View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Huddersfield Repository

University of Huddersfield Repository

Stravoskoufos, Konstantinos, Petrakis, Euripides G.M., Mainas, Nikolaos, Batsakis, Sotiris and Samoladas, Vasilis

SOWL QL: Querying Spatio - Temporal Ontologies in OWL

Original Citation

Stravoskoufos, Konstantinos, Petrakis, Euripides G.M., Mainas, Nikolaos, Batsakis, Sotiris and Samoladas, Vasilis (2016) SOWL QL: Querying Spatio - Temporal Ontologies in OWL. Journal on Data Semantics. pp. 1-21. ISSN 1861-2040

This version is available at http://eprints.hud.ac.uk/28856/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

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

SOWL QL: Querying Spatio - Temporal Ontologies in OWL

Konstantinos Stravoskoufos · Euripides G.M. Petrakis · Nikolaos Mainas · Sotirios Batsakis · Vasilis Samoladas

Received: date / Accepted: date

Abstract We introduce SOWL QL, a query language for spatio-temporal information in ontologies. Building-upon SOWL (Spatio-Temporal OWL), an ontology for handling spatio-temporal information in OWL, SOWL QL supports querying over qualitative spatio-temporal information (expressed using natural language expressions such as "before", "after", "north of", "south of") rather than merely quantitative information (exact dates, times, locations). SOWL QL extends SPARQL with a powerful set of temporal and spatial operators, including temporal Allen topological, spatial directional and topological operations or combinations of the above. SOWL QL maintains simplicity of expression and also, upward and downward compatibility with SPARQL. Query translation in SOWL QL yields SPARQL queries implying that, querying spatio-temporal ontologies using SPARQL is still feasible but suffers from several drawbacks the most important of them being that, queries in SPARQL become particularly complicated and users must be familiar with the underlying spatio-temporal representation (the "N-ary relations" or the "4D-fluents" approach in this work). Finally, querying in SOWL QL is supported by the SOWL reasoner which is not part of the standard SPARQL translation. The run-time performance of SOWL QL has been assessed experimentally in a real data setting. A critical analysis of its performance is also presented.

Konstantinos Stravoskoufos · Euripides G.M. Petrakis · Nikolaos Mainas · Sotirios Batsakis · Vasilis Samoladas
School of Electronic and Computer Engineering,
Technical University of Crete (TUC), Chania, Greece
E-mail: {kgstravo,petrakis,nmainas,batsakis}@intelligence.tuc.gr, vsam@softnet.tuc.gr *Present address:* of S. Batsakis is Department of Informatics, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfiled, U.K.,
E-mail: S.Batsakis@hud.ac.uk

Keywords Query Language · Spatio-Temporal Ontology

1 Introduction

The rapid growth of the World Wide Web (WWW) has generated the need for intelligent tools and mechanisms, which automatically handle tasks that are typically handled manually by users. For example, buying a product requires careful selection among different products that satisfy user needs, at the best available price. In recent years, there is an increasing need for Web services that accomplish these tasks automatically without user intervention. These services must be capable of understanding the meaning of Web pages and reason over their content in a way similar to humans. Semantic Web is a solution to this need. In the Semantic Web, formal definitions of concepts and of their properties form ontologies, which are defined using the OWL language. Ontologies comprise of definitions of concepts and of their properties by means of binary relations (i.e., between two concepts, or a concept and a numerical domain). Query languages such as SPARQL, are typically used for querying information in ontologies.

Handling both static and dynamic information in the Semantic Web is an important problem to deal with. Dynamic ontologies are not only suitable for describing static scenes with static objects (e.g., objects in photographs) but also enable representation of events with objects and properties that change in time and space (e.g., moving objects in a video). However, the syntactic restriction of OWL to binary relations complicates representation of temporal relations. A property holding for a time instant or interval is in fact a ternary relation (involving a temporal instant or interval in ad-

dition to the object and the subject) which cannot be written as a single OWL statement. A standard fix to this problem is to map a temporal relation to a set of binary ones with new classes which are introduced by the temporal model [41].

Representation of dynamic features calls for mechanisms that allow for uniform representation of the notions of time (and of properties varying in time) within a single ontology. Existing methods for achieving this goal include, among others, temporal description logics [5], concrete domains [35], property labeling [25], versioning [32], named graphs [62], reification [41] and the 4D-fluents (perdurantist) approach [66]. Some methods (e.g., [5], [35]) require extending OWL with additional constructs and are not compliant with existing standards of the Semantic Web for crafting, reasoning and querying ontologies.

First, we discuss SOWL [10, 12] (Spatio-temporal OWL), an approach for handling spatio-temporal information in OWL. Complying with existing Semantic Web standards and specifications (e.g., OWL 2.0, SWRL) is a basic design decision in SOWL. The representation of temporal information resorts to 4D-fluents or, equivalently, to N-ary relations. This representation is then enhanced with spatial topological or directional information. The two representations (i.e., 4D-fluents, N-ary relations) are practically equivalent, with 4D-fluents being more suitable for the representation of symmetric, inverse and transitive relations while, the N-ary approach requires fewer additional objects.

SOWL handles also qualitative relations (expressed using natural language terms such as "before", "after" or "below", "above") in addition to quantitative ones. Reasoning in SOWL is implemented in SWRL and is capable of inferring new spatial and temporal relations or detecting inconsistencies. Reasoning implements "path consistency" on tractable sets of spatial and temporal relations [9].

The focus of the present work is on querying spatio-temporal information in OWL ontologies. We propose SOWL Query Language (SOWL QL), a high-level query language for spatio-temporal ontologies in OWL. SOWL QL is independent from the underlying SOWL representation so that, the user need not be familiar with the peculiarities of the spatio-temporal model applied (i.e., 4D-fluents or N-ary relations).

Recent work on query languages for temporal ontologies include st-SPARQL [33], SPARQL-ST [44], τ-SPARQL [62] and TOQL [8]. Similar to SOWL QL, they all rely on the idea of extending SPARQL with spatial and temporal operators. Compared to the work referred to above, SOWL QL has the following advantages: (a) SOWL QL syntax is independent of the underlying model of spatio-temporal representation, (b) the user need not be familiar with the peculiarities of the underlying model allowing for handling dynamic (spatio-temporal) ontologies almost like static ones, (c) supports an exhaustive set of spatial and temporal operators including (temporal) Allen, distance, topological and directional operators, (d) SOWL QL supports querying of qualitative expressions (defined using natural language terms such as "before", "after","East", "West") in addition to quantitative spatio-temporal expressions (a feature which is also supported by SPARQL-ST) and, (e) supports reasoning during the querying process (i.e., queries specifying exact temporal or spatial values call for reasoning support).

GeoSPARQL [42] is the current standard of the Open Geospatial Consortium (OGC) [1] for representing and querying spatial data on the Semantic Web. Similarly to the works referred to above, it extends SPARQL with a complete set of spatial operators and functions for hiding the characteristics of the underlying spatial model (for facilitating query formulation). This feature is similar to SOWL QL which however has this functionality embedded within its syntax. GeoSPARQL is missing a temporal model and, consequently, can not answer temporal or spatio - temporal queries as SOWL QL does.

SOWL QL queries are translated to equivalent SPARQL queries. In fact, SOWL QL and SPARQL are equivalent, meaning that any SOWL QL query can also be expressed in SPARQL and vice versa (although SPARQL spatio-temporal queries are particularly involved). SOWL QL is fully implemented and supported by a Graphical User Interface (GUI). A working version of SOWL QL is available on the Web[2].

Related work in the field of knowledge representation including approaches for dealing with information evolving in time and space are discussed in Sec. 2. The SOWL ontology model and issues related to querying spatio-temporal information in the Semantic Web are discussed in Sec. 3 and Sec. 4 respectively. SOWL QL syntax and semantics as well as its implementation are presented in Sec. 5 and Sec. 6 respectively, followed by conclusions and issues for future research in Sec. 8.

2 Background

The rapid growth of the Semantic Web in recent years has generated additional interest in methods and tools for dealing with time and with concepts (e.g., events) evolving in time and space. Related work for the Semantic Web lays its foundations in earlier work for knowl-

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

[2] http://www.intelligence.tuc.gr/prototypes.php

edge representation such as Description Logics [6] providing a logical formalism for ontologies.

Time can be regarded as discrete or continuous, linear or cyclical, absolute or relative, qualitative or quantitative. In this work, time is described using quantitative or qualitative terms using temporal instances and intervals. Choosing between an interval or a point-based representation is also important [63] with the later being the one adopted in this work. Point-based representations assume a linear ordering of time points with three possible relations the $<$, $>$, $=$ referred to as *before, after* and *equals* respectively. Based on these ordering relations, intervals can also be defined as ordered pairs of points s, e with $s < e$, often referred to as *start* and *end* of an interval respectively. A temporal relation between intervals can be one of the 13 pairwise disjoint Allen's relations [1] of Fig. 1. Building upon a point or an interval-based representation, qualitative relations can still be asserted (e.g., when exact values of temporal instants or intervals are unknown) by means of their relation (e.g., "before", "after") to other temporal instants or intervals.

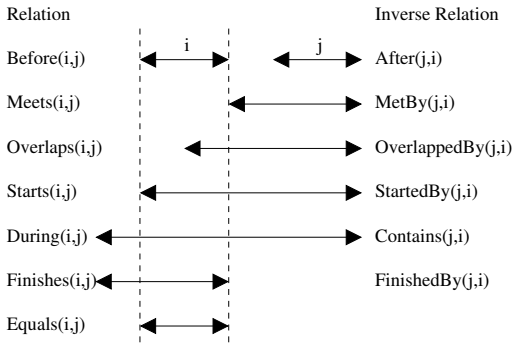

Fig. 1 Allen's Temporal Relations

Space is also an important aspect of knowledge representation. Space can be regarded as two-dimensional (2D) or three-dimensional (3D). Formal spatial, and spatio-temporal representations have been studied extensively in the Database [27, 55] and recently, in the Semantic Web literature [15]. Spatial entities (e.g., objects, regions) in classic database systems are typically represented using points, lines (polygonal lines) or Minimum Bounding Rectangles (MBRs) enclosing objects or regions and their relationships [7].

Spatial relations can be topological, directional or distance relations [9,53]. Topological relations represent the relative position between sets of points or regions on the plane. This research has been motivated mainly by the need for a better and formal understanding of spatial relations in the context of Geographic Information Systems (GIS). Egenhofer [23] suggests a set of nine topological operators between spatial sets of points or regions in which relations are defined in terms of intersections of their boundaries. Along the same lines, Clementini [19] suggests an extension of the above formalism for relations between points, lines and regions by taking also the dimension of their intersections into account, resulting into a set of five general mutually disjoint topological relations namely, *touch, in, cross, overlap, disjoint*. This formalism is most suitable for querying in GIS systems (rather than the Web) where the exact locations of points, lines and regions (polygonal lines) are defined (in meta-data) or can be extracted from maps.

Following the trend of the Web and Semantic Web, where objects and their relations are mostly described qualitatively using natural language expressions (rather than using their coordinates as in st-SPARQL [33]), the most widespread formalism for representing topological relations is the so called Region Connection Calculus (RCC) formalism [49]. RCC has been adopted by the Open Geospatial Consortium (OGC) for querying geospatial data on the Semantic Web [28] and has been implemented in GeoSPARQL [42]. Fig. 2 illustrates the eight RCC8 topological relations namely *(DC, EC, EQ, NTPP, NTPPi, TTP, TPPi, PO)*.

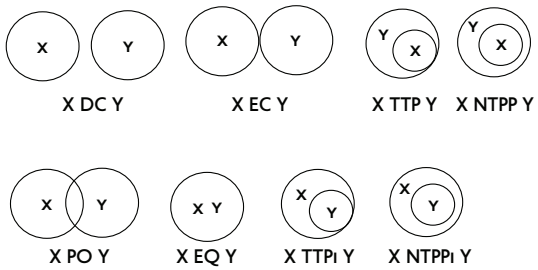

Fig. 2 RCC8 topologic relations.

Representations of topological relations may be complemented with information about direction. Directional relations are defined using cone-shaped areas [38]. Fig. 3 illustrates eight directional relations (nine with the addition of the *Same Location* relation) namely, *North (N), North East(NE), East (E), South East (SE), South (S), South West (SW), West (W)* and *North West (NW)* forming the so called Cone Shaped Directional representation of 9 relations (CSD9). Finally, distance relations are expressed quantitatively (e.g., 3Km away from city A) and can be used in conjunction with topological or directional relations.

Inferring implied relations and detecting inconsistencies is handled by a reasoning mechanism. Relations holding between spatial or temporal entities (e.g., inter-

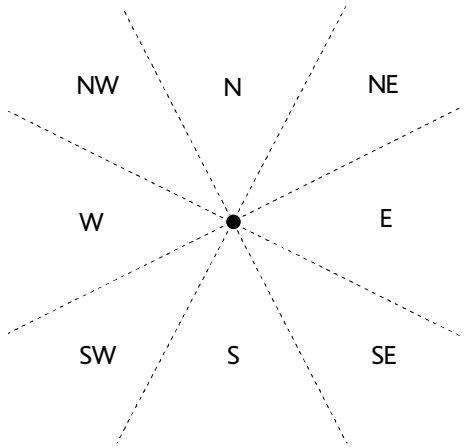

Fig. 3 Directional relations.

vals, points) restrict assertions between other temporal and spatial entities in the knowledge base. Reasoning is realized by means of *compositions* of existing relations: When a relation (or a set of possible relations) R_1 holds between entities A and B and a relation (or a set of relations) R_2 holds between entities B and C then, the composition of relations R_1, R_2 (denoted as $R_1 \circ R_2$) is the set (which may contain only one relation) R_3 of relations holding between A and C. Typically, the set of possible compositions between pairs of relations is defined in a *composition table* for qualitative CSD9 [20], RCC8 [53] or temporal relations [1] respectively.

Reasoning over spatial or temporal relations is known to be an NP-hard problem. Identifying tractable cases of this problem has been in the center of many research efforts over the last few years. Tractable subsets for point algebra have been identified in [63,64,65]. Tractable sets of Allen interval algebra have been identified in [39] and [34]. Tractability of RCC8 topological subsets is analyzed in [50] while, cone-shaped directional relations are examined in [51]. Combining points and intervals for temporal reasoning is analyzed in [31] while, combined reasoning over intervals and their durations is discussed in [47]. Recent results for topological and temporal reasoning are discussed in [14].

2.1 Concepts Evolving in Time

OWL-Time [30] is an ontology providing definitions of time instants, intervals and of their relations as well as, definitions of concepts such as days, weeks, months, years, dates, time zones, duration and measuring units. OWL-Time does not show how dynamic objects evolve in time. This calls for mechanisms such as *Temporal Description logics (TDLs)* [5], *Concrete domains* [35], *Reification*, *labeling of properties* [16,26], *Versioning* [32], *named graphs* [62] and *4D-fluents* [66].

Temporal Description Logics (TDLs) [5, 36] extend standard description logics (DLs) that form the basis of semantic Web standards with additional constructs such as "always in the past", "sometime in the future". TDLs offer additional expressive capabilities over non temporal DLs and retain decidability (with an appropriate selection of allowable constructs) but they require extending OWL syntax and semantics with the additional temporal constructs. *Concrete Domains* [35] require extending OWL with new data types and operators based on an underlying domain (such as decimal numbers). TOWL [24] suggests combining 4D-fluents with concrete domains but does not support qualitative relations or reasoning (as SOWL does) and is not compatible with existing OWL editing, querying and reasoning tools (e.g., Protege, Pellet, SPARQL).

Versioning [32] suggests that the ontology has different versions (one per instance of time). When a change takes place, a new version is created. Changes even on a single attribute require that a new version of the ontology be created. It is not clear how the relation between evolving classes can be represented. *Named Graphs* [62] represent the temporal context of a property by inclusion of a triple representing the property in a named graph (i.e., a subgraph of the RDF graph specified by a distinct name). Named graphs are not part of the OWL specification[3] (i.e., there are not OWL constructs translated into named graphs) and they are not supported by OWL reasoners.

Reification is a general purpose technique for representing n-ary relations using a language such as OWL that permits only binary relations. Specifically, an n-ary relation is represented as a new object that has all the arguments of the n-ary relation as objects of properties. For example if the relation R holds between objects A and B at time t, this is expressed as *R(A,B,t)*. Furthermore, in OWL, using reification this is expressed as a new object with R, A, B and t being objects of properties. Fig. 4 illustrates the relation *WorksFor(Employee, Company, TimeInterval)* representing the fact that an employee works for a company during a time interval. Using reification, the extra class "ReifiedRelation" is created having all the attributes of the relation as objects of properties. Reification suffers mainly from two disadvantages: (a) a new object is created whenever a temporal relation has to be represented (this problem is common to all approaches based on OWL) and (b) offers limited OWL reasoning capabilities [66] since relation R is represented as the object of a property, thus OWL semantics over properties (e.g., inverse properties) are no longer applicable (i.e., the properties of a relation are no longer associated directly with the rela-

[3] http://www.w3.org/TR/owl2-syntax/

tion itself). Examples of temporal representation based on reification (the reified temporal relations are named *Events* or *Actions*) are presented in [18, 56]. In [61] temporal representation is combined with application specific SWRL rules for representing clinical narratives.

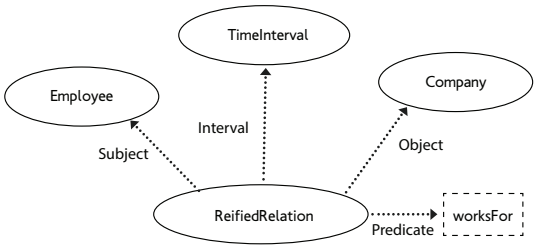

Fig. 4 Example of Reification.

N-ary relations [41] suggest representing an N-ary relation by two properties each related with a new object (i.e., the representation introduces one additional object for every temporal relation). The domains and ranges of properties have to be adjusted taking into account the class of intermediate objects representing the relation (for example the *worksfor* relation in no longer a relation having as object an individual of class *Company* and subject of class *Employee*, as they are now related to object *TemporalEmployment*). This is illustrated in Fig. 5.

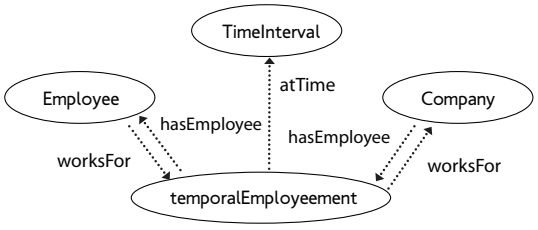

Fig. 5 Example of N-ary Relations.

In the *4D-fluents* (perdurantist) approach [66], concepts in time are represented as four-dimensional objects with the fourth dimension being the time (*time-slices*). Time instances and time intervals are represented as instances of a *TimeInterval* class, which in turn is related with concepts varying in time as shown in Fig. 6. Changes occur on the properties of the temporal part of the ontology keeping the entities of the static part unchanged. The 4D-fluents approach (similarly to N-ary relations) suffers from data redundancy since it introduces two additional objects for each temporal relation (instead of one in the case of N-ary relations).

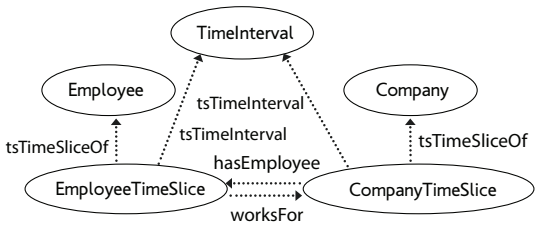

Fig. 6 Example of 4D-fluents.

3 SOWL

SOWL [10] is an ontology for handling spatio - temporal information in OWL. SOWL enables representation of static as well as of dynamic information based on the 4D-fluents [66] (or, equivalently, on the N-ary [41]) approach. Both RCC8 topological and CSD9 relations are integrated in SOWL. Representing both qualitative temporal and spatial information is a distinctive feature of SOWL. Selecting between the two representations is a design decision. For SOWL QL, the instant-based (or point-based) approach for temporal and spatial information representation is adopted: intervals or regions are represented using their end-points.

Reasoning in SOWL is realized by introducing a set of SWRL[4] rules operating on spatial (topological or directional) relations, and separately, by a set SWRL rules for asserting inferred temporal relations. Reasoners that support DL-safe rules (i.e., rules that apply only on named individuals in the knowledge base) such as Pellet [57] can be used for inference and consistency checking over spatio-temporal relations. Checking for restrictions holding on time dependent (fluent) properties requires particular attention: if a fluent property holds between two objects (classes), then these objects are indirectly associated through one or more artificial objects (e.g., *TimeSlice* object in a 4D-fluents representation). For example, for the *worksfor* property in Figure 6, the domain of the property is no longer class *Employee* but *timeslice of Employee*. Checking for property restrictions would require adjusting the domain and range of this property from the artificial to the actual objects. SOWL introduces a rule-based solution to this problem using SWRL [11]. In addition to reasoning applying on temporal and spatial relations, Pellet applies also on the ontology schema for inferring additional facts using OWL semantics (e.g., facts due to symmetric relationships and class-subclass relationships).

Fig. 7 summarizes all types of spatial relations within a single ontology schema. Omitting one or more types of spatial relations is a design decision.

[4] http://www.w3.org/Submission/SWRL/

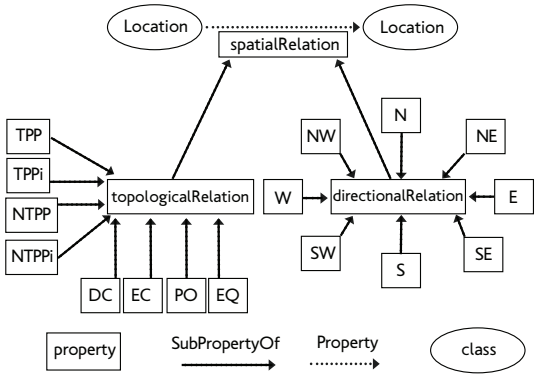

Fig. 7 Ontology schema with spatial relations.

Extending the schema with more relation types is also feasible by instantiating a new class of spatial relations (e.g., the topological relations by [19]) to the *SpatialRelation* class of Fig. 7. Each *spatialRelation* connects two locations and has two sub-properties namely *topologicalRelation* and *directionalRelation*. Any spatial relations formalism can be used to express topological (e.g., [19]), directional or other (e.g., projection-based [58]) relations. Fig. 8 illustrates an extension to the SOWL ontology for incorporating the topological relations by [19].

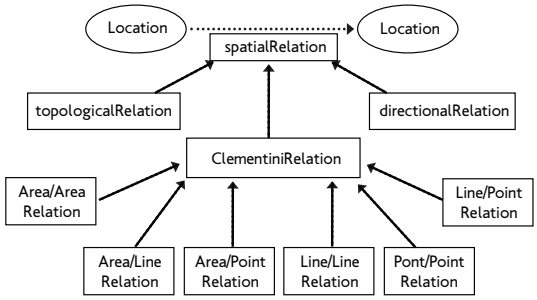

Fig. 8 Ontology schema with a new set of topological relations.

Fig. 9 illustrates a general ontology representation model for spatial information. Class *Location* has attribute *name* (of type string). Also a *Location* object can be optionally connected with a *Geometry* class with sub-classes: *Point, Line, Polyline* and *Minimum Bounding Rectangle (MBR)*. This adds additional low-level (i.e., point, line, polygonal line) representation support to the SOWL model for extracting spatial relations from their raw (pixel) representation. An external software module (i.e., it is not part of the SOWL model) is applied for this task [29]. Then, the symbolic names of spatial relations of Fig. 7 are instantiated to the ontology.

Class *Point* has two (or three in a three-dimensional representation) numerical attributes, namely X, Y (also Z in a three-dimensional representation). For example, *Point* will be the *Geometry* of entities such as cities in a large scale map. Class *Line* has *point1* and *point2* as attributes representing the ending points of a line segment. Class *PolyLine* represents a line or the surrounding contour of an object (or region) as a set of consecutive line segments. An object (or region) may also be represented by its *MBR* specified by the four numerical attributes *Xmax, Ymax, Xmin* and *Ymin*. Both representations may co-exist in SOWL (using one of them or both is a design decision).

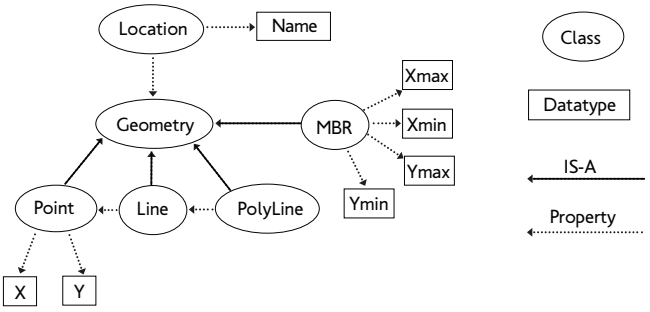

Fig. 9 Ontology representation of spatial objects.

Quantitative distance relations (e.g., "3Km away from city A") are represented as N-ary relations (as illustrated in Fig. 10) by introducing object *Distance* with attributes the two related locations and a numerical attribute representing their distance. Each location object is connected to the *Distance* object with the *locatedAt* property. The *Distance* object is connected to a *TimeInterval* object denoting the duration of the distance relation.

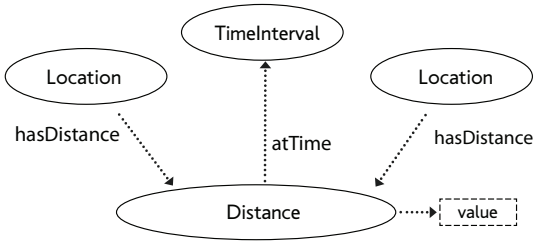

Fig. 10 Ontology representation of distance information.

In the case of a moving object, its location is a property of a *TimeSlice* holding for a specific time interval (Fig. 11). In the case of a static object, its location is a property of the object itself rather than a property of a *TimeSlice*. In the N-ary model, the location of a moving object is a property of the *Event* class. In the case of a static object, its location is a property of the object. As in 4D-fluents, the object can have temporal properties represented by *Event* objects.

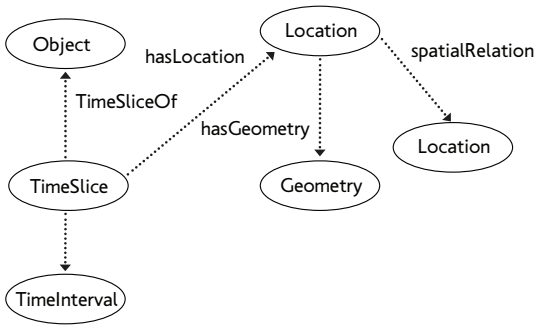

Fig. 11 Ontology representation of moving objects.

Fig. 12 illustrates the dynamic ontology schema representing the scenario "C1 company was located in London from May 2006 to May 2010, since then it is located in Berlin, a city east of London". In this example, it is not known whether the company is still located in Berlin. Only the first temporal interval is defined. The second interval and both locations are unknown and only their qualitative relations are shown. The example illustrates the applicability of the model in the case of missing or inaccurate information (as it is usually the case with natural language descriptions). In these cases, models based on quantitative information only, are insufficient.

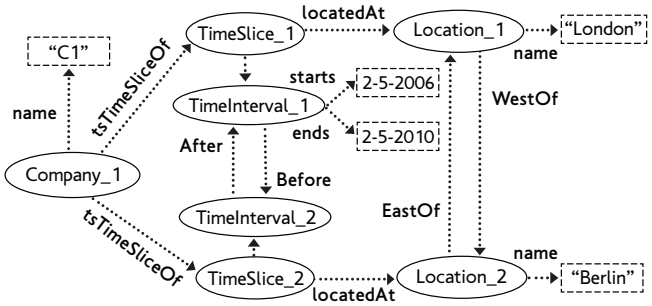

Fig. 12 SOWL example.

4 Querying Spatio - Temporal Information in the Semantic Web

Desirable features of temporal query languages include, upward compatibility (e.g., conventional queries can still be expressed), point and interval-based views of data, and simplicity of expression.

SPARQL [46] is the W3C recommendation query language for RDF. The basic evaluation mechanism of SPARQL queries is based on graph matching. The query criteria are given in the form of RDF triples possibly with variables in the place of the subject, object, or predicate of a triple, referred to as *basic graph patterns*.

Querying spatio - temporal information requires using constructs specific to the underlying temporal model (e.g., *tsTimeSliceOf*, *tsTimeInterval* in the case of 4D-fluents). This leads to complicated expressions and the user has to be familiar with the representation model. This is not required in SOWL QL. The following SPARQL query returns all employees that work for a specific company at any times.

```
SELECT ?employee
WHERE {
 {
?timeSlice_0 ex:hasEmployee ex:Company1.
?timeSlice_0 4dfluents:tsTimeInterval ?interval_0.
?timeSlice_0 ex:hasEmployee ?timeSlice_1.
?timeSlice_1 4dfluents:tsTimeSliceOf ?employee.
?timeSlice_1 4dfluents:tsTimeInterval ?interval_0
 } UNION {
 ex:Company1 ex:hasEmployee ?employee.
  }
}
```

The same query in SOWL QL can be written without using model specific features as

```
SELECT ?employee
WHERE {
ex:Company1 ex:hasEmployee ?employee.
}
```

stSPARQL [33] extends SPARQL with temporal and spatial operators. The underlying temporal representation, referred to as stRDF, is based on Temporal RDF [26]: the RDF statement is extended (by labeling properties) from a triple to a quad where the fourth element is the valid time. However, qualitative temporal information cannot be expressed in stRDF. As a consequence stRDF and stSPARQL do not have any reasoning support. Building-upon ideas from constraint databases, stRDF represents temporal and spatial objects as quantifier-free formulas in a first-order logic of linear constraints using stRDF syntax implying that the user must use stRDF syntax in queries. stSPARQL extends SPARQL with functions that take temporal and spatial terms as arguments and can be used in the SELECT, FILTER, and HAVING clause of a SPARQL query. A spatial term is a spatial literal (i.e., a literal with data type *strdf:geometry)*, a query variable that can be bound to a spatial literal, the result of a set operation on spatio-temporal literals (e.g., union), or the result of a geometric operation on spatial terms (e.g., buffer). stSPARQL uses vectors of points to represent the different geometries. stSPARQL is intended to handle the physical representation of objects (i.e., points or polygonal lines in applications such as cartography and geography) rather than their representation in natural

language as it is the case on the Web (as SOWL QL does).

The following stSPARQL query (example 3 in [33]) retrieves observations recorded at time 11 and the rural area they refer to. Query expressions involve model specific triples (e.g., *hasGeometry*) implying that the user has to be familiar with details of the underlying representation model. This is not the case with the equivalent SOWL QL query (below the stSPARQL query). The query is more concise hiding model specific features from the query expression.

stSPARQL

```
SELECT ?V ?RA
WHERE {
?OBS rdf:type om:Observation.
?LOC rdf:type om:Location.
?R rdf:type om:ResultData.
?RA rdf:type ex:RuralArea.
?OBS om:observationLocation ?LOC.
?OBS om:result ?R.
?R om:value ?V ?T.
?LOC strdf:hasGeometry ?OBSLOC.
?RA strdf:hasGeometry ?RAGEO.
filter(?T contains (t = 11) &&
?RAGEO contains ?OBSLOC)
}
```

SOWL QL

```
SELECT ?V ?RA
WHERE {
?OBS rdf:type om:Observation.
?R rdf:type om:ResultData.
?RA rdf:type ex:RuralArea.
?OBS om:result ?R.
?R om:value ?V.
?RA sowl:NTTPi ?OBS AT(11)
}
```

SPARQL-ST [44], similarly to stSPARQL, applies Temporal RDF and supports querying over spatio - temporal RDF graphs (i.e., temporal RDF graphs that contain spatial objects) using qualitative and quantitative temporal expressions, in conjunction with spatial expressions on the point-set spatial relations by Egenhofer [23]. RDF reification is applied to associate time intervals with RDF statements. Similarly to SOWL-QL, the OWL-Time ontology is integrated into the ontology to model concepts of time. The modeling of spatial properties is based on GeoRSS GML[5]. Two metric operators are introduced for measuring distance and duration respectively. Although syntactically, the language is an extension of SPARQL, aiming at querying RDF graphs stored in triple stores, the ontology is stored in a relational database (SPARQL-ST queries are translated to SQL prior to execution). The prototype implementation of SPARQL-ST is built on top of a commercial DBMS. This, leads to numerous limitations in expressibility (as it is much easier to model complex data in RDF than in SQL), flexibility (i.e., querying over multiple different RDF graphs over HTTP), and reasoning (although SPARQL-ST is not supported by a reasoner) by not taking advantage of OWL semantics as SOWL-QL does.

τ-SPARQL [62] extends the syntax of SPARQL with syntax expressions for addressing temporal information. Building-upon the so called "temporal RDF" expressibility (based on Named Graphs [62]) data model considers time as an additional dimension in data preserving the semantics of time. The language supports two major usage formats referred to as "time point queries" and "temporal queries". Time point queries aim at retrieving information valid at a specified point in time. Temporal queries allow for wild-card intervals and time points. These wild-cards can be used to bind a variable to the validity period of a triple or to express temporal relationships between intervals. τ-SPARQL allows one form of temporal wild-cards $[?s, ?e]$ which binds the literal start and end values.

GeoSPARQL [42] is the solution of the Open Geospatial Consortium (OGC) [6] for representing and querying geospatial information on the Semantic Web. It defines a vocabulary for representing geospatial data in RDF and adds a syntax extension to SPARQL for querying spatial information. Fig. 13 illustrates GeoSPARQL spatial ontology model. The spatial model consists of the *geo:Feature* and *geo:Geometry* classes, both being sub-classes of the *geo:SpatialObject* class.

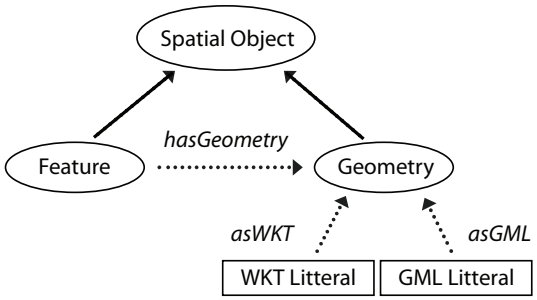

Fig. 13 GeoSPARQL ontology model.

A *Feature* is an object with a spatial location (e.g., a city, a park, a mountain etc.) while *Geometry* is a representation of a spatial location. Geometries and Features are linked by the *geo:hasGeometry* object property which denotes that a Feature's location is represented by a specific Geometry. The *geo:asWKT* and *geo:asGML* properties link a Geometry object with a

[5] http://www.georss.org/gml

[6] http://www.opengeospatial.org/

WKT or GML literal respectively. WKT and GML literals are specifically formatted strings for representing geometries allowing indexing to speed up the query process. The specification supports all RCC-8, Simple Features [13] and Egenhofer [23] topological relations. GeoSparql contains a set of spatial functions for hiding the underlying model from the end-user. A query rewrite component defines rules for translating these functions into SPARQL queries. GeoSPARQL is lacking a temporal model and therefore is appropriate only for static entities (e.g., entities on a map). It is not supported by a spatial reasoner as SOWL QL does.

SOWL QL supports all features referred to above and, in addition, provides a wide arsenal of new temporal and spatial operators. All SOWL QL operators can address either qualitative or quantitative information rather than merely quantitative information expressed using exact values (i.e., dates, times, locations). SOWL QL is also supported by a reasoner, a feature that is not provided by τ-SPARQL, stSPARQL, SPARQL-ST or GeoSPARQL. Unlike τ-SPARQL, stSPARQL or SPARQL-ST, SOWL QL's syntax is model independent.

The approach by Bykau, Mylopoulos, Rizzolo and Velegrakis [17] spans a different research direction. They proposed a model for capturing and querying concept evolution. Concept evolution deals with mereological and causal relationships between concepts rather than merely with concepts whose properties evolve in time (as SOWL QL and all works referred to above). They extend Temporal RDF [26] with additional constructs to model mereological (part-of) and causal relationships between concepts (concept merges, splits and other forms of evolution). Their model enables queries on the history of concepts that cannot be expressed in other query languages (e.g., queries for retrieving concepts that are formed dynamically through merges or splits of other concepts). The query language extends the nested temporal expressions of nSparql [48] with evolution semantics. The problem of query evaluation is modelled as one of finding a Steiner forest and they propose an optimal solution to this problem.

5 SOWL Query Language

SOWL QL introduces a powerful set of spatial and temporal operators which allow users to query spatio-temporal OWL ontologies without dealing with the peculiarities of the underlying representation model. Being an extension of SPARQL, SOWL QL uses the same clauses as SPARQL does and fully supports all SPARQL features. The structure of a SOWL QL query resembles the structure of a SPARQL query with the addition of spatial and temporal operators. SOWL QL operators are discussed in Sec. 5.4.

```
SELECT Variable(s)
WHERE { Triple(s) SPATIAL or TEMPORAL operators
AND Condition(s)
}
```

Similarly to SPARQL, query selection criteria are expressed by RDF triples of the form *Subject - Predicate - Object* referred to as *basic graph patterns*. Basic graph patterns are used inside the body of the *WHERE* clause of a query and this is where SOWL QL operators are applied forming SOWL-QL statements. SOWL QL is capable of querying graph patterns along with their conjunctions and disjunctions. Fig. 14 illustrates the three possible syntax variations of SOWL-QL statements referred to as S_1, S_2 and S_3 respectively. S_1 denotes a simple *Subject Predicate Object* triple (static or dynamic) with no SOWL-QL operators, S_2 denotes a triple (as S_1) followed by a quantitative temporal operator while, S_3 denotes combination of triples related with a qualitative temporal operator.

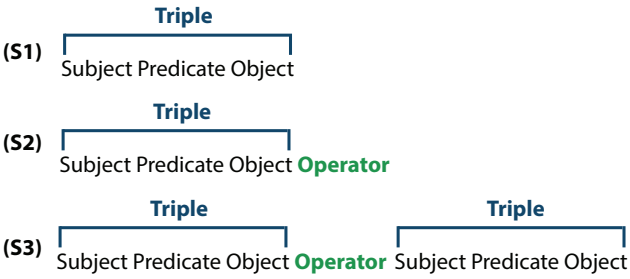

Fig. 14 Syntax variations of SOWL-QL statements.

5.1 S_1 statements

An ontology consists of two parts: the static part (classes, properties, instances) and the dynamic part consisting of additional temporal classes representing time, evolution in time along with properties and instances of the above temporal classes (e.g., *TimeSlice* class and *tsTimeSliceOf* property for the 4D-fluents model or *Event* class and *atTime* property for the N-ary relations model). First, SOWL QL determines if a property (object or data type) in a query is a fluent property (i.e., a property that connects *Timeslice* or *Event* objects) or not (i.e., a property that connects "static" classes or a "static" class with a data type). In the later case, the query is a static one and is handled as an ordinary SPARQL query. In the *subject - predicate - object* triple, a temporal property is referred to as a

"dynamic" predicate. In SOWL QL a triple involving a dynamic predicate resembles a static one.

Query translation determines if the predicate is dynamic or not by checking the ontology for dynamic objects connected to that predicate. If the predicate is a fluent one, the predicate connects two dynamic objects (*Timeslice* or *Event* objects depending on temporal model). If it is static, it connects two static concepts (e.g., a Company and an Employee). SOWL QL is also capable of handling fluent properties connected with static objects in the ontology. In this case, the query retrieves both the dynamic and static objects satisfying the query selection criteria.

The following query retrieves both dynamic and static objects of class "Employee". The dynamic predicate in the *WHERE* clause retrieves "Employees" that work for "Company1".

```
SELECT ?employee
WHERE{
  ex:Company1 ex:hasEmployee ?employee
      }
```

Spatial triples can also be denoted as *subject-predicate-object* triples involving a qualitative spatial operator (as predicate) imposing a constraint on the relation between the subject and the object. The spatial operator connects two static spatial objects (not connected with dynamic temporal *TimeSlice* or *Event* objects) otherwise, the triple is considered to be spatio-temporal. The subject and object of the triple are directly connected with *Location* objects (otherwise a spatial operator cannot be applied). All topological and directional relations are implemented as spatial operators (Sec. 5.4). The spatial query below retrieves all countries in the north (*Nof*) of Greece. The operator addresses static objects (as country locations do not change over time).

```
SELECT ?country
WHERE {
  ?country spatial:Nof ex:Greece
      }
```

The object of the triple can be replaced by *POINT(x,y)*. Here, instead of comparing the *Geometry* objects of two spatial objects we compare the Geometry of the first spatial object with the Geometry of the Point specified. The two arguments of *POINT(x,y)* are float numbers corresponding to the x and y axes in the two-dimensional space. If the point specified does not exist in the knowledge base, the reasoner is invoked (Sec. 5.5). An obvious extension would be to allow using line, polygonal line or MBR in the place of $POINT(x, y)$. This would allow for extending SOWL QL with low-level operators support on lines and regions in the example of stSPARQL, SPARQL-ST and GeoSPARQL.

The following query retrieves countries north of point (x, y). If the point is not in the knowledge base, prior to answering the query the reasoner will be invoked to infer its relations with countries which are instances of the ontology.

```
SELECT ?country
WHERE {
  ?country spatial:Nof POINT(332.2,122.4)
      }
```

Spatial properties can also be dynamic just like all other properties. Being dynamic means that, instead of connecting two *Location* objects, the spatial property connects two dynamic objects (*TimeSlices* or *Events*) which are in turn connected with *Location* objects (Fig. 11). SOWL QL will retrieve all these dynamic objects and combine the results with the static results.

5.2 S_2 statements

Statements of this type are denoted as temporal or spatio-temporal triples followed by a quantitative temporal operator (operators in SOWL QL are distinguished into quantitative and qualitative ones). The predicate of the triple must be dynamic otherwise, no results are returned. Quantitative temporal operators are distinguished into time point operators (if they have one argument) and, into time interval operators (if they have two time points as arguments). In such cases, the spatial dynamic objects (*Timeslices* or *Events* or *Geometry* objects) are retrieved before the restriction is applied. The first query below will find cars (e.g., on a map) parked north of "street1" at a specific time point while, the second query will find cars north of the point with coordinates "(12.5,22.5)" at the time specified.

```
SELECT ?car
WHERE {
  ?car spatial:Nof ex:Street1 AT("2010-02-08T00:00:00")
      }
```

```
SELECT ?car
WHERE {
  ?car spatial:Nof POINT(12.5,22.4) AT("2010-02-08T00:00:00")
      }
```

The following query retrieves companies with employees whose name is "Smith" before time ''2010--02--08T00:00:00''.

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee BEFORE("2010-02-08T00:00:00").
  ?employee ex:employeeName "Smith"
      }
```

Five quantitative time point operators are defined in SOWL QL, namely *AT, STARTSAT, ENDSAT, AFTER, BEFORE* and are discussed in Sec. 5.4. They all specify one time point as argument which is compared against fluent properties (temporal instants or intervals) in the ontology.

Time interval operators take as arguments two time points, denoting the starting and the ending points of a temporal interval. The interval imposes a restriction over temporal intervals in an ontology where the dynamic predicate specified by the triple holds true. SOWL QL implements the following quantitative operators over temporal intervals namely *ALWAYS_AT, SOMETIME_AT* (discussed in Sec. 5.4) as well as all Allen operators with their default meaning (Fig. 1). The following query retrieves companies with employees whose name is "Smith" for temporal intervals overlapping or during the one specified by the operator.

```
SELECT ?company
WHERE {
  ?company ex:hasEmployee ?employee
  SOMETIME_AT("2010-02-08T00:00:00","2012-02-08T00:00:00")
  ?employee ex:EmployeeName "Smith"
    }
```

5.3 S_3 statements

Statements of this type involve a qualitative temporal operator to constraint the relation between two triples. Each triple represents a relation that holds over a specific time interval. S_3 triples can be either temporal or spatio-temporal. All Allen operators can be used as qualitative operators.

The first query below retrieves all employees that were working for "Company1" before the employee named "Smith" was hired by "Company2" while, the second retrieves the employees who have worked for "Company1" before the company has moved its headquarters north of "Company2" headquarters (a temporal and a spatio-temporal statement are combined).

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  BEFORE
  ex:Company2 ex:hasEmployee ?employee2.
  ?employee2 ex:employeeName "Smith"
      }
```

```
SELECT ?employee
WHERE {
?employee  ex:worksFor ex:Company1
BEFORE
ex:Company1Headquarters spatial:Nof ex:Company2Headquarters
}
```

Finally, SOWL QL allows also for conjunctions, disjunctions or negation of statements in the WHERE clause. The following query retrieves the employees who work for "Company1" at time `2010-02-08T00:00:00` or `2011-02-08T00:00:00`.

```
SELECT ?employee
WHERE {
?employee ex:worksFor ex:Company1 AT(2010-02-08T00:00:00)
OR
?employee ex:worksFor ex:Company1 AT(2011-02-08T00:00:00)
}
```

5.4 SOWL QL operators

Temporal operators in SOWL QL are distinguished into quantitative and qualitative. The following quantitative operators specify a single time instant as argument.

AT(timepoint): The fluent holds true during a time interval which contains *timepoint*.

STARTSAT(timepoint): The fluent holds true during a time interval which starts at *timepoint*.

ENDSAT(timepoint): The fluent holds true during a time interval which ends at *timepoint*.

BEFORE(timepoint): The fluent holds true during a time interval which ends before *timepoint*.

AFTER(timepoint): The fluent holds true in a time interval which starts after *timepoint*

The following query retrieves all employees that work for "Company1" during any time interval starting at the time specified.

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  STARTSAT("2010-02-08T00:00:00")
      }
```

The Allen relations of Fig. 1 are also defined as quantitative operators specifying a constraint between two temporal intervals given as arguments.

MEETS(intervalStarts,intervalEnds): Returnstrue if the first time interval meets the second one.

METBY(intervalStarts,intervalEnds): Returnstrue if the second time interval meets the first one.

OVERLAPS(intervalStarts,intervalEnds): Returns true if the first time interval overlaps with the second one.

OVERLAPEDBY(intervalStarts,intervalEnds): Returns true if the second time interval overlaps with the first one.

DURING(intervalStarts,intervalEnds): Returns true if the first time interval is during the second one.

CONTAINS(intervalStarts,intervalEnds): Returns true if the first time interval contains the second one.

STARTS(intervalStarts,intervalEnds): Returns true if the two time intervals start together.

STARTEDBY(intervalStarts,intervalEnds): Returns true if the two time intervals start together.

ENDS(intervalStarts,intervalEnds): Returns true if the two time intervals end together.

ENDEDBY(intervalStarts,intervalEnds): Returns true if the two time intervals end together.

EQUALS(intervalStarts,intervalEnds): Returns true if the first time interval equals the second one.

The following query returns the employees that work for "Company1" for intervals that equal the one specified as argument.

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  EQUALS("2010-02-08T00:00:00","2012-02-08T00:00:00")
      }
```

In addition to Allen, the following two operators are defined:

ALWAYS_AT(intervalStarts,intervalEnds): Returns true for fluents (e.g., events) that occur during the temporal interval specified in the argument (including its starting and ending points). For events occurring in intervals that do not contain all points of the interval in question, the operator returns false. It is defined as a disjunction of the Allen operators *CONTAINS, EQUALS, STARTEDBY, ENDEDBY*. If any of these holds true then *ALWAYS_AT* is also true.

SOMETIME_AT(intervalStarts,intervalEnds): Returns true for events that occur in intervals sharing common points with the interval specified in the argument. *ALWAYS_AT* is therefore a special case of *SOMETIME_AT*. It is defined as a disjunction of the 9 Allen operators, *EQUALS, OVERLAPS, OVERLAPPEDBY, STARTS, STARTEDBY, ENDS, ENDEDBY, CONTAINS, DURING*. If any of these is true then *SOMETIMES_AT* returns true as well.

In all cases of quantitative operators, time points can be replaced by variables in order to retrieve time instants or intervals where the predicate (fluent property) holds true. The following query retrieves the intervals that employee "Johnson" works for Company "C1".

```
SELECT ?x ?y
WHERE {
        ?company ex:hasEmployee ?employee SOMETIME_AT(?x,?y).
        ?company ex:companyName "C1"
        ?employee ex:employeeName "Johnson"
      }
```

All Allen operators can be also used as qualitative operators which can be placed between two temporal or spatio-temporal triple statements. The interval restricting the duration of the predicate of the first triple is defined by the interval that the predicate of the second triple holds true.

The first example below, illustrates the common case (of a quantitative Allen operator) where the interval restricting the duration of the *hasEmployee* relation is restricted by the arguments of the quantitative operator. In the second query, the interval during which the fluent holds true is compared against the interval that the fluent of the second triple holds true (i.e., the interval during which "Company2" has "Employee2").

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  DURING("2010-02-08T00:00:00","2012-02-08T00:00:00")
      }
```

```
SELECT ?employee
WHERE {
  ex:Company1 ex:hasEmployee ?employee
  DURING
  ex:Company2 ex:hasEmployee ex:Employee2
      }
```

Spatial operators in SOWL QL specify a constraint on spatial relations between objects. All 9 directional (i.e., *NoF, SoF, WoF, Eof, NWoF, NEof, SEof, SWof, SameXY*) and all 8 topological relations (*NTTPi, TTPi, DC, EC, EQ, TPP, NTTP, PO*) are defined as operators with their obvious meaning. All operators are discussed in detail in [60]. An extension would be to implement operators for the topological relations by [19]. For points, lines and regions (polygonal lines), these relations are computed by invoking a separate software module [29] and are instantiated to the ontology (Fig. 8).

To support querying on distance information *RANGE* operator is introduced. The operator has two arguments with the first being a numeric value representing the distance and the second being a comparison operator string (i.e., >, <, =, >=, <=) specifying how the distance argument will be used. The following query will retrieve all companies located up to 300Km North of London in 2006.

```
SELECT ?company
WHERE {
?company spatial:nof ex:London RANGE(300,"<=")
SOMETIME_AT("2005-01-01T00:00:00","2006-12-31T00:00:00")
}
```

5.5 Reasoning in SOWL QL

When the ontology is loaded into the memory, the reasoner infers all qualitative relations from the quantitative ones in polynomial time. Temporal and spatial relations are computed by means of data type comparisons and computational geometry algorithms respectively [29]. Moreover, when a quantitative operator is applied in queries, SOWL QL checks whether the values specified as arguments exist in the knowledge base. If not, they are asserted into the knowledge base and the reasoner is invoked so that, the (qualitative) relations of the new values with existing ones are derived and used for answering the query. For example, the following query retrieves employees that worked for a company before the time point specified. If time point ''2010-02-08T00:00:00'' is not in the ontology, prior to answering the query, the SOWL QL reasoner will assert it (temporarily, and remove it after the query is executed) in order to compute its relations with existing fluents.

```
SELECT ?x ?y
WHERE {
        ?x ex1:hasEmployee ?y BEFORE("2010-02-08T00:00:00")
      }
```

The same applies for spatial quantitative operators. The query below, retrieves objects which are north of *POINT(3.4,10.2)*. The point object is asserted into the ontology (if not already there) and the SOWL reasoner will compute all qualitative relations between the new *POINT* and all geometric objects in the ontology.

```
SELECT ?x ?y
WHERE {
        ?x spatial:Nof ?y POINT(3.4 ,10.5)
      }
```

Reasoning in SOWL [10] is realized by means of SWRL rules implementing the allowable compositions over the supported relation (i.e., Allen, RCC-8 or CSD-9) set combined with OWL 2.0 constructs (e.g., for declaring disjoint properties) ensuring path consistency while being sound and complete. Reasoning is embedded within the ontology and relies on a general purpose reasoner such as Pellet[7]. The run-time performance of SOWL declines drastically for large data sets [9].

To deal with the problem of efficiency, reasoning in SOWL QL resorts to CHOROS [37] and CHRONOS [3] reasoners for temporal and spatial information respectively. CHOROS is a dedicated spatial reasoner for directional CSD-9 or RCC-8 relations implemented in Java. Similarly, CHRONOS is a dedicated temporal reasoner for Allen temporal relations. Reasoning is achieved by applying path consistency [59,64] separately for each calculus. Path consistency computes all inferred relations using compositions of existing relations until a fixed point is reached or until an inconsistency is detected. Compositions of basic RCC-8 and CSD-9 relations are defined by Cohn et.al. [21] and by Renz and Mitra [52] respectively. Compositions of basic Allen temporal relations are defined in [2].

The compositions of basic relations may infer disjunctions of such relations because disjunctive entries exist in the composition table (i.e., not all compositions yield a unique relation as a result). Typically, compositions of disjunctions requires computing the composition of disjunctions and storing the result in a structure which may hold up to $2^{13} \times 2^{13}$ entries for Allen relations (i.e., up to 2^{13} disjunctions can appear), or $2^8 \times 2^8$ entries for RC8-8 and $2^9 \times 2^9$ entries for CSD-9 relations . CHOROS and CHOROS implement the following optimization: all disjuctions are computed "on the fly" (i.e., at run-time) by decomposing disjunctive relations into basic ones involving a simple look-up operation in the 13×13 Allen, or 8×8 RCC-8 or 9×9 composition CSD-9 composition table. This results in fewer computations and faster reasoning times.

CHOROS and CHRONOS outperform SOWL in the average and worst cases and scale-up much better than SOWL (i.e., almost linearly in the average case) with the size of the input (i.e., the performance gap between the two reasoner implementations increases with the size of the data set).

5.6 Translation and Equivalence to SPARQL

Syntactically, SOWL QL expressions form a strict superset of SPARQL expressions meaning that every valid SPARQL query is also a valid SOWL-QL query. In the following we show also that every valid SOWL QL query can be translated into an equivalent SPARQL query over the supported representations (i.e., 4D-fluents and N-ary).

SOWL QL uses the same (query) statements with SPARQL with the addition of SOWL-QL operators.

[7] http://clarkparsia.com/pellet/

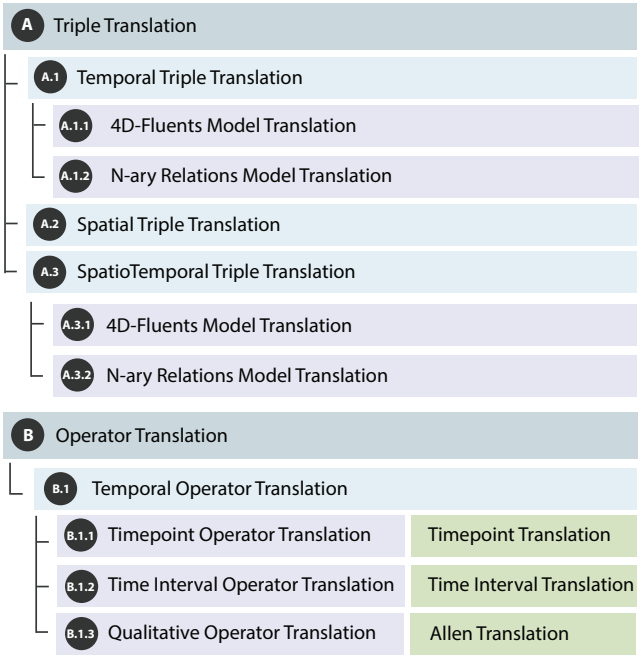

Fig. 15 Translation of SOWL QL to SPARQL.

Three generic query patterns are identified covering the entire range of SOWL QL syntax (Fig. 14). The translation of each SOWL QL statement is realized as a two-stage procedure illustrated in Fig. 15 corresponding to A) the translation of a triple and B) the translation of an operator. As will be shown in the following, triple translation is independent from operator translation and vice-versa.

There are different syntactic variants corresponding to temporal, spatial and spatio-temporal triples (cases A1, A2, A3 in Fig. 15). The translation of temporal and spatio-temporal triples into SPARQL is illustrated in Fig. 16. It results in a conjunction of SPARQL statements and the translation of each spatial expression depends on characteristics of the underlying model (the translation for the 4D-fluents model is different from the translation for the N-ary model). There is only one translation of each spatial expression (its translation is independent of temporal model). In the case of *POINT(x,y)* spatial operator the point is asserted in the ontology as the object of the triple and is translated as above. In the case where two triples are connected with a SOWL QL operator (case S_3) each triple is translated separately and independently from others.

Fig. 16 encompasses also translation of spatial operators. A spatial operator corresponds to the predicate of a spatial or spatio-temporal triple (in cases A.2 and A.3) which is a SPARQL statement already. Translation of temporal operator is more involved and depends on its type (i.e., time point or interval, quantitative or qualitative).

Triple	**Triple Translation** **Syntax:** ?subject ex:predicate ?object
Case A.1: Temporal Triple	**Temporal Model: N-ary** **Case A.1.1** **Spatial Model: none** ?subject ex:predicate ?event. ?event ex:predicate ?object. ?event time:atTime ?interval **Temporal Model: 4D-Fluents** **Case A.1.2** **Spatial Model: -** ?timeSliceX sowl:tsTimeSliceOf ?subject ?timeSliceX sowl:tsTimeInterval ?interval. ?timeSliceX ex:predicate ?timeSliceY. ?timeSliceY sowl:tsTimeSliceOf ?object. ?timeSliceY sowl:tsTimeInterval ?interval
Case A.2: Spatial Triple	**Temporal Model: none** **Spatial Model: SOWL** ?subject spatial:locatedAt ?locationX ?locationX spatial:hasGeometry ?geometryX. ?object spatial:locatedAt ?locationY. ?locationY spatial:hasGeometry ?geometryY. ?locationX ex:predicate ?locationX
Case A.3: SpatioTemporal Triple	**Temporal Model: N-ary** **Case A.3.1** **Spatial Model: SOWL** ?subject spatial:locatedAt ?locationX ?locationX spatial:hasGeometry ?geometryX. ?object spatial:locatedAt ?locationY. ?locationY spatial:hasGeometry ?geometryY. ?locationX ex:predicate ?event. ?event ex:predicate ?locationY. ?event time:atTime ?interval **Temporal Model: 4D-Fluents** **Case A.3.2** **Spatial Model: SOWL** ?subject spatial:locatedAt ?locationX ?locationX spatial:hasGeometry ?geometryX. ?object spatial:locatedAt ?locationY. ?locationY spatial:hasGeometry ?geometryY. ?timeSliceX tsTimeSliceOf ?locationX. ?timeSliceX tsTimeInterval interval. ?timeSliceY sowl:tsTimeSliceOf ?locationY. ?timeSliceY sowl:tsTimeInterval ?interval. ?timeSliceX ex:predicate ?timeSliceY

Fig. 16 Translation of spatio-temporal triples.

Fig. 17 illustrates translation of time point operators (with a single time point value as argument) of case B.1.1, time interval operators (with two time point arguments denoting the starting and ending points of an interval) of case B1.2 and finally, translation of qualitative temporal operators (Allen operators) of case B.1.3. The translation of specific time interval and Allen operators is illustrated in Fig. 19 and Fig. 20 respectively.

SOWL QL statements are translated to SPARQL according to the algorithm of Fig. 21. The algorithm applies to triples in the *While* clause and proceeds by translating spatial triples before temporal ones. The

Operator	Operator Translation
Operator B.1.1: Timepoint	**Syntax:** ?subject ex:predicate ?object operator("timepoint") ?intervalX time:hasBeginning ?intervalXstart. ?intervalX time:hasEnd ?intervalXend. ?timepoint time:inXSDDateTime "timepoint". **Timepoint Operator Translation (Fig. 18)**
Operator B.1.2: TimeInterval	**Syntax:** ?subject ex:predicate ?object operator("intervalStart""intevalEnd") ?intervalX time:hasBeginning ?intervalXstart. ?intervalX time:hasEnd ?intervalXend. ?intervalY time:hasBeginning ?intervalYstart. ?intervalY time:hasEnd ?intervalYend. ?intervalYstart time:inXSDDateTime "intervalStart". ?intervalYend time:inXSDDateTime "intervalEnd". **Time Interval Operator Translation (Fig. 19)** **or** **Allen Operator Translation (Fig. 20)**
Operator B.1.3: Qualitative	**Syntax:** ?subjecX ex:predicateX ?objectX operator ?subjectY ex:predicateY ?objectY ?intervalX time:hasBeginning ?intervalXstart. ?intervalX time:hasEnd ?intervalXend. ?intervalY time:hasBeginning ?intervalYstart. ?intervalY time:hasEnd ?intervalYend. **Allen Operator Translation (Fig. 20)**

Fig. 17 Translation of SOWL QL temporal operators to SPARQL.

Operator	Translation
AT	{ ?intervalXstart time:before ?timepoint. ?intervalXend time:after ?timepoint. } UNION { ?intervalXstart time:equals ?timepoint.} UNION {?intervalXend time:equals ?timepoint.}
AFTER	?intervalXstart time:after ?timepoint..
BEFORE	?intervalXend time:before ?timepoint.
STARTSAT	?intervalXstart time:equals ?timepoint.
ENDSAT	?intervalXend time:equals ?timepoint.

Fig. 18 Translation of timepoint operators.

output is a SPARQL expression for each spatial or temporal SOWL QL triple.

During translation, special variables are generated in order to map queries to the underlying temporal and spatial model concepts (e.g., *timeslices*). A unique identifier *UID* is used after the name of each special variable in order to prevent confusion with user defined variables (e.g., *location1*, *interval30*, etc.). More specifically, the following types of special variables are generated during translation:

Operator	Translation
ALWAYS_AT	A disjunction of the translations of the Allen operators: CONTAINS, EQUALS, STARTEDBY, ENDEDBY.
SOMETIME_AT	A disjunction of the translations of the Allen operators: EQUALS, OVERLAPS, OVERLAPPEDBY, STARTS, STARTEDBY, ENDS, ENDEDBY, CONTAINS, DURING.

Fig. 19 Translation of time interval operators.

Operator	Translation
BEFORE	?intervalXend time:before ?intervalYstart
AFTER	?intervalXstart time:after ?intervalYend.
MEETS	?intervalXend time:equals ?intervalYstart.
METBY	?intervalXstart time:equals ?intervalYend.
OVERLAPS	?intervalXstart time:before ?intervalYstart. ?intervalXend time:after ?intervalYstart. ?intervalXend time:before ?intervalYend.
OVERLAPPEDBY	?intervalXstart time:after ?intervalYstart. ?intervalXstart time:before ?intervalYend. ?intervalXend time:after ?intervalYend.
DURING	?intervalXstart time:after ?intervalYstart. ?intervalXend time:before ?intervalYend.
CONTAINS	?intervalXstart time:before ?intervalYstart. ?intervalXend time:after ?intervalYend.
STARTS	?intervalXstart time:equals ?intervalYstart. ?intervalXend time:before ?intervalYend.
STARTEDBY	?intervalXstart time:equals ?intervalYstart. ?intervalXend time:after ?intervalYend.
ENDS	?intervalXstart time:before ?intervalYstart. ?intervalXend time:equals ?intervalYend.
ENDEDBY	?intervalXstart time:after ?intervalYstart. ?intervalXend time:equals ?intervalYend.
EQUALS	?intervalXstart time:equals ?intervalYstart. ?intervalXend time:equals ?intervalYend.

Fig. 20 Translation of Allen operators.

- An *event* special variable with name *eventUID* for each dynamic temporal triple of the N-ary temporal model.
- Two *timeslice* variables with names *timesliceUID* corresponding to the timeslices of the subject and the object of the 4D-fluents temporal model.
- A *timepoint* variable with name *timepointUID* for each time point in temporal triplets.
- An *interval* variable with name *intervalUID* for each interval used in a temporal triple.
- Two *interval* variables with names *intervalUIDStart* and *intervalUIDEnd* corresponding to the starting and ending points of a time interval.
- A *location* variable with name *locationUID* and a *geometry* variable with name *geometryUID* corre-

```
Translate Spatial Triples:
for each triple in the while clause:
  Read triple
  if ( triple is spatial )
    Apply Case A.2 (Fig.16)
  if ( triple is spatio-temporal )
    if ( model is N-ary )
      Apply Case A.3.1 (Fig.16)
    else
    if( model is 4D-Fluents )
      Apply Case A.3.2 (Fig.16)

Translate Temporal Triples:
for each triple in the while clause:
  Read triple
  if ( triple is temporal )
    if ( model is 4D-Fluents )
      Apply Case A.1.1 (Fig.16)
    else
    if ( model is N-ary )
      Apply Case A.1.2 (Fig.16)
    if (temporal operator exists after triple)
      if( timepoint operator )
        Apply Case B.1.1 (Fig.17)
      else
      if ( time interval operator )
        Apply Case B.1.2 (Fig.17)
      else
      if ( qualitative operator )
        Read next triple
        Apply Case B.1.3 (Fig.17)
```

Fig. 21 Translation Algorithm.

Query

```
SELECT ?company ?employee
WHERE
{
  ?company ex:hasEmployee ?employee B
  BEFORE("2007-02-05T00:00:00")
}
```

Translation (N-ary)

```
SELECT ?company ?employee
WHERE {
?company  ex:hasEmployee ?event0.
?event0 ex:hasEmployee ?employee.
?event0 time:atTime    ?interval0.

?interval0 time:hasBeginning ?interval0start.
?interval0 time:hasEnd ?interval0end.
?timepoint0 time:inXSDDateTime
"2007-02-05T00:00:00"^^xsd:dateTime.

?interval0end time:before ?timepoint0.
}
```

Case A.1.1 (Fig.16) — Case B.1.1 (Fig.17) — Operator (Fig.18)

Translation (4D-Fluents)

```
SELECT ?company ?employee
WHERE
{
?timeSlice0 sowl:tsTimeSliceOf ?company.
?timeSlice0 sowl:tsTimeInterval ?interval0.
?timeSlice0 ex:hasEmployee ?timeSlice1.
?timeSlice1 sowl:tsTimeSliceOf ?employee.
?timeSlice1 sowl:tsTimeInterval ?interval0.

?interval0 time:hasBeginning ?interval0start.
?interval0 time:hasEnd ?interval0end.
?timepoint0 time:inXSDDateTime
"2007-02-05T00:00:00"^^xsd:dateTime.

?interval0end time:before ?timepoint0.
}
```

Case A.1.2 (Fig.16) — Case B.1.1 (Fig.17) — Operator (Fig.18)

Fig. 22 Translation of a temporal SOWL QL query to SPARQL.

sponding to concepts of the underlying spatial SOWL model.

Fig. 22 and Fig. 23 illustrate two SOWL QL queries and their translation to SPARQL queries. The first, is a temporal query that retrieves employees working for a company before a specific time point. The second, is a spatio temporal query that retrieves planes flying South of Italy during a specific time interval. Notice the simplicity of SOWL QL expressions as opposed to the complexity of the equivalent SPARQL queries. SPARQL query expressions involve model specific triples (e.g., *tsTimeSliceOf*, *inXSDDDateTime*, *hasGeometry*, *locatedAT*) implying that the user has to familiar with peculiarities of they underlying temporal or spatial mode. It is almost impossible for ordinary users (even for experts) to formulate such queries.

6 SOWL QL system

SQWL QL queries are issued using a Graphical User Interface (GUI). Fig. 24 illustrates the architecture of SOWL QL translator. It consists of several modules the most important of them being the (a) ontology loader, (b) query parser and (c) the interpreter.

Initially, the user chooses an OWL ontology to load into the memory. The ontology loader determines if a dynamic (temporal or spatial) model is implemented within the ontology. This task is accomplished by parsing the concepts of the ontology and by identifying one by one the URIs (Universal Resource Identifiers are unique strings that describes resources) of the specific classes and properties that are required by each model representation. For example, in order to identify the 4D-fluents model, the *TimeSlice* and *Interval* classes and also the *tsTimeSlice* and *tsTimeInterval* properties must be identified.

Query

```
SELECT ?plane
WHERE
{
 ?plane spatial:Sof ex:Italy
 DURING("2015-02-05T00:00:00","2015-02-06T00:00:00").
}
```

Translation (N-ary)

Case A.3.1 (Fig.16)

```
SELECT ?plane
WHERE {
?plane spatial:LocatedAt ?_location0.
?_location0 spatial:hasGeometry ?_geometry0.
ex:Italy spatial:LocatedAt ?_location1.
?_location1 spatial:hasGeometry ?_geometry1.
?_location0 spatial:SoF ?_location1

?_location0 ex:SoF ?_event0.
?_event0 ex:SoF ?_location1.
?_event0 time:atTime ?_interval0.
```

Case B.1.2 (Fig.17)

```
?_interval0 time:hasBeginning ?_interval0start.
?_interval0 time:hasEnd ?_interval0end.
?_interval1 time:hasBeginning ?_interval1start.
?_interval1 time:hasEnd ?_interval1end.
?_interval1start time:inXSDDateTime
"2015-02-05T00:00:00"^^xsd:dateTime.
?_interval1end time:inXSDDateTime
"2015-02-06T00:00:00"^^xsd:dateTime.
```

Operator (Fig.20)

```
?_interval0start time:after ?_interval1start.
?_interval0end time:before ?_interval1end.
}
```

Translation (4D-Fluents)

Case A.3.2 (Fig.16)

```
SELECT ?plane
WHERE {
?plane spatial:LocatedAt ?_location0.
?_location0 spatial:hasGeometry ?_geometry0.
ex:Italy spatial:LocatedAt ?_location1.
?_location1 spatial:hasGeometry ?_geometry1.
?_location0 spatial:SoF ?_location1

?_timeSlice0 sowl:tsTimeSliceOf ?_location0.
?_timeSlice0 sowl:tsTimeInterval ?_interval0.
?_timeSlice0 spatial:SoF ?_timeSlice1.
?_timeSlice1 sowl:tsTimeSliceOf ?_location1.
?_timeSlice1 sowl:tsTimeInterval ?_interval0.
```

Case B.1.2 (Fig.17)

```
?_interval0 time:hasBeginning ?_interval0start.
?_interval0 time:hasEnd ?_interval0end.
?_interval1 time:hasBeginning ?_interval1start.
?_interval1 time:hasEnd ?_interval1end.
?_interval1start time:inXSDDateTime
"2015-02-05T00:00:00"^^xsd:dateTime.
?_interval1end time:inXSDDateTime
"2015-02-06T00:00:00"^^xsd:dateTime.
```

Operator (Fig.20)

```
?_interval0start time:after ?_interval1start.
?_interval0end time:before ?_interval1end.
}
```

Fig. 23 Translation example of a spatio temporal SOWL QL query to SPARQL.

If a dynamic model is identified, the ontology loader enables the appropriate parser (temporal, spatial or both) and activates the corresponding temporal or spatial interpreter. For example, if the N-ary relations model is recognized then, the ontology loader will enable the

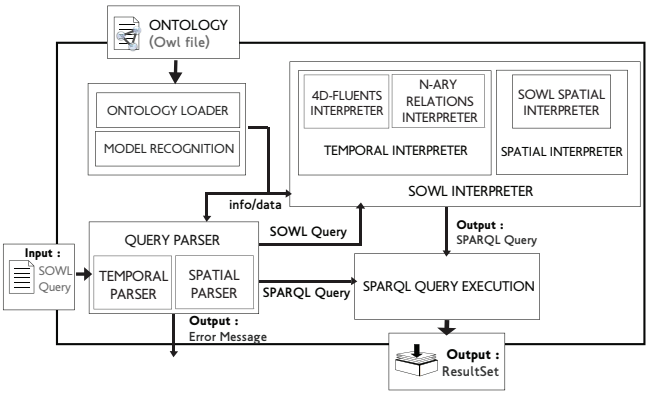

Fig. 24 SOWL QL System Architecture.

temporal parser and the N-ary relations interpreter. The enabled parser searches for existing SOWL QL operators or dynamic predicates in the query. If no SOWL QL operators or dynamic predicates are found then, no translation takes place and the query is executed as an ordinary SPARQL query. If spatio-temporal operators or dynamic predicates are found then, the parser uses the appropriate interpreter to translate the query to SPARQL. When the translation process is completed, the resulted SPARQL query is executed and the results are displayed. Fig. 25 illustrates a snapshot of SOWL QL Graphical User Interface.

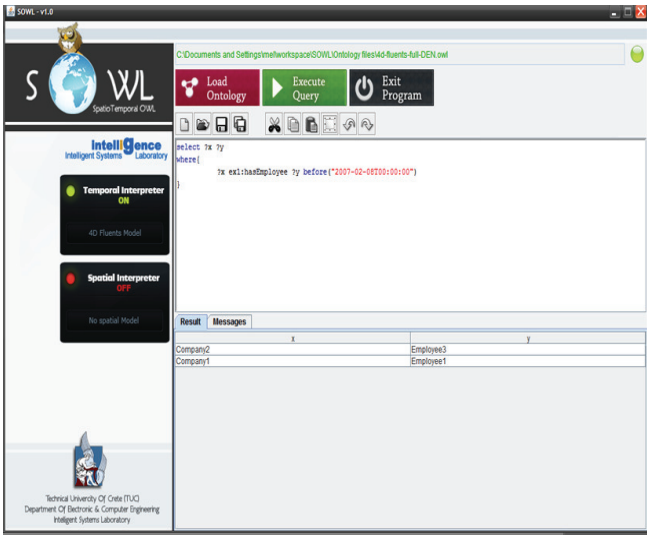

Fig. 25 SOWL QL Graphical User Interface

The parser module is enabled only when a dynamic model is detected by the ontology loader. It consists of a temporal and a spatial parser which can be enabled (or disabled) separately according to spatial model recognized (e.g., if no spatial model is recognized then the spatial parser is never enabled). Each parser is capable of identifying a set of spatio-temporal operators in

the query and enable the appropriate interpreters according to the model that has been recognized in order to translate it to SPARQL. If both the temporal and spatial parsers are enabled, the spatial parser always precedes the temporal parser.

As discussed in Sec. 5, a SOWL QL operator can be used only after a triple pattern or between two triple patterns inside the *WHERE* clause. Moreover, the predicate of each triple can be a fluent one holding over a specific interval (e.g., *ex:Company ex:hasEmployee-ex:Employee*) or it can be a spatial operator (e.g *?x spatial:Nof ?y*). The implementation of the parser is based on these simple acknowledgments. The parser scans the triples in the *WHERE* clause sequentially and uses a look-ahead operator to read the next token after a triple.

The interpreter module translates SOWL QL expressions into equivalent SPARQL queries. The interpreter module is also responsible for invoking the reasoner for triples specifying values which do not exist in the knowledge base. The interpreter is invoked by the parser whenever a triple with a fluent predicate or a SOWL QL operator is recognized. The main interpreter class encompasses two classes implementing the temporal and the spatial interpreters respectively. The temporal and spatial interpreters are also specialized to more specific interpreters according to the ontology dynamic model in use (i.e., 4D-fluents or N-ary relations in this work). For example, if the ontology loader module detects the 4D-fluents model then the corresponding temporal interpreter is enabled. Fig. 26 illustrates a schematic overview of the implementation of the interpreter. Finally, different query patterns call for the appropriate instantiations of the interpreter [60].

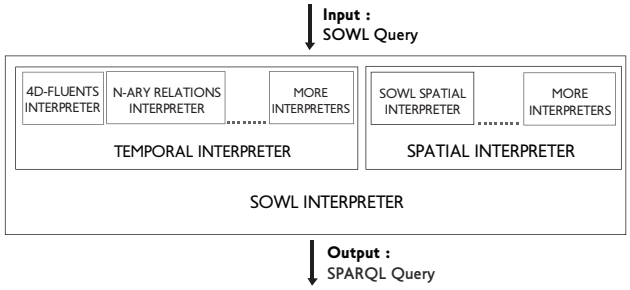

Fig. 26 SOWL QL Interpreter.

7 Evaluation

The run-time efficiency of SOWL QL is assessed experimentally in a real data setting. The SOWL ontology is instantiated with data from AIS Brest dataset [8] using the 4D-fluents temporal model. The dataset contains 1,048,576 position records of ship vessels in Brest area, France, during the year 2009. Each record provides (among others) information such as ship code, speed, position (longitude, latitude) and time. Accordingly, the instantiated ontology represents ship positions at various instances in time. We worked with the first 1,000 data set records referring to positions of 20 ships at various time instances. The CHRONOS [3] and CHOROS [37] reasoners are applied for inferring all implied spatio-temporal relations (including spatial relations between any two ships) and checking the ontology for consistency. All inferred relations are instantiated in the ontology resulting in an ontology with 2,113,835 triples.

Fig. 27 illustrates the representation of a moving ship. Property `movesTo` is a dynamic property as its values change in time (its domain and range is of class `TimeSlice`). Fig. 28 shows the representation of spatial relations between two ships. These spatial relations are handled as dynamic properties.

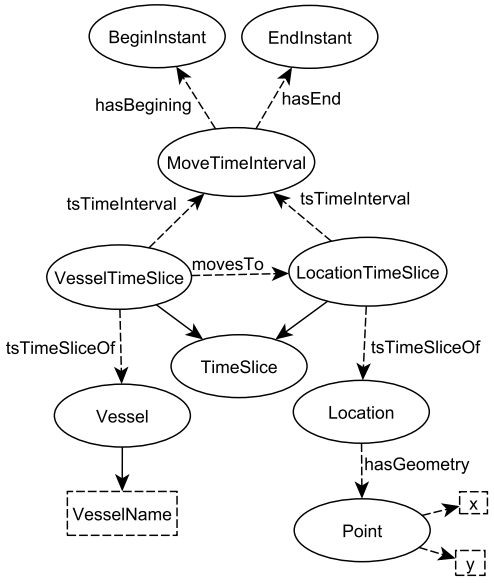

Fig. 27 Representation of a moving ship.

The purpose of the following experiments is to demonstrate the dependence of query performance on both, query complexity and the size of the data set. To this end, we measured the performance of the basic query statements of Section 5. Fig. 29 lists theSOWL QL queries tested on AIS Brest dataset.

[8] http://chorochronos.datastories.org/?q=node/9

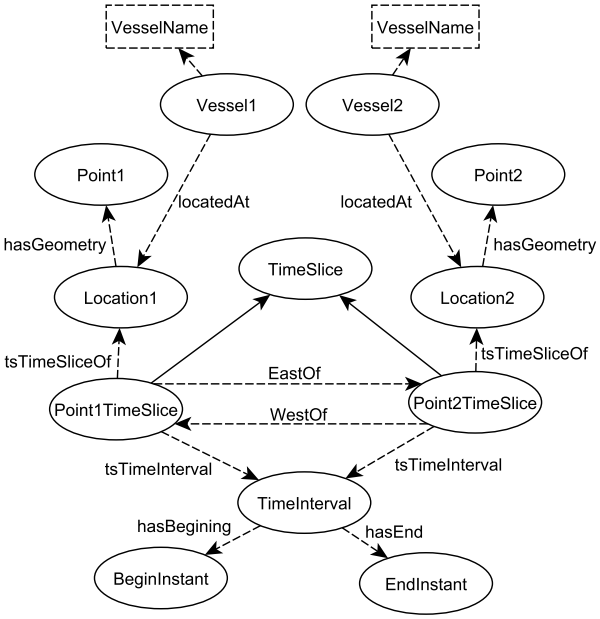

Fig. 28 Representation of dynamic spatial relation.

Query statement	SOWL QL query

Temporal triple
Retrieve movements of vessel "220247000".
	select ?vessel ?location
	where {
	?vessel ex1:movesTo ?location.
	?vessel ex1:vesselName "220247000"
	}

Temporal triple
Time point operator
Retrieve vessel movements after time point "2009-02-11T14:37:11".
	select ?vessel ?location
	where {
	?vessel ex1:movesTo ?location
	AFTER ("2009-02-11T14:37:11")
	}

Temporal triple
Time interval operator
Retrieve vessel movements in temporal intervals overlappingor during the one specified.
	select ?vessel ?location
	where {
	?vessel ex1:movesTo ?location
	SOMETIME AT("2009-02-11T14:37:36",
	"2009-02-11T14:37:45")
	}

Temporal triple
Qualitative operator
Retrieve vessel movements that occur before movements of Vessel "220247000"
	select ?vessel ?location
	where {
	?vessel ex1:movesTo ?location
	AFTER
	?vessel2 ex1:movesTo ?location2.
	?vessel2 ex1:vesselName "220247000"
	}

Spatio-Temporal triple
Retrieve vessels located North of vessel "220247000"
	select ?vessel
	where {
	?vessel1 spatial:Nof ?vessel2.
	?vessel2 ex1:vesselName "220247000"
	}

Spatio-Temporal triple
Temporal Operator
Retrieve vessels located South-East of vessel"220247000" after the time specified.
	select ?vessel
	where {
	?vessel spatial:SEof ?vessel2
	AFTER ("2009-02-11T14:37:11").
	?vessel2 ex1:vesselName "220247000"
	}

Fig. 29 Representative SOWL QL queries tested on AIS Brest dataset.

For every SOWL QL query expression, query response times are reported (each measurement is the average over 10 queries). For queries specifying a spatial location or time point or temporal interval, this quantity is taken from the ontology. This way we avoid invoking the reasoners during query execution; otherwise reasoning times would be included to the response times of these queries. Therefore, all times reported below account solely for query parsing, translation to SPARQL and query execution. The queries were run on a Dell PowerEdge R300 computer, 2.5Ghz with 16GB RAM.

Fig. 30 illustrates the response time as the size of the ontology varies. For all queries, the response time scales linearly with the size of the ontology. As expected, queries specifying a temporal triple alone or temporal triple with a time-point operator perform much better. However, queries with spatio-temporal triples exhibited high response times. In general, the response time of queries with more complicated query statements declined significantly for larger data sets.

As can be seen in Fig. 16, the more involved a query statement is, the longer the size of the resulting SPARQL expression. As SPARQL expressions are evaluated in time proportional to the size of the query pattern [43], our experimental measurements are consistent with expectations.

Consequently, the inefficiency of SOWL QL query response time is a direct consequence of the size of the ontology, which, if materialized, grows quadratically to the size of the input datasets (as the reasoners are invoked after loading the ontology in memory) and requires proportionally high times to search. Motivated by this problem, previous works have limited their scope to situations where spatio-temporal knowledge is completely quantitative, whereas SOWL was specifically designed to handle *qualitative and quantitative* knowledge. By limiting to purely quantitative knowledge, it is possible to avoid materializing the complete set of triples, and instead store the dataset in a spatio-temporal database, computing spatio-temporal triples on-the-fly. However, spatio-temporal database engines for handling both quantitative and qualitative knowledge are not currently available [40].

7.1 Improving the Performance

Having identified the root cause of inefficiency in the current implementation of the SOWL QL query engine, we now focus on a discussion of avenues for future research into ameliorating this problem. There are

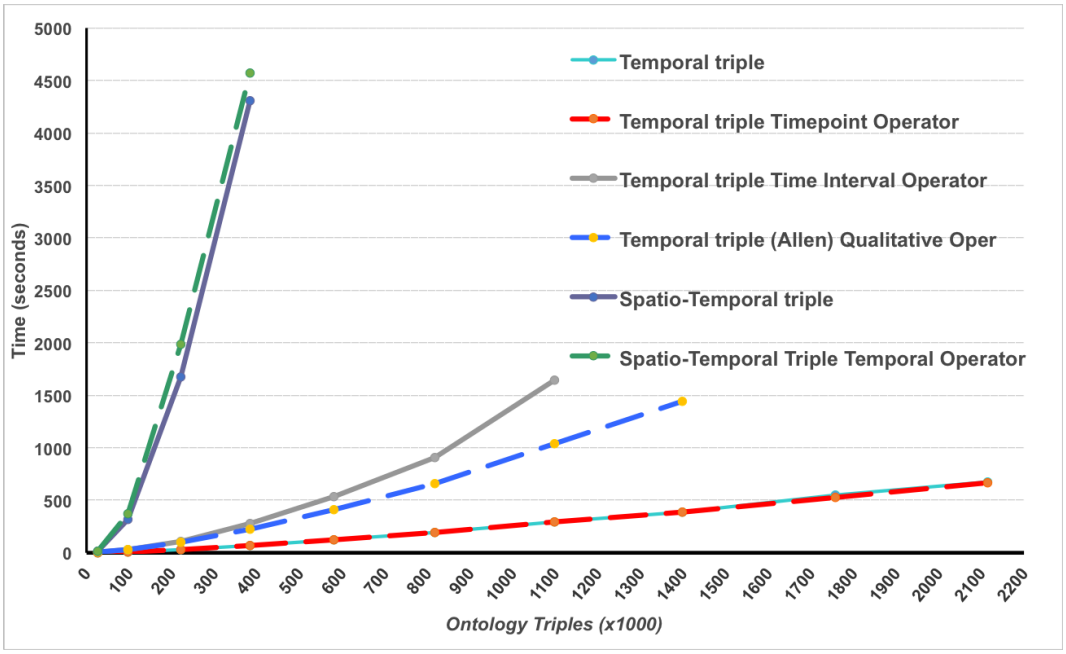

Fig. 30 Average response time of SOWL QL queries on AIS Brest data set.

three main strategies for efficient query evaluation: (a) a query rewriting engine, based on some model of the execution cost, (b) efficient storage structures, including secondary index structures, for organizing the data, and (c) a suite of alternative search algorithms that utilize indexes in order to implement the query language primitives efficiently. Although Jena ARQ [9] has no built-in indexing support for temporal or spatial data types, its extension points allow for custom index mechanisms (via the so-called "magic properties") and search algorithms (termed *filter functions*). Furthermore, it allows customized query rewriting (by user-defined *transformers*). We treat each issue in turn.

In the current implementation of SOWL QL, the spatial parser is always invoked before the temporal one. A more sophisticated query optimizer would exploit selectivity statistics to estimate the size of intermediate results, so to reorder the evaluation of patterns appropriately. Intuitively, the most selective patterns of a query are those that involve literal values; in our case, times, time intervals and spatial geometries. However, because of the transitive nature of spatio-temporal relations, the number of ground triples per value is very high. Therefore, in order to meaningfully perform query rewriting on the spatio-temporal patterns of a query, such triples should not be instantiated explicitly.

An interesting challenge is to develop suitable indexing mechanisms for efficiently storing and searching over qualitative as well as quantitative knowledge. To this end, there are ad-hoc techniques that could be implemented using the current state-of-the-art in data structures, or more principled approaches, into the design of novel data structures.

One ad-hoc but possibly practical solution could be to split spatio-temporal relations into two parts: one containing those triples that relate quantitative entities, and the rest. The first part, containing the bulk of the triples would not be instantiated; instead, a spatio-temporal index structure could serve to represent the triples implicitly and also perform efficient matching. The second part would be explicitly instantiated, but its size would hopefully be small. To this end, an efficient query engine could utilize provably efficient data structures such as the Interval Tree [45] in main memory, or the External Memory Interval Tree [4] when the data is stored on disk, to handle temporal data, or indeed, any of a large number of data structures developed for spatio-temporal databases [54].

Another alternative is to invent new data structures that can store the output of a reasoner in a compressed, quickly searchable form. The crucial observation here is that spatio-temporal relations exhibit high redundancy due to transitivity. Therefore, one could investigate data structures for partial orders. This problem has received some attention in recent years (e.g., [22]). Recent results indicate that the search cost of such techniques is proportional to the so-called width w of the partial order (the cardinality of the maximum set of mutually incomparable elements). Another important concept is the so-called Dushnik-Miller dimension of a

[9] https://jena.apache.org/documentation/query/

partial order: the smallest number of total orders whose intersection gives rise to the partial order (e.g., see [67]). If this dimension is small, one can use standard multidimensional index structures to store qualitative information efficiently. When the spatio-temporal information is complete, this dimension is small for spatio-temporal knowledge, but may be larger in the presence of incomplete information.

In addition to data structures, new matching algorithms are needed, that would be able to exploit these data structures, to perform part of the spatio-temporal reasoning on-the-fly; for example, a memory-efficient version of path consistency, which would utilize existing indexes.

8 Conclusions and Future Work

We introduce SOWL QL, a query language for spatio-temporal ontologies. SOWL QL builds-upon SOWL [9], a representation model for temporal and spatial knowledge in OWL. Addressing qualitative and quantitative information in queries is a unique feature of SOWL QL. SOWL QL is also supported by reasoning for certain query types (i.e., queries specifying exact temporal or spatial values such as points or intervals). In addition, SOWL QL syntax is independent of representation model (i.e., the 4D-fluents or the N-ary model in this work), so users need not be familiar with the peculiarities of the underlying representation model applied. These are distinctive features of SOWL QL not supported by query languages such as stSPARQL, SPARQL-ST or τ-SPARQL.

SOWL QL supports a wide arsenal of temporal and spatial operators (all temporal Allen relations and all spatial topological and directional relations operators are implemented as operators) not supported by other query languages such as those referred to above. stSPARQL, GeoSPARQL in particular, are tailored to the needs of specific application fields (e.g., geography, cartography) where addressing information in their physical (e.g., polygonal) form is more important than addressing natural language (i.e., textual) descriptions implying that it is not particularly meant to be used for the Web. In this direction, we showed how SOWL QL can be enhanced with new low-level operators for lines and polygons in the example of GeoSPARQL, stSPARQL and SPATQL-ST or topological operators (such as those by Clementini [19]).

Query optimization is an important direction for future work: optimizing SOWL QL might require that Jena ARQ is extended (or replaced by a new query mechanism possibly skipping translation to SPARQL) for optimal execution of spatial and temporal expressions. An almost orthogonal issue is speed of search: query response times can be speeded-up significantly by incorporating into the query search process indexing mechanisms for both qualitative and quantitative spatial and temporal information. Although indexing for exclusively quantitative information is feasible using standard techniques, support of qualitative knowledge in SOWL model does not render it directly amenable to indexing techniques.

References

1. Allen, J.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26**(11), 832–843 (1983)
2. Allen, J.F.: Maintaining Knowledge About Temporal Intervals. Commun. ACM **26**(11), 832–843 (1983). DOI http://doi.acm.org/10.1145/182.358434
3. Anagnostopoulos, E., Petrakis, E.G.M., Batsakis, S.: CHRONOS: Improving the Performance of Qualitative Temporal Reasoning in OWL. In: ICTAI, pp. 309–315. IEEE Computer Society (2014)
4. Arge, L., Vitter, J.S.: Optimal dynamic interval management in external memory. In: 37th Annual Symposium on Foundations of Computer Science, pp. 560–569 (1996)
5. Artale, A., Franconi, E.: A Survey of Temporal Extensions of Description Logics. Annals of Mathematics and Artificial Intelligence **30**(1), 171–210 (2000)
6. Baader, F.: Description Logics. In: Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009, *Lecture Notes in Computer Science*, vol. 5689, pp. 1–39. Springer–Verlag (2009)
7. Balbiani, P., Condotta, J.F., del Cerro, L.F.: A New Tractable Subclass of the Rectangle Algebra. In: IJCAI, pp. 442–447 (1999)
8. Baratis, E., Maris, N., Petrakis, E., Batsakis, S., Papadakis, N.: The TOQL System. 11th International Symposium on Spatial and Temporal Databases (SSTD 2009), Demo pp. 450–454 (2009)
9. Batsakis, S.: SOWL: A Framework for Handling Spatio - Temporal Information in OWL. Ph.D. thesis, Dept. of Electronic and Comp. Engineering, Technical Univercity Of Crete (2011)
10. Batsakis, S., Petrakis, E.: SOWL: A Framework for Handling Spatio - Temporal Information in OWL 2.0. In: 5th International Symposium on Rules: Research Based and Industry Focused (RuleML' 2011), pp. 242–249 (2011)
11. Batsakis, S., Petrakis, E.: Imposing Restrictions over Temporal Properties in OWL: A Rule-Based Approach. In: A. Bikakis, A. Giurca (eds.) Rules on the Web: Research and Applications, *Lecture Notes in Computer Science*, vol. 7438, pp. 240–247. Springer Berlin Heidelberg (2012). DOI 10.1007/978-3-642-32689-9_19. URL `http://dx.doi.org/10.1007/978-3-642-32689-9_19`
12. Batsakis, S., Stravoskoufos, K., Petrakis, E.: Temporal Reasoning for Supporting Temporal Queries in OWL 2.0. 15th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES' 2011) **6881**, 558–567 (2011)
13. Beddoe, D., Cotton, P., Uleman, R., Johnson, S., Herring, J.R.: OpenGIS Simple Features Specification For SQL. Tech. rep., OGC (1999)

14. Bodirsky, M., Chen, H.: Qualitative Temporal and Spatial Reasoning Revisited. Journal of Logic and Computation **19**, 1359–1383 (2009)
15. Budak Arpinar, I., Sheth, A., Ramakrishnan, C., Lynn Usery, E., Azami, M., Kwan, M.: Geospatial Ontology Development and Semantic Analytics. Transactions in GIS **10**(4), 551–575 (2006)
16. Buneman, P., Kostylev, E.: Annotation Algebras for RDFS. In: 2nd International Workshop on the Role of Semantic Web in Provenance Management (SWPM-10) (2010)
17. Bykau, S., Mylopoulos, J., Rizzolo, F., Velegrakis, Y.: On Modeling and Querying Concept Evolution. Journal Data Semantics **1**(1), 31–55 (2012). URL http://dblp.uni-trier.de/db/journals/jodsn/jodsn1.html#BykauMRV12
18. Champin, P., Passant, A.: SIOC in Action Representing the Dynamics of Online Communities. In: Proceedings of the 6th International Conference on Semantic Systems, pp. 1–7. ACM (2010)
19. Clementini, E., Felice, P.D., van Oosterom, P.: A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In: D.J. Abel, B.C. Ooi (eds.) SSD, *Lecture Notes in Computer Science*, vol. 692, pp. 277–295. Springer (1993)
20. Cohn, A., Bennett, B., Gooday, J., Gotts, N.: Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. GeoInformatica **1**(3), 275–316 (1997)
21. Cohn, A.G., Bennett, B., Gooday, J., Gotts, N.M.: Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. GeoInformatica **1**(3), 275–316 (1997)
22. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and Selection in Posets. SIAM J. Comput. **40**(3), 597–622 (2011). DOI 10.1137/070697720. URL http://dx.doi.org/10.1137/070697720
23. Egenhofer, M.J., Franzosa, R.D.: Point-Set Topological Spatial Relations. International Journal of Geographical Information Systems **5**(2), 161–174 (1991)
24. Frasincar, F., Milea, V., Kaymak, U.: tOWL: Integrating Time in OWL. Semantic Web Information Management: A Model-Based Perspective pp. 225–246 (2010)
25. Gutierrez, C., Hurtado, C., Vaisman, A.: Temporal RDF. In: Second European Semantic Web Conference (ESWC 2005), pp. 93–107 (2005)
26. Gutierrez, C., Hurtado, C.A., Vaisman, A.: Introducing Time into RDF. IEEE Transactions on Knowledge and Data Engineering **19**(2), 207–218 (2007). DOI 10.1109/tkde.2007.34. URL http://dx.doi.org/10.1109/tkde.2007.34
27. Guting, R.: An Introduction to Spatial Database Systems. The VLDB Journal **3**(4), 357–399 (1994)
28. Hart, G., Dolbear, C.: Linked data: A Geo-Spatial Perspective, chap. 6. CRC Press (2013)
29. Hatzigeorgakidis, G.: Management of Spatio - temporal Information in Semantic Web Applications. Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete (2011)
30. Hobbs, J., Pan, F.: Time Ontology in OWL. W3C Working Draft, September 2006 (2006). URL http://www.w3.org/TR/owl-time/
31. Jonsson, P., Krokhin, A.: Complexity Classification in Qualitative Temporal Constraint Reasoning. Artificial Intelligence **160**(1-2), 35–51 (2004)
32. Klein, M., Fensel, D.: Ontology Versioning on the Semantic Web. In: Proceedings of the International Semantic Web Working Symposium (SWWS), pp. 75–91. Citeseer (2001)
33. Koubarakis, M., Kyzirakos, K.: Modeling and Querying Metadata in the Semantic Sensor Web: the Model stRDF and the Query Language stSPARQL. Proceedings of the 7th Extended Semantic Web Conference (ESWC2010) pp. 425–439 (2010)
34. Krokhin, A., Jeavons, P., Jonsson, P.: Reasoning About Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. Journal of the ACM (JACM) **50**(5), 591–640 (2003)
35. Lutz, C.: Description logics with concrete domains-a survey. In Advances in Modal Logics, volume 4. King's College Publications (2003)
36. Lutz, C., Wolter, F., Zakharyashev, M.: Temporal Description Logics: A Survey. In: 15th International Symposium on Temporal Representation and Reasoning, TIME'08, pp. 3–14. IEEE (2008)
37. Mainas, N., Petrakis, E.G.M.: CHOROS 2: Improving the Performance of Qualitative Spatial Reasoning in OWL. In: ICTAI, pp. 283–290. IEEE Computer Society (2014)
38. Montello, D., Frank, A.: Modeling Directional Knowledge and Reasoning in Environmental Space: Testing Qualitative Metrics. The Construction of Cognitive Maps GeoJournal Library **32**(3), 321–344 (1996)
39. Nebel, B., Burckert, H.: Reasoning About Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. Journal of the ACM (JACM) **42**(1), 43–66 (1995)
40. Nikolaou, C., Koubarakis, M.: Querying incomplete geospatial information in RDF. In: Advances in Spatial and Temporal Databases - 13th International Symposium, SSTD 2013, Munich, Germany, August 21-23, 2013. Proceedings, pp. 447–450 (2013). DOI 10.1007/978-3-642-40235-7_26. URL http://dx.doi.org/10.1007/978-3-642-40235-7_26
41. Noy, N., Rector, A.: Defining N-ary Relations on the Semantic Web (2006). URL http://www.w3.org/TR/swbp-n-aryRelations/
42. Open Geospatial Consortium: GeoSPARQL - A geographic query language for RDF data, A proposal for an OGC Draft Candidate Standard (2010)
43. Perez, J., Arenas, M., Gutierrez, C.: The Semantics and Complexity of SPARQL. In: 5th International Semantic Web Conference, ISWC 2006 (2006). URL http://www.dcc.uchile.cl/~cgutierr/papers/sparql.pdf
44. Perry, M., Jain, P., Sheth, A.P.: SPARQL-ST: extending SPARQL to support spatiotemporal queries. In: N. Ashish, A.P. Sheth (eds.) Geospatial Semantics and the Semantic Web, no. 12 in Semantic Web and Beyond, chap. 3, pp. 61–86. Springer, New York (2011)
45. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag (1985)
46. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C working draft **4** (2006). URL http://www.w3.org/TR/rdf-sparql-query/
47. Pujari, A., Sattar, A.: A New Framework for Reasoning About Points, Intervals and Durations. In: International Joint Conference On Artificial Intelligence, vol. 16, pp. 1259–1267. Lawrence Erlbaum Associates Ltd (1999)
48. Prez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. Journal of Web Semantics **8**(4), 255–270 (2010). URL http://dblp.uni-trier.de/db/journals/ws/ws8.html#PerezAG10
49. Randell, D., Cui, Z., Cohn, A.: A Spatial Logic Based on Regions and Connection. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, (KR 92) **92**, 165–176 (1992)

50. Renz, J.: Maximal Tractable Fragments of the Region Connection Calculus: A Complete Analysis. In: International Joint Conference On Artificial Intelligence, vol. 16, pp. 448–455 (1999)
51. Renz, J., Mitra, D.: Qualitative Direction Calculi with Arbitrary Granularity. In: Trends in Artificial Intelligence: 8th Pacific Rim International Conference on Artificial Intelligence, Proceedings (PRICAI 2004), pp. 65–74 (2004)
52. Renz, J., Mitra, D.: Qualitative Direction Calculi with Arbitrary Granularity. In: C. Zhang, H.W. Guesgen, W.K. Yeap (eds.) PRICAI, *Lecture Notes in Computer Science*, vol. 3157, pp. 65–74. Springer (2004)
53. Renz, J., Nebel, B.: Qualitative Spatial Reasoning using Constraint Calculi. Handbook of Spatial Logics pp. 161–215 (2007)
54. Rigaux, P., Scholl, M., Voisard, A.: Spatial databases - with applications to GIS. Elsevier (2002)
55. Sellis, T.: Research Issues in Spatio - temporal Database Systems. Advances in Spatial Databases (Book Chapter) **1651**, 5–11 (1999)
56. Shaw, R., Troncy, R., Hardman, L.: Lode: Linking Open Descriptions of Events. The Semantic Web pp. 153–167 (2009)
57. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Web Semantics: Science, Services and Agents on the World Wide Web **5**(2), 51–53 (2007)
58. Skiadopoulos, S., Koubarakis, M.: On the Consistency of Cardinal Direction Constraints. Artificial Intelligence **163**(1), 91–135 (2005)
59. Stocker, M., Sirin, E.: PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In: 6^{th} Intern. Workshop on OWL: Experiences and Directions (OWLED 2009), pp. 2–31. Springer-Verlag New York, Inc. (2009)
60. Stravoskoufos, K.: SOWL QL: Querying Spatio - Temporal Ontologies in OWL 2.0 (2013). MSc Thesis
61. Tao, C., Wei, W., Solbrig, H., Savova, G., Chute, C.: CNTRO: A Semantic Web Ontology for Temporal Relation Inferencing in Clinical Narratives. In: AMIA Annual Symposium Proceedings, vol. 2010, pp. 787–91. American Medical Informatics Association (2010)
62. Tappolet, J., Bernstein, A.: Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In: Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications, pp. 308–322. Springer-Verlag (2009)
63. Van Beek, P.: Approximation Algorithms for Temporal Reasoning. Proceedings of the 11th International Joint Conference on Artificial Intelligence- Volume 2 pp. 1291–1296 (1989)
64. van Beek, P., Cohen, R.: Exact and Approximate Reasoning about Temporal Relations. Computational Intelligence **6**, 132–144 (1990)
65. Vilain, M., Kautz, H.: Constraint Propagation Algorithms for Temporal Reasoning. In: Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 377–382 (1986)
66. Welty, C., Fikes, R.: A Reusable Ontology for Fluents in OWL. In: Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006), pp. 226–336 (2006)
67. Yannakakis, M.: The Complexity of the Partial Order Dimension Problem. SIAM J. on Algebraic and Discrete Methods **3**(3), 351–358 (1982)