plain RDF descriptions and RDFS-based ontologies. To store RDF data, users create a model (ontology) to hold RDF triples. The triples are stored after normalization in two tables: an *RDFValues* table that stores RDF terms and a numeric id and an *RDFTriples* table that stores the ids of the subject, predicate and object of each statement. Users can optionally derive a set of inferred triples based on user-defined rules and/or RDFS semantics. These triples are materialized by creating a rules index and stored in a separate *InferredTriples* table. These storage structures are illustrated in Figure 5. A SQL table function is provided that allows issuing graph pattern queries against both asserted and inferred RDF statements.

Oracle Spatial [4] provides facilities to store, query and index spatial geometries. It supports the object-relational model for representing spatial geometries. A native spatial data type, *SDO_GEOMETRY*, is defined for storing vector data. Database tables can contain one or more *SDO_GEOMETRY* columns. Oracle Spatial supports spatial indexing on *SDO_GEOMETRY* columns, and provides a variety of procedures, functions and operators for performing spatial analysis operations.

6.2.2 Indexing Approach

In order to ensure efficient execution of graph pattern queries involving spatial and temporal predicates, we must provide a means to index portions of the RDF graph based on spatial and temporal values. Basically, this is done by building a table mapping Spatial Region instance URIs to their *SDO_GEOMETRY* representation and by building a modified *RDFTriples* table that also stores the temporal intervals associated with a triple. In order to build these indexes, users first load the set of asserted RDF statements into Oracle Semantic Data Store and build an RDFS rules index. After this step, users can run our indexing procedures to build spatial and temporal indexes for the RDF data.

Spatial Indexing Scheme: We provide the procedure *build_geo_index*() to construct a spatial index for a given ontology. This procedure first creates the table *SpatialData* (*value_id NUMBER*, *shape SDO_GEOMETRY*) for storing spatial geometries corresponding to instances of the class Spatial Region in the ontology. *value_id* is the id given to the URI of the Spatial Region instance in Oracle's *RDFValues* table, and *shape* stores the *SDO_GEOMETRY* representation of the Spatial Region instance (see Figure 5). This table is filled by querying the ontology for each Spatial Region instance, iterating through the results and creating and inserting *SDO_GEOMETRY* objects into the spatial indexing table. Finally, to enable efficient searching with spatial predicates on this table, a spatial index (R-Tree) is created on the *shape* column.

Temporal Indexing Scheme: Our temporal indexing scheme is a bit more complicated, as it must account for temporal labels on statements inferred through RDFS semantics. However, we only need to handle a subset of the RDFS inferencing rules. Only a subset is required because we are not interested in handling temporal evolution of the ontology schema. What we need to handle are temporal properties of instance data. Specifically, we need to account for temporal labels of inferred *rdf:type* statements and statements resulting from *rdfs:subPropertyOf*. *rdf:type* statements result from the following rules:

(1) $(x,$ *rdf:type*$, y) \wedge (y,$ *rdfs:subClassOf*$, z) \Rightarrow$
$(x,$ *rdf:type*$, z)$
(2) $(x, p, y) \wedge (p,$ *rdfs:domain*$, a) \Rightarrow (x,$ *rdf:type*$, a)$
(3) $(x, p, y) \wedge (p,$ *rdfs:range*$, b) \Rightarrow (y,$ *rdf:type*$, b)$

We infer instance statements from *rdfs:subPropertyOf* using the following rule:

(4) $(x, p, y) \wedge (p,$ *rdfs:subPropertyOf*$, z) \Rightarrow (x, z, y)$

In each case, if we assume that schema level statements in the ontology are eternally true, the temporal label of an inferred instance statement s is the union of the time intervals of all statements that can be used to infer s.

This temporal inferencing serves an important purpose in our scheme. Consider the example of a *Battle* event (b_1) that three platoons (p_1, p_2, p_3) participate in at different times:

(p_1, *participates_in*, b_1) : $[1, 3]$
(p_2, *participates_in*, b_1) : $[2, 5]$
(p_3, *participates_in*, b_1) : $[1, 4]$

Using rule 3 for generating *rdf:type* statements, we infer:

(b_1, *rdf:type*, *Battle*) : $[1, 5]$

In this case, $[1, 5]$ is the interval union and represents the overall duration or lifetime of b_1. Note that we are using relationships between entities and an event to automatically infer the overall duration of the event.

We provide the procedure *build_temporal_index* (*ontology*, *rules_index_name*, *min_start_time*, *max_end_time*) to construct a temporal index for a given ontology and rules index. The *ontology* parameter identifies the temporal RDF graph stored in Oracle; *rules_index_name* identifies the RDFS rules index associated with the ontology; *min_start_time* and *max_end_time* specify the earliest date and the latest date in the associated time domain. The purpose of these boundary parameters is to act as the start time and end time of statements that are eternally valid. All schema-level statements in the ontology are considered eternally valid. All asserted instance level statements with missing or incomplete temporal properties are also considered eternally valid. The *build_temporal_index* procedure executes in three phases.

The first phase creates the temporary table *asserted_temporal_triples* (*subj_id NUMBER*, *prop_id NUMBER*, *obj_id NUMBER*, *start DATE*, *end DATE*). The ontology is then queried to retrieve all temporal reifications. The subject, property, and object ids of each temporally reified statement and the start time and end time are inserted into this temporary table. Next, those statements with incomplete or missing temporal reifications are added to the *asserted_temporal_triples* table using *min_start_time* and *max_end_time* as a substitution for any missing temporal values. The final step of this phase scans the *asserted_temporal_triples* table and ensures that all asserted schema-level statements have [*min_start_time*, *max_end_time*] as their valid time.

At this point, we have recorded the temporal values for each asserted statement, and the second and third phases perform the temporal inferencing process and create the final *TemporalTriples* table (see Figure 5). Algorithm 1 shows the temporal inferencing procedure. We first create a second temporary table *redundant_triples* (*subj_id NUMBER*, *prop_id NUMBER*, *obj_id NUMBER*, *start DATE*, *end DATE*). Then, we iterate through the *asserted_temporal_triples* table and add any inferred statements to the *redundant_triples* table. In this step, the temporal label of the asserted statement is directly assigned to the corresponding inferred statements. This procedure results in possibly redundant and overlapping intervals for each statement, so a third phase, shown in Algorithm 2, iterates through this table and cleans up the time intervals for each statement. The cleanup phase first sorts *redundant_triples* by (*subj_id*, *prop_id*, *obj_id*, *start_date*) and then makes a single pass over the sorted set to merge overlapping intervals having the same (*subj_id*, *prop_id*, *obj_id*) values. The final result of this process is a table *TemporalTriples* (*subj_id NUMBER*, *prop_id NUMBER*, *obj_id NUMBER*, *start DATE*, *end DATE*) that contains the complete set of asserted and inferred temporal triples.

Algorithm 1 *TemporalInference*

```
1: create temporary table
   redundant_triples (subj_id, prop_id, obj_id, start, end)
2: for each row r ∈ asserted_temporal_triples do
3:    if (r.prop = rdf:type) then
4:       for each Class C ∈ SuperClasses(r.obj) do
5:          insert row (r.subj, rdf:type, C, r.start_date,
            r.end_date) into redundant_triples
6:       end for
7:    else
8:       for each property P ∈ SuperProperties(r.prop) do
9:          insert row (r.subj, P, r.obj, r.start_date, r.end_date)
            into redundant_triples
10:      end for
11:      x ← domain(r.prop)
12:      for each Class C ∈ SuperClasses(x) ∪ {x} do
13:         insert row (r.subj, rdf:type, C, r.start_date,
            r.end_date) into redundant_triples
14:      end for
15:      y ← range(r.prop)
16:      for each Class C ∈ SuperClasses(y) ∪ {y} do
17:         insert row (r.obj, rdf:type, C, r.start_date,
            r.end_date) into redundant_triples
18:      end for
19:   end if
20: end for
```

The complexity of the temporal inferencing procedure is as follows. Assume we have n asserted triples in the dataset and c classes and p property types in the ontology schema. In the worst case, every property would be a subclass of every other property; every

Algorithm 2 *MergeTemporalIntervals*

```
1: create table
   TemporalTriples (subj_id, prop_id, obj_id, start, end)
2: sort redundant_triples by subj_id, prop_id, obj_id, start
3: r ← first row of redundant_triples
4: curr_row ← r
5: for each row r remaining in redundant_triples do
6:    if (r.subj_id = curr_row.subj_id
          and r.prop_id = curr_row.prop_id
          and r.obj_id = curr_row.obj_id) then
7:       if (r.start ≤ curr_row.end
             and r.end > curr_row.end) then
8:          curr_row.end ← r.end
9:       end if
10:      if (r.start > curr_row.end) then
11:         insert row (curr_row.subj_id, curr_row.prop_id,
               curr_row.obj_id, curr_row.start, curr_row.end)
               into TemporalTriples
12:         curr_row.start ← r.start
13:         curr_row.end ← r.end
14:      end if
15:   else
16:      insert row (curr_row.subj_id, curr_row.prop_id,
            curr_row.obj_id, curr_row.start, curr_row.end)
            into TemporalTriples
17:      curr_row ← r
18:   end if
19: end for
20: insert into TemporalTriples
    SELECT (subj_id, prop_id, obj_id, min_start, max_end)
    FROM InferredTriples
    WHERE (subj_id, prop_id, obj_id)
    NOT IN TemporalTriples
```

class would be a subclass of every class, and each property would have every class in its domain and range. In this case, we would add $2c + p$ triples for every asserted triple, yielding $\boldsymbol{O}(n(c + p))$ for Algorithm 1. In Algorithm 2, we must sort this set of statements and then make a single pass over the sorted set, yielding $\boldsymbol{O}(n(c+p)\log(n(c+p))+n(c+p))$. This gives an overall complexity of $\boldsymbol{O}(n(c+p)\log(n(c+p)))$ for the temporal inferencing procedure.

6.2.3 Function Implementation

In this section we discuss the implementation of the SQL table functions defined previously. The table functions were implemented using Oracle's *ODCITable* interface methods. With this scheme, users implement a *start*(), *fetch*() and *close*() method for the table function. In *start*(), the query parameters are parsed; a SQL query is prepared and executed, and a handle to the query is stored in a scan context parameter. The *fetch*() method fetches a subset of rows from the prepared query and returns them. This method is invoked as many times as necessary by the kernel until all result rows are returned. The *close*() method performs cleanup operations after the last *fetch*() call. We also implement an optional *describe*() method, which is used notify the kernel of the structure of the data type to be returned (i.e., columns of the table). This method is necessary because the number of columns in the return type depends on the graph pattern and cannot be determined until query compilation time.

Graph Pattern to SQL Translation: Each of the table functions takes a graph pattern and ontology as input. The conversion of a graph pattern to a SQL query is therefore a central component of each function. The graph pattern is transformed into a self-join query against the *TemporalTriples* table corresponding to the input ontology. The graph pattern translation algorithm is shown in Algorithm 3. The algorithm first parses the graph pattern and builds a mapping between tokens (i.e., variables and URIs) and a list of their occurrences in the graph pattern. To denote an occurrence, we record the triple pattern number and the position within the triple pattern (i.e. subject, predicate or object). We also build a mapping from URIs to their ids in the *RDFValues* table. We then use these mappings to build a self-join query over the *TemporalTriples* table with two sets of conditions in the where clause: (1) restrictions based on the ids of the URIs in the graph pattern and (2) join conditions based on variable correspondences between triple patterns. We must also join with the *RDFValues* table to resolve the ids of URIs bound to variables to actual URI Strings.

The example below illustrates the transformation process. The resulting SQL query assumes that the ids of *on_crew_of* and *used_in* are 1 and 2, respectively.

```
(?a <on_crew_of> ?b)(?b <used_in> ?c)
```

```
SELECT rv1.uri, rv2.uri, rv3.uri
FROM TemporalTriples tt1, TemporalTriples tt2,
   RDFValues rv1, RDFValues rv2,
   RDFValues rv3
WHERE tt1.prop_id = 1 and tt2.prop_id = 2
   and tt1.obj_id = tt2.subj_id
   and rv1.id = tt1.subj_id
   and rv2.id = tt1.obj_id
   and rv3.id = tt2.obj_id;
```

Spatial Functions: Spatial functions are implemented by augmenting the base graph pattern query discussed in the previous section.

Algorithm 4 shows the query processing procedure for *spatial_extent* function. We modify the base query as follows. First we identify the appropriate column (i.e., *subj_id*, *prop_id*, or *obj_id*) in the *RDFTriples* table that corresponds to the position of the *spatial_variable* parameter. Then we add an additional join matching ids

Algorithm 3 *Graph Pattern Translation*

Input:
GP: graph pattern
G_t: temporal RDF graph
Output:
selectStr: select portion of SQL query
$fromStr$: from portion of SQL query
whereStr: where portion of SQL query
$varMap$: mapping between variables and a list of their occurrences in GP

1: *selectStr* ← `'SELECT'`
2: $fromStr$ ← `'FROM'`
3: *whereStr* ← `'WHERE'`
4: declare *mapRecord* as 2-tuple (*triple_pattern_num*, *pos*)
5: declare Map $uriMap$ (String, List of *mapRecord*)
6: declare Map $varMap$ (String, List of *mapRecord*)
7: declare Map $uriIdMap$ (String, Integer)
8: parse GP and populate $uriMap$, $varMap$
9: **for** each var $v \in varMap$ **do**
10: $currList \leftarrow varMap(v)$
11: add `'tt_<currList(1).triple_pattern_num>. <currList(1).pos> as <v>'` to *selectStr*
12: **end for**
13: **for** $i = 1$ to *numTriplePatterns* **do**
14: add `'TemporalTriples tt_<i>'` to $fromStr$
15: **end for**
16: **for** $i = 1$ to *numVars* **do**
17: add `'RDFValues rv_<i>'` to $fromStr$
18: **end for**
19: populate $uriIdMap$ from $RDFValues$
20: **for** each URI $u \in uriMap$ **do**
21: $currList \leftarrow uriMap(u)$
22: **for** $i = 1$ to $length(currList)$ **do**
23: add `'tt_<currList(i).triple_pattern_num>. <currList(i).pos> = <uriIdMap(u)>'` to *whereStr*
24: **end for**
25: **end for**
26: **for** each var $v \in varMap$ **do**
27: $currList \leftarrow varMap(v)$
28: **for** $i = 1$ to $length(currList) - 1$ **do**
29: add `'tt_<currList(i).triple_pattern_num>. <currList(i).pos> = tt_<currList(i+1).triple_pattern_num>. <currList(i+1).pos>'` to *whereStr*
30: **end for**
31: **end for**

from the *TemporalTriples* table with *value_ids* in the *SpatialData* table to select the id of the *SDO_GEOMETRY* object. We must return the id, rather than the *SDO_GEOMETRY* object, from *SpatialData* because object types cannot be returned from table functions. In the case of optional result filtering, we need to modify the where clause so that we filter the spatial features from *SpatialData* according to the input spatial feature and spatial relation. This is done by adding the appropriate *sdo_relate* or *sdo_within_distance* predicate available in Oracle Spatial. For example, given the query:

```
spatial_extent (..., sdo_geometry (...),
'geo_relate (inside)')
```

we would modify the query as follows:

```
WHERE ... AND
sdo_relate (geo.shape,
sdo_geometry (...), 'mask=inside') = 'true'.
```

Algorithm 4 *spatial_extent*

Input:
GP: graph pattern
svar: spatial variable identifier
G_t: temporal RDF graph
$filterParams$: optional filtering parameters
Output:
rows: query results

1: *GraphPatternTranslation* (GP, G_t, *selectStr*, $fromStr$, *whereStr*, $varMap$)
2: add `'SpatialData.id as geom'` to *selectStr*
3: add `'SpatialData'` to $fromStr$
4: $currList \leftarrow varMap(svar)$
5: add `tt_<currList(1).triple_pattern_num>. <currList(1).pos> = SpatialData.value_id'` to *whereStr*
6: **if** ($filterParams$ are present) **then**
7: parse $filterParams$ and add appropriate `sdo_relate` or `sdo_within_distance` predicate to *whereStr*
8: **end if**
9: $sctx \leftarrow$ parse ($selectStr + fromStr + whereStr$)
10: **while** $sctx.results\_remaining()$ **do**
11: $rows \leftarrow sctx.fetch\_rows()$
12: **return** *rows*
13: **end while**

Algorithm 5 shows the query processing procedure for the *spatial_eval* function. We implement what is essentially a nested loop join (NLJ) using the basic *spatial_extent* and filtered *spatial_extent* operators. We first construct and execute a basic *spatial_extent* query in the *start*() routine. Next, in the $fetch()$ routine, we consume a row from the *spatial_extent* query and then construct and execute the appropriate filtered *spatial_extent* query using the second pair of graph pattern and spatial variable parameters and the spatial relation parameter. This is repeated until all rows in the outer *spatial_extent* query are consumed.

Temporal Functions: The implementation of the temporal functions does not translate directly to a SQL query. We must do some extra processing of the base query results in the $fetch()$ routine to form a single time interval for each found graph pattern instance.

Algorithm 6 shows the query processing strategy for the *temporal_extent* function. We first augment the basic graph pattern query in *start*() to also select the start and end values for each temporal triple in the graph pattern instance. In the $fetch()$ routine, to compute the final temporal interval for each graph pattern instance,

Algorithm 5 *spatial_eval*

Input:
GP_1: graph pattern
var_1: spatial variable identifier
GP_2: graph pattern
var_2: spatial variable identifier
spatialRel: spatial relation
G_t: temporal RDF graph
Output:
rows: query result

```
1: sctx ← parse (spatial_extent(GP1, var1, Gt))
2: while sctx.results_remaining() do
3:    outer_rows ← sctx.fetch_rows()
4:    for each row r1 ∈ outer_rows do
5:       inner_rows ← execute(spatial_extent(GP2, var2,
            Gt, r.geom, inverse of spatialRel))
6:       for each row r2 ∈ inner_rows do
7:          add r1.vars, r.geom, r2.vars, r2.geom to rows
8:       end for
9:    end for
10:   return rows
11: end while
```

we examine the start and end times for each triple and select the earliest start and latest end (RANGE) or the latest start and earliest end (INTERSECT). In the case of INTERSECT, if the final start value is later than the final end value then the computed interval is not valid and is not included in the final result. When the optional filtering parameters are specified, we must perform additional checking of the found graph patterns to ensure they satisfy the filter condition. In addition to these extra computations in $fetch()$, we augment the base query in *start*() with a series of predicates involving the start and end times of each statement in the graph pattern. This is done to filter the results as much as possible in the base query to reduce subsequent overhead in $fetch()$. To illustrate these additional predicates, consider the following *temporal_extent* query and corresponding base query:

```
SELECT ...
FROM TABLE(temporal_extent(
'(?x <on_crew_of> ?y) (?y <used_in> ?z)',
'range', 1942, 1944, 'during'));

SELECT ...
FROM ..., TemporalTriples t1,
TemporalTriples t2
WHERE ... and t1.start > 1942
and t2.end < 1944
and t2.start > 1942
and t2.end < 1944;
```

Algorithm 7 shows the query processing strategy for *temporal_eval*. The implementation of the *temporal_eval* operator is similar to the implementation of *spatial_eval*. We first build a basic *temporal_extent* query involving the first pair of graph pattern and interval type parameters, which is executed in the *start*() routine. Next, in $fetch()$, we consume a row from the basic *temporal_extent* query and execute an appropriate filtered *temporal_extent* query using the second pair of graph pattern and interval type parameters. This query uses the time interval from the current outer *temporal_extent* result and the inverse of the temporal relation parameter from the original *temporal_eval* query.

Algorithm 6 *temporal_extent*

Input:
GP: graph pattern
IT: interval type
G_t: temporal RDF graph
$filterParams$: optional filtering parameters
Output:
rows: query result

```
1: GraphPatternTranslation (GP, Gt, selectStr, fromStr,
     whereStr, varMap)
2: for each i in 1 to graphPatternLen do
3:    add 'tt_<i>.start as st_<i>,
      tt_<i>.end as ed_<i>' to selectStr
4: end for
5: if (filterParams are present) then
6:    parse filterParams and add appropriate constraints to
      whereStr
7: end if
8: sctx ← parse(selectStr + fromStr + whereStr)
9: while sctx.results_remaining() do
10:   rows ← sctx.fetch_rows()
11:   for each row r ∈ rows do
12:      if (IT = 'RANGE') then
13:         curr_interval ← [min(r.st), max(r.ed)]
14:      end if
15:      if (IT = 'INTERSECT') then
16:         if max(r.st) ≤ min(r.ed) then
17:            curr_interval ← [max(r.st), min(r.ed)]
18:         end if
19:      end if
20:      if (curr_interval is defined) then
21:         if (filterParams are present and
                curr_interval, t_interval
                satisfies filter condition) then
22:            add r.vars, curr_interval to rows
23:         end if
24:         if (filterParams are not present) then
25:            add r.vars, curr_interval to rows
26:         end if
27:      end if
28:   end for
29:   return rows
30: end while
```

7 Experimental Evaluation

The experimental evaluation of our implementation is described in this section. All code was written in PL/SQL, and all experiments were conducted using Oracle 10g Release 2 running on a Sun Fire V490 server with four

Algorithm 7 *temporal_eval*

Input:
GP_1: graph pattern
IT_1: interval type
GP_2: graph pattern
IT_2: interval type
temporalRel: temporal relation
G_t: temporal RDF graph
Output:
rows: query results

```
1: sctx ← parse (temporal_extent(GP1, IT1, Gt))
2: while sctx.results_remaining() do
3:    outer_rows ← sctx.fetch_rows()
4:    for each row r1 ∈ outer_rows do
5:       inner_rows ← execute( temporal_extent(GP2, IT2,
             Gt, r1.interval, inverse of temporalRel))
6:       for each row r2 ∈ inner_rows do
7:          add r1.vars, r.interval, r2.vars, r2.interval to rows
8:       end for
9:    end for
10:   return rows
11: end while
```

1.8 GHz Ultra Sparc IV processors and 8GB of main memory. The operating system used was 64-bit Solaris 9. The database used an 8 KB block size and was configured with a 512 MB buffer cache and a `pga_aggregate_target` size of 512 MB. The times reported for each query were obtained as follows. The query was run once initially to warm up the database buffers and then timed for 10 consecutive executions. We report the mean execution time over these 10 consecutive executions. Times were obtained by querying for `systimestamp` before and after query execution and computing the difference.

Testing details (e.g., queries used and datasets) are available at http://knoesis.wright.edu/students/mperry/stt_journal/Test-Details.html.

7.1 Datasets

We conducted experiments using two RDF datasets. One consisted of synthetically generated RDF data corresponding to historical analysis of WWII (SynHist), and the other (GovTrack) consisted of real-world RDF data from the political domain that we obtained from http://www.govtrack.us/data/rdf/. Table 1 shows the characteristics of these datasets.

SynHist Dataset: Five synthetically generated datasets (SH1 - SH5) were used in our experiments. The datasets correspond to a historical battlefield analysis ontology schema that we created. The ontology schema defined 15 class types and 9 property types. Each dataset was created in three phases. First we populated the thematic portion of the ontology. Second we added spatial information, and in the final step we generated temporal labels for the statements in the populated ontology.

To populate the thematic portion of the battlefield analysis ontology, we used the ontology population tool described in [54]. This tool inputs an ontology schema and relative probabilities for generating instances of each class and property type. Based on these probabilities, it generates instance data, which, in effect, simulates the population of the ontology. We integrated these RDF graphs with the upper-level ontology described in Section 4.2 by adding a handful of *rdfs:subClassOf* statements to each RDF dataset.

To add spatial aspects to this dataset, we randomly assigned a spatial geometry to each instance of Spatial Region in the ontology. We used year 2000 census block group boundary polygons from the US Census Bureau [6] for the spatial geometries. Differently-sized sets of contiguous US States were chosen in proportion with the ontology size.

The final phase of dataset generation assigned temporal labels to statements in the ontology. Temporal intervals were randomly assigned to each asserted instance statement. Start times and end times for each interval were randomly selected with uniform probability from two overlapping date ranges. We ensured that each interval was valid (i.e., start time earlier than end time) before adding it to the dataset.

GovTrack Dataset: The GovTrack RDF dataset contains data about activities of the US Congress. More specifically, it contains data describing politicians, bills, voting records, political organizations, political offices, and terms held by politicians. The ontologies used for this dataset contained 74 classes and 139 properties. 22 classes and 47 properties were actually used in the instance data.

Some transformations and enhancements of the dataset were needed to make it appropriate for experimentation. We integrated the ontologies used with the upper-level ontology described in Section 4.2 using *rdfs:subClassOf* statements. The GovTrack data contained a significant amount of temporal information. However, this information was encoded using separate properties rather than as temporal RDF. For example, an instance of the class *Term* would have a *start_date* property and an *end_date* property. A preprocessing step was therefore needed to transform the dataset into a temporal RDF graph. This step would, for example, remove the existing *start_date* and *end_date* statements for a *Term* and then add the temporal label [*start_date*, *end_date*] to all statements involving the *Term*. To enhance the dataset with spatial data, we linked *Congressional_District* instances with their corresponding

Table 1 Characteristics of GovTrack and SynHist datasets

Dataset	Num Triples (Asserted + Inferred)	Size of TemporalTriples Table (MB)	Num Spatial Features	Avg Num Points per Polygon	Size of SpatialData Table (MB)
SH1	120,665	6	3,470	98	3
SH2	1,623,404	66	28,488	63	17
SH3	7,002,389	227	77,440	67	50
SH4	19,152,364	754	169,722	56	94
SH5	28,905,693	1,144	244,653	61	145
GT1	5,994,841	264	3,433	2,352	2
GT2	10,471,121	448	3,433	2,352	2
GT3	25,918,237	1,156	3,433	2,352	2

boundary polygons available from the US Census [6]. We used boundary files for the 106th - 110th Congress.

We created three differently-sized subsets of the GovTrack data (GT1 - GT3). GT1 contained information on bills and voting from the 106th Congress. GT2 used the 106th and 107th Congress, and GT3 used the 106th - 110th Congress.

7.2 Experiments

Our experiments were designed to characterize the overall performance of our approach with respect to (1) dataset size and (2) graph pattern complexity.

For testing, B^+-Tree indexes were created on each column of the *TemporalTriples* table and on the *value_id* column of the *SpatialData* table, and an R-Tree index was created on the *shape* column of *SpatialData*. We also created four composite B^+-Tree indexes on the *TemporalTriples* table to allow for efficient index-based joins: (*prop_id*, *subj_id*, *obj_id*) and (*prop_id*, *obj_id*, *subj_id*) for spatial operators and (*prop_id*, *subj_id*, *obj_id*, *start*, *end*) and (*prop_id*, *obj_id*, *subj_id*, *start*, *end*) for temporal operators.

Table 2 shows the execution time for creating RDFS rules indexes using Oracle Semantic Data Store and for executing our temporal inferencing procedure. Times were obtained using the timing option of SQLPlus. The results show that the time required for temporal inferencing is comparable to the time required for RDFS rules index creation. In addition, the procedures take longer on the GovTrack dataset due to its larger ontology schema. The larger schema is also responsible for the greater number of inferred statements relative to the number of asserted statements.

In the following, we refer to two different graph pattern types: unselective and selective. An *unselective graph pattern* contains constant URIs in the predicate position in each triple pattern and variables in each subject and object position, for example:

```
(?x <usgov:cosponsor> ?y)
(?x <usgov:sponsor> ?z)
(?x <usgov:inCommittee> ?c)
```

A *selective graph pattern* has constant URIs in each predicate position and additionally contains a constant URI in the subject and/or object position in at least one triple pattern, for example:

```
(?p <usgov:hasRole> ?y)
(?y <usgov:forOffice>
  <usgov:congress/senate/va>)
```

7.2.1 Scalability with respect to Dataset Size

Tables 3 and 4 summarize the results of our experimentation with respect to dataset size. These experiments were designed to test the general performance of our operators for the GovTrack and SynHist datasets.

Basic temporal_extent: Queries G1 - G4 and H1 - H4 tested the scalability of the *temporal_extent* operator for the GovTrack and SynHist datasets. Query G1, G2 and H1, H2 measure the response time (i.e. time to return the first 1000 rows) for an unselective graph pattern query, and G3, G4 and H3, H4 tested the execution time for a selective graph pattern query. For both query types and both datasets, query execution time is near constant as the dataset size grows. This is a result of the index-based nested loop join (NLJ) strategy used by the DBMS, which tends to have execution times proportional to the result set size. The 5-triple queries are slower than the 3-triple queries as a result of the additional joins needed to evaluate the query.

Filtered temporal_extent: Query G5, G6 and H5, H6 tested the scalability of the *temporal_extent* operator with filtering. These queries used an unselective graph pattern in combination with very selective temporal conditions. The queries show relatively constant execution time for the GovTrack dataset but show more of a linear growth for the SynHist dataset. In each case, the DBMS uses an index-based NLJ strategy over the

Table 2 Execution time for RDFS rules index creation and temporal inferencing

Dataset	Num Triples		Time (HH:MM:SS)	
	Asserted	Inferred	RDFS Idx	Temporal Inference
SH1	70,640	50,025	00:02:52	00:00:26
SH2	980,253	643,151	00:06:35	00:06:27
SH3	4,294,783	2,707,606	00:26:35	00:22:48
SH4	11,593,162	7,559,202	01:02:46	01:00:34
SH5	17,615,502	11,290,191	01:30:57	01:29:29
GT1	2,959,281	3,035,560	00:13:40	00:21:29
GT2	5,245,453	5,225,668	00:24:08	00:27:46
GT3	12,819,641	13,098,596	01:49:06	01:52:03

Table 3 Experimental results for query execution time with respect to ontology size for GovTrack datasets

Query	Operator	Relation	Graph Pattern		Result Size	Execution Time (msec)		
			Num Triples	Num Vars		GT1	GT2	GT3
G1	T-Ext		3	4	1000	386	388	388
G2	T-Ext		5	6	1000	562	540	574
G3	T-Ext		3	3	94	35	35	36
G4	T-Ext		5	5	94	53	53	54
G5	T-Filter	INT/DURING	3	4	451	375	360	380
G6	T-Filter	INT/AFTER	5	6	483	424	421	324
G7	T-Eval	INT/DURING	3 / 3	3 / 3	90	568	580	897
G8	T-Eval	INT/BEFORE	3 / 2	3 / 2	120	196	195	196
G9	S-Ext		3	4	1000	392	411	404
G10	S-Ext		5	6	1000	540	545	547
G11	S-Ext		3	3	437	155	152	153
G12	S-Ext		5	5	428	230	227	226
G13	S-Filter	INSIDE	3	4	166	721	723	719
G14	S-Filter	ANYINTERACT	5	6	559	1088	1072	1087
G15	S-Filter	INSIDE	3	4	283	215	217	215
G16	S-Filter	ANYINTERACT	5	6	442	506	463	503
G17	S-Eval	ANYINTERACT	4 / 1	4 / 2	99	7827	7840	7829
G18	S-Eval	w/in DIST	4 / 2	4 / 2	24	24448	24435	24446
G19	S-Eval	ANYINTERACT	4 / 1	4 / 2	15	80	85	80
G20	S-Eval	w/in DIST	4 / 2	4 / 2	73	790	787	786

Table 4 Experimental results for query execution time with respect to ontology size for SynHist datasets

Query	Operator	Relation	Graph Pattern		Result Size	Execution Time (msec)				
			Num Triples	Num Vars		Mil1	Mil2	Mil3	Mil4	Mil5
H1	T-Ext		3	4	1000	400	403	417	437	516
H2	T-Ext		5	6	1000	608	609	616	611	617
H3	T-Ext		3	3	91	36	36	36	36	37
H4	T-Ext		5	5	178	92	94	94	98	87
H5	T-Filter	INT/OVERLAP	3	4	251	126	170	159	144	353
H6	T-Filter	INT/OVERLAP	5	6	280	107	224	468	1072	1734
H7	T-Eval	INT/OVERLAP	3 / 2	3 / 3	49	85	121	245	697	866
H8	T-Eval	INT/ANYINTERACT	3 / 3	3 / 3	140	226	228	227	229	229
H9	S-Ext		3	4	1000	382	381	384	383	387
H10	S-Ext		5	6	1000	551	550	550	545	549
H11	S-Ext		3	3	183	55	54	54	55	55
H12	S-Ext		5	5	224	108	109	109	109	112
H13	S-Filter	OVERLAP	3	4	449	363	365	367	367	369
H14	S-Filter	w/in DIST	5	6	136	195	197	197	195	197
H15	S-Eval	w/in DIST	3 / 1	3 / 2	130	405	405	409	418	427
H16	S-Eval	w/in DIST	2 / 2	2 / 3	57	228	160	164	168	172

composite indexes containing start date and end date information.

These particular queries represent a challenging case for the *temporal_extent* operator. Because the INTERSECT / RANGE interval derived for a graph pattern instance is constructed dynamically from the temporal labels of each edge in the graph pattern instance, we cannot directly index these derived values. We must instead apply the temporal filtering condition to each graph pattern instance as it is being constructed, which can lead to a very large set of intermediate results that are later discarded. The unnecessary intermediate results are generated because, in many cases, we cannot exclude a graph pattern instance until it is fully constructed and the final derived time interval is known. We try to alleviate this problem by placing limited temporal constraints on each triple pattern in the graph pattern. These initial constraints can reduce the number of intermediate results generated, but the amount of reduction depends on the specific interval type and temporal relation used. This issue is further explored in Section 7.2.3.

The difference in the scalability of the queries over the GovTrack dataset is a result of the characteristics of the time intervals in each dataset. The triples in the SynHist dataset have much longer time intervals with respect to the maximum start and end times of the whole dataset as compared to the GovTrack dataset. As a result, the temporal filtering conditions that can be placed on each triple in the graph pattern are ultimately less selective, leading to larger growth in intermediate results as the dataset size increases.

temporal_eval: Queries G7, G8 and H7, H8 tested the scalability of the *temporal_eval* operator. Selective graph patterns were used for both the left hand side (LHS) and right hand side (RHS) graph pattern in G7, G8 and H7. H8 used a LHS graph pattern and an unselective RHS graph pattern. The results show that execution times for G8 and H7 are relatively constant across each dataset, but queries G7 and H8 show a linear growth in execution time. The growth in execution time for H7 is a result of the larger sets of intermediate results generated by the unselective RHS graph pattern as the dataset size grows. The results for G7 are a result of the DURING temporal relation. This particular relation only allows weak temporal constraints on each triple pattern, leading to a growth in intermediate results. This is explored further in Section 7.2.3.

Basic spatial_extent: Queries G9 - G12 and H9 - H12 tested the scalability of the *spatial_extent* operator. G9, G10 and H9, H10 measured the response time (first 1000 rows) for unselective graph pattern queries, and G11, G12 and H11, H12 measured the execution time of selective graph pattern queries. For both query types and both datasets, query execution time is near constant as the dataset size grows. This is a result of the index-based NLJ strategy used by the DBMS, which tends to have execution times proportional to the result set size. The 5-triple queries are slower than the 3-triple queries as a result of the additional joins needed to evaluate the query. The query execution times are roughly equivalent to those for basic *temporal_extent* queries, as the extra join with the *SpatialData* table needed for the spatial queries is offset by the extra overhead of deriving INTERSECT / RANGE time intervals for the temporal queries.

Filtered spatial_extent: Queries G13 - G16 and H13, H14 tested the scalability of the filtering capability of the *spatial_extent* operator. Each query used an unselective graph pattern in combination with a selective spatial predicate. For each query, execution times are relatively constant across each dataset, which is a result of the index-based NLJ strategy used by the DBMS. The slower times reported in G13 and G14 are a result of the very complex spatial geometries used to represent congressional districts, which increase the time needed to perform the spatial filtering using the R-Tree index. Queries G15 and G16 used the same graph patterns and filtering parameters but were run over a modified dataset substituting random census block group polygons for the congressional district polygons. The execution times are significantly faster using these spatial geometries.

In the SynHist dataset, we see that the spatial filtering queries scale better than temporal filtering queries. Unlike INTERSECT/RANGE intervals, the spatial geometries can be indexed because they are not dynamically created. The spatial filtering queries consequently scale better because we can consistently reduce the search space using the spatial index and do not get as much growth in intermediate results as the dataset size increases.

spatial_eval: Queries G17 - G20 and H15, H16 tested the scalability of *spatial_eval*. G17, G19 and H15 used selective LHS graph patterns and unselective RHS graph patterns. G18, G20 and H16 used selective RHS and LHS graph patterns. In each case, execution times are relatively constant across each dataset due to the index-based join strategy and the consistent filtering from the spatial index. The execution times of G17 and G18 are much slower due to the complexity of the congressional district polygons. To evaluate a *spatial_eval* query over

the GovTrack dataset, we must compute spatial relations between two complex spatial geometries, which is an expensive operation. We had better performance with filtered *spatial_extent* queries because we were computing spatial relations between a complex spatial geometry in the dataset and a simple spatial geometry specified in the query. G19 and G20 are the same *spatial_eval* queries using census block group polygons, which yield much faster execution times.

7.2.2 Scalability with respect to Graph Pattern Size

Our next experiments are designed to test the scalability of various operators with respect to query complexity: that is, the size of the graph pattern used. We have focused on *temporal_extent* and *spatial_extent* operators, as their functionality forms the basis of our implementation.

Filtered temporal_extent: Experiment GP1 tested the scalability of a filtered *temporal_extent* query as the complexity of the graph pattern used in the query increased. We used unselective graph patterns and very selective temporal predicates in each case. We ran one set of queries over the SH5 dataset and one set of queries over the GT3 dataset.

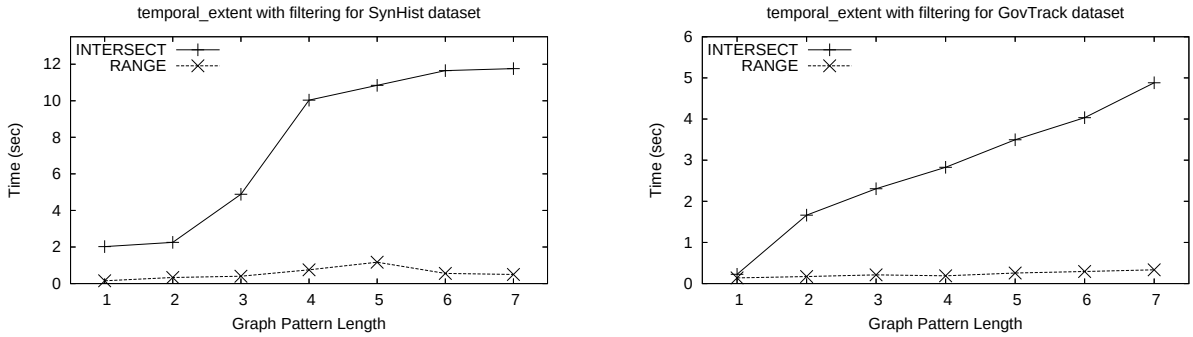
The key to the performance of filtered *temporal_extent* queries is the amount the search space can be reduced by placing partial temporal constraints on each triple pattern in the graph pattern. As we noted earlier, the effectiveness of these partial temporal constraints depends on the particular interval type and temporal relation used in a query.

The objective of this experiment was to characterize the performance of filtered *temporal_extent* queries in both the worst-case scenario (very limited initial temporal filtering) and the best-case scenario (complete initial temporal filtering). An INTERSECT interval type in combination with a DURING temporal relation represented the worst-case. In this situation, we can only enforce that the valid time interval of each triple does not end before the query interval starts or start after the query interval ends. In contrast, with a RANGE interval type and a DURING temporal relation, we can enforce that each triple starts after the query interval starts and ends before the query interval ends. These conditions completely filter out any unwanted graph pattern instances, and this query represents a best-case. Figure 6 shows the execution times for a best-case and worst-case query for unselective graph patterns varying in size from one triple to seven triples. We can see that execution time grows roughly linearly in each case, but performance is significantly worse with the INTERSECT temporal relation. The performance is better for the GovTrack dataset because of the nature of the temporal intervals in each dataset as we discussed in Section 7.2.1. The execution time for queries over the SynHist dataset tends to grow more rapidly at first and then taper off as the graph pattern gets more complex. This trend is a result of the selectivity of the graph pattern itself. In this dataset, there are fewer instances of the more complex graph patterns. This slows the growth in intermediate results, so not as much additional temporal filtering is needed in the *fetch*() method.

Filtered spatial_extent: Experiment GP2 tested the scalability of filtered *spatial_extent* queries. The graphs in Figure 7 show the execution times for queries involving unselective graph patterns and selective spatial filtering conditions. As the graph pattern size grows, the query execution times show linear scalability on both datasets and are much faster than the worst-case temporal queries. Because the spatial values in our dataset are not dynamically derived, we can effectively index them. The faster execution times result from the more effective spatial indexing. The spatial index is used initially to select the nodes satisfying the spatial filtering condition, which reduces the search space for evaluating the rest of the graph pattern. The queries over the GovTrack dataset have slower execution times because spatial computations are more expensive for the complex spatial geometries in the GovTrack dataset.

Basic temporal_extent: Experiment GP3 tested the scalability of basic *temporal_extent* queries using selective graph patterns. Figure 8 shows query execution time for basic *temporal_extent* queries as graph pattern size ranges from 1 triple to 10 triples. The number of result rows returned from the query is also shown in the graphs. These graphs show that performance is quite good for selective graph pattern queries even as the graph patterns grow relatively large. In each case, the execution times grow roughly linearly as the graph pattern size increases when the effects of the result set size are taken into account. The DBMS starts with the most selective triple pattern and uses an index-based join to construct the rest of the graph pattern instance. The initial selection dramatically cuts down the search space and results in the fast execution times for these queries.

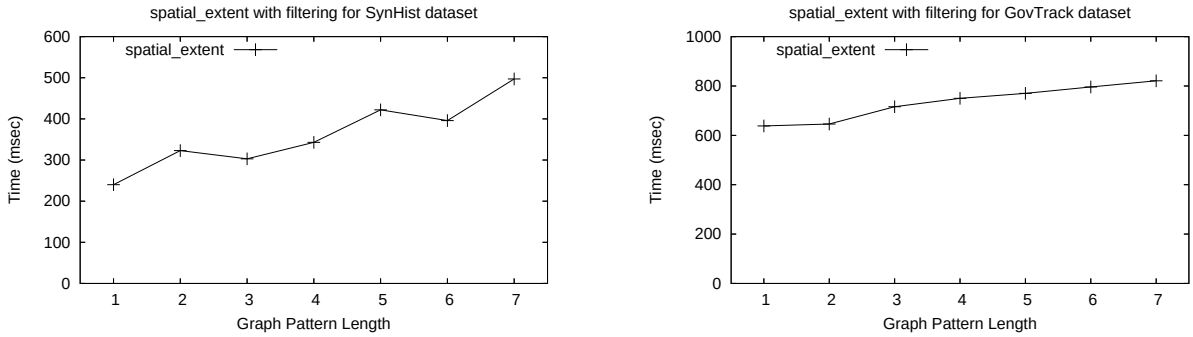
Basic spatial_extent: Experiment GP4 tested the scalability of basic *spatial_extent* queries using selective graph patterns. Figure 9 shows the execution time of basic *spatial_extent* queries as graph pattern size ranges from 2 to 10 triples. The result set size of each query is also shown in the figure. Execution time grows linearly as



(a) SynHist Dataset

(b) GovTrack Dataset

Fig. 6 Experiment GP1: filtered temporal_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.



(a) SynHist Dataset

(b) GovTrack Dataset

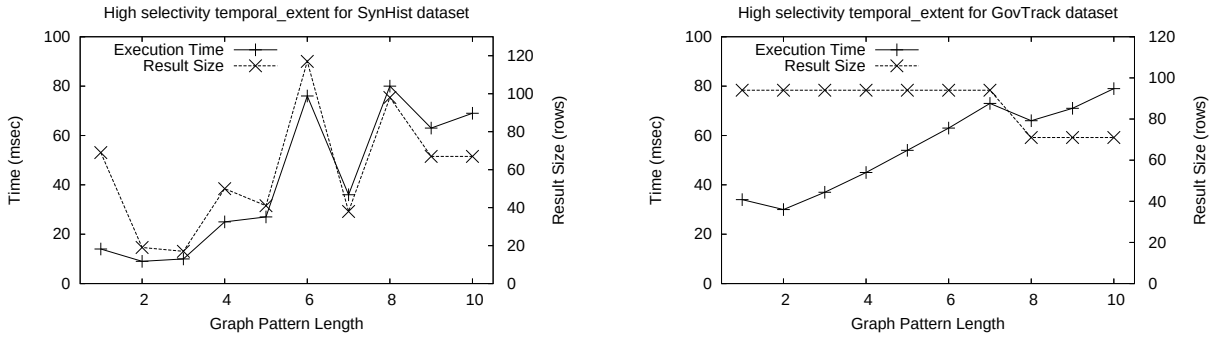
Fig. 7 Experiment GP2: filtered spatial_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

graph pattern size increases when the result set size is taken into account. Again, the DBMS starts with the most selective triple pattern and grows the graph pattern instance from there using an index-based NLJ strategy. The initial selection reduces the search space and is responsible for the good performance that we see. The times reported in this experiment are a bit slower than those in GP3 due to the larger result set sizes.

7.2.3 Scalability of Spatiotemporal Queries

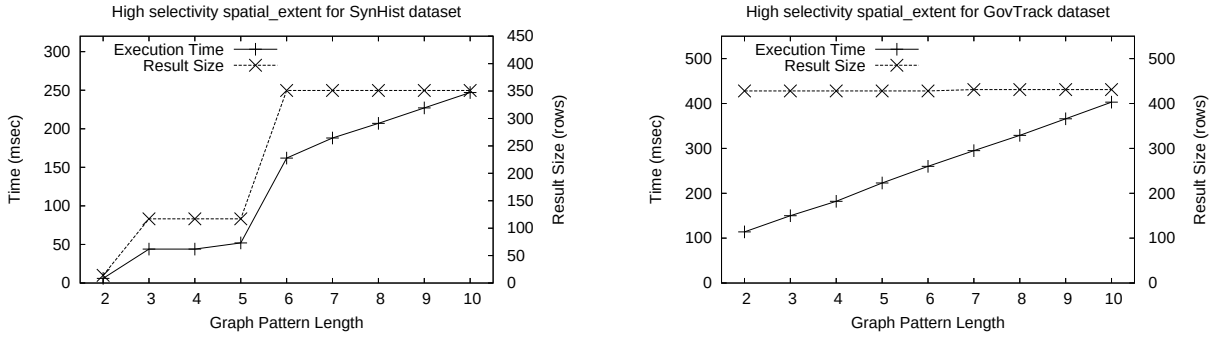
We performed some basic experiments to demonstrate the scalability of spatiotemporal queries that combine a spatial operator and a temporal operator in a single SQL query.

Spatiotemporal Queries w.r.t. Dataset Size: Our first spatiotemporal experiment tested scalability with respect to dataset size. Tables 5 and 6 show the execution times for a query involving both a filtered *temporal_extent* operator and a filtered *spatial_extent* opera-



(a) SynHist Dataset (b) GovTrack Dataset

Fig. 8 Experiment GP3: highly selective basic temporal_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.



(a) SynHist Dataset (b) GovTrack Dataset

Fig. 9 Experiment GP4: highly selective basic spatial_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

tor. Each query used one filtered *spatial_extent* operator invocation and one filtered *temporal_extent* operator invocation. The same unselective graph pattern was used in each operator invocation, and the results of each operator invocation were joined based on equality of variable values (i.e. along the lines of the spatiotemporal query example in Section 6.1). The results show that execution times are significantly slower than queries involving a single operator because the results for each individual function invocation must be retrieved and then joined based on variable correspondences to form the final result. This slowdown occurs for both datasets. However, the queries show good scalability with respect to dataset size. Execution time is near constant as the dataset size increases for the GovTrack dataset, but the execution time grows linearly for the SynHist dataset. The growth in execution time for the SynHist dataset is due to the scalability of queries involving a filtered *temporal_extent* operator on this dataset as discussed previously.

Table 5 Execution time for filtered spatial_extent plus filtered temporal_extent for GovTrack dataset

Query	Operator	Relation	Graph Pattern		Result Size	Execution Time (msec)		
			Num Triples	Num Vars		GT1	GT2	GT3
STG1	ST-Filter	INSIDE INT/DURING	3	4	122	4490	4732	4740
STG2	ST-Filter	ANYINT INT/DURING	5	6	397	4590	4608	4602

Table 6 Execution time for filtered spatial_extent plus filtered temporal_extent for SynHist dataset

Query	Operator	Relation	Graph Pattern		Result	Execution Time (msec)				
			Num Triples	Num Vars	Size	SH1	SH2	SH3	SH4	SH5
STH1	ST-Filter	OVERLAP INT/OVERLAP	3	4	43	1843	1916	2143	2687	3113
STH2	ST-Filter	w/in DIST INT/OVERLAP	5	6	84	2012	2028	2045	2171	2189

Spatiotemporal Queries w.r.t. Graph Pattern Size: Experiment ST1 tested the scalability of a spatiotemporal query with respect to graph pattern complexity. The spatiotemporal queries involved both a *spatial_extent* operator invocation and a *temporal_extent* operator invocation. Within a spatiotemporal query, the same selective graph pattern was used for each operator and the results of the two operator invocations were joined on equality of variable values. Figure 10 shows the execution times for one such spatiotemporal query of each graph pattern size. The results of this experiment show that execution time tends to grow linearly with graph pattern complexity when result set size is taken into account. Execution times are roughly twice as long as a query involving a single operator (i.e. as in experiments GP3 and GP4), as results for both function invocations must be retrieved and then joined.

8 Conclusions

This paper discussed an approach for realizing spatial and temporal query operators for Semantic Web data. Our work was motivated by a lack of support for spatial and temporal relationship analysis in current semantic analytics tools. Spatial and temporal data is critical in many analytical applications and must be effectively utilized for semantic analytics to reach its full potential. In addition, a framework that allows integrated analysis of spatial, temporal and thematic information is needed to realize many visions of the next generation World Wide Web, such as the Event Web [**?**], and, as we discuss in [**?**], the framework presented in this paper can help realize such a vision.

Our approach built upon existing support for storage and querying of RDF data and spatial geometries in Oracle DBMS. A set of experiments using both synthetic and real-world RDF datasets of over 25 million triples showed that our implementation exhibited good scalability for a large populated ontology. Basic *temporal_extent* and *spatial_extent* queries were quite fast in all circumstances. The worst performance was seen with filtered *temporal_extent* queries using low selectivity graph patterns with highly selective temporal predicates. However, the resulting execution times were manageable.

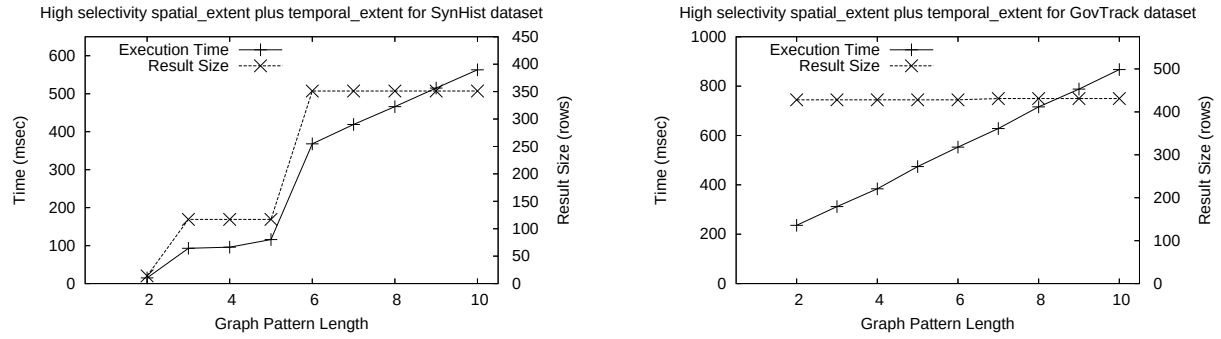
A possible limitation of this work is that Oracle Semantic Data Store does not support incremental maintenance of RDFS rules indexes. Consequently, our indexing scheme inherits this limitation. However, incremental maintenance of a materialized set of inferred triples upon updates of asserted triples is possible (e.g., [73]), and existing algorithms could be extended to incorporate temporal information.

In the future, we plan investigate this incremental maintenance issue and to investigate extensions of the SPARQL query language that support the types of operations discussed in this paper.

Acknowledgements We thank Professor T. K. Prasad for his helpful comments on our formalizations, and we thank Farshad Hakimpour and Prateek Jain for their help. This work is partially funded by NSF-ITRIDM Award #0325464 & #0714441 entitled "SemDIS: Discovering Complex Relationships in the Semantic Web."

References

1. Open geospatial consortium geospatial semantic web interoperability experiment. http://www.opengeospatial.org/projects/initiatives/gswie
2. Oracle database data cartridge developer's guide, 10g release 2. URL http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b14289/toc.htm
3. Oracle spatial resource description framework (rdf) 10g release 2. URL http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b19307/toc.htm
4. Oracle spatial user's guide and reference 10g release 2. URL http://download-



(a) SynHist Dataset

(b) GovTrack Dataset

Fig. 10 Experiment ST1: basic spatial_extent plus temporal_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

east.oracle.com/docs/cd/B19306_01/appdev.102/b14255/toc.htm
5. Semantic web activity. URL http://www.w3.org/2001/sw/
6. United states census year 2000 cartographic boundary files. URL http://www.census.gov/geo/www/cob/bdyfiles.html
7. W3c geospatial incubator group. URL http://www.w3.org/2005/Incubator/geo/
8. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable semantic web data management using vertical partitioning. In: 33rd International Conference on Very Large Data Bases (2007)
9. Abdelmonty, A.I., Smart, P.D., Jones, C.B., Fu, G., Finch, D.: A critical evaluation of ontology languages for geographic information retrieval on the internet. Journal of Visual Languages and Computing **16**(4), 331–358 (2005)
10. Agarwal, P.: Ontological considerations in giscience. International Journal of Geographical Information Science **19**(5), 501–536 (2005)
11. Aleman-Meza, B., Nagarajan, M., Ramakrishnan, C., Ding, L., Kolari, P., Sheth, A., Arpinar, I.B., Joshi, A., Finin, T.: Semantic analytics on social networks: Experiences in addressing the problem of conflict of interest detection. In: 15th International World Wide Web Conference. Edinburgh, Scotland (2006)
12. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D.: On storing voluminous rdf descriptions: The case of web portal catalogs. In: 4th International Workshop on the Web and Databases. Santabarbara, Califonia, USA (2001)
13. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
14. Allen, J.F.: Towards a general theory of action and time. Artificial Intelligence **23**(2), 123–154 (1984)
15. Allen, J.F., Ferguson, G.: Actions and events in interval temporal logic. Journal of Logic and Computation **4**(5), 531–579 (1994)
16. Angles, R., Gutierrez, C.: Querying rdf data from a graph database perspective. In: 2nd European Semantic Web Conference. Heraklion, Greece (2005)
17. Anyanwu, K., Sheth, A.: r-queries: Enabling querying for semantic associations on the semantic web. In: The 12th International World Wide Web Conference. Budapest, Hungary (2003)
18. Aref, W.G., Samet, H.: Extending a dbms with spatial operations. In: 2nd International Symposium on Advances in Spatial Databases. Zurich, Switzerland (1991)
19. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vahrenhold, J., Vitter, J.S.: A unified approach for indexed and non-indexed spatial joins. In: 7th International Conference on Extending Database Technology. Konstanz, Germany (2000)
20. Beckett, D.: The design and implementation of the redland rdf application framework. Computer Networks **39**(5), 577–588 (2002)
21. Beckman, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. In: ACM SIGMOD international conference on Management of data. Atlantic City, New Jersey, USA (1990)
22. Brickley, D., Guha, R.V.: Rdf vocabulary description language 1.0: Rdf schema. w3c recommendation. URL http://www.w3.org/TR/rdf-schema/
23. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: ACM SIGMOD International Conference on Management of Data. ACM Press, Washington, D.C. (1993)
24. Broekstra, J., Kampman, A., Harmelen, F.v.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In: International Semantic Web Conference. Sardinia, Italy (2002)
25. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An efficient sql-based rdf querying scheme. In: 31st International Conference on Very Large Data Bases. Trondheim, Norway (2005)
26. Egenhofer, M.J.: Toward the semantic geospatial web. In: 10th ACM International Symposium on Advances in Geographic Information Systems. McLean, VA (2002)
27. Egenhofer, M.J., Herring, J.R.: Categorizing binary topological relations between regions, lines, and points in geographic databases. Tech. Rep. 94-1, University of Maine, National Center for Geographic Information and Analysis (1994)

28. Fonseca, F.T., Egenhofer, M.J., Agouris, P., Camara, G.: Using ontologies for integrated geographic information systems. Transactions in GIS **6**(3), 231–257 (2002)
29. Gao, D., Jensen, C.S., Snodgrass, R.T., Soo, M.D.: Join operations in temporal databases. The International Journal on Very Large Data Bases **14**(1), 2–29 (2005)
30. Grenon, P., Smith, B.: Snap and span: Towards dynamic spatial ontology. Spatial Cognition and Computation **4**(1), 69–104 (2004)
31. Gunther, O.: Efficient computation of spatial joins. In: 9th International Conference on Data Engineering. IEEE Computer Society, Vienna, Austria (1993)
32. Gutierrez, C., Hurtado, C., Vaisman, A.: Temporal rdf. In: European Conference on the Semantic Web. Heraklion, Crete, Greece (2005)
33. Gutierrez, C., Hurtado, C., Vaisman, A.: Introducing time into rdf. IEEE Transactions on Knowledge and Data Engineering **19**(2), 207–218 (2007)
34. Guting, R.H.: An introduction to spatial database systems. International Journal on Very Large Data Bases **3**(4), 357–399 (1994)
35. Guting, R.H., Bohlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M.: A foundation for representing and querying moving objects. ACM Transactions on Database Systems **25**(1), 1–42 (2000)
36. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD International Conference on Management of Data. Boston, MA, USA (1984)
37. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A comparison of rdf query languages. In: 3rd International Semantic Web Conference. Hiroshima, Japan (2004)
38. Hadjieleftheriou, M., Kollios, G., Tsotras, V.J., Gunopulos, D.: Efficient indexing of spatiotemporal objects. In: 8th International Conference on Extending Database Technology. Prague, Czech Republic (2002)
39. Harris, S., Gibbins, N.: 3store: Efficient bulk rdf storage. In: 1st International Workshop on Practical and Scalable Semantic Systems. Sanibel Island, Florida, USA (2003)
40. Harth, A., Decker, S.: Optimized index structures for querying rdf from the web. In: 3rd Latin American Web Congress (2005)
41. Hayes, P.: Rdf semantics. URL http://www.w3.org/TR/rdf-mt/
42. Hobbs, J., Pan, F.: An ontology of time for the semantic web. ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing **3**(1), 66–85 (2004)
43. Jones, C.B., Abdelmonty, A.I., Finch, D., Fu, G., Vaid, S.: The spirit spatial search engine: Architecture, ontologies, and spatial indexing. In: 3rd International Conference on Geographic Information Science. Adelphi, MD, USA (2004)
44. Kammersell, W., Dean, M.: Conceptual search: Incorporating geospatial data into semantic queries. In: Terra Cognita - Directions to the Geospatial Semantic Web. Athens, GA, USA (2006)
45. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: Rql: a declarative query language for rdf. In: 11th International World Wide Web Conference. Honolulu, Hawaii, USA (2002)
46. Klyne, G., Carroll, J.J.: Resource description framework (rdf): Concepts and abstract syntax. URL http://www.w3.org/TR/rdf-concepts/
47. Kochut, K., Janik, M.: Sparqler: Extended sparql for semantic association discovery. In: 4th European Semantic Web Conference. Innsbruck, Austria (2007)
48. Kolas, D., Hebeler, J., Dean, M.: Geospatial semantic web: Architecture of ontologies. In: 1st International Conference on GeoSpatial Semantics. Mexico City, Mexico (2005)
49. Mokbel, M.F., Ghanem, T.M., Aref, W.G.: Spatio-temporal access methods. IEEE Data Engineering Bulletin **26**(2), 40–49 (2003)
50. Mukherjea, S., Bamba:, B.: Biopatentminer: An information retrieval system for biomedical patents. In: 30th International Conference on Very Large Data Bases, pp. 1066–1077. Toronto, Canada (2004)
51. Ozsoyoglu, G., Snodgrass, R.T.: Temporal and real-time databases: A survey. IEEE Transactions on Knowledge and Data Engineering **7**(4), 513–532 (1995)
52. Pelekis, N., Theodoulidis, B., Kopanakis, I., Theodoridis, Y.: Literature review of spatio-temporal database models. The Knowledge Engineering Review **19**(3), 235 – 274 (2004)
53. Perez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. In: 5th International Semantic Web Conference. Athens, GA, USA (2006)
54. Perry, M.: Tontogen: A synthetic data set generator for semantic web applications. AIS SIGSEMIS Bulletin **2**(2), 46–48 (2005)
55. Perry, M., Hakimpour, F., Sheth, A.: Analyzing theme, space and time: an ontology-based approach. In: 14th ACM International Symposium on Geographic Information Systems. Arlington, VA, USA (2006)
56. Perry, M., Sheth, A.P., Hakimpour, F., Jain, P.: Supporting complex thematic, spatial and temporal queries over semantic web data. In: 2nd International Conference on Geospatial Semantics. Mexico City, MX (2007)
57. Peuquet, D.J.: Making space for time: Issues in space-time data representation. GeoInformatica **5**(1), 11–32 (2001)
58. Price, R., Tryfona, N., Jensen, C.S.: Extending UML for Space- and Time-Dependent Applications, vol. 1, pp. 342–366. Idea Group (2002)
59. Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf, w3c recommendation (2008). URL http://www.w3.org/TR/rdf-sparql-query/
60. Pugliese, A., Udrea, O., Subrahmanian, V.S.: Scaling rdf with time. In: 17th International World Wide Web Conference (2008)
61. Ramakrishnan, C., Milnor, W.H., Perry, M., Sheth, A.P.: Discovering informative connection subgraphs in multi-relational graphs. SIGKDD Explorations **7**(2), 56–63 (2005)
62. Salzberg, B., Tsotras, v.J.: Comparison of access methods for time-evolving data. ACM Computing Surveys **31**(2), 158–221 (1999)
63. Samet, H.: The quadtree and related hierarchical data structures. ACM Computing Surveys **16**(2), 187–260 (1984)
64. Seaborne, A.: Rdql - a query language for rdf (2004). URL http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
65. Sintek, M., Decker, S.: Triple - a query, inference, and transformation language for the semantic web. In: 1st International Semantic Web Conference. Sardinia, Italy (2002)
66. Smart, P.D., Abdelmonty, A.I., El-Geresy, B.A., Jones, C.B.: A framework for combining rules and geo-ontologies. In: 1st International Conference on Web Reasoning and Rule Systems. Innsbruck, Austria (2007)
67. Souzis, A.: Rxpath specification proposal (2004). URL http://rx4rdf.liminalzone.org/RxPathSpec
68. Tanasescu, V., Gugliotta, A., Domingue, J., Villarias, L.G., Davies, R., Rowlatt, M., Richardson, M.: A semantic web gis based emergency management system. In: International Workshop on Semantic Web for eGovernment. Budva, Montenegro (2006)
69. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking database representations of rdf/s stores. In: 5th International Semantic Web Conference. Galway, Ireland (2005)

70. Tryfona, N., Jensen, C.S.: Conceptual data modeling for spatiotemporal applications. GeoInformatica **3**(3), 245–268 (1999)
71. Tryfona, N., Jensen, C.S.: Using abstractions for spatio-temporal conceptual modeling. In: ACM Symposium on Applied Computing. Como, Italy (2000)
72. Udrea, O., Pugliese, A., Subrahmanian, V.S.: Grin: A graph based rdf index. In: 22nd AAAI Conference on Artificial Intelligence, pp. 1465–1470 (2007)
73. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. Journal on Data Semantics **2**, 1–34 (2005)
74. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient rdf storage and retrieval in jena2. In: VLDB Workshop on Semantic Web and Databases. Berlin, Germany (2003)
75. Worboys, M.F., Hornsby, K.: From objects to events: Gem, the geospatial event model. In: 3rd International Conference on Geographic Information Systems. Adelphi, MD (2004)
76. Yuan, M.: Wildfire conceptual modeling for building gis space-time models. In: GIS/LIS. Pheonix, AZ (1994)
77. Yuan, M.: Modeling semantical, temporal and spatial information in geographic information systems. In: M. Craglia, H. Couclelis (eds.) Geographic Information Research: Bridging the Atlantic, pp. 334–347. Taylor and Francis, London (1996)