NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

PhD THESIS

Efficient Management for Geospatial and Temporal Data using Ontology-based Data Access Techniques

Konstantina Ch. Bereta

ATHENS

JUNE 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Αποτελεσματική Διαχείριση Γεωχωρικών και Χρονικών Δεδομένων με τεχνικές Ανάκτησης Δεδομένων βάσει Οντολογιών

Κωνσταντίνα Χ. Μπερέτα

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2019

PhD THESIS

Efficient Management for Geospatial and Temporal Data using Ontology-based Data Access Techniques

Konstantina Ch. Bereta

SUPERVISOR: Manolis Koubarakis, Professor NKUA

THREE-MEMBER ADVISORY COMMITTEE:

Manolis Koubarakis, Professor NKUA

Yiannis Ioannidis, Professor NKUA

Dimitrios Gunopoulos, Professor NKUA

SEVEN-MEMBER EXAMINATION COMMITTEE

Manolis Koubarakis,
Professor NKUA

Yiannis Ioannidis,
Professor NKUA

Dimitrios Gunopoulos,
Professor NKUA

Nikolaos Mamoulis,
Professor University of Ioannina

Yannis Theodoridis,
Professor Piraues University

Dimitrios Plexousakis,
Professor University of Crete

Diego Calvanese,
Professor University of Bolzano

Examination Date: 6 20, 2019

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Αποτελεσματική Διαχείριση Γεωχωρικών και Χρονικών Δεδομένων με τεχνικές Ανάκτησης Δεδομένων βάσει Οντολογιών

Κωνσταντίνα Χ. Μπερέτα

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Εμμανουήλ Κουμπαράκης, Καθηγητής ΕΚΠΑ

ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ:

Εμμανουήλ Κουμπαράκης, Καθηγητής ΕΚΠΑ

Γιάννης Ιωαννίδης, Καθηγητής ΕΚΠΑ

Δημήτριος Γουνόπουλος, Καθηγητής ΕΚΠΑ

ΕΠΤΑΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Εμμανουήλ Κουμπαράκης,
Καθηγητής ΕΚΠΑ

Γιάννης Ιωαννίδης,
Καθηγητής ΕΚΠΑ

Δημήτριος Γουνόπουλος,
Καθηγητής ΕΚΠΑ

Νικόλαος Μαμουλής,
Καθηγητής Πανεπιστημίου Ιωαννίνων

Γιάννης Θεοδωρίδης,
Καθηγητής Πανεπιστημίου Πειραιά

Δημήτριος Πλεξουσάκης,
Καθηγητής Πανεπιστημίου Κρήτης

Diego Calvanese,
Καθηγητής Πανεπιστημίου Bolzano

Ημερομηνία Εξέτασης: 20 6 2019

ABSTRACT

The data model RDF and query language SPARQL have been widely used for the integration of data coming from different souces. Due to the increasing number of geospatial datasets that are being available as linked open data, a lot of effort focuses in the development of geospatial (and temporal, accordingly) extensions of the framework of RDF and SPARQL. Two highlights of these efforts are the query language GeoSPARQL, that is an OGC standard, and the framework of stRDF and stSPARQL. Both frameworks can be used for the representation and querying of linked geospatial data, and stSPARQL also includes a temporal dimension.

Although a lot of geospatial (and some temporal) RDF stores started to emerge, converting geospatial data into RDF and then storing it into an RDF stores is not always best practice, especially when the data exists in a relational database that is fairly large and/or it gets updated frequently.

In this thesis, we propose an Ontology-based Data Access (OBDA) approach for accessing geospatial data stored in geospatial relational databases, using the OGC standard GeoSPARQL and R2RML or OBDA mappings. We introduce extensions to an existing SPARQL-to-SQL translation method to support GeoSPARQL features. We describe the implementation of our approach in the system Ontop-spatial, an extension of the OBDA system Ontop for creating virtual geospatial RDF graphs on top of geospatial relational databases. Ontop-spatial is the first geospatial OBDA system and outperforms state-of-the-art geospatial RDF stores. We also show how to answer queries with temporal operators in the OBDA framework, by utilizing the framework stRDF and the query language stSPARQL which we extend with some new features. Next, we extend the data sources supported by Ontop-spatial going beyond relational database management systems, and we present our OBDA solutions for creating virtual RDF graphs on top of various Web data sources (e.g., HTML tables, Web APIs) using ontologies and mappings. We compared the performance of our approach with a related implementation and evaluation results showed that not only does Ontop-spatial support more functionalities (e.g., more data sources, more simple workflow), but it also achieves better performance. Last, we describe how the work described in this thesis is applied in real-world application scenarios.

SUBJECT AREA: Geospatial and Temporal Knoweledge Bases

KEYWORDS: Linked spatiotemporal data, Spatiotemporal databases, Semantic Web

ΠΕΡΙΛΗΨΗ

Το μοντέλο δεδομένων RDF και η γλώσσα επερωτήσεων SPARQL είναι ευρέως διαδεδομένα για την χρήση τους με σκοπό την ενοποίηση πληροφορίας που προέρχεται από διαφορετικές πηγές. Ο αυξανόμενος αριθμός των γεωχωρικών συνόλων δεδομένων που είναι πλέον διαθέσιμα σαν γεωχωρικά διασυνδεδεμένα δεδομένα οδήγησε στην εμφάνιση επεκτάσεων του μοντέλου δεδομένων RDF και της γλώσσας επερωτήσεων SPARQL. Δύο από τις σημαντικότερες επεκτάσεις αυτές είναι η γλώσσα GeoSPARQL, η οποία έγινε OGC πρότυπο, και το πλαίσιο του μοντέλου δεδομένων stRDF και της γλώσσας επερωτήσεων stSPARQL. Και οι δύο προσεγγίσεις μπορούν να χρησιμοποιηθούν για την αναπαράσταση και επερώτηση διασυνδεδεμένων γεωχωρικών δεδομένων, ενώ το μοντέλο stRDF και η γλώσσα stSPARQL παρέχουν επίσης επιπλέον λειτουργικότητα για την αναπαράσταση και επερώτηση χρονικών δεδομένων.

Παρότι ο αριθμός των δεδομένων που είναι διαθέσιμα σαν γεωχωρικά ή και χρονικά διασυνδεδεμένα δεδομένα αυξάνεται, η μετατροπή των γεωχωρικών δεδομένων σε RDF και η αποθήκευσή τους σε αποθετήρια RDF δεν είναι πάντα η βέλτιστη λύση, ειδικά όταν τα δεδομένα βρίσκονται εξαρχής σε σχεσιακές βάσεις οι οποίες μπορεί να έχουν αρκετά μεγάλο μέγεθος ή και να ενημερώνονται πολύ συχνά.

Στα πλαίσια αυτής της διδακτορικής διατριβής, προτείνουμε μια λύση βασισμένη στην ανάκτηση πληροφορίας με χρήση οντολογιών και αντιστοιχίσεων για την επερώτηση δεδομένων πάνω από γεωχωρικές σχεσιακές βάσεις δεδομένων. Επεκτείνουμε τεχνικές επανεγγραφής GeoSPARQL ερωτημάτων σε SQL ώστε η αποτίμηση των επερωτήσεων να γίνεται εξολοκλήρου στο γεωχωρικό σύστημα διαχείρισης βάσεων δεδομένων. Επίσης, εισάγουμε επιπλέον λειτουργικότητα στη χρονική συνιστώσα του μοντέλου δεδομένων stRDF και της γλώσσας επερωτήσεων stSPARQL, προκειμένου να διευκολυνθεί η υποστήριξη χρονικών τελεστών σε συστήματα OBDA. Στη συνέχεια, επεκτείνουμε τις παραπάνω μεθόδους με την υποστήριξη διαφορετικών πηγών δεδομένων πέρα από σχεσιακές βάσεις και παρουσιάζουμε μια OBDA προσέγγιση που επιτρέπει τη δημιουργία εικονικών RDF γράφων πάνω από δεδομένα που βρίσκονται διαθέσιμα στο διαδίκτυο σε διάφορες μορφές (πχ. HTML πίνακες, web διεπαφές), με χρήση οντολογιών και αντιστοιχίσεων. Συγκρίναμε την απόδοση του συστήματός μας με ένα σχετικό σύστημα και τα αποτελέσματα έδειξαν ότι πέραν του ότι το σύστημά μας παρέχει μεγαλύτερη λειτουργικότητα (πχ. υποστηρίζει περισσότερα είδη πηγών δεδομένων, περιλαμβάνει απλούστερες διαδικασίες) και εξασφαλίζει καλύτερη απόδοση. Τέλος, παρουσιάζουμε την εφαρμογή των μεθόδων και συστημάτων που περιγράφονται στη διατριβή σε πραγματικά σενάρια χρήσης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Γεωχωρικές και Χρονικές Βάσεις Γνώσεων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Διασυνδεδεμένα χωροχρονικά δεδομένα, Χωροχρονικές βάσεις δεδομένων, Σημασιολογικός Ιστός

ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΗΣ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

Εισαγωγή

Ο όγκος των γεωχωρικών και χρονικών δεδομένων που παράγονται και καταναλώνονται από εφαρμογές αυξάνεται ραγδαία τα τελευταία χρόνια, καθώς ολοένα και περισσότερες εφαρμογές χρησιμοποιούν γεωχωρικά δεδομένα που παράγονται στα πλαίσια διαφόρων επιστημονικών πεδίων (πχ. τηλεπισκόπηση, περιβαλλοντικές επιστήμες, γεωλογία), ενώ πολλές εφαρμογές και υπηρεσίες (πχ. Google maps), στηρίζονται στην αποτελεσματική ενοποίηση και διαχείριση γεωχωρικών δεδομένων από διαφορετικές πηγές.

Το πρόβλημα της διαχείρισης και αποτελεσματικής ανάκτησης γεωχωρικών δεδομένων έχει απασχολήσει την ερευνητική κοινότητα μερικές δεκαετίες και αρκετές διεπιστημονικές ομάδες έχουν αναπτύξει μοντέλα δεδομένων και συστήματα που στοχεύουν στην αποτελεσματική διαχείριση γεωχωρικών και χρονικών δεδομένων. Στον χώρο των βάσεων δεδομένων, το σχεσιακό μοντέλο επεκτάθηκε με τη δυνατότητα να αναπαριστά γεωχωρικά δεδομένα (πχ. γεωμετρίες) και χρονικά δεδομένα, ενώ αντίστοιχα η γλώσσα επερωτήσεων SQL επεκτάθηκε με γεωχωρικούς και χρονικούς τελεστές. Οι προσπάθειες αυτές οδήγησαν στη δημιουργία μιας πληθώρας συστημάτων διαχείρισης βάσεων δεδομένων που υποστηρίζουν χωρικούς και χρονικούς τελεστές (πχ. PostGIS και PostgreSQL-temporal, Spatialite, Oracle Spatial and Graph), ενώ μέχρι και σήμερα η βιβλιογραφία βρίθει τεχνικών βελτιστοποίησης γεωχωρικών και χρονικών ερωτημάτων.

Παράλληλα, υπάρχουν καθιερωμένα Γεωχωρικά Πληροφοριακά Συστήματα (GIS), όπως είναι το εμπορικό σύστημα ArcGIS και το αντίστοιχο σύστημα ανοικτού κώδικα QGIS, τα οποία χρησιμοποιούνται σε επιχειρησιακό επίπεδο για καθημερινές εργασίες που αφορούν την επεξεργασία και την οπτικοποίηση γεωχωρικών συστημάτων. Στον τομέα της Τεχνητής Νοημοσύνης υπάρχουν επίσης εργασίες που επικεντρώνονται στην εφαρμογή μεθόδων συμπερασμού σε γεωχωρικά και χρονικά δεδομένα με σκοπό την εξαγωγή συμπερασμάτων από την υπάρχουσα γεωχωρική και χρονική πληροφορία, αντίστοιχα.

Προϋπόθεση για την ενοποίηση δεδομένων από διαφορετικές πηγές είναι τα δεδομένα να είναι σε ένα κοινό μοντέλο δεδομένων. Για το σκοπό αυτό, συχνά χρησιμοποιείται το μοντέλο δεδομένων RDF [51]. Η κεντρική ιδέα του μοντέλου δεδομένων RDF είναι ότι κάθε οντότητα στο διαδίκτυο (και όχι μόνο) είναι ένας πόρος, στον οποίο ανατίθεται ένα μοναδικό προσδιοριστικό (URI). Ο πόρος αυτός μπορεί να περιγραφεί με τη χρήση τριπλετών, δηλαδή δηλώσεων που αποτελούνται από τρία μέρη:Το *υποκείμενο*, που είναι ο πόρος που θέλουμε να περιγράψουμε, το *κατηγόρημα*, και το *αντικείμενο*. Το κατηγόρημα περιγράφει μια ιδιότητα του υποκειμένου, και είναι και αυτό ένας πόρος που μπορεί να περιγραφεί, ενώ το αντικείμενο είναι η τιμή της ιδιότητας που περιγράφει το κατηγόρημα. Η τιμή αυτή μπορεί να είναι είτε ένας άλλος πόρος (URI), είτε ένα λεκτικό ενός τύπου δεδομένων (πχ. αριθμός, συμβολοσειρά, ημερομηνία). Το μοντέλο δεδομένων RDF χρησιμοποιείται για να διευκολύνει τη συναλλαγή πληροφορίας μεταξύ εφαρμογών,

καθώς εισάγει σημασιολογία στα δεδομένα.

Το σχήμα των δεδομένων που είναι εκφρασμένα στο μοντέλο RDF δηλώνεται με τη χρήση οντολογιών. Οι οντολογίες περιγράφουν τα είδη των οντοτήτων που μπορεί να υπάρχουν σε ένα σύνολο δεδομένων, δηλαδή τις *κλάσεις*, τις σχέσεις μεταξύ των κλάσεων αυτών, τα κατηγορήματα και τη σχέση μεταξύ των κατηγορημάτων, ενώ μπορεί να περιλαμβάνουν και αξιώματα που εισάγουν πρόσθετους ορισμούς και περιορισμούς στα δεδομένα που μπορούν να χρησιμοποιηθούν για την εξαγωγή νέας πληροφορίας από την υπάρχουσα με χρηση κατάλληλων κανόνων συμπερασμού. Η γλώσσσα που χρησιμοποιείται για τη διατύπωση επερωτήσεων σε δεδομένα RDF είναι η γλώσσα SPARQL [40], η οποία είναι μια δηλωτική γλώσσα που μοιράζεται αρκετά κοινά στοιχεία με τη γλώσσα επερωτήσεων SQL.

Το μοντέλο RDF Και η γλώσσα επερωτήσεων SPARQL δεν υποστηρίζουν ειδικούς τύπους δεδομένων και τελεστές για την αναπαράσταση και επερώτηση γεωχωρικών και χρονικών δεδομένων. Για το λόγο αυτό, τα τελευταία χρόνια άρχισαν να προτείνονται οι πρώτες γεωχωρικές και χρονικές επεκτάσεις τους. Στη συνέχεια θα αναφερθούμε στις δύο κυριότερες επεκτάσεις του μοντέλου RDF και της γλωσσας SPARQL που είναι η γλώσσα GeoSPARQL [26] και η γλώσσα stSPARQL [47].

Η γλώσσα GeoSPARQL είναι μια επέκταση του μοντέλου RDF και της γλώσσας SPARQL η οποία προτάθηκε για την υποστήριξη νεών τύπων δεδομένων και τελεστών για την αναπαράσταση και ανάκτηση γεωχωρικών δεδομένων και προτυποποιήθηκε από το Open Geospatial Consortium (OGC). Η GeoSPARQL επεκτείνει το μοντέλο δεδομένων RDF εισάγοντας δύο νέους τύπους δεδομένων για την αναπαράσταση γεωμετριών σαν λετικά (literals). Αυτοί οι τύποι δεδομένων είναι ο τύπος δεδομένων Well-Known-Text (WKT), και ο τύπος δεδομένων Geography Markup Language (GML). Οι τύποι δεδομένων WKT και GML αποτελούν ήδη πρότυπα OGC για την αναπαράσταση γεωμετριών υπό τη μορφή κειμένου. Η γλώσσα GeoSPARQL επίσης ορίζει κατηγορήματα που αφορούν τοπολογικές σχέσεις μεταξύ πόρων που έχουν γεωμετρία ή απευθείας μεταξύ γεωμετριών, ενώ επίσης ορίζει και ένα σύνολο από τελεστές που μπορούν να χρησιμοποιηθούν σαν συναρτήσεις που επεκτείνουν την SPARQL 1.1. Οι συναρτήσεις αυτές λαμβάνουν σαν είσοδο μία ή δύο γεωμετρίες, υπολογίζουν γεωμετρικές ιδιότητες (πχ. εμβαδό) ή τοπολογικές σχέσεις (πχ. επικάλυψη δύο γεωμετριών), και μπορούν να περιληφθούν σε φίλτρα ή στις δηλώσεις προβολής (SELECT) των SPARQL ερωτημάτων.

Το μοντέλο δεδομένων stRDF και η γλώσσα επερωτήσεων stSPARQL είναι άλλη μια επέκταση του μοντέλου RDF και της γλώσσας SPARQL με τύπους δεδομένων και τελεστές για την αναπαράσταση και επερώτηση γεωχωρικών και χρονικών δεδομένων. Το μοντέλο RDF και η γλώσσα stSPARQL αναπτύχθηκαν την ίδια χρονική περίοδο αλλά ανεξάρτητα από τη γλωσσα GeoSPARQL, και μοιράζονται αρκετά κοινά στοιχεία. Η γλωσσα stSPARQL προβλέπει επίσης την αναπαράσταση γεωμετριών σαν λεκτικά εκφρασμένα στα πρότυπα WKT και GML, και προτείνει τύπους δεδομένων που βασίζονται στα πρότυπα αυτά. Στην stSPARQL ορίζονται επίσης αντίστοιχοι τελεστές με την GeoSPARQL, με τη διαφορά ότι ορίζονται επιπλέον και συναθροιστικές συναρτήσεις, δηλαδή συναρτήσεις που παίρνουν σαν είσοδο μία λίστα από γεωμετρίες που είναι λύσεις στο κύριο μέρος ενός stSPARQL ερωτήματος, κάνουν έναν υπολογισμό (πχ. ένωση όλων

των γεωμετριών), και προβάλλουν το αποτέλεσμα.

Μία άλλη σημαντική διαφορά ανάμεσα στις γλώσσες GeoSPARQL και stSPARQL είναι ότι το μοντέλο stRDF και η γλωσα stSPARQL περιλαμβάνουν και χρονικές επεκτάσεις στο πλαίσιο των RDF και SPARQL, ενώ η γλώσσα GeoSPARQL δεν παρέχει επιπλέον δυνατότητες για χρονικά δεδομένα. Ειδικότερα, χρησιμοποιώντας το μοντέλο stRDF μπορεί κανείς να μοντελοποιήσει τον *χρόνο εγκυρότητας* τριπλετών, δηλαδή τον χρόνο κατά τον οποίο μια τριπλέτα είναι έγκυρη. Πέραν του χρόνου εγκυρότητας όμως, ορίζονται και πιο γενικοί χρονικοί τελεστές που λαμβάνουν σαν είσοδο χρονικές περιόδους ή χρονικά σημεία (timestamps) και κάνουν αντίστοιχους υπολογισμούς.

Πολύ σύντομα μετά την προτυποίηση της γλώσσας GeoSPARQL, άρχισαν να αναπτύσσονται τα πρώτα συστήματα ή επεκτάσεις συστημάτων που την υλοποιούν, όπως είναι τα συστήματα: GraphDB[1], Oracle Spatial and Graph[2], USeekM[3], ενώ άλλα συστήματα υποστηρίζουν δικούς τους ειδικούς τελεστές για την αποθήκευση και επερώτηση γεωχωρικών δεδομένων. Το σύστημα Strabon[4] [48] είναι το πιο αποδοτικό από αυτά τα συστήματα, σύμφωνα με πειραματικές μελέτες [34] αλλά και το πληρέστερο, καθώς υποστηρίζει τόσο τη γλώσσα GeoSPARQL όσο και τη γλώσσα stSPARQL.

Συστήματα ανάκτησης δεδομένων με οντολογίες

Παρόλο που το μοντέλο RDF χρησιμοποιείται ευρέως από την ακαδημαϊκή κοινότητα και από εμπορικές εφαρμογές για την ενοποίηση δεδομένων από διαφορετκές πηγές, πολλοί χρήστες που έχουν τα δεδομένα τους σε σχεσιακές βάσεις πολλές φορές δεν επιθυμούν να μεταφράζουν τα δεδομένα τους σε RDF για να τα ενώσουν με άλλα δεδομένα και να εκμεταλλευθούν τα οφέλη των διασυνδεδεμένων δεδομένων. Ειδικά στις περιπτώσεις που οι βάσεις αυτές περιέχουν μεγάλο όγκο δεδομένων ή δεδομένα που ενημερώνονται συχνά, οι χρήστες αποθαρρύνονται από το να πραγματοποιούν μετατροπές των δεδομένων τους σε RDF κάθε φορά που οι βάσεις ενημερώνονται προκειμένουν να συντηρούν ενημερωμένα τα RDF αποθετήριά τους. Επίσης, συχνά τα αποθετήρια RDF δεν είναι το ίδιο αποδοτικά με τις σχεσιακές βάσεις δεδομένων.

Προκειμένου να αντιμετωπιστούν τα προβλήματα αυτά, δημιουργήθηκε η ερευνητική περιοχή της ανάκτησης δεδομένων με οντολογίες [75]. Η περιοχή αυτή μελετά το πρόβλημα της δημιουργίας εικονικών RDF τριπλετών πάνω από σχεσιακές βάσεις δεδομένων, χωρίς να μετατρέπονται τα δεδομένα των βάσεων εξαρχής σε RDF και να αποθηκεύονται υπό την μορφή RDF αρχείων σε κάποιο RDF αποθετήριο. Χρησιμοποιώντας τις τεχνικές αυτές, οι χρήστες μπορούν να διατυπώνουν SPARQL ερωτήματα πάνω από τους εικονικούς RDF γράφους, τα οποία μετά επανεγγράφονται αυτόματα στα αντίστοιχα SQL ερωτήματα που αποτιμώνται στο αντίστοιχο σύστημα

[1] https://www.ontotext.com/

[2] https://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html

[3] https://www.w3.org/2001/sw/wiki/USeekM

[4] http://www.strabon.di.uoa.gr/

διαχείρισης βάσεων δεδομένων και τα αποτελέσματα επιστρέφονται στον χρήστη σαν εικονικοί RDF όροι, ακριβώς με τον ίδιο τρόπο με τον οποίο κανείς θα έθετε τα ίδια ερωτήματα σε ένα αποθετήριο RDF.

Για να δημιουργήσει κανείς ένα εικονικό αποθετήριο σε ένα σύστημα ανάκτησης δεδομένων με οντολογίες, πρέπει να παρέχει την οντολογία που μοντελοποιεί τα δεδομένα σαν είσοδο στο σύστημα, καθώς και ένα αρχείο αντιστοιχίσεων (mapping file). Στο αρχείο αυτό δηλώνονται οι RDF όροι που αντιστοιχίζονται σε κάθε στοιχείο του σχεσιακού μοντέλου, για κάθε σύνολο από πλειάδες των πινάκων της βάσης που θέλουμε να αντιστοιχίσουμε σε εικονικούς RDF όρους. Για το σκοπό αυτό, υπάρχουν διάφορες γλώσσες αντιστοιχίσεων, καθώς κάθε σύστημα ανάκτησης δεδομένων με οντολογίες που αναπτυσσόταν αρχικά υποστήριζε τη δική του γλώσσα. Καθώς όμως αναπτύχθηκαν αρκετά τέτοια συστήματα, η γλώσσα επερωτήσεων R2RML προτυποποιήθηκε από το W3C [29].

Αρκετά συστήματα που υλοποιούν τις τεχνικές ανάκτησης δεδομένων πάνω από οντολογίες πραγματοποιώντας αυτόματη SPARQL-σε-SQL επανεγγραφή χρησιμοποιώντας οντολογίες και αντιστοιχίσεις. Ωστόσο, κανένα από αυτά δεν περιελάμβανε υποστήριξη για γεωχωρικά και χρονικά δεδομένα, πριν τη δημιουργία του συστήματος το οποίο περιγράφεται στην παρούσα διατριβή.

Συνεισφορά διδακτορικής διατριβής

Οι συνεισφορές της παρούσας διδακτορικής διατριβής, είναι οι ακόλουθες:

- Περιγράφουμε δημιουργία του πρώτου συστήματος επεξεργασίας ερωτημάτων GeoSPARQL πάνω από σχεσιακές βάσεις δεδομένων, επεκτείνοντας τεχνικές επανεγγραφής SPARQL ερωτημάτων σε SQL ερωτήματα, με χρήση οντολογιών και αντιστοιχίσεις σχεσιακών όρων σε εικονικούς όρους RDF. Οι τεχνικές μας υλοποιούνται σαν επέκταση του συστήματος Ontop, το οποίο ονομάζουμε Ontop-spatial[5] [10].

- Παρουσιάζουμε τα αποτελέσματα της διεξαγωγής πειραματικής μελέτης που συγκρίνει το Ontop-spatial με παραδοσιακά αποθετήρια που περιλαμβάνουν υποστήριξη για γεωχωρικά δεδομένα. Τα αποτελέσματα των πειραμάτων έδειξαν ότι το σύστημά Ontop-spatial επιτυγχάνει καλύτερους χρόνους απόκρισης από το καλύτερο χωροχρονικό RDF αποθετήριο συχνά κατά δύο τάξεις μεγέθους.

- Επεκτείνουμε το μοντέλο δεδομένων RDF και τη γλώσσα επερωτήσεων stSPARQL με χαρακτηριστικά που διευκολύνουν την υποστήριξη χρονικών τελεστών σε συστήματα ανάκτησης δεδομένων με οντολογίες.

- Προτείνουμε τεχνικές για τη δημιουργία εικονικών RDF γράφων πάνω από πηγές δεδομένων διαθέσιμες στο διαδίκτυο, πέρα από σχεσιακές βάσεις δεδομένων,

[5] http://ontop-spatial.di.uoa.gr/

για παράδειγμα HTML πίνακες ή Web διεπαφές. Περιγράφουμε την υλοποίηση των τεχνικών αυτών σαν επέκταση του συστήματος Ontop-spatial και διεξάγουμε πειραματική μελέτη που συγκρίνει τη δική μας προσέγγιση με μια άλλη σχετική εργασία. Τα αποτελέσματα των πειραμάτων δείχνουν ότι η δική μας προσέγγιση επιτυγχάνει καλύτερους χρόνους απόκρισης σε συνδυασμό με καλύτερη λειτουργικότητα.

- Παρουσιάζουμε ενδεικτικές περιπτώσεις χρήσης του συστήματος σε πραγματικά σενάρια εφαρμογών. Πολύ σύντομα μετά τη δημιουργία του συστήματος ενσωματώθηκε σε ένα ευρύ φάσμα εφαρμογών που εντάσσονται σε διαφορετικούς τομείς, όπως είναι η διαχείριση γης, η αστική ανάπτυξη, θαλάσσια ασφάλεια, καθώς και δημιουργία διεπαφών για τα δεδομένα Copernicus.

Ανάκτηση γεωχωρικών δεδομένων με χρήση οντολογιών

Στα πλαίσια αυτής της διδακτορικής διατριβής παρουσιάζουμε την πρώτη προσέγγιση υλοποίησης συστήματος που απαντά GeoSPARQL ερωτήματα πάνω από γεωχωρικές βάσεις δεδομένων επανεγγράφοντας τα ερωτήματα αυτά αυτόματα στα αντίστοιχα ερωτήματα SQL. Υλοποιούμε την προσέγγισή μας σαν γεωχωρική επέκταση του συστήματος Ontop, που είναι ένα σύστημα ανάκτησης δεδομένων με οντολογίες και αντιστοιχίσεις. Το σύστημα Ontop υλοποιεί έναν αλγόριθμο επαναγραφής SPARQL ερωτημάτων σε ερωτήματα SQL, τον οποίο επεκτείνουμε για την υποστήριξη γεωχωρικών ερωτημάτων. Το νέο σύστημα, το οποίο ονομάζουμε Ontop-spatial, αποτιμά GeoSPARQL ερωτήματα ως εξής:

- Αρχικά, ένα GeoSPARQL ερώτημα έχει υποβληθεί στο σύστημα και γίνεται έλεγχος συντακτικού. Έχουμε τροποποιήσει το συντακτικό δέντρο το οποίο χρησιμοποιούσε έτοιμο το σύστημα Ontop ώστε να αναγνωρίζονται οι τύποι δεδομένων και οι τελεστές της γλώσσας GeoSPARQL.
- Έπειτα το σύστημα Ontop μετατρέπει τα SPARQL ερωτήματα σε ένα πρόγραμμα datalog.
- Οι αντιστοιχίσεις μετατρέπονται και αυτές σε datalog και προστίθενται στο πρόγραμμα ώστε να γίνουν βελτιστοποιήσεις (πχ., απλοποίηση, ανίχνευση κενών ερωτημάτων πριν φτάσουν στο συστημα διαχείρισης βάσεων δεδομένων). Για το σκοπό αυτό, έχουμε επεκτείνει τον μετατροπέα επαναγραφής SPARQL ερωτημάτων σε datalog, ορίζοντας γεωχωρικά κατηγορήματα datalog. Κάθε κατηγόρημα datalog αντιστοιχεί σε καθέναν από τους γεωχωρικούς τελεστές που ορίζονται στη γλώσσα GeoSPARQL. Σε αυτό το στάδιο υλοποιούνται επίσης οι κανόνες επαναγραφής *ποιοτικών* επερωτήσεων GeoSPARQL σε *ποσοτικές* επερωτήσεις GeoSPARQL, όπως ορίζει η επέκταση με τίτλος “Query rewrite” του προτύπου.
- Στη συνέχεια, το datalog πρόγραμμα που έχει δημιουργηθεί μετατρέπεται σε μια επερώτηση SQL, η οποία προωθείται στο γεωχωρικό σύστημα διαχείρισης βάσεων

δεδομένων με το οποίο είναι συνδεδεμένο το σύστημα Ontop-spatial. Αφού αποτιμάται η επερώτηση, τα αποτελέσματα προωθούνται πίσω στο Ontop-spatial, όπου μετατρέπονται σε εικονικοί RDF όροι, σύμφωνα με την οντολογία και τις αντιστοιχίσεις, και τέλος επιστρέφονται στον χρήστη.

Πραγματοποιήσαμε πειραματική μελέτη συγκρίνοντας το σύστημα Ontop-spatial, με δύο από τα πιο αποδοτικά αποθετήρια RDF που υποστηρίζουν επερωτήσεις GeoSPARQL, το σύστημα ανοιχτού λογισμικού Strabon και ένα εμπορικό σύστημα. Τα αποτελέσματα της πειραματική μελέτης δείχνουν ότι το σύστημα Ontop-spatial επιτυγχάνει καλύτερους χρόνους απόκρισης από τα άλλα δύο συστήματα, συχνά με δύο τάξεις μεγέθους διαφορά. Το περιεχόμενο αυτής της ενότητας αναλύεται περισσότερο στο Κεφάλαιο 3, καθώς και στη δημοσίευση [10].

Ανάκτηση χρονικών δεδομένων με χρήση οντολογιών

Την ίδια προσέγγιση με την ανάκτηση γεωχωρικών δεδομένων με χρήση οντολογιών και αντιστοιχίσεων θα μπορούσαμε να εφαρμόσουμε και για την ανάκτηση χρονικών δεδομένων με ερωτήματα SPARQL πάνω από εικονικούς χρονικούς γράφους RDF. Ωστόσο, στην περίπτωση των χρονικων δεδομένων υπάρχουν πρόσθετες προκλήσεις. Πρώτον, δεν υπάρχει προτυποποιημένη γλώσσα που να επεκτείνει τη γλώσσα SPARQL με χρονικούς τελεστές, σε αντίθεση με την GeoSPARQL που είναι πρότυπο. Δεύτερον, οι επεκτάσεις που έχουν προταθεί για μοντελοποίηση του χρόνου εγκυρότητας τριπλετών δεν ταιριάζουν στο μοντέλο της ανάκτησης δεδομένων που βασίζεται στη χρήση οντολογιών, καθώς απαιτεί λειτουργικότητα που δεν υποστηρίζεται από τις υπάρχουσες τεχνικές.

Για να διευκολύνουμε την υλοποίηση χρονικών τελεστών στα συστήματα ανάκτησης δεδομένων που βασίζονται σε οντολογίες, ορίζουμε μια επέκταση στη χρονική συνιστώσα του μοντέλου RDF και της γλώσσας stSPARQL, που περιλαμβάνει τα εξής:

- *Χρονικά κατηγορήματα*. Ορίζουμε ένα κατηγόρημα για κάθε χρονική συνάρτηση που ορίζεται στην stSPARQL και μπορεί να χρησιμοποιηθεί σαν χρονικό φίλτρο.
- *Επανεγγραφή χρονικών τελεστών*. Ορίζουμε ένα σύνολο από κανόνες που επανεγγράφουν ένα ερώτημα που περιέχει ένα χρόνικό κατηγόρημα, σε ένα ισοδύναμο ερώτημα που περιέχει την αντίστοιχη χρονική συνάρτηση σε φίλτρο.

Η σχεδιαστική απόφαση στην οποία βασίζεται η παραπάνω απόφαση είναι να αποφευχθεί η επέκταση του συντακτικού που υποστηρίζεται από ένα σύστημα ανάκτησης δεδομένων με οντολογίες με ένα μη προτυποποιημένο συντακτικό. Αντ' αυτού, χρησιμοποιούνται χρονικα κατηγορήματα, τα οποία, αν και για εμας έχουν ειδική σημασιολογία η οποία συμπληρώνεται από το τμήμα της επανεγγραφής χρονικών τελεστών, δεν παραβιάζουν το πρότυπο του μοντέλου δεδομένων RDF και το πρότυπο συντακτικό της γλώσσας SPARQL. Έτσι, η διαδικασία αποτίμησης ενός χρονικό ερωτήματος stSPARQL (δηλαδή, ερωτήματος που περιλαμβάνει ένα χρονικό κατηγόρημα) περιγράφεται ως εξής:

- Το ερώτημα περνά από συντακτικό έλεγχο χωρίς διαφοροποίηση καθώς πρόκειται για ένα ερώτημα που δεν παραβιάζει το πρότυπο της γλώσσας SPARQL

- Το ερώτημα μεταφράζεται σε ένα πρόγραμμα datalog. Ορίζουμε ειδικά χρονικά κατηγορήματα που αντιστοιχούν σε κάθε χρονικό κατηγόρημα stSPARQL, και άρα σε κάθε χρονική συνάρτηση stSPARQL.

- Στο επίπεδο της datalog υλοποιείται το τμήμα της stSPARQL για μετατροπή των χρονικών ερωτημάτων που περιλαμβάνουν χρονικό κατηγόρημα σε χρονικά ερωτήματα που περιλαμβάνουν την αντίστοιχη χρονική συνάρτηση.

- Το τελικό πρόγραμμα datalog μετασχηματίζεται σε μία SQL επερώτηση η οποία αποτιμάται τελικά στο σύστημα διαχείρισης βάσεων δεδομένων που βρίσκονται τα χρονικά δεδομένα. Η επερώτηση αυτή περιλαμβάνει τον χρονικό τελεστή που αντιστοιχεί στον χρονικό datalog τελεστή.

- Τα αποτελέσματα επιστρέφονται στον χρήστη.

Το περιεχόμενο της ενότητας αυτής παρουσιάζεται αναλυτικά στο Κεφάλαιο 4.

Ανάκτηση δεδομένων από το Web με χρήση οντολογιών

Αφού προτείναμε τις δικές μας λύσεις για την ανάκτηση χωρικών και χρονικών σχεσιακών δεδομένων χρησιμοποιώντας οντολογίες και αντιστοιχίσεις, το επόμενο πρόβλημα ήταν να επιχειρήσουμε να επεκτείνουμε το σύστημά μας με τη δυνατότητα να επεξεργάζεται ερωτήματα σε εικονικούς RDF γράφους που δημιουργούνται πάνω από δεδομένα τα οποία δεν ανήκουν τοπικά σε κάποια βάση δεδομένων, αλλά υπάρχουν στο διαδίκτυο σε διάφορες μορφές, όπως πίνακες HTML και Web διεπαφές.

Το πρόβλημα αυτό είναι ιδιαιτέρως απαιτητικό καθώς το σχήμα των δεδομένων που είναι διαθέσιμα στις μορφές αυτές δεν είναι συγκεκριμένο (όπως είναι το σχεσιακό σχήμα).

Η πρότασή μας για την επίλυση αυτού του προβλήματος βασίζεται στην χρήση ενός συστήματος ροής δεδομένων που παρέχει SQL διεπαφή. Χρησιμοποιώντας αυτό το σύστημα, το οποίο ονομάζεται MadIS, μπορεί κανείς να δημιουργήσει συναρτήσεις που δημιουργούν εικονικούς σχεσιακούς πίνακες. Οι συναρτήσεις αυτές περιέχονται σε SQL ερωτήματα και όταν κληθούν συμπεριφέρονται σαν κανονικοί σχεσιακοί πίνακες, με τη διαφορά ότι ανακτούν τα δεδομένα κατευθείαν από την πηγή και τα κάνουν διαθέσιμα σαν εικονικούς σχεσιακούς πίνακες. Σημειώνεται ότι σε κανένα όμως βήμα της επεξεργασίας τα δεδομένα αυτά δεν αποθηκεύονται.

Επεκτείναμε λοιπόν το σύστημα MadIS με συναρτήσεις που δημιουργούν εικονικούς πίνακες ανακτώντας δεδομένα από Web APIs, όπως είναι για παράδειγμα οι διεπαφές που υλοποιούν το πρωτόκολλο OPeNDAP για την ανάκτηση γεωχωρικών δεδομένων στο διαδίκτυο. Επίσης, υλοποιήσαμε τις ακόλουθες επεκτάσεις στο σύστημα Ontop-spatial:

- Υλοποιήσαμε διεπαφή που συνδέει το σύστημα Ontop-spatial με το σύστημα MadIS, ώστε το δεύτερο να μπορεί να λειτουργήσει σαν σύστημα διαχείρισης δεδομένων.
- Τροποποιήσαμε το Ontop-spatial ώστε να διαφοροποιείται στην περίπτωση που τα δεδομένα δεν βρίσκονται τοπικά σε ένα σύστημα βάσεων δεδομένων. Συγκεκριμένα, το Ontop συνδέεται με την υπάρχουσα βάση δεδομένων όπως ορίζεται από το αρχείο αντιστοιχήσεων, προκειμένου να πραγματοποιήσει προεργασία των δεδομένων που αναμένεται να εμπλακούν στα ερωτήματα (πχ. συλλογή στατιστικών). Αυτό δεν είναι δυνατόν να γίνει στην περίπτωσή μας, καθώς επικοινωνούμε με την πηγή δεδομένων μετά την υποβολή του SPARQL ερωτήματος.

Σημειώνεται ότι τα SQL ερωτήματα που περιέχουν τους τελεστές δημιουργίας εικονικών σχεσιακών πινάκων περιέχονται στις αντιστοιχίσεις που περιλαμβάνονται στο αρχείο αντιστοιχίσεων.

Έτσι, όταν υποβάλεται ένα ερώτημα στο σύστημα Ontop-spatial, πραγματοποιείται η ακόλουθη διαδικασία:

- Το ερώτημα μετατρέπεται σε datalog, όπως περιγράψαμε παραπάνω. Οι αντιστοιχίσεις περιλαμβάνονται και αυτές στο τελικό datalog πρόγραμμα.
- Το datalog πρόγραμμα μετατρέπεται στο τελικό SQL ερώτημα που αποτιμάται στο σύστημα MadIS. Αν στο ερώτημα περιλαμβάνεται τελεστής δημιουργίας εικονικών πινάκων, ο τελεστής καλείται *την ώρα της αποτίμησης του SQL ερωτήματος* και προωθεί τα ενδιάμεσα απότελέσματα στον τελεστή που έπεται κατά το πλάνο εκτέλεσης του ερωτήματος στο σύστημα MadIS. Η απόφαση αυτή έχει σαν συνέπεια (i) να ανακτάται πάντα η τελευταία έκδοση των δεδομένων από την πηγή δεδομένων χωρίς να υπάρχουν προβλήματα συγχρονισμού, και (ii) τα δεδομένα ανακτώνται από το σύστημα που πραγματοποιεί τελικά την αποτίμηση του ερωτήματος κάτι που ελαχιστοποιεί το χρόνο διακίνησης δεδομένων.

Εφαρμόσαμε την τεχνική που υλοποιήσαμε σε πραγματικές περιπτώσεις χρήσης, χρησιμοποιώντας τις ακόλουθες διεπαφές σαν πηγές δεδομένων: Twitter, Foursquare, Yelp, OPeNDAP server, καθώς και HTML πίνακες. Διεξάγαμε πειραματική μελέτη και συγκρίναμε τα αποτελέσματα με μια άλλη υλοποίηση που έχει αναπτυχθεί σε μια σχετική εργασία και υποστηρίζει παρόμοια λειτουργικότητα με τη δική μας προσέγγιση (αν και αρκετά περιορισμένη). Τα ευρύματα της πειραματικής μελέτης έδειξαν ότι η προσέγγιση μας, εκτός του ότι παρέχει μεγαλύτερη λειτουργικότητα και είναι πιο εύκολη στη χρήση, υποστηρίζοντας περισσότερες πηγές δεδομένων, είναι και πιο αποδοτική.

Το περιεχόμενο αυτής της ενότητας αναλύεται με λεπτομέρειες στο Κεφάλαιο 5.

Εφαρμογές

Το σύστημα Ontop-spatial, που αποτελεί το κύριο παράγωγο αυτής της διδακτορικής διατριβής εφαρμόστηκε σε πολλά πραγματικά σενάρια χρήσης προερχόμενα από

διαφορετικούς τομείς. Αφορμή της δημιουργίας του αποτέλεσε αρχικά το έργο OPTIQUE, και συγκεκριμένα η περίπτωση χρήσης της εταιρίας Statoil, στα πλαίσια της οποίας το σύστημα χρησιμοποιήθηκε για την ανάκτηση γεωχωρικών δεδομένων από μεγάλες σχεσιακές γεωχωρικές βάσεις χρησιμοποιώντας οντολογίες. Πολύ σύντομα μετά τη δημιουργία του συστήματος, εφαρμόστηκε επίσης στα πλαίσια του έργου MELODIES σε συνεργασία με τις εταιρίες VISTA και GISAT σε εφαρμογές διαχείρισης γης και αστικής ανάπτυξης αντίστοιχα.

Σε συνεργασία με την εταιρία AIRBUS και συγκεκριμένα με το τμήμα Άμυνας και Διαστήματος (Defence and Space), το σύστημα χρησιμοποιήθηκε σαν κομμάτι εφαρμογής με στόχο την ανάκτηση δεδομένων που περιγράφουν πλοία καθώς και την τρέχουσα θέση και κατάστασή τους. Αρχικά, τα δεδομένα αυτά αποθηκεύονται σε σχεσιακή γεωχωρική βάση δεδομένων. Η δυναμική φύση των δεδομένων (πχ. θέσεις πλοίων) αυτών καθιστά συχνές τις ενημερώσεις της βάσης με νέα δεδομένα συνεπώς μια λύση δημιουργίας εικονικών γράφων RDF πάνω από τα δεδομένα αυτά είναι προτιμότερη από την μετατροπή και αποθήκευση των δεδομένων σε RDF κάθε φορά που αυτά ενημερώνονται.

Μια ακόμα επέκταση και χρήση του συστήματος Ontop-spatial έγινε στα πλαίσια του έργου Copernicus App Lab. Το έργο είχε σαν στόχο την ανάκτηση δεδομένων Copernicus μέσω διεπαφών διαθέσιμων στο διαδίκτυο, σε μορφή που είναι εύκολα κατανοητή από χρήστες και προγραμματιστές εφαρμογών. Για το λόγο αυτό χρησιμοποιήθηκαν οντολογίες για την αναπαράσταση των δεδομένων, καθώς και το σύστημα Ontop-spatial. Το σύστημα Ontop-spatial επεκτάθηκε ώστε να μπορεί να υποστηρίζει δεδομένα που είναι διαθέσιμα χρησιμοποιώντας το πρότυπο OPeNDAP. Με τον τρόπο αυτό μπορεί κανείς να θέτει GeoSPARQL ερωτήματα με τον ίδιο τρόπο που θα έκανε σε ένα γεωχωρικό αποθετήριο RDF, με τη διαφορά οτι θα ερωτήματα αυτά εσωτερικά μετατρέπονται σε κλήσεις OPeNDAP και ανακτώνται κατευθείαν από τον αντίστοιχο εξυπηρετητή, και άρα στην πιο ενημερωμένη έκδοσή τους.

Το περιεχόμενο αυτής της ενότητας αναλύεται με λεπτομέρειες στο Κεφάλαιο 6.

Συμπεράσματα και μελλοντικές επεκτάσεις

Στην παρούσα διδακτορική διατριβή περιγράφονται τεχνικές για την αποτελεσματική ενοποίηση και επερώτηση γεωχωρικών και χρονικών δεδομένων. Δίνεται έμφαση στις τεχνικές ανάκτησης γεωχωρικών και χρονικών δεδομένων πάνω από γεωχρονικές βάσεις δεδομένων χρησιμοποιώντας οντολογίες. Περιγράφουμε το πρώτο σύστημα ανάκτησης γεωχωρικών δεδομένων χρησιμοποιώντας οντολογίες και προσαρμόζουμε τη μέθοδο αυτή και για την υποστήριξη χρονικών δεδομένων, επεκτείνοντας το μοντέλο δεδομένων stRDF και τη γλώσσα επερωτήσεων stSPARQL. Τέλος, επεκτείνουμε τις μεθόδους μας πέρα από τις σχεσιακές βάσεις και προτείνουμε τη δική μας προσέγγιση στο πρόβλημα της ανάκτησης πληροφορίας απευθείας από ετερογενείς πηγές που υπάρχουν στο διαδίκτυο με χρήση οντολογιών.

Επεκτάσεις των τεχνικών που περιγράφονται στη διατριβή αυτή είναι οι ακόλουθες: (i) κατανεμημένη επεξεργασία γεωχωρικών και χρονικών ερωτημάτων, (ii) υποστήριξη ειδικών τυπων δεδομένων και τεχνικών για την αναπαράσταση και επερώτηση δεδομένων σε μορφή Raster.

To my parents, Christos and Eufimia

ACKNOWLEDGEMENTS

Part of the work described in this PhD thesis was funded by the EU projects Optique (http://optique-project.eu/) and Copernicus App Lab (https://www.app-lab.eu/). Although the work described in this thesis started after my involvement in the project TELEIOS (http://www.earthobservatory.eu/), it gave me the background knowledge and motivation I needed for most of the work described in this thesis.

I would like to thank my advisor, professor Manolis Koubarakis, for giving me the opportunity to be a member of his research team, which is a leading team in linked geospatial data science. Being the advisor of my bachelor, master and now my PhD theses, he constantly challenges me to think and go beyond the state-of-the-art and he has unlimited eagerness to teach me how to properly write technical documents and make good presentations. Also, I will always appreciate his remarkable honesty and directness and his continuous encouragement to us, his students, to express our opinions and follow our own paths. I would also like to thank prof. Ioannidis and prof. Gunopoulos, for their useful feedback and guidance.

Joining the knowledge representation and reasoning group of the department was the most influential move I have done, not only for my career, but also for my personal development. Apart from my supervisor, I had the honour to collaborate with exceptional colleagues, and be part of a great, hard-working team that stayed remarkably bonded in good and bad times. My colleagues were my inspiration, and as I do not intend to mention all their names, I cannot but mention Kostis Kyzirakos, our “knowledge repository” whose personality, work ethics and attitude influenced my professional career and the entire group, although he graduated before I started my PhD.

I would also like to thank my close friends and my life companion, Manos, for putting up with me and the side effects of doing a PhD (e.g., emotional roller coasters and very limited free time). Manos is my number one fan, my anchor, and the person that gives my faith back when needed.

Last but not least, I thank my family; My father, Christos, my mother, Eufimia, and my brother, Yannis. Without their endless support all these years and their continuous encouragement and belief in me, nothing of what is described in this thesis would have been possible. But most importantly, I thank my parents for being my role models for working hard and fighting against all odds, and for their principle that defeat is stop trying. This thesis is devoted to them.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Due to recent technological advances (Satellite missions, location-aware search engines, mobile applications), the availability of geospatial data has been increased considerably. At the same time, the data model RDF is widely used in order to integrate data coming from different sources. The Linked Data paradigm allows for publishing data as linked open data, in order to exploit the semantic links between datasets, increasing the overall value of the data, as it allows users to pose rich queries over the combined, interlinked datasets. As a result, more and more geospatial datasets are becoming available as linked geospatial data.

Following to this trend, the first extensions of the framework of RDF and SPARQL appeared, highlighted by the establishment of GeoSPARQL, a geospatial extension of SPARQL, as an OGC standard. At the same time, the evolution of geospatial features that changes over time is equally important, which also led to the creation of temporal extensions of the data model RDF and the SPARQL query language. For example, the data model stRDF and the query language stSPARQL extend RDF and SPARQL respectively with spatial and temporal features.

At the moment, there is a wide variety of RDF stores that support the OGC standard GeoSPARQL. The state-of-the art of these systems is the open source spatiotemporal RDF store Strabon, that supports both GeoSPARQL and stSPARQL.

However, the following challenges remain in the area of linked geospatial and temporal data:

- Geospatial triple stores are not as efficient as geospatial databases (e.g., the PostGIS extension of PostgreSQL)
- There is no standard data model and query language for the representation and querying of linked temporal data
- Geospatial and temporal data is often stored in geospatially and/or temporally enabled geospatial databases. In the cases when the size of these databases is large, and/or when the databases get updated frequently, users are often discouraged to maintain semantic replicas of their original datasets by converting the data each time it gets updated and storing the new datasets in a triple store.
- A great part of data that is available on the Web is accessed via Rest APIs, or is available in other formats, such as HTML tables. In order to access this data and make it available as linked open data, one needs to access it through public APIs , parse it, convert it into RDF and store it in a triple store in order to query it combined with other data.

In the context of this thesis, we address the challenges described above.

1.1 Contributions

The contributions of the work described in this thesis are summarised as follows:

- We present the first approach of extending the OBDA paradigm with geospatial support, enabling the creation of virtual geospatial RDF graphs on top of geospatial relational databases. In this context, we describe the implementation of the geospatial extension of the OBDA system Ontop, which we named Ontop-spatial, that is able to process GeoSPARQL queries on top of geospatial relational data on-the-fly, using ontologies and mappings.
- We perform an experimental evaluation of the system Ontop-spatial using heavily geospatial workload and queries, and we compare its performance with two state-of-the-art RDF stores with GeoSPARQL support, an open source system and a commercial. The results of the experimental evaluation showed that Ontop-spatial outperforms geospatial RDF stores often by two orders of magnitude. In this way, Ontop-spatial is now considered as the most efficient GeoSPARQL query engine.
- We describe our approach for a temporally-enhanced OBDA system that is able to process temporal SPARQL queries on top of temporal databases. To achieve this, we introduce two more components in the temporal dimension of the data model RDF and the query language stSPARQL, that can be easily implemented in OBDA systems, in order to facilitate the implementations of temporal features in the OBDA paradigm. Ontop-spatial was extended in this direction with support for the temporal extension PostgreSQL.
- We extend further the OBDA paradigm in the direction of supporting more kinds of data sources, apart from relational databases, that can be found on the Web (e.g., HTML tables, Web APIs) and enable users to pose SPARQL queries on top of them, without fetching the data a priori and storing it locally. For this purpose, we extend SQL with virtual table operators that retrieve data from their original sources (e.g., Rest APIs, HTML tables, etc.) and make it available in tabular format, as virtual tables. We allow for the use of virtual table operators in R2RML mappings in the same way that static relational tables could be used embedded in SQL queries contained in the mappings. We implemented this approach as an extension of the system Ontop-spatial using the data flow system MadIS as back-end. MadIS is an SQLite python wrapper and can be used to implement virtual table operators on top of SQLite as python user-defined functions. We also implemented a caching mechanism that can be enabled as a virtual table operator parameter, so it can be configured at the mappings level. The caching mechanism is able to use cached data in the cases when the same API call needs to be made in a future query, until the data expire. The expiration time of the data is configured as a parameter in the virtual table operator.
- We compare our approach with related work both in terms of functionality and performance. The findings of this comparison are that our system not only offers more

functionality (e.g., less workflow steps, extensibility, support for more kinds of data sources), but it is also more efficient.

- We finally present real-world scenarios where the approaches and systems described in this thesis were used. Ontop-spatial has been used in various applications, two of which are described in this thesis: The Copernicus use case and the maritime awareness use case. In the Copernicus use case, we describe how we used the extension of Ontop-spatial with virtual tables to be able to retrieve Copernicus data using SPARQL queries from Rest APIs on-the-fly with ontologies and mappings, without materialising and converting the data into RDF. In the maritime awareness use case, we describe how Ontop-spatial is used to integrate information about vessels that originally reside in a geospatial relational database with linked open data in order to express rich queries against the unified dataset.

1.2 Publications

The content of the present thesis is partially covered in the following publications:

- Konstantina Bereta, Guohui Xiao, Manolis Koubarakis. *Ontop-spatial: Ontop of geospatial databases*. Journal of Web Semantics, 58 (2019)
- Konstantina Bereta and Manolis Koubarakis. *Ontop of Geospatial Databases*. International Semantic Web Conference (ISWC), Kobe, Japan, 17-21 October 2016
- Konstantina Bereta, George Papadakis and Manolis Koubarakis. *Querying the Web On-the-fly Using Ontologies and Mappings*. 31st International Workshop on Description Logics (DL 2018). Arizona, USA, 27-29 October, 2018
- Konstantina Bereta and Manolis Koubarakis. *Creating Virtual Semantic Graphs on top of Big Data from Space*. Conference on Big Data from Space (BiDS-17), Toulouse, France, 28-30 November, 2017
- Konstantina Bereta, Herve Caumont, Ulrike Daniels, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus, Firman Wahyudi and Dirk Daems. *The Copernicus App Lab project: Easy Access to Copernicus Data*. The 22nd International Conference on Extending Database Technology (EDBT 2019). Lisbon, Portugal, 26-29 March, 2019
- Stefan Brüggemann, Konstantina Bereta, Guohui Xiao and Manolis Koubarakis. *Ontology-based data access for Maritime Security*. In 13th Extended Semantic Web Conference (ESWC), Crete, Greece, May 30- June 2, 2016.
- Konstantina Bereta, Guohui Xiao, Manolis Koubarakis, Martina Hodrius, Conrad Bielski, and Gunter Zeug. *Ontop-spatial: Geospatial Data Integration using GeoSPARQL-to-SQL Translation*. International Semantic Web Conference (ISWC), Kobe, Japan, 17-21 October 2016. Demo paper.

- Konstantina Bereta, Hervé Caumont, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus and Firman Wahyudi. *From Copernicus Big Data to Big Information and Big Knowledge: a Demo from the Copernicus App Lab Project*. 27th ACM International Conference on Information and Knowledge Management (CIKM 2018). Turin, Italy, 22-26 October, 2018. Demo paper.

Part of the work described in this thesis is also covered in the following submitted paper:

- Konstantina Bereta, George Papadakis, Manolis Koubarakis. *Querying the Web On-the-fly Using Ontologies and Mappings*. Submitted as conference paper.

1.3 Thesis structure

The rest of this thesis is organised as follows. In Chapter 2 we present background knowledge and related work. In Chapter 3 we describe our extension of the OBDA paradigm with spatial support, which we implemented in the system Ontop-spatial. In Chapter 4 we introduce our extension of the framework of stRDF and stSPARQL in the direction of supporting temporal features in the OBDA paradigm. In Chapter 5 we go beyond relational databases as data sources and we describe our OBDA approach for creating virtual semantic graphs on top of various of Web data sources, such as HTML tables and Web APIs. In Chapter 6 we describe real-world applications in which the approaches described in the context of this thesis were applied. Last, in Chapter 7 we conlude this thesis and we present future extensions.

2. BACKGROUND AND RELATED WORK

In this section we provide background knowledge that forms the ground which the work described in this thesis is built on. In Section 2.1 we provide some preliminary knowledge on the data model RDF and SPARQL query language. In Section 2.2 we describe geospatial extensions of RDF and SPARQL.

2.1 RDF and SPARQL

2.1.1 The data model RDF

The Resource Description Framework (RDF) is a W3C standard that was created for describing resources on the Web. These resources can be anything that is available on the web, whether it can be directly retrieved or not. For example, this framework can be used to describe metadata about a Web page (e.g., the author of the Web page) or entities that can be identified in that Web page. The primary purpose of the data model RDF is to facilitate data exchanging, suggesting a way of encoding information about Web resources and adding semantics to it, so that it can be exchanged between applications preseving its original meaning and in an interoperable way so that the data can be reused by applications and be combined with other data in a transparent, interoperable way, regardless of the underlying schema of the original data.

In the data model RDF, resources are identified using Uniform Resource Identifiers (URIs). URIs are also used to encode the properties and the respective values that describe these resources. Properties and Values are Web resources as well.

RDF statements are used to describe resources in the same way that sentences are used in natural language to describe facts. RDF statements consist of the following three elements: The subject, the predicate and the object of the statement.

Let us now consider the following statement statement in English, describing a Web page.

```
http://ontop-spatial.di.uoa.gr/ has a creator whose value is Konstantina Bereta
```

The statement described above could be encoded in RDF as follows:

```
<http://ontop-spatial.di.uoa.gr/> <http://purl.org/dc/elements/1.1/creator>
<https://dblp.uni-trier.de/pers/hd/b/Bereta:Konstantina>
```

The RDF statement decribed above follows the triples notation, having a subject, a predicate and an object (in that order). The object of the RDF statement provided above is a unique identifier of a Web page, that is essentially the URL of the page. The predicate of the statement is a property that links the Web page to its creator. The value of this

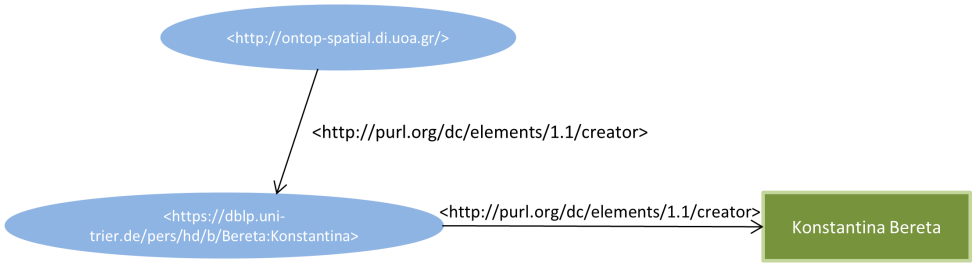


Figure 2.1: Example of an RDF graph

property is another URL. This URL is the DBLP page of the creator of the page. Notably, the name and surname of the person that created this web page could have been used instead,i.e.,"Konstantina Bereta". This would be compliant with the standard RDF syntax and closer to the natural language statement. However, The name of the creator of the page is not a unique identifier (e.g., there can be more than one people with the same name), while the URL that corresponds to their DBLP page is (i) unique (i.e., no other person can have the same DBLP page), and (ii), it can potentially link the page to the DBLP profile of its creator, enabling applications that could consume this data to have access to enriched information across many datasets. In order also include information about the name of the creator of the page, we can add the following triple:

```
<https://dblp.uni-trier.de/pers/hd/b/Bereta:Konstantina>
<http://www.example.org/terms/name> "Konstantina Bereta"
```

Apart from the triples notation, RDF statements can also be represented as a directed, labeled graph. Figure 2.1 shows an example of an RDF graph that represents the triples described above.

2.1.2 SPARQL query language

SPARQL is the W3C standard language for querying RDF graphs. As it essentially performs graph matching, *graph pattern expressions* are core components of a SPARQL query.

As described in [60], a SPARQL graph pattern expression is defined recursively as follows.

- A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (a triple pattern).
- If $P1$ and $P2$ are graph patterns, then expressions $(P1\ AND\ P2)$, $(P1\ OP\ P2)$, and $(P1\ UNION\ P2)$ are graph patterns (*conjunction graph pattern*, *optional graph pattern*, and *union graph pattern*, respectively).
- If P is a graph pattern and R is a SPARQL built-in condition, then the expression $(P\ FILTER\ R)$ is a graph pattern (a filter graph pattern).

Apart from graph patterns, another important component of SPARQL queries is the *solution modifiers*. The solution modifiers are operators that operate on the output of the graph pattern matching evaluation, e.g., it can be a projection, an `order by`, `distinct`, and/or a limit clause.

The *output* of SPARQL queries can be a yes/no value (i.e., when using the `ASK` primitive in the projection part), a set of bindings (i.e., mappings of variables included in the `SELECT` clause with their corresponding values), constructed RDF graphs (e.g., using the `CONSTRUCT` keyword), or a description of resources (i.e., using a `DESCRIBE` clause).

The work described in [60] also presents the semantics and complexity of SPARQL.

We now provide some SPARQL examples against the RDF graph described in Listing 2.1.

Listing 2.1: Example RDF graph

```
PREFIX dblp:<https://dblp.uni-trier.de/pers/hd/b/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX ex:<http://www.example.org/terms/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

<http://ontop-spatial.di.uoa.gr/> dc:creator dblp:Bereta:Konstantina .
dblp:Bereta:Konstantina ex:name "Konstantina Bereta" .
dblp:Bereta:Konstantina ex:age "31"^^xsd:integer .
```

Example 2.1.1. Retrieve the creator of the web page `http://ontop-spatial.di.uoa.gr/`, and project also their name.

Listing 2.2: Example of a SPARQL query

```
PREFIX pu: <http://purl.org/dc/elements/1.1/>
PREFIX ex: <http://www.example.org/terms/>

SELECT ?cr ?name
WHERE { <http://ontop-spatial.di.uoa.gr/> pu:creator ?cr .
        ?cr                               ex:name ?name }
```

In the query described in Listing 2.3, the variables `?cr` and `?name` are projected using the respective `SELECT` clause described in the first line of the query. The variable `?cr` gets bound with the value of the object part of the triple whose object and predicate parts match with the URIs `<http://ontop-spatial.di.uoa.gr/>` and `pu:creator` respectively, thus, a solution binding of this triple pattern is
$\omega 1 = \{?cr \leftarrow$ `<https://dblp.uni-trier.de/pers/hd/b/Bereta:Konstantina>`$\}$. In a similar way, the second pattern of the query matches to a triple included in the dataset that includes `ex:name` as a predicate. The evaluation of this triple pattern produces the following binding:
$\omega 2 = \{?cr \leftarrow$ `<https://dblp.uni-trier.de/pers/hd/b/Bereta:Konstantina>`$, ?name \leftarrow$ `'Konstantina Bereta'`$\}$

The result of the graph pattern matching process of the query evaluation is $\omega = \omega 1 \bowtie \omega 2$, where $\bowtie$ is a the set-theoretic left outer-join operator.
$\omega = \{?cr \leftarrow$ <https://dblp.uni-trier.de/pers/hd/b/Bereta:Konstantina>$, ?name \leftarrow$ 'Konstantina Bereta'$\}$

Example 2.1.2. Retrieve the pages whose creator is less than 40 years old.

Listing 2.3: Example of a SPARQL query

```
PREFIX pu: <http://purl.org/dc/elements/1.1/>
PREFIX ex: <http://www.example.org/terms/>

SELECT ?page
WHERE { ?page pu:creator ?cr .
        ?cr   ex:age ?age
        FILTER (?age < 40}
```

The query described in Listing 2.3 contains two triple patterns and a FILTER clause. Following the approach explained above, we can easily infer that the solution of the evaluation of the two triple patterns are the following bindings:
$\omega = \{?page \leftarrow$ <http://ontop-spatial.di.uoa.gr/>$, ?age \leftarrow 31\}$
The FILTER clause of the SPARQL query, however, introduces a restriction. The binding ω will be included in the result set only if it qualifies the condition on the variable $?age$. As $31 < 40$, the outcome of the FILTER condition is positive, so the binding ω is indeed included in the result set. However, only the variable $?page$ is included in the projection, so the final result of the query is the following:
$\omega' = \{?page \leftarrow$ <http://ontop-spatial.di.uoa.gr/>$\}$

2.2 Geospatial and temporal extensions of RDF and SPARQL

2.2.1 GeoSPARQL

The query language GeoSPARQL is a geospatial extension of the query language SPARQL and it is standardized by OGC.

It comprises the following components:

- The *Core* component, which defines high level RDFS/OWL classes for spatial objects.
- The *Topology Vocabulary extension*, which defines RDF properties for asserting and querying topological relations between spatial objects.
- The *Geometry extension*, which defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects.

- The *Geometry topology extension*, which defines topological query functions.
- The *Query rewrite extension*, which defines topological query functions.
- The *RDFS entailment extension*, which includes the RDF and RDFS reasoning requirements.

We will now provide some examples that highlight the geospatial functionalities provided in GeoSPARQL.

Example 2.2.1. Describe the geometry of a location of an area.

Listing 2.4: Spatial triple

```
:Area geo:asWKT  "Point(-83 33)"^^geo:wktLiteral
```

The triple provided in Listing 2.4 describes the WKT serialisation of the geometry of an area. As defined in the Geometry extension component in GeoSPARQL, geometries can be serialised using the standard Well-known-text (WKT) and Geography Markup Language (GML) formats.

Example 2.2.2. Select features that overlap with each other

Listing 2.5: Spatial predicates in GeoSPARQL triple patterns

```
SELECT  ?x WHERE { ?x  geo:sfOverlaps ?y}
```

The GeoSPARQL query provided in Listing 2.5 uses the predicate `geo:sfOverlaps`, that is defined in Topology Vocabulary extension of the GeoSPARQL specification [26]. Using this predicate one can retrieve entities for which the `overlap` relationship holds. Notably, to receive a non-empty result set of this query, this information should either exist in the knowledge base or can be inferred using spatial reasoning (which might or might not be supported depending on the implementation).

Example 2.2.3. Select features whose geometries overlap with each other

Listing 2.6: Spatial join in GeoSPARQL

```
SELECT  ?x WHERE {
?x1 geo:asWKT ?g1 . ?x2 geo:asWKT ?g2 .
FILTER (geof:sfOverlaps(?g1, ?g2))}
```

An alternative way to retrieve overlapping geometries using GeoSPARQL is to pose the query described in Listing 2.6, which uses the function `geof:sfOverlaps` instead of the predicate `geo:sfOverlaps`. The difference is that in this case, the geometries of the corresponding spatial objects need to be retrieved and the query processor needs to consider the calculation of the `overlap` condition using the actual geometries. This approach is normally used when the respective `qualitative` information does not exist explicitly in

the knowledge base in order to be retrieved directly using the predicate `geo:sfOverlaps`. This query is of course more expensive in terms of query execution time, as it includes the evaluation of the overlap condition for every pair of geometries included in the intermediate results (spatial join).

In Chapter 3 we provide more details and formalisation of the GeoSPARQL components.

2.2.2 stSPARQL

The framework of stRDF and stSPARQL is an extension of the framework of RDF and SPARQL with spatial and temporal support and it has been developed in the same time, but independently, with GeoSPARQL, before it was standardised. The spatial dimension of the data model stRDF and the query language stSPARQL is very similar to the ones proposed in GeoSPARQL. For example, all geosptial functions defined in GeoSPARQL are also defined in stSPARQL (using the strdf namespace instead[1]). The representation of geometries in the data model stRDF is also very similar to the one proposed in GeoSPARQL, as again, the geometries are serialised as WKT and GML literals.

However, stRDF and stSPARQL support the following additional functionalities: (i) spatial aggregates, and (ii) the representation and query of the valid time of triples.

Spatial aggregates. The language stSPARQL, as opposed to GeoSPARQL, considers also the use of spatial aggregates in queries, i.e., a set of functions that perform spatial operations over a set of bindings that are solutions of the corresponding triple patterns of a stSPARQL query. More specifically, it defines the aggregate functions provided in Listing 2.7.

Listing 2.7: Spatial aggregates in stSPARQL

```
strdf:geometry strdf:union(set of strdf:geometry A)
strdf:geometry strdf:intersection(set of strdf:geometry A)
strdf:geometry strdf:extent(set of strdf:geometry A)
```

Notably, although the `geof:union` operator is defined in the GeoSPARQL specification [26], it is not an aggregate, but an operator that takes a pair of geometries as input and returns their union. This version of the `geof:union` is also supporting in the corresponding `strdf:union` operator, but is also overloaded with the ability to be used as an aggregate. The `strdf:union` operator shown in the first line of the Listing 2.7 takes a set of geometries as input (e.g., it can be a variable bound of a set of geometry bindings that are solutions to the triple patterns of an stSPARQL query), it computes the union of all these geometries and it returns the result as a single, unified geometry.

Valid time. Another major feature of the framework of stRDF and stSPARQL is the temporal dimension, e.g., the “t” in “st”, which is described in more detail in [13]. The data model RDF only considers user-defined time and it does not provide support for the representation of *valid time* of triples, which is the time when a triple is valid. GeoSPARQL does

[1] `http://strdf.di.uoa.gr/ontology`

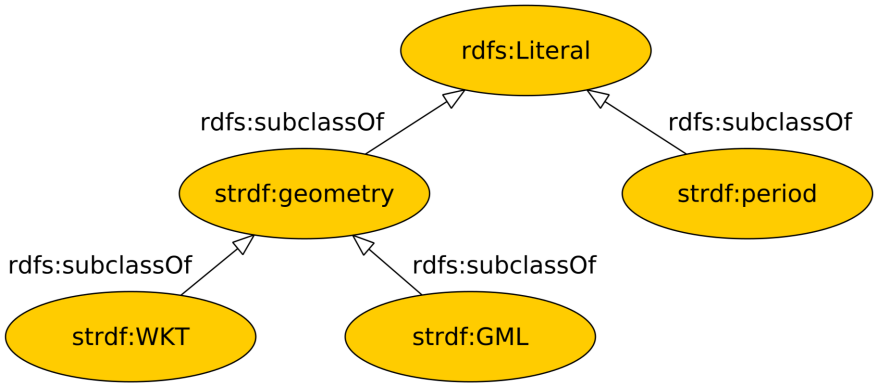


Figure 2.2: The datatypes of stRDF

not offer any additional temporal features to RDF and SPARQL. The valid time support of the framework of stRDF and stSPARQL enables users to model the spatial data that changes over time. Below we describe the temporal features of the framework of stRDF and stSPARQL, which we extend in the context of this thesis.

The *timeline* assumed is the (discrete) value space of the datatype `xsd:dateTime` of XML-Schema. Two kinds of time primitives are supported: time instants and time periods. A *time instant* is an element of the time line. A *time period* (or simply period) is an expression of the form $[B,E)$, $(B,E]$, (B,E), or $[B,E]$ where B and E are time instants called the *beginning* and the *ending* of the period respectively. Since the time line is discrete, we often assume only periods of the form $[B,E)$ with no loss of generality. Syntactically, time periods are represented by literals of the new datatype `strdf:period` that we introduce in stRDF. The value space of `strdf:period` is the set of all time periods covered by the above definition. The lexical space of `strdf:period` is trivially defined from the lexical space of `xsd:dateTime` and the closed/open period notation introduced above. Time instants can also be represented as closed periods with the same beginning and ending time.

Values of the datatype `strdf:period` can be used as objects of a triple to represent *user-defined time*. In addition, they can be used to represent *valid times* of temporal triples which are defined as follows. A *temporal triple (quad)* is an expression of the form `s p o t.` where `s p o.` is an RDF triple and `t` is a time instant or a time period called the *valid time* of a triple. An *stRDF graph* is a set of triples and temporal triples. In other words, some triples in an stRDF graph might not be associated with a valid time.

We also assume the existence of temporal constants `NOW` and `UC` inspired from the literature of temporal databases [25]. `NOW` represents the current time and can appear in the beginning or the ending point of a period. `UC` means “Until Changed” and is used for introducing valid times of a triple that persist until they are explicitly terminated by an update. For example, when John becomes an associate professor in 1/1/2013 this is assumed to hold in the future until an update terminates this fact (e.g., when John is promoted to professor).

Example 2.2.4. The stRDF graph described in Listing 2.8 consists of temporal triples that represent the land cover of an area in Spain for the time periods [2000, 2006) and [2006,

UC) and triples which encode other information about this area, such as its code and the WKT serialization of its geometry extent. In this and following examples, namespaces are omitted for brevity. The prefix `strdf` stands for `http://strdf.di.uoa.gr/ontology` where one can find all the relevant datatype definitions underlying the model stRDF.

Listing 2.8: Example of temporal graph

```
clc:Area_4 rdf:type clc:Area .
clc:Area_4 clc:hasID "EU-101324" .
clc:Area_4 clc:hasGeometry "POLYGON((-0.66 42.34,..))"^^strdf:WKT .
clc:Area_4 clc:hasLandCover clc:coniferousForest
        "[2000-01-01T00:00:00,2006-01-01T00:00:00)"^^strdf:period .
clc:Area_4 clc:hasLandCover clc:naturalGrassland
        "[2006-01-01T00:00:00,UC)"^^strdf:period .
```

The stRDF graph provided in Listing 2.8 is written using the N-Quads format[2] which has been proposed for the general case of adding context to a triple. The graph has been extracted from a publicly available dataset provided by the European Environmental Agency (EEA) that contains the changes in the CORINE Land Cover dataset for the time period [2000, UC) for various European areas. According to this dataset, the area `corine:Area_4` has been a coniferous forest area until 2006, when the newer version of CORINE showed it to be natural grassland. Until the CORINE Land cover dataset is updated, `UC` is used to denote the persistence of land cover values of 2006 into the future. The last triple of the stRDF graph gives the WKT serialization of the geometry of the area (not all vertices of the polygon are shown due to space considerations).

The new features of the language are:

Temporal Triple Patterns. Temporal triple patterns are introduced as the most basic way of querying temporal triples. A *temporal triple pattern* is an expression of the form `s p o t.`, where `s p o.` is a triple pattern and `t` is a time period or a variable.

Temporal Extension Functions. Temporal extension functions are defined in order to express temporal relations between expressions that evaluate values of the datatypes `xsd:dateTime` and `strdf:period`. The first set of such temporal functions are 13 Boolean functions that correspond to the 13 binary relations of Allen's Interval Algebra. stSPARQL offers nine functions that are "syntactic sugar" ,i.e., they encode frequently-used disjunctions of these relations.

There are also three functions that allow relating an instant with a period:

- `xsd:Boolean strdf:during(xsd:dateTime i2, strdf:period p1)`: returns true if instant `i2` is during the period `p1`.
- `xsd:Boolean strdf:before(xsd:dateTime i2, strdf:period p1)`: returns true if instant `i2` is before the period `p1`.

[2] `http://sw.deri.org/2008/07/n-quads/`

- `xsd:Boolean strdf:after(xsd:dateTime i2, strdf:period p1)`: returns true if instant `i2` is after the period `p1`.

The above point-to-period relations appear in [53]. The work described in [53] also defines two other functions allowing an instant to be equal to the starting or ending point of a period. In our case these can be expressed using the SPARQL 1.1. operator $=$ (for values of `xsd:dateTime`) and functions `period_start` and `period_end` defined below.

Furthermore, stSPARQL offers a set of functions that construct new (closed-open) periods from existing ones. These functions are the following:

- `strdf:period strdf:period_intersect(period p1, period p2)`: This function is defined if `p1` intersects with `p2` and it returns the intersection of period `p1` with period `p2`.
- `strdf:period strdf:period_union(period p1, period p2)`: This function is defined if period `p1` intersects `p2` and it returns a period that starts with `p1` and finishes with `p2`.
- `strdf:period strdf:minus(period p1, period p2)`: This function is defined if periods `p1` and `p2` are related by one of the Allen's relations `overlaps, overlappedBy, starts, startedBy, finishes, finishedBy` and it returns the a period that is constructed from period `p1` with its common part with `p2` removed.
- `strdf:period strdf:period(xsd:dateTime i1, xsd:dateTime i2)`: This function constructs a (closed-open) period having instant `i1` as beginning and instant `i2` as ending time.

There are also the functions `strdf:period_start` and `strdf:period_end` that take as input a period p and return an output of type `xsd:dateTime` which is the beginning and ending time of the period `p` respectively.

Finally, stSPARQL defines the following functions that compute temporal aggregates:

- `strdf:period strdf:intersectAll(set of period p)`: Returns a period that is the intersection of the elements of the input set that have a common intersection.
- `strdf:period strdf:maximalPeriod(set of period p)`: Constructs a period that begins with the smallest beginning point and ends with the maximum endpoint of the set of periods given as input.

The query language stSPARQL, being an extension of SPARQL 1.1, allows the temporal extension functions defined above in the SELECT, FILTER and HAVING clause of a query. A complete reference of the temporal extension functions of stSPARQL is available on the Web[3].

[3] `http://www.strabon.di.uoa.gr/stSPARQL`

Temporal Constants. The temporal constants `NOW` and `UC` can be used in queries to retrieve triples whose valid time has not ended at the time of posing the query or we do not know when it ends, respectively.

The new expressive power that the valid time dimension of stSPARQL adds to the version of the language presented in [48], where only the geospatial features were presented, is as follows. First, a rich set of temporal functions are offered to express queries that refer to temporal characteristics of some non-spatial information in a dataset (e.g., see Examples 2.2.5, 2.2.6 and 2.2.9 below). In terms of expressive power, the temporal functions of stSPARQL offer the expressivity of the qualitative relations involving points and intervals studied by Meiri [53]. However, we do not have support (yet) for quantitative temporal constraints in queries (e.g., $T_1 - T_2 \leq 5$). Secondly, these new constructs can be used together with the geospatial features of stSPARQL (geometries, spatial functions, etc.) to express queries on geometries that change over time (see Examples 2.2.7 and 2.2.8 below). The temporal and spatial functions offered by stSPARQL are orthogonal and can be combined with the functions offered by SPARQL 1.1 in arbitrary ways to query geospatial data that changes over time (e.g., the land cover of an area) but also moving objects [38] (we have chosen not to cover this interesting application in this chapter).

Below we provide some representative examples that demonstrate the expressive power of stSPARQL.

Example 2.2.5. *Temporal selection and temporal constants*. Return the current land cover of each area mentioned in the dataset.

Listing 2.9: stSPARQL query retrieving the current land cover of an area

```
SELECT ?clcArea ?clc
WHERE {?clcArea rdf:type corine:Area .
       ?clcArea corine:hasLandCover ?clc ?t .
FILTER(strdf:during(NOW, ?t))}
```

The query described in Listing 2.9 is a temporal selection query that uses an extended Turtle syntax that we have devised to encode temporal triple patterns. In this extended syntax, the fourth element is optional and it represents the valid time of the triple pattern. The temporal constant `NOW` is also used.

Example 2.2.6. *Temporal selection and temporal join*. Give all the areas that were forests in 1990 and were burned some time after that time.

Listing 2.10: stSPARQL query including temporal selection and join

```
SELECT ?clcArea
WHERE{ ?clcArea rdf:type clc:Area ;
        clc:hasLandCover clc:ConiferousForest ?t1 ;
        clc:hasLandCover clc:BurnedArea ?t2  .
FILTER(strdf:during(?t1, "1990-01-01T00:00:00"^^xsd:dateTime)
        && strdf:after(?t2,?t1))}
```

The query described in Listing 2.10 shows the use of variables and temporal functions to join information from different triples.

Example 2.2.7. *Temporal join and spatial metric function.* Compute the area occupied by coniferous forests that were burnt at a later time.

Listing 2.11: stSPARQL temporal join query

```
SELECT ?clcArea (SUM(strdf:area(?geo)) AS ?totalArea)
WHERE {?clcArea rdf:type clc:Area;
                clc:hasLandCover clc:coniferousForest ?t1 ;
                clc:hasLandCover clc:burntArea ?t2 ;
                clc:hasGeometry ?geo .
FILTER(strdf:before(?t1,?t2))}
GROUP BY ?clcArea
```

In the query described in Listing 2.11, a temporal join is performed by using the temporal extension function `strdf:before` to ensure that areas included in the result set were covered by coniferous forests *before* they were burnt. The query also uses the spatial metric function `strdf:area` in the SELECT clause of the query that computes the area of a geometry. The aggregate function `SUM` of SPARQL 1.1 is used to compute the total area occupied by burnt coniferous forests.

Example 2.2.8. *Temporal join and spatial selection*. Return the evolution of the land cover use of all areas contained in a given polygon.

Listing 2.12: stSPARQL temporal join and spatial selection query

```
SELECT ?clc1 ?t1 ?clc2 ?t2
WHERE {?clcArea rdf:type corine:Area ;
                clc:hasLandCover ?clc1 ?t1 ;
                clc:hasLandCover ?clc2 ?t2 ;
                clc:hasGeometry ?geo .
FILTER(strdf:contains(?geo,"POLYGON((-0.66 42.34,..))"^^strdf:WKT)
FILTER(strdf:before(?t1,?t2))}
```

The query provided in Listing 2.12 performs a temporal join and a spatial selection. The spatial selection checks whether the geometry of an area is contained in the given polygon. The temporal join is used to capture the temporal evolution of the land cover in pairs of periods that preceed one another .

Example 2.2.9. *Update statement with temporal joins and period constructor*. Apply a coalesce step over periods during which the land cover use of a CORINE area stayed the same by unifying the periods for which the land cover of an area stays the same.

Listing 2.13: stSPARQL temporal update

```
UPDATE {?area corine:hasLandCover ?clcArea ?coalesced}
WHERE {
    SELECT (?clcArea AS ?area) ?clcArea
    (strdf:period_union(?t1,?t2) AS ?coalesced)
    WHERE {?clcArea rdf:type clc:Area ;
            clc:hasLandCover ?clcArea ?t1;
            clc:hasLandCover ?clcArea ?t2 .
    FILTER(strdf:meets(?t1,?t2) || strdf:overlaps(?t1,?t2))}}
```

The update statement provided in 2.13 performs an operation called *coalescing* in the literature of temporal relational databases: two temporal triples with exactly the same subject, predicate and object, and periods that overlap or meet each other can be "joined" into a single triple with valid time the union of the periods of the original triples [15].

2.3 Ontology-based Data Access (OBDA)

In ontology-based data access (OBDA) [76], we can view existing relational (geospatial) databases as RDF graphs with the help of mappings and ontologies. Formally, we start from an *OBDA specification* $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, consisting of a set $\mathcal{T}$ of OWL axioms (called the *TBox*), a relational database schema $\mathcal{S}$, and a set $\mathcal{M}$ of *mapping assertions*. An *OBDA instance* $(\mathcal{P}, \mathcal{D})$ is given by an OBDA specification $\mathcal{P}$ and a relational database instance $\mathcal{D}$ compliant with $\mathcal{S}$.

A mapping $\mathcal{M}$ is a declarative specification relating symbols in the ontology (classes and properties) to (SQL) views over the data. The W3C standard RDB2RDF Mapping Language (R2RML) was created with the goal of providing a language for such mappings. The ontology $\mathcal{T}$, together with the mapping $\mathcal{M}$, exposes a high-level conceptual view of the underlying data in terms of a virtual RDF graph, which users can query using the SPARQL query language. In this chapter, we extend R2RML mapping with supports of datatypes defined in GeoSPARQL standard.

To ease the presentation, instead of the concrete Turtle serialization of an R2RML mapping, in the following we use an equivalent compact form. A mapping assertion m consists of a source part sql which is a SQL query and a target part t which is an RDF triple template with placeholders inside.

By applying all mapping assertions in $\mathcal{M}$ to $\mathcal{D}$, one can derive a (*virtual*) RDF graph $\mathcal{AMD}$ [62]. Then, SPARQL query answering over an OBDA instance $(\mathcal{P}, \mathcal{D})$ is defined as query answering over $(\mathcal{T}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$.

The derived RDF graph can be materialized as RDF triples, or alternatively it can be kept virtual, in which case the user queries are translated by the OBDA system into queries over the data sources. In the virtual approach, one avoids the cost of materialization, and one can rely on the maturity of relational database systems for efficient query answering, with support for security, robust transactions, etc. Among the state-of-the-art systems support-

ing the virtual OBDA approach we mention Ontop [19], D2RQ[4], Morph [63], Mastro [22], and Stardog[5].

[4] http://d2rq.org/
[5] http://stardog.com/

3. GEOSPATIAL ONTOLOGY-BASED DATA ACCESS

In this chapter we present the first Ontology-based data access approch for querying linked geospatial data using the language GeoSPARQL on top of geospatial relational databases. The work that is presented in this chapter is covered in the publications [11, 14, 7].

3.1 Introduction

Currently, there is an emerging interest of scientific communities from various domains that produce and process geospatial data (e.g., earth scientists) to publish this data as linked data and combine it with other data sources. Responding to this trend, the Semantic Web community has been very active in the geospatial domain, proposing data models, query languages, and systems for the representation and management of geospatial data. Notably, this research has led to the development of extensions of RDF and SPARQL, such as stRDF/stSPARQL [47] and GeoSPARQL [26], that handle geospatial data. Similarly, research on geospatial relational databases has been going on for a long time and has resulted in the implementation of several efficient geospatial DBMS.

Ontology-based data access (OBDA) [76] is a popular paradigm for providing a convenient and user-friendly access to data repositories. In OBDA, an OWL ontology describes the domain of interest, which is connected to a data source through a declarative R2RML mapping specification. Then the underlying data source is *virtualized* as an RDF graph using the vocabulary from the ontology, and the SPARQL queries over the ontologies are automatically rewritten by an OBDA engine into SQL queries expressed over the underlying database. Despite the extensive research performed in the fields of relational databases and the Semantic Web on the development of solutions for handling geospatial data efficiently, to the best of our knowledge, there is no OBDA system that enables the creation of virtual, geospatial RDF graphs on top of geospatial databases. This would be very useful for scientists that produce and process geospatial data, as they mainly store this data in relational geospatial databases (e.g., PostGIS) or in other geospatial data formats that are easily imported into such databases (e.g., shapefiles). With the existing solutions in place, these scientists are forced to materialize their data as RDF in order to publish it as linked data and/or use it in combination with other data sources. However, this is often not practical and discourages users from using Semantic Web technologies. This issue applies to the OBDA paradigm in general, but it has more impact in the geospatial domain due to the reasons we have just described. We address these issues by extending the OBDA paradigm with geospatial support.

The contributions to the state-of-the art described in this chapter are the following:

- On the theoretical side, we provide a formalization of the OGC GeoSPARQL standard in terms of SPARQL entailment regime.

- For a practical query answering algorithm, we introduce an extension to the existing SPARQL-to-SQL translation method in order to support GeoSPARQL queries.
- We describe the implementation of our approach in the system Ontop-spatial, which to the best of our knowledge is the first OBDA system for GeoSPARQL.
- We present an experimental evaluation of our system by extending the benchmark Geographica [34], comparing the performance of Ontop-spatial with the state-of-the-art geospatial RDF store Strabon [48] and the free version of a state-of-the-art commercial triple store with GeoSPARQL support. Due to license reasons, in the context of this chapter we will refer to the commercial system using the alias "System-X". The results show that, in most cases, Ontop-spatial outperforms both of them.

Ontop-spatial is available as free and open source software at the following link: `https://github.com/ConstantB/ontop-spatial`.

3.2 Preliminaries

In this section, we recall the basic notions needed for the rest of this chapter.

3.2.1 RDF and SPARQL

We consider a vocabulary of three pairwise disjoint and countably infinite sets of symbols: **I** for *IRIs*, **L** for *RDF literals*, and **V** for *variables*. In line with previous work on ontology-based data access, we do not consider blank nodes. Intuitively, an IRI represents an object, and a literal represents a typed value. A literal ℓ is of the form $value$^^$type$ where $value$ is the *lexical value* of the ℓ, and $type$ is the *type* IRI of ℓ. The supported types defined in the RDF 1.1 Concepts document [28] is largely based on the XML Schema Definition Language (XSD) [61]. An RDF term is an element in $\mathbf{T} = \mathbf{I} \cup \mathbf{L}$. An (RDF) *triple* is an element in $\mathbf{T} \times \mathbf{I} \times \mathbf{T}$. An *(RDF) graph* is a set of triples.

SPARQL [39] is the W3C standard language designed to query RDF graphs. A *triple pattern* is an element of $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$. A *basic graph pattern* (*BGP*) is a finite set of triple patterns. A filter expression F is a Boolean function, which restricts on solutions over the whole group where the filter appears. We consider the fragment of SPARQL queries defined by Q in the following EBNF grammar[1]:

$$\begin{aligned} P &::= B \mid Q \mid P \texttt{ FILTER } F \mid P \texttt{ UNION } P \mid \\ &\quad (P, P) \mid P \texttt{ OPT } P \\ Q &::= \texttt{SELECT } \{ \mathbf{V} \texttt{ AS } \mathbf{V} \} \texttt{ WHERE } P \end{aligned}$$

where B is a BGP and F is a *filter expression* (we refer to [39] for details).

The semantics of SPARQL queries is given in terms of *solution mappings*, which are *partial* maps $s\colon \mathbf{V} \to \mathbf{T}$ with (possibly empty) domain $dom(s)$. Here, following [60, 45], we use

[1] Recall that in EBNF "{ A }" means any number of repetitions of A.

the set-based semantics for SPARQL (rather than the bag-based one, as in the W3C specification). More specifically, for a BGP B, the *answer* $[\![B]\!]_G$ to B over a graph G is $[\![B]\!]_G = \{s\colon \mathit{var}(B) \to \mathbf{T} \mid s(B) \subseteq G\}$, where $\mathit{var}(B)$ is the set of variables occurring in B and $s(B)$ is the result of substituting each variable u in B by $s(u)$. Then, the *answer* to a SPARQL query Q over a graph G is the set $[\![Q]\!]_G$ of solution mappings defined by induction using the SPARQL algebra operators (filter, join, union, optional, and projection) starting from BGPs; cf. [44]. This semantics is known as *simple entailment*.

3.2.2 SPARQL Entailment Regimes

SPARQL entailment regimes allow for querying RDF graphs with reasoning capabilities [36]. Specifically, an entailment regime E specifies how to obtain from an RDF graph G an entailed graph $\mathit{eg}^E(G)$ [77]. Then, the answer $[\![B]\!]^E_G$ to a BGP B under the entailment regime E is defined as $[\![B]\!]_{\mathit{eg}^E(G)}$. Similarly, the answer $[\![Q]\!]^E_G$ to a SPARQL query Q under the entailment regime E is defined as $[\![Q]\!]_{\mathit{eg}^E(G)}$. In this way, entailment regimes only modify the evaluation of BGPs but not that of other SPARQL operators.

We present now the standard W3C semantics for SPARQL queries over OWL ontologies. Under the *OWL 2 direct semantics entailment regime*, one can query an RDF graph G that consists of two parts: the *intensional* sub-graph (i.e., TBox or ontology) T representing the background knowledge in terms of class and property axioms, and an *extensional* sub-graph (i.e., ABox) $\mathcal{A}$ representing the data as class and property assertions. We write such a graph G, which represents a *knowledge base*, as $(\mathcal{T}, \mathcal{A})$ to emphasize the partitioning when necessary. Moreover, for convenience, we use the triple notations $(s, \texttt{rdf:type}, C)$ and (s, p, o) and the ABox assertion notations $C(s)$ and $p(s, o)$ interchangeably. We are particularly interested in the OWL 2 QL profile [56] of OWL 2, which induces the *OWL 2 QL* entailment regime. For an OWL 2 QL knowledge base G we have $\mathit{eg}^{QL}(G) = \{t \mid G \models_{DL} t\}$, where $\models_{DL}$ denotes the standard OWL 2 entailment, defined in terms of description logics semantics, cf. [74].

3.2.3 Geospatial extensions of RDF and SPARQL

There are several research works on the spatial extensions of the data model RDF and the query language SPARQL. The data model *stRDF* and the query language *stSPARQL* are extensions of RDF and SPARQL 1.1 respectively, developed for the representation and querying of spatial [48] and temporal data (i.e., the valid time of triples [13]). Another framework that has been developed for the representation and querying of geospatial data on the Semantic Web is the OGC standard *GeoSPARQL* [26]. Although GeoSPARQL and stSPARQL were developed independently, they share a lot of features in common. They both adopt the OGC standards Well-known Text (WKT) and Geography Markup Language (GML) for representing geometries. Also both of them extend SPARQL with the topological functions defined in the OGC standard "OpenGIS Simple Feature Access for SQL" [1], and the Egenhofer [32] and the RCC-8 [65] topological relation families. The main difference

	gid integer	code_00 character varying(100)	id character varying(18)	remark character varying(20)	area_ha numeric	shape_leng numeric	shape_area numeric	geom geometry
1	20440	BroadLeavedForest	EU-1900387		69169698	772.0513230	5691.69698	010300002
2	20512	BroadLeavedForest	EU-1900769		83331793	235.0022857	9833.31793	010300002
3	20543	BroadLeavedForest	EU-1900881		89247076	16.17039328	189.247076	010300002
4	20797	BroadLeavedForest	EU-1901587		07822436	63.55011923	107.822436	010300002
5	20904	BroadLeavedForest	EU-1901816		61899830	997.0454395	0618.99830	010300002

Figure 3.1: table `crc` that contains CORINE land cover data

between stSPARQL and GeoSPARQL is that stSPARQL also provides support spatial updates and spatial aggregates, and offer valid time support.

Since in the rest of this chapter we will refer to the notation and concepts defined or followed by stSPARQL and GeoSPARQL, we briefly present them below for the convenience of the reader.

Spatial literal. A spatial literal represents the serialization of a geometry. In stSPARQL, it is a literal of type `strdf:geometry` or its subtypes `strdf:WKT` or `strdf:GML`, as defined in [48]. Similarly, in GeoSPARQL, it is a literal of type `geo:wktLiteral` or `geo:gmlLiteral`.

Spatial term. A spatial term is either a spatial literal or a variable that can be bound to a spatial literal.

Spatial filter. A spatial filter is a Boolean binary function *SF*(t_1, t_2), where t_1, t_2 are spatial terms and *SF* is one of the Boolean functions of the Geometry extension of GeoSPARQL, e.g., `geof:sfEquals`.

Spatial selection. A spatial selection in GeoSPARQL/stSPARQL is a SELECT query with a FILTER which is a Boolean binary function with arguments *a variable and a constant*.

Spatial join. A spatial join in these languages is a query with a FILTER which is a Boolean binary function whose *all arguments are variables*. The definition of the spatial join in SPARQL corresponds to the definition of the spatial join in the geospatial extensions of the relational model. In the rest of this chapter, spatial joins will often be denoted as $\bowtie_{sf}$, where *sf* is a spatial filter.

Example 3.2.1. Consider the table `crc` shown in Figure 3.1 and the following mapping assertion.

```
clc:{gid} rdf:type clc:CorineLandCoverArea;
          geo:hasGeometry clc:geometry/{gid} .
clc:geometry/{gid} geo:asWKT {geom}^^geo:wktLiteral.
← SELECT gid, geom FROM clc
```

The source part of the first mapping assertion is a query over the table `clc`. Intuitively, for each row in the table `clc`, it generates two triples sharing the same subject $s =$ `clc:{gid}`. The first triple declares that the type of s is an IRI `clc:CorineLandCoverArea` and the second triple declares that the geometry of s is an IRI $o =$ `clc:geometry/{gid}` . The third triple declares that the WKT serialization of o is the literal `{geom}^^geo:wktLiteral`. Below is the resulting virtual RDF graph that is populated with information of the first row

of table clc.

```
clc:20440 rdf:type clc:CorineLandCoverArea .
clc:20440 geo:hasGeometry clc:geometry/20440/ .
clc:geometry/20440 geo:asWKT {geom}^^geo:wktLiteral .
```

Consider another mapping assertion:

```
gag:{gid} rdf:type gag:AdministrativeDivision;
          hasGeometry gag:geometry/{gid} .
gag:geometry/{gid} geo:asWKT   {geom_4326}^^geo:wktLiteral.
← SELECT gid,ST_Transform(geom,'4326') AS geom_4326 FROM gag
```

The source part is a query over the table gag, which contains information about administrative divisions. In a similar way to the above, for each row in the table gag, it generates two triples sharing the same subject $s =$ gag:{gid}. The first triple declares that the type of s is an IRI gag:AdministrativeDivision and the second triple declares that the geometry of s is an IRI $o =$ gag:geometry/{gid}. The third triple declares that the WKT serialization of o the literal {geom}^^geo:wktLiteral. Notably, the source query contains a row SQL function in the select clause named ST_Transform. This function is widely used in the area of GIS and it transforms a geometry from its original Coordinate Reference System (CRS) to another. The geometries original shapefile which is imported to our database are expressed using the CRS 2100. Instead of transforming and materializing the geometries into the WGS84 (the universal CRS), we incorporated this function in the mappings to showcase the flexibility of our approach: Users can incorporate geospatial data manipulation functions in the source part of the mappings, so that the virtual semantic views that will be constructed on top of their data is an improved version of the original data.

3.3 A formalization of GeoSPARQL

In this section we present in detail the features of GeoSPARQL and provide a formalization in terms of the SPARQL entailment regime.

3.3.1 OGC GeoSPARQL Standard

In the context of this chapter, we will only consider GeoSPARQL (and, as a result, the geospatial part of stSPARQL). The main features of GeoSPARQL are specified in Clauses 6 to 10 of [26] consisting of one core component and four extensions. Specifically, the core component, the topology vocabulary, and the geometry extension defines an OWL ontology[2] (denoted by $\mathcal{T}_{geo}$); the geometry topology extension defines SPARQL functions for spatial selection and spatial filter; and the query write extension defines additional rules for computing spatial relations. In the following, we summarize these features of GeoSPARQL following the structure of specification.

[2] http://www.opengis.net/ont/geosparql

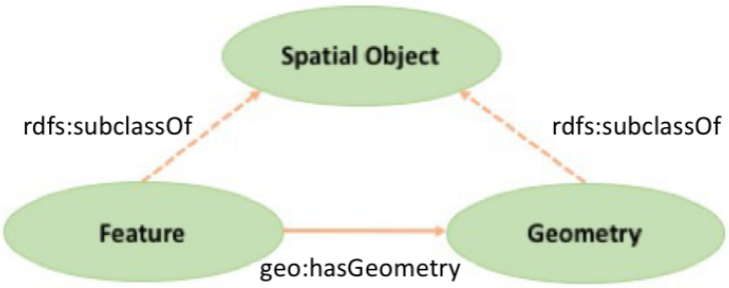


Figure 3.2: GeoSPARQL class hierarchy

3.3.1.1 Core component [Clause 6, OGC-GeoSPARQL standard]

This component defines high-level RDFS/OWL classes for spatial objects. The main GeoSPARQL classes are shown in Figure 6.3. Classes `Feature` and `Geometry` are subclasses of the general class `SpatialObject` (`SubclassOf` properties are represented using dotted arrows). Features have geometries, and this is expressed using the object property `geo:hasGeometry`.

3.3.1.2 Topology vocabulary extension [Clause 7, OGC-GeoSPARQL standard]

This component defines RDF properties for asserting and querying topological relations (i.e., object properties in OWL) between spatial objects. Specifically, it covers different families of topological relations including *Simple Features Access* (e.g., `geo:sfOverlaps`), *RCC8* (e.g., `geo:rcc8po`), and *Egenhofer* (e.g., `geo:ehOverlap`).

3.3.1.3 Geometry extension [Clause 8, OGC-GeoSPARQL standard]

The Geometry extension component defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects. More specifically, this component of GeoSPARQL defines that serializations of geometries are RDF literals, introducing the datatypes `geo:asWKT` and `geo:GML` that correspond to the respective OGC standards WKT and GML that are used to represent geometries as text. It also defines a set of properties that associate features with their geometries, such as the properties `geo:hasGeometry`, `geo:hasSerialization`, `geo:isSimple`, etc. The same component also defines a set of functions that perform non-topological spatial operations, such as the functions `geof:distance` and `geof:intersection`.

3.3.1.4 Geometry Topology extension [Clause 9, OGC-GeoSPARQL standard]

This component defines topological query functions that take two geometry literal and return a boolean value. Topological query functions can be used for spatial selections and spatial joins. The standard defines functions in the families of Simple Features

(1) feature – feature rule

$\texttt{geo:sfOverlaps}(\texttt{?f1}, \texttt{?f2}) \leftarrow \texttt{geo:hasDefaultGeometry}(\texttt{?f1}, \texttt{?g1}),\ \texttt{geo:asWKT}(?g1, \texttt{?g1WKT}),$
$\texttt{geo:hasDefaultGeometry}(\texttt{?f2}, \texttt{?g2}),\ \texttt{geo:asWKT}(\texttt{?g2}, \texttt{?g2WKT}),$
$\texttt{geof:sfOverlaps}(\texttt{?g1WKT}, \texttt{?g2WKT}).$

(2) feature –- geometry rule

$\texttt{geo:sfOverlaps}(\texttt{?f1}, \texttt{?f2}) \leftarrow \texttt{geo:hasDefaultGeometry}(\texttt{?f1}, \texttt{?g1}),\ \texttt{geo:asWKT}(?g1, \texttt{?g1WKT}),$
$\texttt{geo:asWKT}(\texttt{?f2}, \texttt{?g2WKT}),$
$\texttt{geof:sfOverlaps}(\texttt{?g1WKT}, \texttt{?g2WKT}).$

(3) geometry – feature rule

$\texttt{geo:sfOverlaps}(\texttt{?f1}, \texttt{?f2}) \leftarrow \texttt{geo:asWKT}(\texttt{?f1}, \texttt{?g1WKT}),$
$\texttt{geo:hasDefaultGeometry}(\texttt{?f2}, \texttt{?g2}),\ \texttt{geo:asWKT}(\texttt{?g2}, \texttt{?g2WKT}),$
$\texttt{geof:sfOverlaps}(\texttt{?g1WKT}, \texttt{?g2WKT}).$

(4) geometry – geometry rule

$\texttt{geo:sfOverlaps}(\texttt{?f1}, \texttt{?f2}) \leftarrow \texttt{geo:asWKT}(\texttt{?f1}, \texttt{?g1WKT}),$
$\texttt{geo:asWKT}(\texttt{?f2}, \texttt{?g2WKT}),$
$\texttt{geof:sfOverlaps}(\texttt{?g1WKT}, \texttt{?g2WKT})$

Figure 3.3: The Rules $\mathcal{R}_{overlaps}$ for computing overlaps relations

Access, RCC8, and Egenhofer. These functions belong to the namespace `geof` and have the same names as the topological predicates of the Topology extension. For example, `geof:sfOverlaps` is the topological query function corresponding to the relation `geo:sfOverlaps`.

3.3.1.5 Query Rewrite extension [Clause 10, OGC-GeoSPARQL standard]

This clause defines a set of transformation rules, denoted $\mathcal{R}_{geo}$, for computing spatial relations between spatial objects based on their associated geometries. The rules $\mathcal{R}_{geo}$ use the topological extension functions defined in Clause 9 to establish the existence of topological predicates defined in Clause 7.

For example, the rules $R_{overlaps}$ in Figure 4.1 specifies how to compute `geo:sfOverlaps` relations of two spatial objects `?f1` and `?f2`. The first feature-feature rule deals with the situation where `?f1` and `?f2` are features, and the overlaps relation can be computed by calling the `geof:sfOverlaps` function over the WKT serializations `?g1WKT` and `?g2WKT` of the geometries `?g1` and `?g2` of the objects `?f1` and `?f2`. We use $\mathcal{R}_{geo}(G)$ to denote the minimal model of applying rules $\mathcal{R}_{geo}$ to a set of facts G.

The rules $\mathcal{R}_{geo}$ are also used for transforming *qualitative* spatial queries into equivalent *quantitative* queries. When using $\mathcal{R}_{geo}$ for query rewriting, it transforms the triple patterns that contain the topological relation (e.g., `ogc:sfOverlaps`) into an equivalent query that describes the same relation using the respective function (e.g., `geof:overlaps`) in the filter clause of the query. Given a SPARQL query q, we denote by $rew_{geo}(q)$ the SPARQL query

derived by applying all rules $\mathcal{R}_{geo}$ in query rewrite extension.

Example 3.3.1. The GeoSPARQL query q that is described in Listing 3.2 retrieves all pairs of Corine Land Cover areas (s1) and the administrative divisions (s2) such that $s1$ overlaps with $s2$.

Listing 3.1: GeoSPARQL overlap query

```
SELECT ?s1 ?s2 WHERE {
    ?s1 a clc:CorineLandCoverArea .
    ?s2 a gag:AdministrativeDivision .
    ?s1 geo:sfOverlaps ?s2.  }
```

The topological relation `overlaps` in q is expressed using the predicate `geo:sfOverlaps`. Query q can be transformed to the following query q' using the equivalent quantitative function in the example transformation rules of $\mathcal{R}_{overlaps}$.

Listing 3.2: Query rewriting

```
SELECT ?s1 ?s2 WHERE {
  ?s1 a clc:CorineLandCoverArea .
  ?s2 a gag:AdministrativeDivision .
  {
    { ?s1 geo:sfOverlaps ?s2.  }
    UNION
    # feature - feature
    { ?s1 geo:hasDefaultGeometry ?g1 .
      ?g1 geo:asWKT ?g1WKT .
      ?s2 geo:hasDefaultGeometry ?g2 .
      ?g2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    UNION
    # feature - geometry
    { ?s1 geo:hasDefaultGeometry ?g1 .
      ?g1 geo:asWKT ?g1WKT .
      ?s2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    # geometry - feature
    UNION
    { ?s1 geo:asWKT ?g1WKT .
      ?s2 geo:hasDefaultGeometry ?g2 .
      ?g2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    UNION
    # geometry - geometry
    { ?s1 geo:asWKT ?g1WKT. ?s2 geo:asWKT ?g2WKT.
```

```
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
  }
}
```

3.3.2 GeoSPARQL entailment regime

Now we develop a formal framework capturing the semantics of the family of GeoSPARQL query languages in terms of SPARQL entailment regime [73, 46, 77]. Here we introduce a more general version of the formal semantics of GeoSPARQL. Given an entailment regime **E**, we can augment it with the GeoSPARQL capabilities, to derive a new entailment regime (geo-**E**). Intuitively, the geo-**E**-entailment regime captures the core component, the topology vocabulary and the geometry extension by the built-in ontology $\mathcal{T}_{geo}$, the geometry topology extension by corresponding SPARQL functions, and the query rewrite extent by the rules $\mathcal{R}_{geo}$.

Definition 1. Let geo be the GeoSPARQL ontology, $(\mathcal{T}, \mathcal{A})$ a DL ontology, **E** an entailment regime,

$$\mathit{eg}^{\text{geo-}\mathbf{E}}(\mathcal{T}, \mathcal{A}) = \mathcal{R}_{geo}(\mathit{eg}^{\mathbf{E}}(\mathcal{T} \cup geo, \mathcal{A})).$$

It is easy to see that the "query rewrite extension" can be indeed realized by query rewriting.

Proposition 1. Let $\mathcal{T}_{geo}$ be the GeoSPARQL ontology, $(\mathcal{T}, \mathcal{A})$ a DL ontology, q a BGP, then

$$[\![q]\!]^{\text{geo-}\mathbf{E}}_{\mathcal{T},\,\mathcal{A}} = [\![rew_{geo}(q)]\!]^{\mathbf{E}}_{\mathcal{T}\cup\mathcal{T}_{geo},\,\mathcal{A}}\,.$$

We note that OGC GeoSPARQL standard does not explicitly specify which level of reasoning is required. The reasoning capabilities listed in the requirements essentially only require RDFS. Therefore, GeoSPARQL roughly corresponds the geo-RDFS entailment regime. However, the geo ontology is actually classified as $\mathcal{SHIF}$, which is much more expressive than RDFS.

Proposition 1 can be readily applied to the OBDA setting. Given an OBDA instance $(\mathcal{P}, \mathcal{D})$ where $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, the answers of a SPARQL query q under the geo-**E**-entailment regime is $[\![q]\!]^{\text{geo-}\mathbf{E}}_{\mathcal{T},\mathcal{A}_{\mathcal{M},D}}$ and consequently $[\![rew_{geo}(q)]\!]^{\mathbf{E}}_{\mathcal{T}\cup geo,\,\mathcal{A}}$. When **E** is first-order rewritable (e.g. OWL 2 QL), the geo-**E** entailment regime can be implemented in an OBDA system by modifying the workflow. Details of such techniques are discussed in the next section.

3.4 GeoSPARQL-to-SQL

In this section, we present the techniques of answering GeoSPARQL queries in OBDA by translating to SQL queries, which is based on the SPARQL-to-SQL algorithm used in

Ontop [46, 19]. The pseudo code of algorithm is outlined in Figure 1. As in the classical case, the algorithm takes as inputs a (Geo)SPARQL query, an ontology $\mathcal{T}$, and a mapping $\mathcal{M}$, and returns a SQL query. The algorithm consists of *(1)* an offline step, which is query-independent and preprocesses the mapping and ontology and generates the so-called saturated mapping or T-mapping, and *(2)* an online step, which translates the input SPARQL query into an SQL query. We refer the readers to [19] for more details of the workflow. In the following, we discuss the GeoSPARQL specific steps, which are underlined in the pseudo code.

Algorithm 1 Algorithm of Translating GeoSPARQL into SQL

Input: GeoSPARQL query q, Ontology $\mathcal{T}$, Mapping $\mathcal{M}$
Output: An SQL expression

1: // offline phase
2: $\mathcal{T}_{classified} =$ classify$(\mathcal{T} \cup geo)$ ▷ ontology classification
3: $\mathcal{M}_{\mathcal{T}} \leftarrow saturate(\mathcal{M}, \mathcal{T}_{classified})$ ▷ mapping saturation
4: // online phase
5: $Q \leftarrow rew_{geo}(q)$ ▷ GeoSPARQL query write
6: $S \leftarrow$ list of nodes in Q in a bottom-up topological order
7: sql $\leftarrow$ empty map from nodes to SQL expressions
8: **for** node $n \in S$ **do**
9: **if** n is triple pattern **then** ▷ translating leaves
10: sql$[n] \leftarrow$ replace-Tmap-def(n, $\mathcal{M}_{\mathcal{T}}$)
11: **else** ▷ translating non-leaf nodes
12: **if** n = JOIN(n_1,n_2) **then**
13: sql$[n] \leftarrow$ InnerJoin(sql$[n_1]$, sql$[n_2]$)
14: **else if** n = OPTIONAL(n_1, n_2, e) **then**
15: sql$[n] \leftarrow$ LeftJoin(sql$[n_1]$, sql$[n_2], e$)
16: **else if** n = UNION(n_1, n_2) **then**
17: sql$[n] \leftarrow$ Union(sql$[n_1]$, sql$[n_2]$)
18: **else if** n = FILTER(n_1, e) **then**
19: sql$[n] \leftarrow$ Filter(sql$[n_1], e$)
20: **else if** n = PROJECT(n_1, p) **then**
21: sql$[n] \leftarrow$ Project(sql$[n_1], p$)
22: **end if**
23: **end if**
24: **end for**
25: **return** sql$[S$.last$()]$

Ontology classification At line 2, the algorithm classifies the input ontology $\mathcal{T}$ union with the GeoSPARQL ontology geo, and construct an explicit hierarchy of classes and properties. By assuming the built-in ontology geo, the algorithm is able to support the Clauses 6 – 8 of the GeoSPARQL standard.

Table 3.1: GeoSPARQL Simple Feature functions of to SQL Functions

GeoSPARQL function	OGC SFS SQL function
geof:sfEquals	ST_Equals
geof:sfDisjoint	ST_Disjoint
geof:stIntersects	ST_Intersects
geof:sfTouches	ST_Touches
geof:sfCrosses	ST_Crosses
geof:sfWithin	ST_Within
geof:sfContains	ST_Contains
geof:sfOverlaps	ST_Overlaps

GeoSPARQL query-rewrite At line 5, the algorithm expands the input GeoSPARQL query q using the rules $\mathcal{R}_{geo}$ as in Example 3.3.1. We note that the resulting query is of polynomial size of the input query.

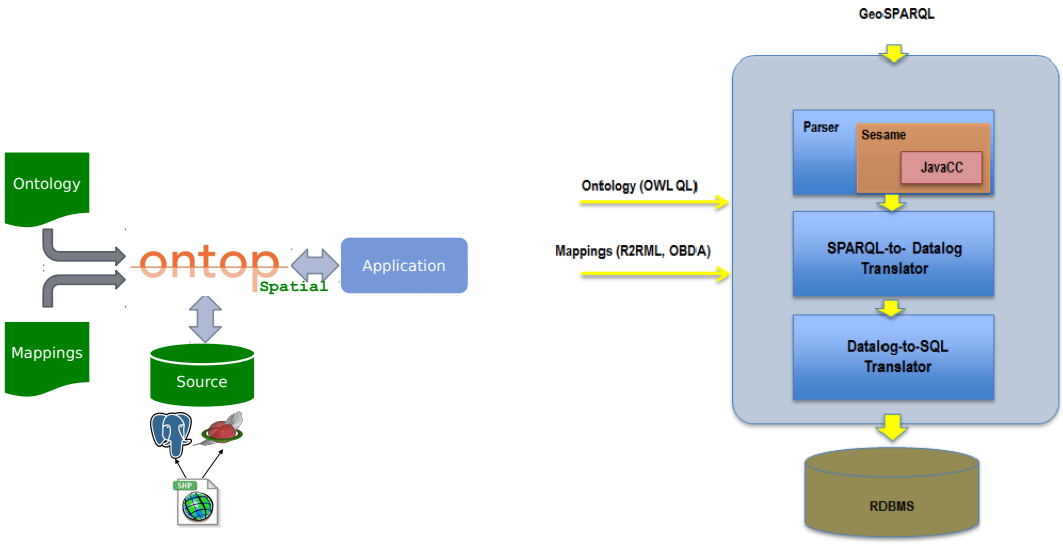
Spatial filter expressions At line 19, the algorithm transforms the SPARQL filters to its SQL equivalences. Now it also translates GeoSPARQL functions to the corresponding functions in the spatial extension of SQL. In Table 3.1, we provide a list of SPARQL Simple Feature functions defined in GeoSPARQL and their equivalences in SQL functions defined in OpenGIS SQL standard [41].

3.5 Implementation

We implemented the GeoSPARQL-to-SQL translation framework discussed in Section 3.4 as an extension of the system Ontop with geospatial features focusing on *spatial selections* and *spatial joins*. We chose to extend Ontop instead of systems offering similar functionality because *(i)* it is open source, robust and extensible, *(ii)* it offers a wide range of functionalities that are useful for geospatial applications (reasoning, multiple APIs), and *(iii)* it implements significant SPARQL-to-SQL optimizations, producing queries that can be executed efficiently by the underlying DBMS as reported in [67]. Ontop-spatial is available as free and open source software at the following link: `https://github.com/ConstantB/ontop-spatial`.

3.5.1 System overview

An abstract overview of the system as well as a high-level architecture diagram can be seen in Figures 3.4(a) and 3.4(b) respectively. In the following, we highlight the components of Ontop that we have extended as they are placed in the query processing workflow:



(a) Abstract overview (b) Ontop-spatial architecture

Figure 3.4: Ontop-spatial

- The virtual Ontop repository takes as input an ontology and a mapping file. Mappings can be either R2RML or the Ontop native OBDA mapping language.
- Once Ontop-spatial receives a GeoSPARQL query, the query gets parsed. We modified the Sesame (now known as rdf4j) parser used by Ontop (and the javacc parser that the respective Sesame library uses), in order to extend its syntax to support geospatial operations in the filter clause of the query. Additionally, qualitative geospatial queries, (i.e., queries containing geospatial triple patterns such as `ex:feauture1 geo:overlaps ex:feature2`) are also supported as standard SPARQL triple patterns, and get transformed into their quantitative equivalents (i.e., queries with spatial filters) in the following step.
- Conventionally, the next step in Ontop is to translate the SPARQL query and the R2RML mappings into a Datalog program so that the query can be represented formally and optimized following a series of optimization steps described in detail in [67]. Ontop-spatial inherits these optimizations and extends the SPARQL-to-Datalog translation module. As explained in the previous section, the geospatial filters are transformed into Datalog using distinguished geospatial predicates. The same distinguished geospatial predicates are used in the case of the qualitative geospatial queries as well using the Query Rewrite extension of GeoSPARQL. As a result, both quantitative and qualitative representations of a GeoSPARQL query are transformed into the same SQL query in the following step.
- The optimized version of the Datalog query, as derived from the previous step, gets translated into SQL. Every geospatial Datalog predicate is mapped to the respective geospatial SQL operator, following the syntax of the underlying DBMS. The DBMS adapter has been extended in order to be able to identify geospatial columns in

the database of the user. The PostgreSQL adapter of Ontop has been modified to support the PostGIS extension and an adapter of the open source DBMS Spatialite has been added.

- The SQL query gets eventually executed in the underlying DBMS. Currently, the spatially-enabled databases that ontop supports are the geospatial extensions of PostgreSQL and Sqlite, namely PostGIS[3], Spatialite[4], and Oracle Spatial[5] respectively. More geospatial databases will be supported in the future.

- After the evaluation of the spatial SQL query in the DBMS, Ontop-spatial gets the results and sends them to the user. If geometries need to be projected, the SQL query that is produced returns the result as WKT. This enables Ontop-spatial to be used as a GeoSPARQL endpoint, that could serve as input endpoint for applications like linked geospatial data visualizers [57] to display the geometries that are returned as a result of a GeoSPARQL query.

Like the default version of Ontop, ontop-spatial can be used as a web application (using Sesame workbench), as a Sesame library, as a Protege plugin, or it can be executed from the command line. The virtual geospatial graphs created by Ontop can also be materialized, creating an RDF dump, so that it can then be imported in a geospatial RDF store.

3.5.2 Compliance with GeoSPARQL

In the following, we explain the parts of GeoSPARQL that are supported in Ontop-spatial.

Core component. Ontop-spatial supports SPARQL and the RDFS classes of the GeoSPARQL ontology.

Topology Vocabulary extension. Ontop-spatial supports the properties `geo:sfEquals`, `geo:sfDisjoint`, `geo:sfIntersects`, `geo:sfTouches`, `geo:sfCrosses`, `geo:sfWithin`, `geo:sfContains`, `geo:sfOverlaps` and the respective Egenhofer and RCC relations to be used in SPARQL graph patterns.

Geometry extension. Ontop-spatial supports the Geometry classes and properties defined in the Geometry topology component of the GeoSPARQL specification. It also supports the serializations of geometries as literals of the datatypes `geo:wktLiteral` and `geo:gmlLiteral` respectively, as well as the serialization properties `geo:asWKT` and `geo:asGML`. Furthermore, Ontop-spatial supports the non-topological query functions `geof:distance`, `geof:buffer`, `geof:convexHull`, `geof:intersection`, `geof:union`, `geof:difference`, `geof:symDifference`, `geof:envelope` and `geof:boundary` as SPARQL extension functions.

[3] `http://www.postgis.net`
[4] `http://www.gaia-gis.it/gaia-sins/`
[5] `https://www.oracle.com/database/spatial/index.html`

Geometry topology extension. Ontop-spatial supports all of the topological relation functions defined in the Geometry topology extension.

Query Rewrite extension. To the best of our knowledge, Ontop-spatial is the first GeoSPARQL implementation that supports this extension of GeoSPARQL.

RDFS entailment extension. Ontop-spatial supports this GeoSPARQL component as Ontop supports RDFS reasoning.

3.5.3 Beyond GeoSPARQL: Raster data support

Raster data support. In the raster data model, the geospatial data are represented in a different way than in the vector data model. Essentially, they are represented as pixels, with each pixel containing a set of values. This data format is more compact and it is very common in scientific data. For example, a value of a raster cell could indicate a measurement value, such as temperature, moisture level, etc. Due to the popularity of this data model, the geospatial databases incorporated the implementation of adapters for Raster data, introducing specialized data types for representing raster data formats and a set of extension functions for their processing and manipulation, handling them in a similar way to how they handle vector data.

However, none of the geospatial extensions of the framework of RDF and SPARQL, such as stRDF and stSPARQL and GeoSPARQL have considered support for raster data. The main challenge that lies behind this is twofold: First, a raster file is associated with a geometry only as a whole. It is not straight-forward to associate separate raster cells to a geometry, they have to be vectorized first (i.e., translated into polygons). Second, every raster cell is associated with one or more values. In order to convert all information contained in a raster file into RDF, then multiple triples should describe a raster cell, producing a large amount of triples for a whole raster file. However, not all of this information is needed. In most of the use cases, only the information that derives from a raster file and satisfies certain criteria (e.g., value constraints) is all that is needed to be converted into RDF. This means that the raster file needs to be processed and then the results of this processing are useful as RDF, while any other information is redundant. These challenges have discouraged the scientific community from converting and materializing raster data to RDF. Recently, OGC and W3C have established a working group on Spatial Data on the Web[6]. One of the working notes published by the working group recently is called “Coverages in linked data” and, among other, this discusses these challenges of raster data.

In the work described in this chapter, we address these challenges by following the OBDA paradigm:

- Ontop-spatial can connect to a geospatial relational database with a raster adapter.

[6] https://www.w3.org/2015/spatial/wiki/Main_Page

- The raster datatype is internally handled in the same way as its vector counterpart (e.g., the Geometry datatype).
- The following GeoSPARQL operators are overloaded for supporting the respective operations having raster data as arguments in addition to vector data: `ST_Contains`, `ST_Covers`, `ST_Within`, `ST_Overlaps`, `ST_Intersects`, `ST_Touches`.
- PostGIS operators can be added in the mappings in order to process the raster data and create virtual geospatial RDF views above them. For example, certain operators can be used in the SQL query of a mapping in order to refine the results, refining the information from the original raster file that will be virtually translated into RDF. An example is given below.

In this example, we will try to combine both vector and raster sources. First, we import a shapefile containing USA second-level administrative divisions to a PostGIS database. The table that contains this information is named `USA_ADM2` and it has the following columns: the gid column that stores the id of the respective entries of the shapefile, the id_0 column that stores an identifier of the administrative division, and the $geom$ column that stores the boundaries of the administrative divisions as vector geometries in Well-known-binary format (WKB). Then, we import a raster file that is a GeoTIFF image of Chicago. With the raster extension of PostGIS enabled in the database, the raster file is imported into a table named `CHICAGO`. The column $rast$ of this table contains the raster cells of the GeoTIFF image and it is of $raster$ datatype, which is supported by the raster extension of PostGIS. According to the PostGIS reference[7], each raster has one or more bands each having a set of pixel values and it can be georeferenced. To summarize, the schema of these two tables are:

```
USA_ADM2 (gid, id_0, geom)
CHICAGO (rid,  rast)
```

The mapping provided below encodes how data stored in tables `USA_ADM2` and `CHICAGO` can be mapped into (virtual) RDF terms.

Listing 3.3: Mapping vector and raster geometries

```
mappingId chicago2
target geo:{geom} rdf:type f:rasterCell ;
       geo:{geom} geo:asWKT {geom} .
source SELECT (ST_DumpAsPolygons(rast)).geom FROM chicago;
```

In the mapping shown in Listing 3.3, the geometries (that are of the raster datatype in the database) are mapped to the WKT format, after they are vectorized, using the PostGIS `ST_DumpAsPolygons` function in the source part. This is a procedure that allows domain experts to use all geometries that they may have in a database uniformly, and execute

[7] `https://postgis.net/docs/raster.html`

spatial operations involving vector and raster geometries, for example. Domain experts usually perform this vectorization step as part of pre-processing. In the mapping described above, we show how this can be done on-the-fly, using Ontop-spatial. In the following, we provide an example of a GeoSPARQL query that involves the combination of vector and raster data sources.

Example 3.5.1. Retrieve administrative divisions that intersect with raster cells of the GeoTIFF image of Chicago.

Listing 3.4: Query intersecting verctor and raster geometries

```
SELECT ?adm
WHERE { ?r rdf:type :rasterCell . ?r :hasGeometry ?rast .
      ?adm rdf:type :AdministrativeDivision .
      ?adm geo:hasGeometry ?g . ?g geo:asWKT ?geom .
FILTER(geof:sfIntersects(?geom,?rast))}
```

Using the query described in Listing 3.4, we retrieve the geometries that correspond to the boundaries of second level administrative divisions whose serializations get bound to the variable $?geom$. We also retrieve the geometries of raster cells that get bound to the variable $?rast$. In Ontop-spatial, the GeoSPARQL function `geof:sfIntersects` is overloaded so that it can evaluate the condition that checks the spatial intersection of vector and raster geometries. As a result, we retrieve tha administrative divisions whose boundaries intersect with the GeoTIFF image.

In this way, a user can handle geospatial data sources regardless of their original format. The query described above is identical to a query that we would pose if only vector data sources were involved. Notably, there is no other system that is able to perform spatial joins between vector and raster geometries in such transparent way, even in the case of specific software solutions that specialize in raster data management like Rasdaman[8].

3.6 Evaluation

We conducted an empirical evaluation of our implementation based on the philosophy of Geographica[9], a benchmark for testing the performance of geospatial RDF stores [34]. Geographica consists of a *micro benchmark* and *a macro benchmark*. The *micro benchmark* is designed for testing basic geospatial operators, such as spatial selections and spatial joins. The *macro benchmark* tests the performance of the evaluated systems using queries that correspond to real application scenarios. As our aim is not to test geospatial RDF stores as done in [34], we use a modified benchmark based on the micro benchmark of Geographica as we explain later in this section.

Since there was no alternative OBDA system that allow for posing GeoSPARQL queries over geospatial relational databases, we decided to evaluate Ontop-spatial in comparison

[8] `http://www.rasdaman.com/`

[9] `http://geographica.di.uoa.gr/`

with a geospatial RDF store. We consider that the spatiotemporal RDF store Strabon [48] is a good representative of the family of the geospatial RDF stores to compare with as *(i)* it is a state-of-the-art geospatial RDF store both in terms of functionality and performance [48, 34], *(ii)* it supports a big subset of GeoSPARQL (apart from stSPARQL), and *(iii)* it uses a spatially-enabled DBMS as back-end, performing a SPARQL-to-SQL translation following a specific storage scheme as explained in [48]. This enables us to use the same DBMS (PostGIS with the same configuration and tuning) and perform a comprehensive comparison. We also consider the system System-X (the Free version) which is one of the most efficient triple stores and recently added support for GeoSPARQL. Since this system is more recent than the study described in [34], we included it in our experimental evaluation to find out how it compares to both Strabon and Ontop-spatial.

3.6.1 Datasets

Geospatial data come, in most cases, in native geospatial data formats. In a real-world scenario, a user that works with geospatial data obtains it as files in a geospatial data format (e.g., a shapefile) and stores it either in a GIS or a spatially-enabled relational database. Later on, he may convert the data into RDF and store it in a geospatial RDF store in order to combine it with other linked data.

The benchmark Geographica is based on such real-world geospatial application scenarios and for the experimental evaluation of Ontop-spatial we will also follow this approach: We will import real geospatial datasets in a spatially-enabled relational database and use it as the back-end of Ontop-spatial.

We chose to use the datasets of Geographica that are available in their original format (shapefiles). These datasets are the Corine Land Cover dataset of Greece, which is provided by the European Environment Agency (EEA), the Greek Administrative Geometry (GAG), and the Hotspots dataset provided by the National Observatory of Athens. We complemented these data sources with the original raw files of OpenStreetMap data about Greece which are available as shapefiles[10]. Geographica uses the RDF versions of the same subset of the OSM datasets created by the project LinkedGeoData[11]. For the rest of this Chapter, we will refer to this dataset using the acronym of the resulting, RDF-ized version (LGD). We added more OSM categories to our workload (e.g., buldings, waterways, etc.), as we will exploit the fact that each one is contained in a different shapefile (so it will be imported into a different table), to stress our system as we explain later on in this section.

For the evaluation of Ontop-spatial, we imported the shapefiles in a PostGIS database using the `shp2pgsql` command as described here: `https://github.com/ConstantB/Ontop-spatial/wiki/Shapefiles`. In this way, each shapefile is loaded into a separate table in the database. Each one of these tables contains a column where geometries are stored in binary format (WKB) and an index has been built on that column. Then, we cre-

[10] `http://download.geofabrik.de/europe/greece.html`
[11] `http://linkedgeodata.org/`

ated the minimum set of mappings in order to pose the queries of the benchmark. We used PostgreSQL version 9.1.13 and PostGIS 2.0.3, performing the fine tuning configurations suggested here: `http://geographica.di.uoa.gr`.

Table 3.4 shows information about the datasets described above, such as the disk size that each of these tables occupy, the number of tuples and the average number of points per geometry. Notice that the LGD dataset consists of 7 shapefiles/tables which is important in the OBDA setting as we will explain later on. Also, LGD-Places and LGD-Points contain only point geometries.

In order to compare the performance of our system with Strabon and System-X, we materialized the virtual geospatial RDF graphs produced by Ontop-spatial and stored them in Strabon and System-X, so that both the virtual RDF graphs produced by Ontop-spatial and the graphs stored in Strabon contain exactly the same information. The produced RDF dump consists of 5.620.482 triples and contains 855.502 geometries. The total PostGIS database size (in terms of disk usage) of Ontop-spatial is 700 MB. The respective size of the PostGIS database that was produced after loading the RDF dump to Strabon is 1665 MB, which is more than twice the disk space compared to the original database produced by importing the shapefiles directly. The reason is that in the first case the database stores the data, while in the second case the database stores the equivalent set of triples. This kind of overhead is common in RDF stores that use a relational database as backend. Also, Strabon inherits the *per_predicate* storage scheme of the Sesame RDBMS package, so every predicate is stored in a different table and additional tables are used for dictionary encoding. According to this storage scheme, all geometries are stored in a table called *geo_values* in WKB format and the respective column is indexed using an R-tree-over-GiST index, as described in [48].

3.6.2 Queries

The GeoSPARQL queries that we used for the experimental evaluation of our system are a set of *spatial selections* and a set of *spatial joins*. We used some of the queries of Geographica, and some queries that are appropriate in the OBDA setting as we will explain in the rest of this section. The queries used in our evaluation are presented in Tables 3.2 and 2. Each query has a numeric identifier, a mnemonic label, a number that shows how many BGPs it consists of and a number that shows how many results it returns.

Both spatial selection and spatial join queries contain a spatial filter that checks if a spatial relation holds between two geometries that are given as arguments to the respective GeoSPARQL function. In the case of spatial selections, one of the arguments is a variable and the other one is a constant, which can be either a line (queries suffixed with “L” in the query label) or a polygon (using “P” suffix). In spatial join queries, both arguments of the respective spatial binary operator are variables. The first set of queries that we consider contains simple geospatial queries, i.e., queries consisting of a single triple pattern to retrieve the geometries of a dataset and a spatial filter (spatial selections 00-14 and spatial joins 00-03). Note that spatial joins require at least two triple patterns to retrieve the ge-

ometries that will be bound to the variables that are involved in the spatial filter. This kind of queries test the response time of the compared systems to perform “pure” geospatial queries (i.e., with the minimum amount of non-spatial triple patterns involved, focusing as much as possible on the evaluation of the spatial condition).

The next set of queries that we consider tackles an important issue that is crucial in OBDA systems: the generation of `Union` operators, deriving from the ontology and the schema of the database in the SPARQL-to-SQL translation phase. For example, the LGD dataset consists of 7 shapefiles, each one containing a column where geometries are stored. But according to the ontology, the data property that connects a spatial object with its geometry is universal for all spatial objects in the dataset. We present the mappings for two of these tables/shapefiles in Listing 3.5.

Listing 3.5: Examples of geospatial mappings for two LGD tables

```
mappingId   lgd_buildings_geometry
target      lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral.
source      SELECT gid, geom FROM buildings

mappingId   lgd_landuse_geometry
target      lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral.
source      SELECT gid, geom FROM landuse
```

Listing 3.6: Template for spatial selection queries

```
SELECT ?s1 ?o1
WHERE { ?s1 lgd:asWKT ?o1 .
FILTER(geosparql:FUNCTION(SPATIAL_CONSTANT,?o1)).}
```

Listing 3.7: Spatial selection query 19

```
SELECT distinct ?s1
WHERE { ?s1 lgd:asWKT ?o1 . { {?s1 rdf:type lgd:Road}
UNION {?s1 rdf:type lgd:Waterway}.}
FILTER(geof:sfIntersects(GEOMETRY,?o1))
```

Listing 3.8: Spatial join query 6

```
SELECT ?s1 ?s2
WHERE {  ?s1 lgd:asWKT ?o1 .  ?s2 lgd:asWKT ?o2 .
FILTER(geo:sfIntersects(?o1,?o2))}
```

Let us now consider the template for spatial selection queries in Figure 3.6. The translated SQL query corresponding to a GeoSPARQL query following this template would create unions in order to fetch results deriving from all the tables it has been mapped to, that is, all seven LGD tables, and then apply the spatial selection to this union. This is the case for spatial selection queries 15-19. In order to test how our system responds by increasing/decreasing the number of unions produced in the translated query, we add an additional, thematic filter that selects a different number of LGD categories each time,

thus affecting a different number of tables, and producing different number of unions, respectively. For example, consider query 19 which is shown in Listing 3.7, which contains a union to retrieve both waterways and roads (coming from different shapefiles, thus different tables in the database).

The queries 15, 16, 17, and 18 produce 6, 4, 3, and 4 unions respectively. The presence of unions has a negative impact on the query response time, but things get even worse when unions appear in spatial joins (e.g., spatial join query 6). Since variables appear in the spatial filters that serve as the conditions of the spatial joins, all combinations of the respective tables that are involved in the corresponding mappings should be spatially joined pairwise.

For example, consider the spatial join query 6 which is given in Listing 3.8. This query performs a spatial join with the condition `intersects` in *all* LGD tables that are involved in the mappings containing the predicate *lgd:asWKT*. This join is translated into the corresponding relational algebra expression as follows:

$$(L_{buildings} \cup L_{luse} \cup ... \cup L_{waterways}) \bowtie_{sf} (L_{buildings} \cup L_{luse} \cup ... \cup L_{waterways})$$

where $L_{buildings}$, L_{luse},..., $L_{waterways}$, etc are LGD tables and sf is spatial operator corresponding to `geof:sfIntersects` from the query. The query engine evaluates this relational algebra expression as unions of joins and all involved tables get spatially joined pairwise.

Last, in order to measure how the selectivity of the queries affect the performance of the systems, we included the spatial selection queries 20 and 21 involve the computation of the intersection of all kinds of LGD areas with a specific polygon. This polygon is large in the case of spatial selection query 20 so that many geometries will be returned, while in spatial selection query 21 this polygon is small enough so that very few LGD areas intersect with it.

3.6.3 Results

Experimental set up. The experiments were carried out on a server with the the following specifications: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, 12MB L3, RAID 5, 32GB RAM and OS: Ubuntu 12.04. All experiments were carried out with both cold and warm cache. Queries are first executed in cold cache and then in warm cache. The queries for which the system under test times out (the time out threshold is set to 40 minutes) are not executed in warm cache. All queries and code we used to execute the experiments in both systems, can be found in the “experiments” branch of the github repository of Ontop-spatial (folder “benchmark”) at `https://github.com/ConstantB/Ontop-spatial`.

Query response time. The results of our experimental evaluation can be seen in Figures 3.5 - 5. Response time is measured in nanoseconds and presented in logarithmic scale. A general observation is that the query response time of Ontop-spatial is better than the one of Strabon and System-X, especially when big datasets are involved, both for spatial selections and spatial joins. Strabon times out after 40 minutes in spatial join queries 6

Table 3.2: Spatial selections description

No	Query	#BGPs	results
00	Equals_GADM_P	1	0
01	Contains_GADM_P	1	9
02	Contains_GADM_P	1	0
03	Equals_GADM_L	1	1
04	Overlaps_GADM_L	1	0
05	Contains_GADM_L	1	0
06	Intersects_CLC_L	1	5
07	Contains_CLC_L	1	0
08	Equals_CLC_L	1	5
09	Overlaps_CLC_L	1	0
10	Overlaps_CLC_P	1	132
11	Intersects_CLC_P	1	533
12	Contains_CLC_P	1	401
13	Equals_CLC_P	1	0
14	Intersects_LGD_P	2	2749
15	Intersects_LGD_B	2	2749
16	Intersects_LGD_PL	2	2626
17	Intersects_LGD_P	2	2522
18	Intersects_LGD_LU	2	2722
19	Intersects_LGD_ROA	2	2387
20	Intersects_LGD_bigP	1	729189
21	Intersects_LGD_P2	3	5

Table 3.3: Spatial joins description

No	Query	#BGPs	results
00	Within_CLC_GADM	2	34114
01	Intersects_GADM_GADM	2	1556
02	Overlaps_GADM_CLC	2	17035
03	Intersects_LGD_GADM	3	154725
04	Intersects_LGD_LGD_Mus	4	2
05	Intersects_LGD_GADM	2	819319
06	Intersects_LGD_LGD	1	3686229
07	Crosses_LGD_LGD_Roads	4	178602

Table 3.4: Workload characteristics

Dataset	Size	Tuples	$Avg\frac{\#points}{geometry}$
CLC	283MB	44834	187.84
Hotspots	35 MB	37048	5
GAG	24 MB	326	3020.14
LGD-Buildings	42 MB	155474	6.5
LGD-Landuse	20 MB	40220	19.4
LGD-Places	2.4 MB	13043	1
LGD-Points	12 MB	61664	1
LGD-Railways	2 MB	4996	13.3
LGD-Roads	250 MB	514403	19
LGD-Waterways	16 MB	20565	39.84

and 7. System-X times out after 40 minutes in spatial join queries 0,1,2,3,6 and 7. In spatial selection queries 2–5, although Ontop-spatial achieves better response time than Strabon in cold cache, it gets outperformed in warm cache, as intermediate results (which are not many as the dataset involved in this query is relatively small), are more likely to be found in the cache, increasing the hit rate of the cache and decreasing I/O requests. However, such differences between executions in warm and cold cache are eliminated in larger datasets. System-X performs worse than Ontop-spatial and Strabon in both cases.

In what follows we explain why Ontop-spatial outperforms Strabon.

Listing 3.9: Spatial join query 2

```
SELECT ?s1 ?s2
WHERE {
  ?s1 clc:asWKT ?o1 .
  ?s2 gag:asWKT ?o2 .
  FILTER(geof:sfWithin(?o1, ?o2))
  }
```

Listing 3.10: Spatial join query 4

```
 select ?s1 ?s2 where {
 ?s1 lgd:asWKT ?o1 .
?s1 rdf:type lgd:Building .
 ?s1 lgd:type "Museum" . ?s2 lgd:asWKT ?o2 .
 ?s2 rdf:type lgd:Landuse .
 filter(geof:sfIntersects(?o1,?o2))}
```

Listing 3.11: Ontop-spatial SQL query

```
SELECT 1 AS "s1QuestType", NULL AS "s1Lang",
('http://geo.linkedopendata.gr/clc/' || REPLACE(..... || '/') AS "s1",
 1 AS "s2QuestType", NULL AS "s2Lang",
('http://geo.linkedopendata.gr/gag/ont/' || REPLACE(...'/') AS "s2"
FROM clc QVIEW1, gag QVIEW2
WHERE  QVIEW1."gid" IS NOT NULL AND QVIEW1."geom" IS NOT NULL AND
QVIEW2."gid" IS NOT NULL AND QVIEW2."geometry" IS NOT NULL AND
       (ST_Within(QVIEW1."geom",QVIEW2."geometry"))
```

Listing 3.12: Strabon SQL query

```
 SELECT a0.subj, u_s2.value, a2.subj,
 u_s1.value FROM aswkt_855211 a0
 INNER JOIN geo_values l_o2 ON (l_o2.id = a0.obj)
 INNER JOIN geo_values l_o1
```

```
ON (ST_Within(l_o1.strdfgeo, l_o2.strdfgeo))
INNER JOIN aswkt_135992 a2 ON (a2.obj = l_o1.id)
LEFT JOIN uri_values u_s2 ON (u_s2.id = a0.subj)
LEFT JOIN uri_values u_s1 ON (u_s1.id = a2.subj)
```

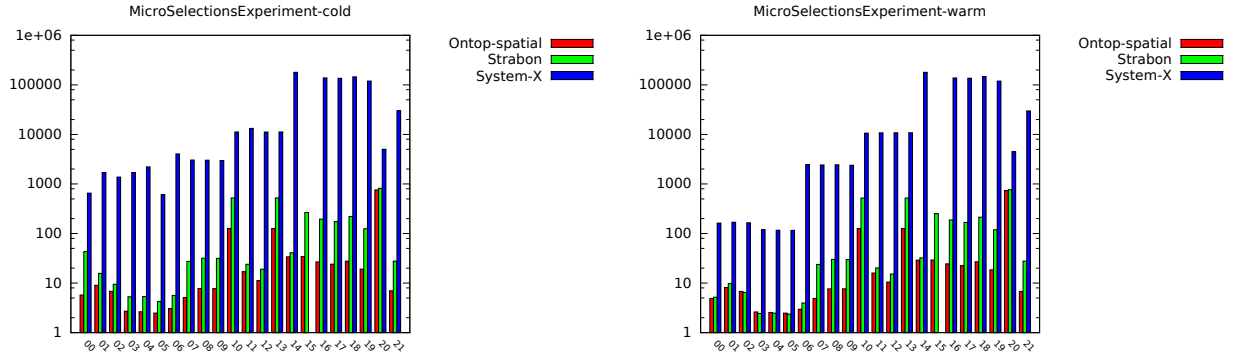


Figure 3.5: Spatial Selections experiment (cold and warm cache)

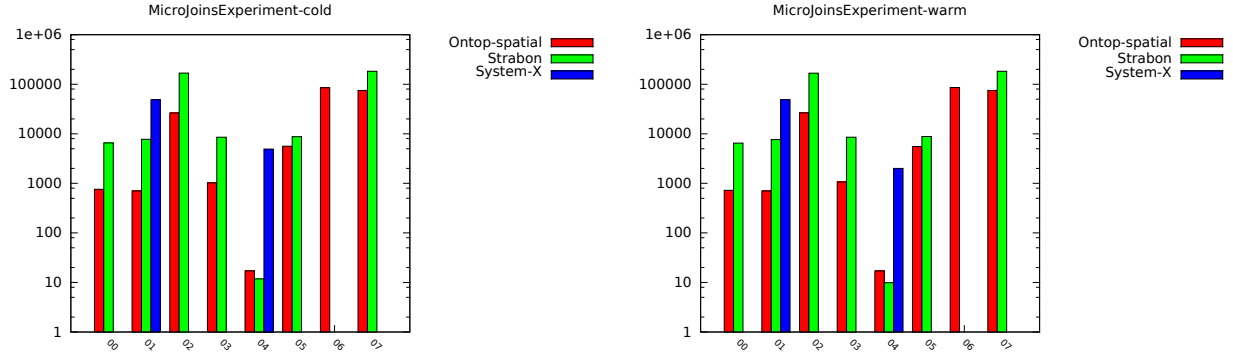


Figure 3.6: Spatial Joins experiment (cold and warm cache)

The queries provided in Listings 3.11 and 3.12 are the SQL translations of the GeoSPARQL spatial join query 2, which is provided in Listing 3.9. One can observe that Ontop-spatial produces the same query as one would have written by hand in a geospatial relational database. Strabon produces some extra joins, as a result of the star schema that it follows in the database (and has been inherited from the Sesame RDBMS that Strabon is built on), i.e., each predicate is stored in a different table and there are some additional tables used for dictionary encoding (tables storing URIs, one table for each different datatype, etc.). This has a negative impact on performance when many intermediate results are produced. In Strabon, geometries are stored in a single table, named geo_values, and are indexed on the geometry column using an R-tree-over-GiST index. On the other hand, Ontop-spatial stores each shapefile in a different table, and geometries are stored in a sepate column for each table, and a separate R-tree-over-GiST index is constructed for the geometries of each shapefile/table. As Table 3.4 shows, there are cases where geometries of a shapefile/table are of the same type (e.g., all contain points/-linestrings/polygons), allowing Ontop-spatial to build smaller and more efficient indices.

Nevertheless, in spatial join query 4, Strabon outperforms Ontop-spatial. The query is provided in Listing 3.10. Using this query, we want to retrieve the land use of areas that intersect with Museums. This is a very selective query with respect to the thematic condition, so the PostgreSQL optimizer correctly chooses to perform the thematic conditions first so that only the geometries of Museums will be checked in the spatial condition that follows, and the R-tree index will be used. Both systems execute the query very fast, with Strabon achieving nearly 4 times better performance than Ontop-spatial, as the overhead of the extra joins it performs, as described above, is reduced because very few intermediate results are produced. Also, dictionary decoding helps Strabon to perform string comparison (for value “Museum”) only once, in order to retrieve the id of that value and then perform thematic joins efficiently using the id (numeric) value.

Queries 15-19 have filters that select different kinds of LGD categories. Query response time increases every time many LGD categories are involved (Query 15 asks for all categories), producing the respective number of unions in the case of Ontop-spatial and more intermediate results for Strabon, forcing more geometries to be checked in the spatial filter. On the contrary, query response time decreases when less LGD categories need to be selected.

The results of union-queries are more interesting in the case of spatial joins, shown in Figure 5. One would expect that unions with spatial joins, as in the case of the spatial join query 6, would dramatically decrease the performance of Ontop-spatial. Indeed, query response time increases in the case of queries like query 6, but Ontop-spatial still performs better than Strabon. The explanation for this lies in the fact that each time a spatial join is performed between two different LGD tables, the optimizer chooses the one having the smaller index (and usually smaller geometries, in this case) to be nested inside the inner branch of the nested loop, where it performs an index scan. This has greater impact on the execution time of geospatial queries, as the evaluation of spatial joins is more expensive due to the cost of the evaluation of the spatial conditions.

In spatial selection query 20, the performance of the two systems is very close, while in the more selective version of the same query, i.e., spatial selection query 21, the gap in the execution times between Ontop-spatial and Strabon increases again. This happens because nearly every geometry in the workload is included in the results, so spatial indices are not useful in this case.

System-X performs worse than the other two systems in all cases mainly because of the fact that its geospatial index relies on Apache Lucene[12], that does not support R-trees but simpler forms of spatial indices, such as quad-trees. The spatially-enabled RDBMS (i.e., PostgreSQL with PostGIS extension enabled) that serves as the back-end for Ontop-spatial and Strabon, on the other hand, incorporates more advanced and mature techniques for efficient geospatial query processing which are considered stadard for the relational database community, such as support for R-tree indices and spatially-enabled optimizer.

Overall, we observe that importing the shapefiles to a database and then using an OBDA

[12]https://lucene.apache.org/core/

approach is very efficient, as in most cases, the information that is contained in a shapefile is compact and homogeneous, as we often have one shapefile per data source. So, the SQL queries that are produced based on such a schema contain reduced amount of joins and can be executed efficiently. It is also evident from the experiments that forwarding geospatial query processing to a spatially-enabled DBMS as back-end can improve performance significantly, as they incorporate well-established optimization techniques in the area of geospatial query processing that have not yet been incorporated in native geospatial triple stores up to date.

3.7 Related Work

The work on extending RDF and SPARQL with geospatial functionality also gave rise to the implementation of geospatial RDF stores such as Parliament, uSeekM and Virtuoso, that implement a subset of GeoSPARQL, and Strabon [48] that implements both GeoSPARQL and stSPARQL.

There have also been systems that enable the translation of geospatial data from their native formats to RDF. GeoTriples [49] is a tool for the conversion of geospatial data from a variety of source formats (shapefiles, relational databases, XML files, etc.) to RDF using GeoSPARQL and stSPARQL vocabularies and R2RML mappings.

Another category of systems that are related to our work is SPARQL-to-SQL systems such as Ontop [67], Ultrawrap [69], D2RQ[13] and Morph [63]. These systems offer no geospatial functionality.

In the area of description logics, the work described in [33] extends DL-Lite with spatial primitives and presents a rewriting mechanism to standard DL-Lite, preserving FOL-rewritability. The work described in [58] examines the FOL rewritability of spatial calculi (e.g., RCC8, RCC2) combined with DL-Lite. PelletSpatial [70] is a qualitative spatial reasoner implemented on top of Pellet.

3.8 Discussion and Findings

In this chapter, we describe how we extended the techniques of [67] to develop the first geospatially-enabled OBDA system, named Ontop-spatial. By extending the OBDA system Ontop, Ontop-spatial inherits the advantages of using RDB2RDF systems in real use cases: *(i)* RDB-to-RDF workflow becomes less complicated, without having to use different tools for converting data into RDF and then storing it in RDF stores, *(ii)* no data needs to be transfered, as existing databases are used as input to the system, and *(iii)* mappings provide a layer of abstraction between the data manipulation/database experts and the end users.

[13] `http://d2rq.org/`

These advantages have even greater impact when dealing with geospatial data. The domains where geospatial data are produced and used are dominated by geospatial databases and other tabular file formats that could easily be imported to a database (e.g., shapefiles). GIS practitioners use geospatial relational databases in their day-to-day tasks, either directly or as the back-end of applications to store and manipulate data (e.g., GIS have connectors for geospatial relational databases). Ontop-spatial provides a solution for combining the advantages of geospatial relational databases, for example, the wide variety of geospatial data operators and the performance achieved by the use of spatial indices, with the data modeling advantages of the RDF data model. Moreover, Ontop-spatial allows for encapsulating geospatial data manipulation functions offered by geospatial extensions to SQL (e.g., functions for transforming geometries to a different coordinate reference system) in the mappings.

On the other hand, Ontop-spatial inherits the disadvantages of the OBDA systems as well. First, in order to combine information coming from different geospatial sources, the data should be imported in databases. Second, as the database is given as input to the system, it is read-only and Ontop-spatial does not support SPARQL store or update operations; all updates should be done directly on the database level. Third, the performance of the system is heavily dependent on the ontology, the schema of the database, and the mappings, as we explained in the previous sections, which applies for OBDA approaches in general. However, our experiments showed that in many cases, our geospatially enchanced OBDA approach achieves significantly better performance than the state-of-the-art geospatial RDF store Strabon. The main reasons for this are summarized as follows:

- The database schema that is produced simply by importing the shapefiles to the database is in most cases suitable for OBDA approaches, as shapefiles contain compact and homogeneous information per dataset.
- The database produced by storing the materialized RDF dump that ontop exports in Strabon is bigger than the database that results from importing the shapefiles, even though only the RDF triples that were involved in the OBDA mappings (i.e., the *virtual* RDF triples) were exported. This happens because of i) the normalization imposed by the RDF data model itself (i.e., triples) and ii) the additional tables used for dictionary encoding.
- The additional joins that are created in the translated SQL queries of Strabon and the fact that geometries are stored in a single table where geospatial operators are performed increase even by more than an order of magnitude in very large workloads with many and complicated geometries, when many intermediate results are produced in queries.

3.9 Summary

In the work described in this chapter we introduced the system Ontop-spatial, this is now the most efficient GeoSPARQL query engine, as it is able to outperform state-of-the-art

GeoSPARQL query engines, without the need to covert geospatial relational data into RDF and store it geospatial RDF stores.

In the next chapter we present how we followed a similar approach in order to extend the OBDA paradigm with temporal support. However, unlike GeoSPARQL, there is no standard extension of the framework of RDF and SPARQL with temporal support. For this reason, as we describe in the following chapter, we follow the framework of stRDF and stSPARQL which we adjust in the OBDA setting and we develop further its temporal dimension.

4. TEMPORAL ONTOLOGY-BASED DATA ACCESS

In this chapter, we present our approach for creating virtual temporal semantic graphs on top of temporal relational databases, employing an Ontology-based data access technique that we used for the spatial case, as described in Chapter 3. The problem of posing temporal SPARQL queries on top of temporal databases on-the-fly using the OBDA paradigm is even more challenging. First, there is no standard temporal extension of the framework of RDF and SPARQL, as in the case of the OGC standard GeoSPARQL. Second, in order to support valid time in the data model RDF, none of the proposed approches for translating temporally-enhanced queries into standard SPARQL queries suits the OBDA paradigm.

The first approach is reification, that is proposed in [37]. Reification exploits the fact that every RDF triple is a resource and it can be described using the `rdf:statement` RDF primitive. Therefore, one can use additional triples to declare the subject, the predicate and the object of the triple, using the RDF properties `rdf:subject`, `rdf:predicate`, and `rdf:object` of the triple. In the same way, one can use another triple to denote the valid time of the triple. Although this is a straightforward approach that can be used to map temporally-ehanced triples into standard RDF ones using reification, the use of so many additional triples to describe a statement is not very user-friendly and also not very efficient, as it increases considerably the number of triples for a temporal dataset. To address these issues, Tappolet et al. proposed in [71] the use of named graphs to denote the valid time of triples. In this approach, a triple that has a valid time belongs to a named graph that corresponds to this valid time. Then, this named graph is associated with a triple describing the actual serialisation of the temporal value it corresponds to. This information, i.e., the temporal values of named graphs, is stored in the default graph of the dataset. The temporal dimension of the framework of stRDF and stSPARQL, as described in Section 2.2.2 is based on the named graph approach of [71]. In the OBDA setting, both approaches are challening to implement. First, the reification approach means that the user needs to write the reified syntax on their own when writing the mappings. On the other hand, OBDA systems and SPARQL-to-SQL translation techniques that have been studied so far do not support named graphs.

Another feature that the OBDA systems lack is the support for SPARQL1.1. extension functions, which is needed for the support of the temporal extension functions defined in stSPARQL that we explained in 2.2.2 and are documented in [13].

Due to these challenges, we will not consider valid time for the rest of this chapter and we will focus in addressing these issues for supporting user-defined time. We propose an extension of the data model stRDF and the query language stSPARQL with additional, but *lightweight* temporal features that can be adopted by both RDF stores and OBDA systems.

$$\texttt{strdf:periodContains}(\texttt{?s1}, \texttt{?s2}) \leftarrow \texttt{strdf:hasTime}(\texttt{?s1}, \texttt{?t1}), \texttt{strdf:hasTime}(\texttt{?s2}, \texttt{?t2}), \texttt{strdf:periodContains}(\texttt{?t1}, \texttt{?t2}).$$

Figure 4.1: The $\mathcal{R}_{periodContains}$ transformation rule

4.1 Temporal extensions of stSPARQL

We define the following two additional temporal components in the query language stSPARQL.

Temporal predicates. We extend stSPARQL by defining a set of *temporal predicates* that are based on Allen's interval algebra [3]. These predicates are the following: `strdf:peridEquals`, `strdf:after`, `before`, `periodOverlaps`, `starts`, `finishes`, `periodContains`, `strdf:meets`, `during`, and `isMetBy`. The temporal predicates are equivalent to the temporal extension functions that are defined in stSPARQL and are described in [13]. These functions can either operate on intervals or time instants, where suits the case. For example, an interval can either contain another time interval (e.g., a literal of the `strdf:period` datatype or a literal of the `xsd:dateTime` datatype).

Temporal query rewrite component. The temporal query rewrite component of stSPARQL defines a set of rules for translating *qualitative* temporal queries, i.e., queries with temporal predicates, into *quantitative* ones, i.e., queries with temporal operators. This component of stSPARQL is similar to the query rewrite component of GeoSPARQL [26]. We denote as $R_{temporal}$ the set of rewriting rules R_i that we define for each temporal predicate i. Using these rules, a query q that contains a temporal predicate i will get transformed into the equivalent query q, by applying rule R_i to q. The query q' contains the temporal extension function of stSPARQL that corresponds to the temporal operator i. In the following section we provide an example of how this new query rewriting component of stSPARQL participates in the overall evaluation of temporal stSPARQL queries. This component of stSPARQL is important, as it allows any OBDA system that implements it to answer temporal SPARQL queries transparently, without modifying their syntax.

Rewriting rules. We now explain how queries that contain temporal predicates (qualitative) get translated into queries that contain functions. We define a rule for each temporal prediate that we define, that corrseponds to a temporal function in stSPARQL. Table 4.1 describes this mapping. For example, in Figure 4.1 we describe the rule that translates statement patterns that contain the temporal predicate `strdf:periodContains` into statement patterns that contain the respective quantitative function `strdf:periodContains` of stSPARQL. In the rest of this chapter, we explain how we implement this set of rules in the OBDA setting and how this feature enables the creation of virtual temporal graphs on top of temporal relational data. Notably, this extension of the temporal dimension of stSPARQL does not only apply for the OBDA setting, but it can be implemented in RDF stores with temporal features as well, in order to provide temporal support without extending the standard SPARQL syntax.

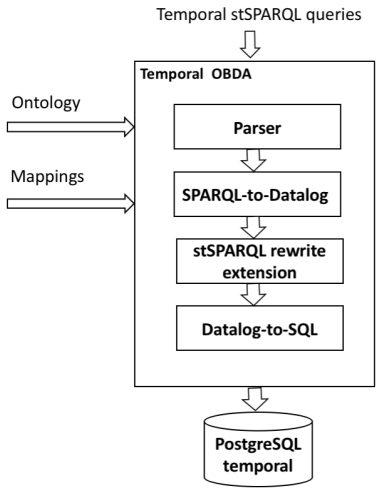


Figure 4.2: Workflow of the execution of temporal queries

4.2 Translation of temporal stSPARQL queries to SQL

In this section we describe the workflow of answering stSPARQL queries in an OBDA system like Ontop [67]. As we have already explained in Chapter 3, the system Ontop introduces an intermediate layer between the syntax level (SPARQL) and the data level (SQL), which is the Datalog layer. Once a temporal stSPARQL query q that contains the temporal predicate i arrives, it gets processed as follows:

- First, the query gets parsed as standard SPARQL query, as the temporal predicates are standard RDF predicates. This ensures compliance with all OBDA engines since they support standard SPARQL syntax.

- Second, the query gets translated into datalog, and so as the mappings. To achieve this, we have defined a set of temporal datalog predicates that correspond to the temporal predicates and temporal operators defined in stSPARQL. These operators and their equivalent stSPARQL and SQL operators are shown in Table 4.1.

- In the next step, the datalog representation of the query q gets translated into the datalog representation of query q', by applying rule R_i to q. The datalog representation of the query q' now contains the stSPARQL temporal operator that corresponds to the predicate i.

- Then, the datalog program that is created in the previous step, that contains the transformed query q' and the mappings (as described in [67], gets translated into SQL. To achieve this, we have map each datalog temporal predicate to the corresponding temporal SQL operator, as shown in Table 4.1.

The workflow of the temporal query execution in Ontop-spatial is shown in Figure 4.2.

Table 4.1: Mapping stSPARQL temporal operators to SQL and Datalog

Operator	stSPARQL	Datalog
`equals(period1, period2)`	strdf:periodEquals	PERIODEQ
after(period1, period2)	strdf:after	AFTER
before(period1, period2)	strdf:before	BEFORE
overlaps(period1, period2)	strdf:PeriodOverlaps	PERIODOVERLAPS
starts(period1, period2)	strdf:starts	STARTS
finishes(period1, period2)	strdf:finishes	FINISHES
periodContains(period1, period2)	strdf:periodContains	PERIODCONTAINS
meets(period1, period2)	strdf:meets	MEETS
during(period1, period2)	strdf:during	DURING
isMetBy(period1, period2)	strdf:isMetBy	ISMETBY

4.3 Example

In this section we provide an example that showcases the approach that we presented in this chapter. We assume the temporal relational table shown in 4.2. This is a PostgreSQL table that is temporally-enabled (it supports the PostgreSQL temporal extension). The datatype of the column named `duration` is the `period` datatype supported PostgreSQL-temporal and it is used to denote intervals. Timestamps can also be stored in this table, as intervals with the same beginning and end.

id (Integer)	name (varchar)	duration (Period)

Table 4.2: Schema of table Meeting

The mappings shown in Listing 4.1 describe how the relational data of Table 4.2. The source part of the mapping consists of a simple SQL query that retrieves all columns of the table described in 4.1. The target part of the mappings consists of a set of virtual triple templates. In the third triple template, the predicate `strdf:hasTime` is used to connect an entity to the time it is valid for. This time can either be an interval or a timestamp, i.e., a literal of the `strdf:period` datatype, or the `xsd:dateTime` datatype respectively. As in this case we have intervals, the values stored in the `duration` column of the table are mapped into literals of the datatype `strdf:period`.

We use the predicate `strdf:hasTime` as a generic predicate that can capture any temporal dimension, i.e., valid time, transaction time, or user-define time. Our method is generic enough to capture all different time dimensions. For example, if we would like to represent valid time, we would use the predicate `strdf:hasValidTime` instead.

Listing 4.1: Temporal mappings

```
mappingId Meeting-Mapping
target strdf:Meeting/{id} a strdf:Meeting ;
       strdf:hasId {id} ;strdf:hasName {name} ;
strdf:hasTime {duration}^^strdf:period .
source SELECT id, name, duration FROM meeting
```

Let us now pose a temporal query over the virtual graph that will be generated based on the mappings provided in Listing 4.1. The query is described in Listing 4.2 and it retrieves meetins whose durations overlap with each other.

Listing 4.2: Temporal stSPARQL query

```
SELECT DISTINCT *
WHERE { ?x1 a strdf:Meeting . ?x2 a strdf:Meeting .
        ?x1 strdf:hasTime ?p1 . ?x2 strdf:hasTime ?p2 .
        ?p1 strdf:contains ?p2  . ?x1 strdf:hasId ?id1 .
        ?x2 strdf:hasId ?id2}
```

Listing 4.3: Datalog translation of temporal query

```
ans10000000000(x1,x2) :- TemporalMeeting.owl#Meeting(x1), TemporalMeeting.owl#Meeting(x2)
ans1000000000(p1,x1,x2) :- ans10000000000(x1,x2), TemporalMeeting.owl#hasTime(x1,p1)
ans100000000(p1,p2,x1,x2) :- ans1000000000(p1,x1,x2), TemporalMeeting.owl#hasTime(x2,p2)
ans10000000(p1,p2,x1,x2) :- ans100000000(p1,p2,x1,x2), TemporalMeeting.owl#contains(p1,p2)
ans1000000(p1,p2,id1,x1,x2) :- ans10000000(p1,p2,x1,x2), TemporalMeeting.owl#hasId(x1,id1)
ans100000(p1,p2,id2,id1,x1,x2) :- ans1000000(p1,p2,id1,x1,x2), TemporalMeeting.owl#hasId(x2,id2)
ans10000(p1,p2,id2,id1,x1,x2) :- ans100000(p1,p2,id2,id1,x1,x2), NEQ(id1,id2)
ans10(x1,x2,p1,p2,id1,id2) :- ans100(p1,p2,id2,id1,x1,x2)
ans1(x1,x2,p1,p2,id1,id2) :- ans10(x1,x2,p1,p2,id1,id2)
```

Listing 4.4: Translated temporal SQL query

```
SELECT *
FROM public.meeting qview1, public.meeting qview2
WHERE qview1.id IS NOT NULL AND qview2.id IS NOT NULL AND
(qview1.duration @> qview2.duration) AND (qview2.id <> qview1.id) AND
qview1.duration IS NOT NULL AND qview2.duration IS NOT NULL;
```

4.4 Summary

In this chapter we describe how we used and extended the framework of stRDF and stSPARQL with more temporal features and we also documented how these features were implemented in the temporal extension of the system Ontop-spatial. In the following chapter we tackle the challenging problem of extending the OBDA paradigm to support SPARQL queries over data that is not materialised in a relational database, but exists in the Web in various kinds of sources, such as HTML tables and Rest API's.

5. QUERYING THE WEB USING ONTOLOGIES AND MAPPINGS

This chapter describes how we further extended the OBDA paradigm with the capability to pose SPARQL queries on top of heterogeneous data sources available on the Web in different formats, such as Web APIs, HTML tables, etc. The work that is presented in this chapter is also covered in [12].

5.1 Introduction

Ever since its creation, the World Wide Web has been populated with a constantly increasing amount of data. In the era of the Web of Data, being able to retrieve information from different sources and correlate it to extract knowledge is of paramount importance. The linked data paradigm was proposed with the purpose of integrating RDF data coming from different sources so that they can be queried through the SPARQL query language [40].

Despite the rapidly increasing availability of open data as linked data on the Web, a lot of data is still published in other, non-RDF formats, such as HTML forms/tables, or via Rest APIs [55, 54]. To use this data as linked data, one should transform it into RDF triples to be stored in a triple store. Depending on the format of this data, this transformation can be performed either by using a specialized tool, or by writing custom code for retrieving the data from the Web and materializing the corresponding RDF triples. However, this can be a challenging task in cases where the data is large and/or the data sources get frequently updated.

During the last decade, the problem has been *partially* addressed by the Ontology-based Data Access (OBDA) paradigm [21]. In essence, the OBDA systems create virtual RDF graphs on top of relational data using ontologies and mappings. The mappings encode how the relational data is mapped into the RDF terms that are described in the ontology. Various mapping languages exist for encoding mappings, with R2RML being recently established as a W3C standard [29]. Using ontologies and mappings, OBDA systems process SPARQL queries by translating them into SQL queries that are executed in the underlying DBMS. The results are translated into the respective RDF terms so that OBDA systems can be used as triple stores - except that they do not need to transform and materialize the data as RDF triples. The OBDA paradigm is particularly useful when the amount of data to be processed is large and/or the database is updated frequently [75].

Recent research efforts have concentrated on the *automatic* conversion of more data formats into RDF, aiming to reduce the transformation overhead and to facilitate the synchronization of the RDF versions of the data with its original sources [75, 31, 23, 54, 55]. One of the highlights of these efforts is the creation of the mapping language RML [31], a superset of R2RML that captures data transformation for many formats other than relational tables. Indeed, the adoption of RML in various use cases has significantly simplified and improved their performance [52, 50].

In this work, we consider a different problem, namely the task of *querying on-the-fly non-RDF web data using SPARQL* [55, 54, 23]. To address it, [55] proposes an extension of the query language SPARQL to query RDF data combined with data coming from Web APIs as JSON files. [54] proposes an architecture based on micro-services that extends the SPARQL protocol with the ability to query APIs on-the-fly. Finally, [23] proposes an extension of the R2RML mapping language that is inspired by RML and aims at providing primitives for querying different kinds of data sources available on the Web, such as APIs.

However, these works merely support relational data or specific file formats (e.g., XML, CSV). They also rely on custom SPARQL/R2RML extensions that hamper their adoption, while their extension with third-party added-value services is a very complicated procedure. Lastly, some of them implement a caching mechanism [55, 54], but they cannot make the most of it, as demonstrated by our experimental evaluation in Section 5.6.3.

In this chapter, we go beyond these state-of-the-art works, introducing a framework for extending existing OBDA techniques to support querying of data from different sources that are available on the Web, such as webtables and Rest APIs. It relies on virtual relational tables that allow for executing any SPARQL query on top of an OBDA system, using the necessary ontology and mappings, but without requiring the data to be available a-priori, i.e., before the query is posed. This approach offers a series of unique characteristics: (i) It accommodates any format of web data, like the increasingly popular HTML tables and the omnipresent REST APIs. (ii) It is easy to use and incorporate into any Semantic Web application, as it relies on standard SPARQL and R2RML (and its equivalents). (iii) It is transparent to the user, allowing the seamless enrichment of retrieved data with third-party added-value services (e.g., sentiment analysis). (iv) It is suitable for non-relational web data with frequent updates. (v) It is able to fully exploit an effective caching mechanism.

In short, the contributions to the state-of-the-art described in this chapter are as follows:
• We propose an OBDA-based framework for posing SPARQL queries on top of non-RDF Web on-the-fly, i.e., without requiring them to be a-priori imported or downloaded - they are fetched at query time. To achieve this, virtual table operators are embedded in SQL queries that take part in R2RML mappings. These mappings specify which part and source of web data will be fetched as well as how they will be mapped to virtual RDF terms. Combining these mappings with an ontology allows for returning the virtual relational data that are involved in the query as RDF results.

• We showcase the applicability of our approach in three use cases that (i) involve significant amount of crowd-sourced information, (ii) are widely used by application developers, and (iii) get updated so frequently that a snapshot of the respective information at a given time might become outdated soon. Using traditional OBDA approaches, we would have to convert the data into RDF every time it gets updated, storing it in a triple store, or to ingest the data in a DBMS when it gets updated, using an OBDA system to query the data. Our aim is not to outperform existing SPARQL query engines, but to complement them, targeting the *velocity* and *variety* dimensions of the Web of data, rather than the *volume*.

• We provide a thorough experimental evaluation of our approach, demonstrating its feasibility and scalability in three, realistic, highly diverse and demanding applications we

consider. The results show that our approach is able to process queries on webtables of up to 100,000 rows in size within minutes. We also compared the performance of our approach to the state-of-the-art method described in [55], with the results verifying that our framework provides more functionality, while being more efficient, as well.

The rest of the chapter is organized as follows: Section 5.2 discusses the state-of-the-art in the field, while Section 5.3 describes our approach and methodology. In Section 5.4, we document the implementation of our approach, which is applied to three practical scenarios in Section 5.5. Section 5.6 presents our experimental evaluation, whereas Section 5.7 concludes the paper along with directions for future work.

5.2 Related Work

OBDA systems are primarily useful in cases where users store their data in relational databases, but do not want to materialize them as RDF triples, particularly when these databases are large or/and get frequently updated [21]. As a result, many OBDA and RDB2RDF systems have been developed in the recent years, such as Ontop [20], Ultrawrap [69], Morph [63], Sparqlify[1], and Oracle Spatial and Graph[2]. These systems are able to connect to *existing* relational data sources and create virtual RDF graphs using ontologies and mappings. The common assumption of these systems is that the data source should already exists and thus, connection details should be provided in the mappings. Most of them support the R2RML mapping language or provide translators from their native mappings languages to R2RML. For example, Ontop also supports its own native OBDA language for encoding mappings. Once connected to the data source, OBDA systems make the most of the underlying database by collecting information about data characteristics (e.g., statistics, constraints).

On another line of research, there are RDB2RDF systems that focus on converting data into RDF using mappings producing RDF dumps. Initially, only relational data sources were supported through the R2RML language [29]. Given, though, that data can be found in many formats other than relational, the RML language was created as a superset of R2RML, encoding how various data formats, like XML and CSV, can be mapped to RDF triples [31]. Another recent work in this direction is the approach described in [50], which aims at converting Web data from various formats (e.g., CSV, JSON) into RDF, using SPARQL queries - SPARQL 1.1 primitives and extension functions were extended, too.

A recent work described in [23] proposed a mapping language called D2RML which is inspired from R2RML and RML. This work extends R2RML to support more data formats, including REST APIs. Although an implementation of a D2RML processor exists, it is not part of a standalone SPARQL query engine, to the best of our knowledge.

Closer to our work is the approach presented in [55]. It proposes an extension of SPARQL

[1] `http://aksw.org/Projects/Sparqlify.html`

[2] `http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html`

that enables users to combine the responses of JSON APIs with results from the evaluation of standard triple patterns. We deviate from this approach in that: (i) we do not extend SPARQL syntax, (ii) we allow users to query APIs using standard SPARQL triple patterns directly, without having to combine them with stored RDF data, (iii) we provide a general approach that is not limited to JSON APIs, and (iv) we produce significantly fewer API calls that the current state-of-the-art [55], which translates to improved performance. In Section 5.6.3, we provide a detailed functionality and performance comparison between the two approaches.

Another work close to ours is described in [54]. It proposes an architecture that is based on the development of SPARQL wrappers for Web APIs. To this end, it extends HTTP requests to SPARQL endpoints to include arguments that are used to retrieve a fragment of the data that can be accessed via the Web API. This fragment is converted into RDF and stored using an in-memory triple store. In this way, the SPARQL query that is contained in the original SPARQL HTTP request is evaluated against the RDF graph that is stored in the triple store, which is only a fragment of the original dataset. This fragment can be considered as a linked data fragment (LDF) interface, as described in [72]. Note that the original linked data fragment approach considers the evaluation of single triple patterns on the server-side, leaving the rest to the client, in order to improve the sustainability of linked data endpoints [72]. However, there is no limit to the expressivity of queries that can be executed on the server in [54].

In short, [54] converts a fragment of the dataset into RDF and stores the converted data into an in-memory triple store. In our approach, the conversion is performed on-the-fly using mappings and an in-memory virtual table is constructed instead. In the former case, the schema of the virtual RDF terms can be changed only by modifying part of the system code, whereas in our approach, it suffices to change the mapping file. Additionally, [54] requires the user to specify the fragment of the Web API that will be accessed using SPARQL passing through the SPARQL endpoint parameters, which depends on the API. This means that the end user should be fully aware of the Web API documentation, whereas in our case, the query level is completely transparent to the end-user.

5.3 Approach

We begin with background information on the formalisation of RDF and SPARQL (Section 5.3.1) as well as the Ontology-based data access paradigm (Section 5.3.2). This preliminary knowledge forms the basis, on which we build our approach. In Section 5.3.3, we explain how we extend SQL with virtual table operators that access different data sources available on the Web, while Section 5.3.4 explains how we evaluate standard SPARQL queries on top of web APIs. Section 5.3.5 summarizes our methodology.

5.3.1 RDF and SPARQL

We denote as I, B and L the pairwise disjoint infinite sets of IRIs, blank nodes and literals, respectively. V stands for the infinite set of varibles that are disjoint from I, B and L. Based on [40], we provide the following definitions:

RDF triple. An RDF triple is an element of the form (s, p, o) of $(I \cup B) \times I \times (I \cup B \cup L)$, where s is the subject, p is the predicate and o is the object of the triple.

RDF graph. An RDF graph is finite set of RDF triples.

Triple pattern. A triple pattern is an element of the form $(I \cup L) \cup V) \times (I \cup V) \times (T \cup V)$.

Graph pattern. A (basic) graph pattern (BGP) is a finite set of triple patterns.

Evaluation of triple patterns over an RDF graph. Let D be an RDF graph over $I \cup B \cup L$, t a triple pattern and $P1, P2$ graph patterns. The evaluation of a graph pattern over D, denoted by $[[.]]_D$, is defined recursively as follows:

- $[[t]]_D = \{\mu | dom(\mu) = var(t)$ and $\mu(t) \in D\}$, where $var(t)$ is a set of variables occurring in t.
- $[[(P_1$ AND $P_2)]]_D = [[P_1]]_D \bowtie [[P_2]]_D$.
- $[[(P_1$ OPT $P_2)]]_D = [[P_1]]_D \mathbin{⟕} [[P_2]]_D$.
- $[[(P_1$ UNION $P_2)]]_D = [[P_1]]_D \cup [[P_2]]_D$.

The mapping μ is a partial function $\mu : V \mapsto (I \cup B \cup L)$. We denote as $\mu(t)$ the triple obtained if we replace every variable u of the variables included in t (i.e., $var(t)$) with their bindings according to μ, i.e., $\mu(u)$.

5.3.2 Ontology-based data access

The ontology-based data access paradigm [75] proposes the creation of virtual RDF graphs on top of relational databases using ontologies and mappings. Given a database schema S, an ontology O, and a set of mappings M, an OBDA specification is defined as $P = (O, M, S)$. Then, an *OBDA instance* (P, D) is defined given the OBDA specification P and the database D that follows the database schema S. Mappings encode how relational data get mapped into RDF terms. A virtual RDF graph $VG_{M,D}$ of the database instance D is produced if we apply the mappings M to D. Then, if $[[Q]]_{(P,D)}$ is the evaluation of the SPARQL query Q over the OBDA instance (P, D), it is equavalent to $[[Q]]_{(P,VG_{(M,D)})}$.

R2RML [29] is a W3C standard language for encoding mappings. Nevertheless, many OBDA systems support their own native mappings languages. In this work, we will present examples on mappings using the native language of Ontop [20], as it is more compact and user-friendly, combining brevity with readability.

5.3.3 Extending SQL with virtual table operators

The core concept of our approach is to model a data source as a virtual relational table. For this reason, we define a virtual table operator for each kind of data source. Each *virtual table operator* has the syntax: **VT** $::=$ **vtable**($\mathbf{args}[][,\mathbf{f}]$), where the vector $args$ denotes the arguments that are given as input to the virtual table operator, while f is optional, denoting the *cache update rate*.

To understand the form of the SQL queries that use virtual tables, consider the extension of the SQL syntax provided in Listing 5.1.

Listing 5.1: SQL syntax for virtual tables

```
<query specification> ::= SELECT [ <set quantifier> ] <select list>
                                  <table expression>
<table expression>    ::=  <from clause> [ <where clause> ]
                          [ <group by clause> ] [ <having clause> ]
<from clause>         ::= FROM <table references>
<table references>    ::= <table reference>
                           [ { <comma> <table reference> }..]  |
                           vtable_operator_name(args[,f])
```

We extend the SQL syntax provided in Listing 5.1 with virtual table support as shown in the last lines. The SQL standard defines two types of tables: the *base* ones, which are materialized in a database, and the *derived* ones, which are produced from relational algebra expressions. At the relational algebra level, $vtable$ is just another relational algebra operator. Thus, we consider virtual tables generated by virtual table operators as another kind of derived tables, and any mapping language that is able to use SQL queries in mappings (e.g., R2RML, OBDA) is compatible.

To improve performance, each virtual table can optionally use a cache. The cache feature is useful in cases where: (i) not all data sources get updated with the same frequency, (ii) some data sources might not be accessible at the next query time (e.g., due to API limitations), or (iii) a minimal query execution time is required, due to a large number of queries, i.e., the frequency of queries is much higher than the update frequency of data sources. To support these cases, f indicates the length of the time window (in milliseconds), during which the retrieved data are temporarily stored. If the virtual table operator with the *same* input parameters ($args$) is invoked twice (or more) before this time window ends, the cached data will be used, improving query time. If the query is repeated after the end of the time window, the fresh data is fetched from the data source and gets stored in the system. If f has a negative value or is completely absent, nothing is stored and the virtual table operator fetches fresh data every time it is invoked. To support this functionality, we store meta-data that contain information about when and where data resulting from a virtual table signature was stored last time.

Since our approach and our caching mechanism deviates considerably from related works [55, 54], we now explain in more detail how virtual tables work. Each virtual table operator is implemented differently, but a generalized description is provided in Algorithm 2. First,

Algorithm 2 Virtual Table Operator

Input: $args[]$, frequency f
Output: T, the generated virtual table

1: $T \leftarrow \emptyset$
2: $t \leftarrow getLastUpdate(args[])$
3: **if** $|t - NOW| < f$ **then**
4: $\quad T \leftarrow getTableFromCache(args[])$ ▷ Get table from cache if current
5: **return** T;
6: **end if**
7: $E \leftarrow retrieveData(args[])$
8: **for** $e \in E$ **do**
9: $\quad row \leftarrow \{tupleID\}$
10: $\quad W \leftarrow getAttributes(e)$
11: $\quad$ **for** $w \in W'$ **do**
12: $\quad\quad w'[] \leftarrow process(w)$ ▷ Process attributes of tuple
13: $\quad\quad row \leftarrow row \cup \{w'[]\}$
14: $\quad$ **end for**
15: $\quad T \leftarrow T \cup row$
16: **end for**
17: $UpdateCache(args[], NOW)$ ▷ Store the new data in cache for time f
18: **return** T

the operator checks the time the last query with the same arguments was executed (Line 3). If it is within the given cache update rate, f, the already retrieved results are returned as output (Lines 4-6). Otherwise, the operator retrieves the data from scratch, using the given arguments (Line 7). For each record, it creates a new tuple with a unique id (Lines 8-9). Next, it iterates over its attribute values, adding them to the tuple after the necessary processing (Lines 11-14). Note that the functionality of the `processAttribute` function ranges from simple tasks (e.g., data manipulation functions like value transformation/correction), to more complicated tasks (e.g., data mining tasks), as we explain in Section 5.5.2. For this reason, w' may contain more than one element. For example, it can be the text of a tweet together with information about its polarity, i.e., whether its sentiment is positive or negative. Finally, after all tuples have been processed and added to the virtual table (Line 13), the cache is updated (Line 15) and the table is returned as output.

The result of a virtual table operator is a virtual table with the following schema: **VT**[**tupleID**, **cols**], where $tupleID$ is the unique identifier of a tuple and $cols$ are the requested attributes. Note that some of these attributes might not exist originally in the data source, but they could introduce new knowledge derived from processing of the original data (as we explain in Section 5.5.2).

5.3.4 Evaluation of SPARQL queries on top of web APIs

As described above, the semantics of RDF and SPARQL [40] assume that the evaluation of SPARQL queries is performed over an RDF knowledge base and the OBDA paradigm [75] defines the creation of virtual RDF graphs on top of materialised databases for which the schema is known a-priori.

Our aim is to support the evaluation of SPARQL queries on top of different kinds of web data (e.g., APIs, webtables, etc.) without extending SPARQL or the mapping languages, as suggested by the related work [55, 54, 23]. Instead, we extend the OBDA paradigm to support virtual relational data, for which the schema is not known a-priori, i.e., not before a SPARQL query is fired.

Let us now model the response of an API call as a set of sets of $< attribute, value >$ pairs. Let $ATTR$ be the set of all attributes of a response of an API call. For each $attr_i \in ATTR_I \subset ATTR$, we define a mapping $\mu : attr_i \mapsto pred_i$ that maps $attr_i$ to a virtual predicate $pred_i \in I$. Then, the value of $attr_i$ defines obj_i as follows:
$obj_i := \mu(v(attr_i))|\mu(URI(v(attr_i)))$, where $v(attr_i)$ is the value of $attr_i$ and $URI(v(attr_i))$ is a URI template populated by the (API) value of $attr_i$, as the object of a triple can either be a literal or a URI. All URI templates are defined in the mappings.

Then, we create a virtual graph $VG_{API,M}$ that consists of triples of the form $(subj_i, pred_i, obj_i)$. The evaluation of a SPARQL triple pattern t over a virtual RDF graph on top of an API given the set of mappings M, is provided below:
$[[t]]_{VG_{API,M}} = \{\mu | dom(\mu) = var(t)$ and $\mu(t) \in VG_{API,M}\}$

Notably, although we only mention web APIs as a data source in this section, the same approach applies to other non-RDF data sources as well, such as HTML tables, as we explain in Section 5.5.

5.3.5 Methodology

We now describe the steps that should be performed in order to pose SPARQL queries to non-RDF data sources on-the-fly with the help of the virtual table operator.

1) We construct an ontology that models the data of interest.

2) We create a virtual table operator (if it is not available) for the data source at hand (e.g., a specific REST API), applying Algorithm 2.

3) We create the mappings, where the source part comprises an *extended-SQL query*, i.e., an SQL query that uses the virtual table operator for the selected data source along with the respective parameters.

3) Given the ontology and the mappings, we set up an *OBDA repository* in combination with an SQL engine that is able to process the extended-SQL queries included in the mappings. Note that the selected OBDA system should be (made) “database-agnostic” in the sense that it does not require access to the data beforehand. This feature goes beyond

the existing RDB2RDF/OBDA systems, which require that the data to be mapped already reside in a database, to which they connect in order to a-priori extract data characteristics (e.g., schema, integrity constraints, primary keys) [69, 67]. In our case, the data is fetched on-the-fly, after a SPARQL query is fired.

4) Once a SPARQL query arrives, the OBDA system translates it to SQL. The resulting SQL embeds the virtual table operator(s) involved in the query. By the time these operators are invoked as part of the extended-SQL query evaluation, the involved data source(s) will be accessed and made available as virtual table(s). Next, the query result returns back to the OBDA system to be presented as virtual RDF terms.

5) If the OBDA system supports reasoning, the reasoning process will also be applied for the new data sources (e.g., OWL-QL reasoning is performed in [67]).

Given that our approach is generic, we do not associate it with a specific mapping language or OBDA system. Instead, we set the specifications such that, once they are met, any RDB2RDF mapping language or system can implement our approach. Our own implementation is described in Section 3.5, followed by three examples that apply and extend it in Section 5.5.

5.4 Architecture and Implementation

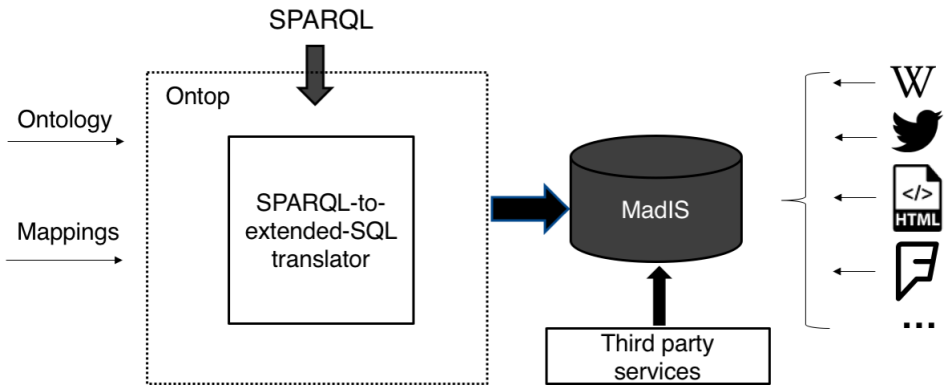


Figure 5.1: System architecture.

We now describe the implementation of the methodology described above. As shown in Figure 5.1, its architecture consists of the following components:

• As back-end, it uses the MadIS[3] [24] system, an extensible relational database system built on top of the SQLite[4] database, with extensions implemented in Python via the SQLite wrapper APSW[5]. The SQLite database can be extended with user-defined operators that

[3] http://madgik.github.io/madis
[4] http://www.sqlite.org
[5] https://github.com/rogerbinns/apsw

can be used as row, aggregate, or virtual table operators. The APSW SQLite wrapper provides an interface for implementing these operators in an extensible way through Python. Using MadIS, we define our own operators to create virtual tables and populate them with data that we retrieve from the Web. To query them, we use MadQL, the MadIS implementation of the extended-SQL language we described above, which contains the virtual table operators. We implemented a MadIS virtual table operator for each of the data sources we query in Section 5.5 (i.e., Twitter, Foursquare, webtables). A virtual table operator for webtables is already provided by MadIS, but we extended it so that it implements Algorithm 2. Instead of using MadIS, we could implement the same virtual table operators in C, extending SQLite directly, but this would be less user-friendly and re-usable than the plug-and-play MadIS Python operators and it would affect the modularity and extensibility of the architecture that we propose.

• Third party applications are external micro-services that could be invoked by a virtual table operator in MadIS. For example, in the Twitter use case, the `twitterapi` virtual table operator communicates with a Sentiment Analysis classifier to identify the sentiment of each tweet (see Section 5.5.2). In this way, we suggest an architecture for performing data analysis tasks that eliminates compatibility issues between the virtual table operator and any data analysis software: the server can be written in any language or platform, but the client can still use it as a service.

• The system Ontop[6] [20], a state-of-the-art, open-source OBDA system that supports both R2RML and the OBDA mapping language. Most specifically, we extended its geospatial extension named Ontop-spatial [10] in order to have geospatial support. To this end, we extended the MadIS JDBC connector so that it complies with Ontop, while Ontop was extended to use MadIS as a back-end. The latter modification is the most significant one, enabling Ontop to operate in a "database-agnostic" manner that supports non-materialized databases and relies on MadIS as back-end. The reason is that Ontop, like all other OBDA systems, originally connects only with populated and materialized databases, using their data for optimization, *before* a query is actually fired. Instead, our framework retrieves data only *after* a query is fired, creating a virtual table on-the-fly. As a result, no prior knowledge of the data can be used.

5.5 Practical Scenarios

We now showcase how we can pose SPARQL queries on data coming from HTML tables or REST APIs using ontologies and mappings.

5.5.1 Querying Webtables on-the-fly with SPARQL

HTML tables constitute one of the most common tabular formats for publishing data on the Web. A lot of research activities and applications have focused on retrieving, min-

[6] `https://github.com/ontop/ontop`

Figure 5.2: Tables with 100 movies from Rotten Tomatoes and Wikipedia.

ing, annotating, and semantically-enriching information available in webtables [66]. As an example, consider a semantic-based recommendation engine that tries to address the cold-start problem for new users. To make meaningful suggestions for users with no history and an empty profile, it uses the American Film Institute list of the 100 best movies from Wikipedia[7] in combination with the latest list of user reviews from Rotten Tomatoes[8], as shown in Figure 5.2. This is expressed with the SPARQL query described in Listing 5.2.

Listing 5.2: Querying webtables using SPARQL

```
PREFIX wiki: <http://en.wikipedia.org/movies/ontology#>
PREFIX r: <http://www.rottentomatoes.com/top/bestofrt/>

select distinct  ?title ?rrank ?wrank
where { ?s r:title ?title .
        ?s2 wiki:title ?title .
        ?s r:rank ?rrank .
        ?s2 wiki:rank ?wrank  }
```

The SPARQL query provided in Listing 5.2 retrieves the titles of movies that are included in both tables and the respective ranks. This is performed by executing a join on the “title” column of both tables. We now explain how we can accommodate this application using our approach to *query* data contained in HTML tables based on ontologies and mappings.

First, we use the virtual table operator `webtable`, extending the respective MadIS operator. This operator creates a virtual table and populates it with data contained in the HTML table that is given as input so that this data can be queried using MadQL queries. These queries can then be embedded in mappings as a data source, so that virtual RDF graphs can be created.

The mappings provided in Listing 5.3 describe how the information contained in these tables is translated into RDF terms. From the Rotten Tomatoes webtable, we retrieve the rank number of reviews along with the title of the film. From the Wikipedia webtable, we retrieve the title, the ranks for years 1998 and 2007 and the release date. To retrieve this

[7] http://en.wikipedia.org/wiki/AFI%27s_100_Years...100_Movies
[8] http://www.rottentomatoes.com/top/bestofrt/

information, we use the `webtable` virtual table operator that parses an HTML table and returns the results as a virtual table. The MadQL query that uses this operator can be seen in both mappings. Its first argument is the HTML page that contains the respective webtable, while the second one is the index of the webtable in the page. In our example, we want the third HTML table that appears in the Rotten Tomatoes page and the second one that appears in the respective Wikipedia page.

Note that the Rotten Tomatoes website includes the release date of every film in parenthesis next to the film title, while the Wikipedia table provides it in a separate column. Since we want to join the two webtables on the "Title" field, we align this attribute so that it has the same format in both tables. To achieve this, we concatenate the columns "Title" and "Release year" of the Wikipedia table so that the format of the resulting title is exactly the same with the one in the Rotten Tomatoes webtable.

Listing 5.3: Mappings for webtables

```
mappingId webtable_rotten_tomatoes
target rot:{rank} rot:rank {rank}; rot:title {Title};
       rot:reviews  {reviews}^^xsd:int;
       rot:rating {RatingTomatometer}^^xsd:int .
source select  rid as rank, "No. of Reviews" as reviews, Title,
       RatingTomatometer from
       webtable('http://www.rottentomatoes.com/top/bestofrt/',3)

mappingId webtable_wikipedia
target wiki:{rid}  wiki:title {Title};  wiki:rank98 {rank98}^^xsd:int ;
       wiki:rank {rank} .
source select rid, rank,Title from (select rid,Film||"
       ("||"Release year"||")"as Title, "2007 rank" as rank from
        webtable('http://en.wikipedia.org/wiki/AFI\%27s\_100 \_Years
        ...100\_Movies',2))]]
```

5.5.2 Querying Twitter data. What is happening now?

Twitter is a popular social network whose popularity is increasing to the extent that many people use it as a news stream [43]. Collecting its data is important for many academic and commercial activities to perform data mining, integration, and analysis tasks [5]. Twitter data sources have the following characteristics: (i) They get frequently updated (about 8,000 tweets are posted per second and around 700M are posted per day[9]), (ii) They are more important when they are fresh - the primary use of Twitter is to find out information about *what is happening now*. (iii) They are frequently used by data scientists as input datasets to data analysis and data mining tasks (e.g., sentiment analysis [68]).

Typically, users write crawlers to retrieve Twitter data and store it in files or in a database. Since the Twitter API has a limit of 100 tweets per request, the crawlers perform multiple

[9] `http://www.internetlivestats.com/twitter-statistics/`

requests and accumulate data over a large period of time. Let us now imagine a semantic-based application that tracks user-generated content about active Semantic Web events, collecting the latest relevant tweets and processing them with a sentiment analysis service that identifies their polarity. For example, it uses the SPARQL query described in Listing 5.4 to retrieve positive tweets about the Web Conference 2019.

Listing 5.4: SPARQL query for twitter

```
SELECT distinct ?s
WHERE {  ?s twitter:tweetsAbout <http://www2019.thewebconf.org/> .
         ?s twitter:sentiment "positive"}
```

Traditionally, this query would be answered through the following steps: (i) retrieve the relevant Twitter data, (ii) transform it into RDF, and (iii) store it in a RDF store. Alternatively, one would store the data in a database, using an OBDA system to query it with mappings. The sentiment analysis task would be performed as a pre- or a post- processing step.

In contrast, using our approach requires less steps for answering this query. After a SPARQL query like the one described above is fired, a virtual table is created, containing information about every tweet along with its `sentiment`, i.e., whether its sentiment is positive or negative. Then, this information gets mapped into virtual RDF terms. To this end, we implemented a virtual table operator that (i) searches data using the Twitter REST API, (ii) uses a binary classifier to identify whether it is positive of negative, and (iii) populates a virtual table with the results. This data can then be accessed using MadQL queries that can be incorporated in mappings so that virtual RDF triples can be produced on-the-fly. An example of such a mapping is given in Listing 5.5.

Listing 5.5: Mappings for twitter

```
mappingId twitter_mapping
target    twitter:{username} twitter:tweetsAbout
          <http://www2019.thewebconf.org/>; twitter:sentiment {s}.
source    select id, sentiment as s from (twitterapi key:www2019)
```

The source part of this mapping contains a MadQL query that uses the virtual table operator named `twitterapi`. This virtual table operator takes as input a search keyword, which in our example is *www2019*. The result of this query is the creation of a virtual table with information about tweets for the Web Conference 2019. Note that the attribute `sentiment` is not part of the data retrieved from the Twitter API, but is derived from the sentiment analysis classifier that is used internally, in the `twitterapi` virtual table operator.

The above SPARQL query is translated into the SQL query provided in Listing 5.6.

Listing 5.6: SQL query

```
SELECT * FROM ( SELECT DISTINCT 1 AS "sQuestType", NULL AS "sLang",
('http://twitter.com/' || REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
```

```
CAST(QVIEW1.id AS CHAR),'␣', '%20'),'!', '%21'),...)) AS "s"
FROM
(select distinct id, sentiment from (twitterapi key:www2019)) QVIEW1,
(select distinct id, sentiment from (twitterapi key:www2019)) QVIEW2
WHERE  QVIEW1.id IS NOT NULL AND (QVIEW1.id = QVIEW2.id) AND
(QVIEW2.sentiment = 'positive')) SUB_QVIEW;
```

This query contains the virtual table operator `twitterapi` that creates a virtual table. The columns `id` and `sentiment` of this table populate a view that is created on-the-fly by the OBDA system. In traditional OBDA systems, the views are constructed on-the-fly from existing, materialized tables (or other views). In our system, this table does not exist, but is created and populated on-the-fly, *after* the SPARQL query is fired and translated into MadQL. The MadQL query will create and populate the table, but this procedure is completely invisible to the user: exactly the same SPARQL query would be used even if the data did not come from a REST API, but was stored in a database, or a triple store.

To classify the tweet according to its polarity, we employed an open-source sentiment classifier for Twitter[10], which uses an SVM model that is already trained with the following datasets: (i) The Stanford Sentiment140 dataset[11], (ii) the Polarity Dataset (v2.0)[12], and (iii) a dataset from the University of Michigan[13] that contains 7,086 sentences extracted from various social media.

We have modified this classifier so that it follows a client-server model, where the server and the client communicate through a socket. In this way, we avoid incorporating the whole classifier into the virtual table operator and save the cost of loading the classifier every time the virtual table operator is invoked. When the server starts, it loads the classifier and waits for connection. The client part is incorporated into the `twitterapi` virtual table operator and sends every tweet of the results to the server for classification through a socket. The server performs sentiment analysis and returns whether the tweet is positive or negative. The result is returned as an additional column of the produced virtual table, called `sentiment`.

5.5.3 Querying Foursquare data on-the-fly

Foursquare is a mobile application that offers location-based search for venues with multiple criteria (e.g., nearby restaurants ranked by rating or distance). The descriptions of these venues are enriched with user reviews and ratings, thus facilitating location recommendations. Foursquare also allows users to share their location (e.g., park) with their friends, informs them how many other users are simultaneously at the same location, and alerts them when many people have checked in at the same time in a place nearby.

[10] https://github.com/dkakkar/Twitter-Sentiment-Classifier
[11] http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip
[12] http://www.cs.cornell.edu/people/pabo/movie-review-data/
[13] https://inclass.kaggle.com/c/si650winter11

Having around 55 million monthly users and a platform that contains crowd-sourced information for 105 million venues worldwide (according to its website), Foursquare soon became a useful data source for application development. Developers can access its API[14] and get part of this information for free (e.g., venue description, location, rating, check-ins), while more data is available on charge. Foursquare has approximately 40,000 registered developers using its API[15]. As an example of applications built on top of Foursquare, consider the "Mr Jitters" app, which uses Foursquare data to find the best coffee places nearby[16].

Semantic Web agents could also exploit this valuable data source. Imagine a semantic-web alternative to "Mr Jitters" that uses Foursquare venues as RDF so as to interlink them with datasets from the linked open data cloud (e.g., DBpedia, LinkedGeodata, Geonames). Supposing that it searches information about coffee places in Chicago, it would pose the SPARQL query described in Listing 5.7:

Listing 5.7: Querying Foursquare using SPARQL

```
SELECT ?venue ?checkins
WHERE {?venue four:name ;  four:hereNow ?checkins;
       four:category "Coffee"; four:near "Chicago"}
```

We now explain how our framework can be used to map the free Foursquare data to virtual RDF graphs and perform this query on top of them. First, we create an ontology that describes all venue categories that appear in the Foursquare venue category taxonomy[17]. The resulting ontology contains 961 classes that represent venue categories, enabling us to perform reasoning over this rich class hierarchy of venues.

Next, we implement a virtual table operator, called `foursqr`, that receives as input some keywords for searching venues and returns as output a list of venues. The operator is implemented as a Python MadIS virtual table operator, which internally uses a Python library for the Foursquare API[18]. When the Foursquare virtual table operator is invoked, it accesses the Foursquare API with the input parameters as arguments, and the result is presented as a virtual table that in turn gets mapped into RDF terms using the mapping described in Listing 5.8.

Listing 5.8: Foursquare mappings

```
mappingId foursquare_mapping
target    four:{id} four:hasID {id} ; four:name {name} ;
          four:hereNow {h}^^xsd:integer; four:category four:{category};
          four:near "Chicago"  .
source    select id, category, name, hereNow_count as h, contact
          from (foursqr key:coffee near:Chicago)
```

[14] https://developer.foursquare.com/
[15] https://en.wikipedia.org/wiki/Foursquare#Foursquare_API
[16] https://developer.foursquare.com/docs/sample-apps
[17] https://developer.Foursquare.com/docs/resources/categories
[18] https://github.com/mLewisLogic/foursquare.git

In this mapping, we want to retrieve coffee places in Chicago. The `foursqr` operator used in the MadQL query of the mapping takes the respective parameters as input. It generates a virtual table populated with information about coffee places in Chicago. The target part of the mapping encodes how these attributes are translated into RDF terms according to the Foursquare Ontology.

5.6 Experimental evaluation

In this section, we empirically evaluate the time efficiency of our approach. Section 5.6.1 describes the setup of our experiments, while Section 5.6.2 evaluates the three operators that were implemented using our approach. We conclude with a performance comparison against the sate-of-the-art system described in [55]. The process we followed to perform this comparison and the respective results are described in detail in Section 5.6.3

5.6.1 Experimental setup

Execution environment. All experiments were executed in an Intel Core(TM) 2 Quad CPU Q9650 machine at 3.00GHz with Ubuntu 14.04 and 8GB RAM. In all experiments, we measure the query execution time, which includes a full iteration of the result set. We execute all experiments in both cold and warm cache. In warm cache, we execute a query once before all executions of the same query that we measure. In cold cache, we configure all virtual tables that are involved so that they do not use the caching mechanism described in Section 5.3 (i.e., we set a negative value to the `rate` parameter of the virtual table operator in the mappings). These two configurations allow for measuring the impact of the caching mechanism on the query execution time.

Data sources and queries. We query data from HTML tables about films, Twitter and Foursquare, posing queries that are similar to the ones described in Section 5.5. More specifically, we pose queries for tweets that contain the `WWW2019` hashtag, retrieving also the sentiment for each tweet. We look for coffee places in Chicago from Foursquare and we join two HTML tables with films, one from Wikipedia and one from Rotten Tomatoes. The mappings and part of the queries that we used are explained in Section 5.5. For each data source, we begin with a query that involves a single triple pattern and then, we increment the number of triple patterns to increase the complexity of the query.

However, the amount of tweets and Foursquare entries that one can obtain for free through a single request is limited, due to the restrictions of the respective APIs. The HTML tables that exist in the Web are also relatively small. For these reasons, we evaluate the scalability of our system using synthetic webtables. We employed an original Wikipedia table about Italian election opinion polls[19] as a template, which we multiplied so that we can ex-

[19] `https://en.wikipedia.org/wiki/Opinion_polling_for_the_Italian_general_election,_2018`

ecute queries for tables with 10, 100, 1,000, 10,000, and 100,000 rows. Then, we posed the same queries over these tables in order to measure the scalability of our system.

Comparison with related work. We compared our approach with the system described in [55]. To achieve this, we additionally implemented an operator of Yelp, which allows for comparing the two systems on a equal basis (note that the system of [55] does not apply to webtables). In Section 5.6.3, we describe in detail the set up of the comparison and report the evaluation results.[20]

5.6.2 Experimental Results

Real workload. The query execution times of the real workload experiments in both cold and warm cache are presented in Figure 5.3. The label of each query is suffixed by the number of triple patterns it incorporates (e.g., Q2 indicates two triple patterns). We observe that in warm cache, the execution times are at least an order of magnitude lower than in cold cache. Apparently, the reason is that in the former case, the virtual tables are cached (cf. Section 5.3), whereas in the latter case, the data must be fetched (e.g., calling the respective APIs).

Reasonably, as the number of triple patterns used in the queries increases, the execution time increases. This happens because more triple patterns yield more joins in the translated SQL query. When these joins produce more intermediate results, instead of filtering them down, they introduce additional cost in the evaluation. In other words, we add triple patterns to retrieve more information, rather than to pose restrictions. The main reason for this performance cost is that the data is not materialized in the database and, thus, the OBDA system is not aware of database constraints, or other hints that could accelerate SQL translation and execution, as described in [20].

Note also that all films queries use two data sources, joining the HTML tables described in Section 5.5.1 to retrieve the movies that are common between the two tables. This is a more costly operation than the selections performed in Twitter and Foursquare use cases, resulting in higher execution times, as the query execution time also includes the time to parse the HTML table(s).

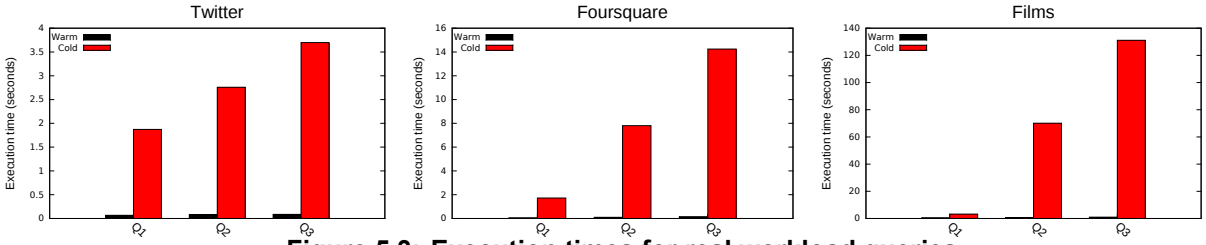


Figure 5.3: Execution times for real workload queries.

Synthetic workload. Recall that our approach does not target the *volume* of the Web of

[20]Note that we also attempted to compare our approach with the work described in [54], but we could not build an instance of their platform, following the online instructions.

data, but rather the *variety* and *velocity* dimensions. Nevertheless, we included a scalability experiment in our evaluation so as to assess the maximum size of input data that we can query efficiently. For the scalability experiment, we used a query with two triple patterns posed against the synthetic webtables of varying size. The first query, that is provided in Listing 5.9, is not selective, returning as many results as the rows of the table. The second query , that is described in Listing 5.10 is very selective, returning two results at all cases.

Listing 5.9: Query of low selectivity for webtables

```
SELECT distinct ?s1 ?d ?l
WHERE { ?s1 :date ?d .
        ?s1 :lead ?l }
```

Listing 5.10: Query of high selectivity for webtables

```
SELECT distinct ?s1 ?d
WHERE { ?s1 :date ?d .
        ?s1 :lead "1.5"^^xsd:float  }
```

The outcomes of the scalability test appear in Figure 5.4. We observe that as the number of rows in a webtable increases, the query execution time increases superlinearly, but the extent of this increase depends heavily on the selectivity of the query. We can safely conclude, though, that our system can query webtables with up to 100,000 rows within minutes.

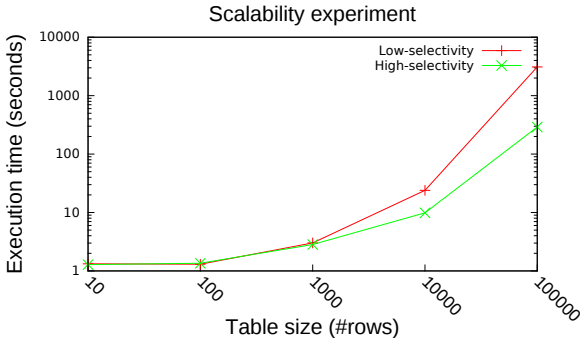


Figure 5.4: Query execution time as dataset size increases.

5.6.3 Comparison with the state-of-the-art

We now compare our approach with the system described in [55], which we call *SERVICE-to-API* in the following[21]. Recall that its goal is to enrich RDF data with data from external

[21]We also attempted to compare our approach with the work described in [54], but we could not build an instance of their platform, following the online instructions.

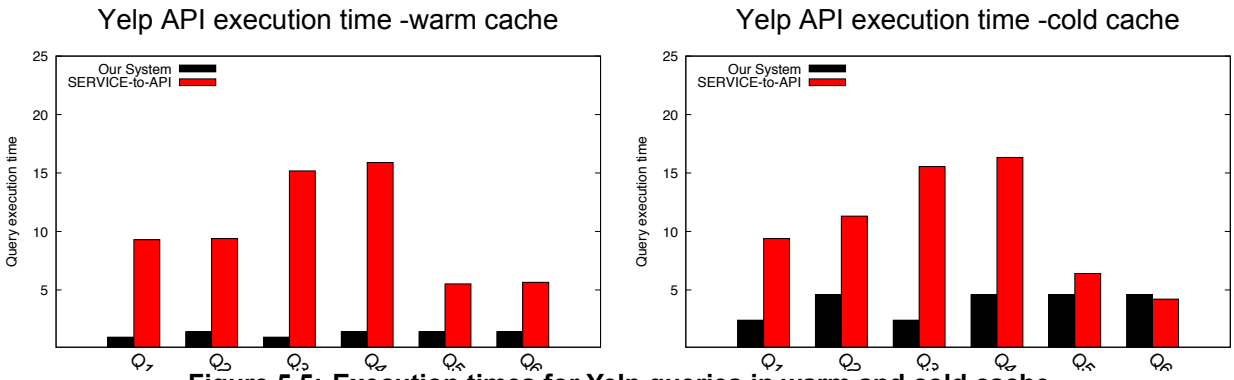


Figure 5.5: Execution times for Yelp queries in warm and cold cache.

sources, such as REST APIs. Thus, its query language requires at least one triple pattern that is evaluated in the RDF repository and its variables are bounded to values that populate their URI templates. Every variable binding actually yields a separate API call. A cache mechanism aims to minimize the API calls.

An example of its query language is presented in Listing 5.11. The value of keyword $SERVICE$ creates a URI template for each one of the values bound to the variable l, which is used in the query's triple pattern. In this case, a call to the Yelp API is produced for each binding of the variable l, returning a JSON file. This JSON file is parsed according to the JSON pattern included in the query, which bounds the variables i, $name$ and $rating$ to the values of the respective attributes of the JSON file.

Listing 5.11: SERVICE-to-API query, equivalent to SPARQL query Q1

```
SELECT   ?i ?name ?rating WHERE {
?x <http://www.w3.org/2000/01/rdf-schema#label> ?l .
SERVICE <https://api.yelp.com/v3/businesses/{l}>{
( $.[\"id\"], $.[\"name\"], $.[\"rating\"]) AS (?i, ?name, ?rating)}}
```

In this context, there are two major *qualitative* differences between our approach and SERVICE-to-API [55]:

(i) The query language. For SERVICE-to-API, the JSON attributes are directly bound to variables by parsing the JSON response, as instructed by the JSON patterns included in the query. As a result, the users need to know the documentation of the API in order to identify the information they need. Only in this way are they able to combine API data with the RDF data in the triplestore, formulating accurate queries that extend SPARQL with JSON patterns [55]. In contrast, our approach creates virtual semantic graphs on top of APIs using mappings, thus allowing users to pose standard SPARQL queries as if the contents of the APIs were transformed into RDF.

(ii) Every API call in our approach retrieves an entire virtual table, which is mapped to a virtual RDF graph. In contrast, SERVICE-to-API [55] merely retrieves one entry of this table per API call, which has a significant impact on performance, as explained below.

Regarding the *quantitative* comparison of the two systems, we consider data retrieved from the REST API of Yelp[22], as SERVICE-to-API does not apply to WebTables. We

[22] `https://www.yelp.ie/dublin`

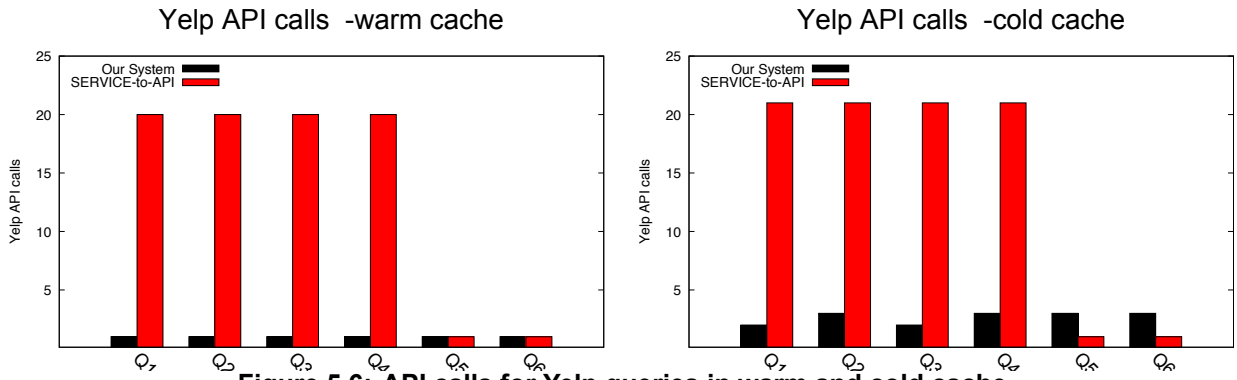


Figure 5.6: API calls for Yelp queries in warm and cold cache.

chose the Yelp API, as it is the only data source for which both systems offer the same functionality (our Twitter operator involves a microservice for performing sentiment analysis). However, the findings of this experiment are representative of the general behaviour of the two systems - the differences between the two systems as exposed by the following experiment are the same against any Web API.

For SERVICE-to-API, we stored data about businesses (burger joints in Chicago) in an RDF repository, as it does not allow for queries that include API calls without triple patterns included in the query. Then, we used the SERVICE keyword to join them with their names and IDs that are retrieved from the REST API of Yelp. Note that we used the original implementation of SERVICE-to-API, which was kindly provided to us by the authors of [55]. For our approach, we implemented a virtual table operator of Yelp and pose the SPARQL query Q1, which appears in Listing 5.12, to retrieve the same data.

SPARQL Query Q2 contains one more triple pattern (i.e., we also retrieve the rating of businesses) and it is described in Listing 5.12. There are different ways to express this query in the SERVICE-to-API, depending on the configuration of the repository. The closest definition seems to be the query shown in Listing 5.15, which is query Q5. However, the fact that it returns different results suggests that this is not the case. Instead of returning the name and rating of the requested businesses, it returned the Cartesian product of all different burger businesses and all different rating values. So, if the SPARQL query Q2 is expected to return $|N|$ results, the query in Listing 5.15 returns $|N \times S|$ results, where $|S|$ is the number of different rating values. We could briefly describe this phenomenon as *a difference in semantics between SPARQL and the new language proposed in [55]*.

In order to follow the semantics of [55], we create variations of queries Q1 and Q2, in which we have partially stored the data to be retrieved by Q1 and Q2 in the repository supported by SERVICE-to-API. Once we have at least one triple for each entity stored, we retrieve only the missing values using the queries described in Listings 5.11 and 5.14, respectively. In this way, SERVICE-to-API returns the correct results, since the underlying triple store is forced to perform a JOIN between the materialized and the values that are returned from the API, instead of a Cartesian product. The trade-off, on the other hand, is that *SERVICE-to-API cannot pose a query to retrieve the results directly through the API, as some form of materialization needs to be performed in order to retrieve correct results*.

Query Q6 differs from query Q5 only in that it uses the `BIND` operator instead of triple

pattern (i.e., instead of storing the respective triple in a triple store).

Listing 5.12: SPARQL query Q1

```
select distinct ?id ?name
where {
 ?s yelp:name ?name .
 ?s yelp:hasID ?id }
```

Listing 5.13: SPARQL query Q2

```
select distinct ?id ?name
where { ?s yelp:name ?name .
 ?s yelp:rating ?rating .
 ?s yelp:hasID ?id }
```

Listing 5.14: SERVICE-to-API query Q3

```
SELECT distinct  ?i ?name    WHERE {
 ?x <http://www.w3.org/2000/01/rdf-schema#label> ?l .
  ?x <http://yelp.com/ontology#name> ?name .
 ?x <http://yelp.com/ontology#rating> ?rating .
 SERVICE <https://api.yelp.com/v3/businesses/{l}>{
 ( $.[\"id\"]) AS (?i)}}
```

Listing 5.15: SERVICE-to-API query Q5

```
SELECT distinct ?id ?b  WHERE {
?x <http://www.w3.org/2000/01/rdf-schema#label> ?l  SERVICE
<https://api.yelp.com/v3/businesses/search?term=Burgers&location={l}&sort=2>
($.[\"businesses\"][0:20][\"id\"], $.[\"businesses\"][0:20][\"name\"],
$.[\"businesses\"][0:20][\"rating\"] ) AS (?id, ?b, ?r)}"
```

We evaluated these queries in both systems and we present the results in Figures 5.5 and 5.6. The former depicts the query execution times and the latter the number of API calls invoked. In both cases, we consider warm and cold caches (on the left and right, respectively). We observe that our approach is three times faster than SERVICE-to-API. The main reason is that we retrieve a set of tuples for each API call, which are then mapped into virtual RDF graphs. In contrast, SERVICE-to-API retrieves one entry for each API call, yielding many more API calls in order to get the same information.

Another observation is that our system by design benefits more from caching than the system in comparison. We cache the entire table for each API call, while SERVICE-to-API performs an API call for each tuple, which means that only one tuple is cached each time. Hence, for a result set consisting of N tuples, our system will cache the entire result set as a virtual table that is retrieved from a single API call. On the other hand, SERVICE-to-API needs at least N calls, of which at most one will be cached,

5.7 Summary

In this chapter, we presented a methodology for querying Web data on-the-fly using SPARQL, based on extending SQL with virtual table operators, embedding them into mappings, and making an OBDA system compliant to them. We also performed an experimental evaluation of our approach, showing that we go beyond the state of the art, not only in terms of functionality, but also in terms of performance. Our approach complements traditional approaches of querying data using SPARQL and targets the *diversity* and *velocity* of web data.

In the next chapter, we present how the work described so far in this thesis was used and extended in real-world applications. In one of these applications, the requirement was to retrieve Copernicus data from a Rest kind of API, named OPeNDAP, without materialising the data. For this reason, as we explain in the following chapter, we implemented another virtual table operator that serves as an adapter with the Copernicus OPeNDAP APIs and can be used in mappings to retrieve Copernicus data on-the-fly and make them available as virtual RDF triples using the system that we described in this section. We refer the reader to the next chapter for more details.

6. APPLICATIONS

The system Ontop-spatial was very useful to many real-world applications, as many end-users did not wish to materialise their data as RDF triples, but used mainly geospatial relational databases. Ontop-spatial is applicable to any domain where integration and processing of geospatial and temporal data is first class citizen. Ontop-spatial has been used in the following domains: land management (collaboration with the german company ViSTA), urban planning (collaboration with the czech company GISAT), oil industry (collaboration with the international company Statoil), and maritime security (collaboration with AIRBUS defence and Space in Bremen). It was also used and extended in the project Copernicus App Lab in order to create semantic views on top of Copernicus data on-the-fly. This chapter highlights two of the main applications where Ontop-spatial was used, the maritime application and the Copernicus App Lab application. The material of this chapter is partially covered in [8], [9] and [17].

6.1 Querying Copernicus Data on-the-fly using ontologies and mappings

Earth observation (EO) is the gathering of data about our planet's physical, chemical and biological systems via satellite remote sensing technologies supplemented by Earth surveying techniques. The Landsat program of the US was the first international program that made large amounts of EO data open and freely available. *Copernicus*, the European programme for monitoring the Earth, is currently the world's biggest EO programme. It consists of a set of complex systems that collect data from satellites and in-situ sensors, process this data and provide users with reliable and up-to-date information on a range of environmental and security issues. The data of the Copernicus programme is provided by a group of missions created by ESA, which is called *Sentinels*, and the *contributing missions*, which are operated by national, European or international organizations. Copernicus data is made available under a free, full and open data policy. Information extracted from this data is also made freely available to users through the *Copernicus services* which address six thematic areas: land, marine, atmosphere, climate, emergency and security.

The Copernicus programme offers myriad forms of data that enable citizens, businesses, public authorities, policy makers, scientists, and entrepreneurs to gain insights into our planet on a free, open, and comprehensive basis. By making the vast majority of its data, analyses, forecasts, and maps freely available and accessible, Copernicus contributes to the development of innovative applications and services that seek to make our world safer, healthier, and economically stronger. However, the potential (in both societal and economic terms) of these huge amounts of data can only be fully exploited if using them is made as simple as possible. Therefore, the straightforward data access every downstream service developer requires must also be combined with in-depth knowledge of EO data processing. The Copernicus App Lab[1] aims to address these specific challenges by

[1] http://www.app-lab.eu/

bridging the digital divide between the established, science-driven EO community and the young, innovative, entrepreneurial world of mobile development.

Copernicus App Lab goes beyond these projects in the following important ways:

- It develops a software architecture that enables on demand access to Copernicus data using the well-known OPeNDAP framework and the geospatial ontology-based data access system Ontop-spatial. Now users and application developers do not need to worry about having to download data or having to learn the details of sophisticated data formats for EO data. All they need to develop is an ontology describing the data they are interested in and R2RML mappings that capture the correspondence between the ontology and the data sources containing the data. Using traditional approaches, application developers would have to implement different clients/adapters in their applications corresponding to the different file formats their data is in, in order to access and process the data. Instead of implementing custom code, they can now use GeoSPARQL queries to access the geospatial data, regardless of their formats.
- It brings computing resources close to the data by making the Copernicus App lab tools available as Docker images that are deployed in the Terradue cloud platform as cloud services.
- It enables search engines like Google to treat datasets produced by Copernicus as “entities” in their own right and store knowledge about them in their internal knowledge graph.

In this context, the following section describes how the techniques presented in Chapter 5 were extended to support Copernicus data that has been available through OPeNDAP services.

6.1.1 Mapping Copernicus Data

The geospatial ontology-based data access (OBDA) system Ontop-spatial [10] is used to make Copernicus data available via OPeNDAP as linked geospatial data, without the need for downloading files and transforming them into RDF. Ontop-spatial is also used in order to create virtual semantic RDF graphs on top of geospatial relational data sources using ontologies and mappings.

Then, the Open Geospatial Consortium standard GeoSPARQL can be used to pose queries to the data using the ontology. As documented in [10], Ontop-spatial also achieves significantly better performance than state-of-the-art RDF stores, and as we explained earlier, it also supports raster data as well.

In the context of the work described in this section, we extend the approach described in Chapter 5. Ontop-spatial has been extended with an adapter that enables it to retrieve data from an OPeNDAP server, create a table view on-the-fly, populate it with this data and create virtual semantic geospatial graphs on top of them. In order to use OPeNDAP as a

new kind of data source, Ontop-spatial utilizes the system MadIS as backend, following the architecture of the system that is presented in Chapter 5. We implemented a new MadIS virtual table operator named `Opendap`, that is able to create and populate a virtual table on-the-fly with data retrieved from an OPeNDAP server. In this way, Ontop-spatial enables users to pose GeoSPARQL queries on top of OPeNDAP data sources without materializing any triples or tables. We stress that the relational view that is created is not materialized. The intermediate SQL layer facilitates the data manipulation process, in the sense that we can manipulate the data before they get RDF-ized. By using this approach, we can perform "data cleaning" without (i) changing the data that arrive from the server (ii) changing any intermediate code, such as the opendap function and (ii) without requiring any extra pre-processing steps.

To improve performance, the OPeNDAP adapter also implements a caching mechanism that stores results of an OPeNDAP call in a cache for a time window w, so that if another, identical OPeNDAP call needs to be performed within this time window, the cached results can be used directly. The length of the time window w is configured by the user in the mappings and it is optional.

Listing 6.1: Example of mappings

```
mappingId opendap_mapping
target lai:{id} rdf:type lai:Observation .
       lai:{id} lai:lai {LAI}^^xsd:float;
                time:hasTime {ts}^^xsd:dateTime .
       lai:{id} geo:hasGeometry _:g .
       _:g geo:asWKT {loc}^^geo:wktLiteral .
source SELECT id, LAI, ts, loc
       FROM (ordered opendap
      url:https://analytics.ramani.ujuizi.com/
      thredds/dodsC/Copernicus-Land-timeseries-
      global-LAI%29/readdods/LAI/, 10)
       WHERE LAI > 0
```

In the example mappings provided in 6.1, the `source` is the LAI dataset discussed above which provided through the RAMANI OPeNDAP server of the Copernicus App Lab software stack. The dataset contains observations that are LAI values as well as the time and location for each observation. The MadIS operator `Opendap` retrieves this data and populates a virtual SQL table with schema `(id,LAI,ts,loc)`. The column `id` was not originally in the dataset but it is constructed from the location and the time of observation. The LAI column stores LAI values of an observation as `float` values. The attribute `ts` represents the timestamp of an observation in date-time format. In the original dataset times are given as numeric values and their meaning is explained in the metadata. For example, it can be days or months after a certain time origin. The `Opendap` virtual table operator converts these values to a standard format. Because of the fact that the `Opendap` operator is implemented as an SQL user-defined operator, it can be embedded into any SQL query. In the above mapping, we also refine the data that we want to be translated into virtual RDF terms by adding an filter to the query to eliminate (noisy) negative or zero

LAI values. The value `10` that is passed as argument to the `Opendap` virtual table operator is the length of the time window w of the cache that is used (in minutes). In this case, if $|w|$ is the length of the time window w, then $|w| = 10$ minutes. This means that results of a every OPeNDAP call get cached every 10 minutes. If a query arrives resulting in an OPeNDAP in time t, where $t < 10$ minutes later than a previous *identical* OPeNDAP call (resulting from a same or similar query that involves the same OPeNDAP call), then the cached results can be used directly, eliminating the cost of performing another call to the OPeNDAP server.

The `target` part of the mapping encodes how the relational data is mapped into RDF terms. Every row in the virtual table describes an instance of the class `lai:Observation`. The values of the LAI column populate the triples that describe the LAI values of the observation, and the values of the columns `ts` and `loc` populate the triples that describe the time and location of the observations accordingly.

6.1.2 Querying Copernicus Data using GeoSPARQL

Given the mapping provided above, we can pose the GeoSPARQL query provided in Listing 6.2 to retrieve the LAI values and the geometries of the corresponding areas.

Listing 6.2: Query retrieving LAI values and locations

```
SELECT DISTINCT ?s ?wkt ?lai
WHERE { ?s lai:hasLai ?lai .
        ?s geo:hasGeometry ?g .
        ?g geo:asWKT ?wkt }
```

Using queries like the one described in Listing 6.2, Sextant can again visualise the various datasets and build layered maps like the one in Figure 6.1. The visualisation of the case study in Sextant is available on line at the following URL: `http://test.strabon.di.uoa.gr/SextantOL3/?mapid=m8s4kilcarub1mun_`

Similarly, in Figure 6.1, we have used Sextant to build a temporal map that uses the following datasets:

- *Leaf area index (LAI)* is a dimensionless quantity that characterizes plant canopies and it is defined as the one-sided green leaf area per unit ground surface area in broadleaf canopies[2]. LAI information from the global land service of Copernicus is made available as a NetCDF file giving LAI values for points expressed by their lat/long co-ordinates. Part of this dataset is described above. We access this dataset through the OpeNDAP adapter which we implemented for Ontop-spatial and we automatically retrieve the data on-the-fly, using SPARQL queries.

- The *Urban Atlas dataset* of 2012, which provides land cover data for European urban

[2] `https://en.wikipedia.org/wiki/Leaf_area_index`

areas with more than 100.000 inhabitants[3].

- OpenStreetMap data, which we used as shapefiles, provided by the company Geofabrik[4]. We imported the shapefiles into a PostGIS database and we used Ontop-spatial to create virtual geospatial RDF graphs on top of them. Insted, we could have used the RDF versions of this dataset, as for example the RDF datasets provided by the project LinkedGeoData[5]. We opted for the OBDA solution in order to (i) retrieve the most current version of the data without materialising the OpenStreetMap data every time it gets updated, and (ii) achieve better query execution times, as we explained in Chapter 3.
- GADM is an open and free dataset giving us the geometries of administrative divisions of various countries[6]. It is available in various formats (e.g., ESRI geodatabase, SQLite database, shapefiles), which we converted and materialised as RDF files, which we stored in Strabon and made it available in a GeoSPARQL endpoint. Since GADM data does not change frequently, we opted for the materialised solution in this case.

We show how the LAI values (small circles) change over time in each administrative area of Paris (administrative areas are delineated by magenta lines) and correlate these readings with the land cover of each area (taken from the CORINE land cover dataset or Urban Atlas). This allows us to explain the differences in LAI values over different areas. For example, Paris areas belonging to the CORINE land cover class `clc:greenUrbanAreas` overlap with parks in OpenStreetMap and show higher LAI values over time than industrial areas. Paris enthusiasts are invited to locate the Bois de Boulogne park in the figure.

6.2 Maritime Security application

The maritime security domain is challenged by a number of data analysis needs with a focus on increasing the maritime situation awareness. Vessel movements are of major importance for maritime data analysts and decision makers. Abnormal vessel behaviors and suspicious vessel movements need to be detected and understood to properly increase the maritime domain awareness. More than two-thirds of the overall volume of cargo worldwide is transported seaborne. This massively increases the number of ships traveling on the seas. Next, the continuously increasing number of offshore wind parks has an impact on the security of the citizens. The energy supply must be ensured even without fossil fuels. This turns offshore wind park into assets with a strong demand for protection. Moreover, industrial nations worldwide use the potential of the seas, but are threatened by pirates and terrorists.

[3] `https://land.copernicus.eu/local/urban-atlas/view`
[4] `http://download.geofabrik.de/`
[5] `http://linkedgeodata.org/About`
[6] https://gadm.org/

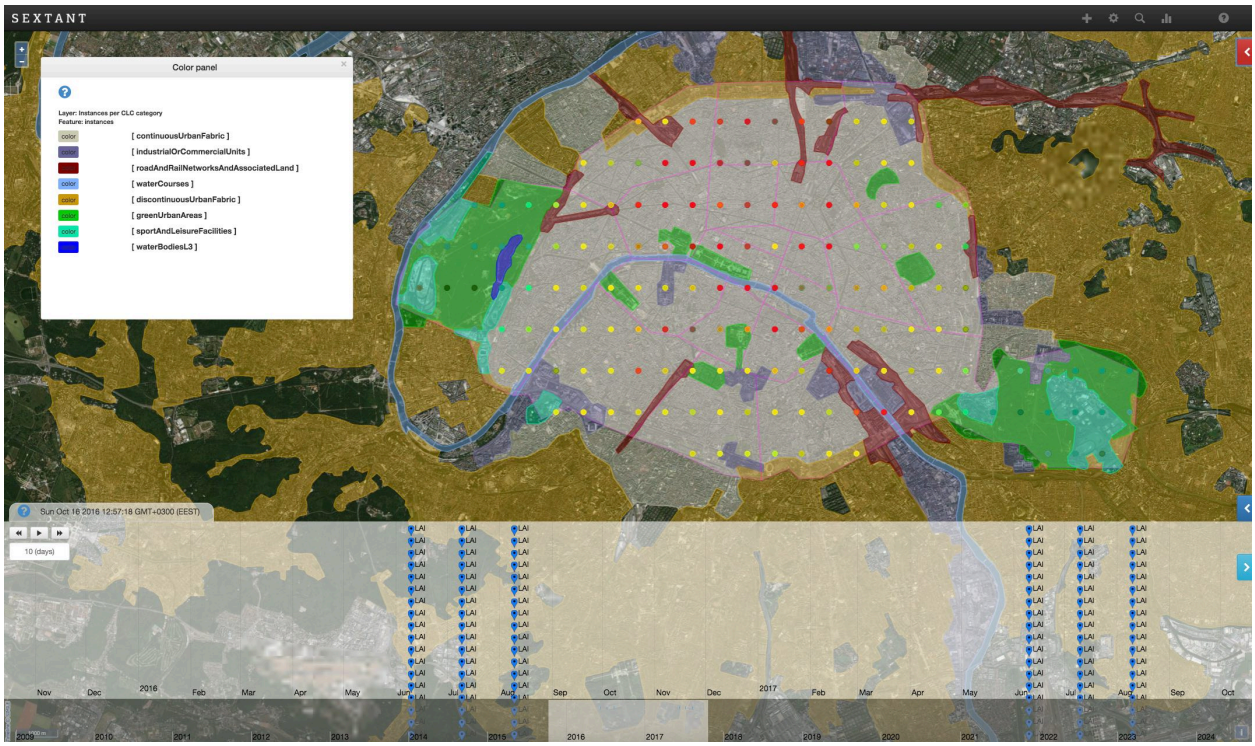


Figure 6.1: The "greenness" of Paris

The project EMSec[7] (real-time services for the maritime security) has the aim to support the maritime security by improving the availability and accessibility of relevant data and information ashore and offshore. The central data management component of EMSec is the "Real-time Maritime Situation Awareness System" (RMSAS), which is in charge of integrating various types of data from different sources.

This section focuses on the integration and analysis of data for vessel movements in RMSAS. For the proper analysis a storage capability is needed to properly analyze vessel data and to identify vessel trajectories and their stops and movements for the last days. However, the vessel data integration and management task in RMSAS is challenged by the following requirements: *(a)* The vessel data is heterogeneous and in particular consists of dynamic position data or static metadata. *(b)* There is a need for integrating third party data, i.e., open data like GeoNames and OpenStreetMap. *(c)* The size of the data is large, deriving from the acquisition and processing of large radar and satellite images. *(d)* The data about vessels are produced in real-time, i.e., approximately 1000 vessel positions are acquired per second.

The motivation of this use case was to address the above needs for combining data from heterogeneous sources, such as in-situ data, AIS data, and open data developing an automated solution avoiding manual work as much as possible. We considered that the conceptualized model offered by ontologies would meet the requirements of that purpose, so we used state-of-the-art technologies and tools into this direction. The rationale behind

[7] http://www.emaritime.de/projects/emsec/

our design choices was to avoid the creation of replicas of the same data in other formats and also to avoid storing natively data that are already available as open data. For the first, the system Ontop-spatial was used, instead of using a triple store, to avoid the cost (in disk space and response time) of materializing our frequently updated data to RDF and storing them natively. For the second, we used federation to query data coming from different endpoints (e.g., in-house data and linked data like geonames).

The RMSAS system has been implemented on top of these state-of-the-art Semantic Web techniques and tools. The evaluation has shown that RMSAS eases the data analysis by using virtual triples and standardized vocabularies. Next, the integration of several heterogeneous data sources is a benefit for maritime decision makers and the maritime security. Finally, the approach contributes significantly to the detection of routine traffic and abnormal vessel behavior.

6.2.1 RMSAS: Real-time Maritime Situation Awareness System

RMSAS is a real-time maritime situation awareness system that has been implemented in EMSec as a system of systems to integrate vessel data coming from various sensors, enrich this data with data from other sources (e.g. open data), harmonize these datasets using established maritime standards and infer new knowledge from the integrated data and deliver this it in near real-time. The final results are displayed in a user-friendly interface that enables maritime decision makers to handle maritime situations more efficiently.

Situation-aware data shall be presented to the user in near real-time. To achieve this, data have to be integrated and consolidated from several different sources. This allows for properly displaying the combination of this data to the user via an application. Providing data faster and in further detail shall allow the involved parties to identify critical situations better and earlier, to avoid these situations, and to manage them efficiently.

Being the central data management component in EMSec, the RMSAS aims at integrating and consolidating data. RMSAS uses the “System of Systems” approach and implements a federated information system based on separate services (SOA). Data are integrated in RMSAS in near real-time, next they are consolidated based on semantic data models and techniques and provided to the end user as information products. Ontologies are used in the consolidation of these heterogeneous data.

Data sources. The data sources that are used in the maritime awareness scenario that is described in this section

Data streams. Data streams are data which are created continuously in real-time. These data streams can be AIS-data that continuously report new arriving vessels in the German bay.

Static data. Data are static when they are stored in databases, on FTP-servers, or in external systems. These can be metadata about vessels as for example the vessel type, cargo, port of departure, and historical data about previous routes. After transmission to the earth, satellite data are made available as packages.

More specifically, data sources that are used in the context of this use case are described below.

AIS. AIS messages are not only used as reference data, they are additionally used for quality inspections and – wherever possible – analyzed to identify certain movement pattern, for instance for ferries. Several AIS types are available: Terrestric AIS, satellite AIS[8], and the AIS signal that comes from the Columbus-module of the ISS[9]. In EMSec we receive AIS data about 800-1000 vessels in the German bay every 1-3 seconds.

Satellite SAR. TerraSAR-X provides satellite-based synthetic aperture radar (SAR) and creates radar images with a high resolution. Algorithms can be used to detect objects (e.g., vessels) and to link these detected objects with previously collected AIS messages. The radar images can also be analyzed to extract wind and wave information and connect them with conventional secondary weather information.

Airborne systems. The EMSec consortium utilizes an airplane that comes with an AIS receiver and a radar system. The AIS messages are used as described before. The radar system provides objects and their movements as plots and tracks. Next, another airborne system provides optical images that are used to detect vessels in these images. RMSAS is capable of providing these object detections together with weather information and geospatial information to the end user applications.

Open Data. Open data are data coming from the linked open data cloud or from other external data sources. Using these data can improve certain kinds of analysis. For example, these data sources contain information about real existing harbors (as in GeoNames[10]), or information about certain points of interest (as in DBpedia[11] or OpenStreetMap[12]), or that contain weather data (as in OpenWeatherMap[13]).

GeoNames. is a gazetteer that collects both spatial and thematic information for various place names around the world. GeoNames data is available through various Web services but it is also published as linked data. The features in GeoNames are interlinked with each other defining regions that are inside the underlined feature (children), neighboring countries (neighbors) or features that have certain distance with the underlined feature (nearby features).

OpenStreetMap (OSM). maintains a global editable map that depends on users to provide the information needed for its improvement and evolution. OpenStreetMap datasets are available in RDF format from the LinkedGeoData project[14]. However, it was more convenient for us to download the most up-to-date original OpenStreetMap data about

[8] http://www.esa.int/Our_Activities/Space_Engineering_Technology/ESA_satellite_receiver_brings_worldwide_sea_traffic_tracking_within_reach

[9] http://www.esa.int/Our_Activities/Space_Engineering_Technology/Space_Station_keeps_watch_on_world_s_sea_traffic

[10] http://geonames.org

[11] http://dbpedia.org

[12] http://openstreetmap.org/

[13] http://openweathermap.org/

[14] http://linkedgeodata.org

Bremen, available as Shapefiles[15]. We imported the Shapefiles into a PostGIS database and created virtual geospatial RDF views on top of them using *Ontop*-spatial, as described at `https://github.com/ConstantB/ontop-spatial/wiki/Shapefiles`.

6.2.2 Request Management in EMSec

This section describes the concept of managing and answering requests in EMSec. Figure 6.2 describes that these requests may come from a user, a SOA-architecture or else. The main concept is that requests are formulated using the Top level ontology (TLO) and are posed using SPARQL. This enables the end user to use the described high level semantics of the TLO. The semantic data processing component utilizes *Ontop* to translate the queries to SQL queries in order to be evaluated in the underlying RDBMS, e.g., a PostgreSQL database. This work focuses on relational data sources, so other input data formats such as CSV will not be discussed here.

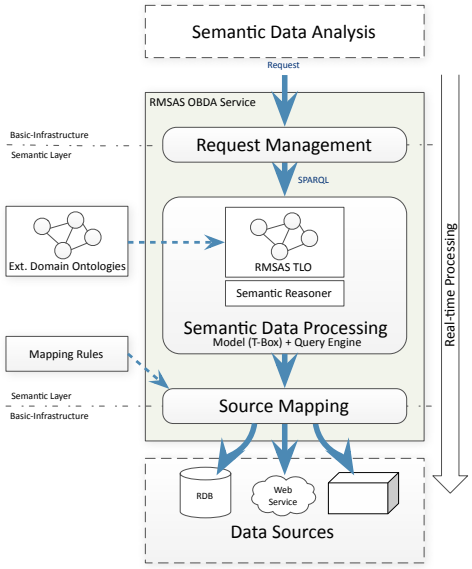


Figure 6.2: Request Management within EMSec

6.2.3 Scenarios in EMSec

The validation of the created methods, architectures, algorithms, and concepts will be done in a campaign, where two maritime security scenarios are executed. First, a concrete satellite mission is utilized. Second, both airborne missions are requested and executed. The generated data are transferred to RMSAS in near real time and integrated, analyzed, consolidated and finally transferred to the user. Several maritime regions are deserving

[15] `http://download.geofabrik.de/europe/germany/bremen.html`

protection. Restricted areas can be off-shore platforms, wind parks, or preserved areas. These call for limited vessel traffic with certain restrictions. *Geographic fences* can be created to analyze the vessel traffic focusing in specific areas of interest. Possible scenarios are to check that the speed over ground is within a limited range in these regions, that certain vessel types like oil-tanker may not pass these regions, or that under certain sea conditions no vessel traffic is allowed.

6.2.4 Modeling the maritime domain

Maritime domain models are results of several research projects, both national and international. The CoopP-project has created the CISE-ontology[2], which is reused in RMSAS and adopted to meet the project's specific requirements.

Object. Objects can be any involved parts of the maritime domain. They can be physical elements that are airborne, onshore and offshore, such as vessels, containers, planes, icebergs, or satellites. Vessels are central elements of interest and modeled in greatest detail with a special focus on the information that are available in AIS.

Geometry. Geometry is dedicated to deal with information about space and geographical localizations of the maritime objects. The geometries contained in our data are encoded in WKT format, which is an OGC standard for the serialization of geometries. The geometries encountered in our dataset are mainly polygons and points. The geometries of areas, for example, are represented as polygons. These areas can be marked regions ashore, for instance. Dimension describes the specifics of an object like length, width, or height. A location describes places with a geographical name like cities or harbours. They can be identified using a URI which makes it possible to interlink them with external sources like GeoNames or DBPedia. Movements describe the track of an object including its course and speed over ground and optionally its rate of turn. Points describe a dedicated geographical point described using its geographical coordinates and its height. A position then is a point combined with a timestamp.

Time. Time is used to describe timestamps that can be used to model positions of objects, to label data during data integration and to support temporal data analysis.

In order to model our data, we have constructed an ontology that is shown in figure 6.3. In this work, we focus on the aspects of vessel movements and trajectories.

The movement ontology defines the necessary structures for modeling object movements like vessel, satellites or aircrafts. The ontology allows for enriching native position data with semantics. This allows to model vessel positions as being moves or stops. Any moving object has position data and consists of trajectories that reflect the historic positions of an object. The use of semantics to these positions facilitates the monitoring of the status of the moving object, i.e. whether it has stopped or was moving.

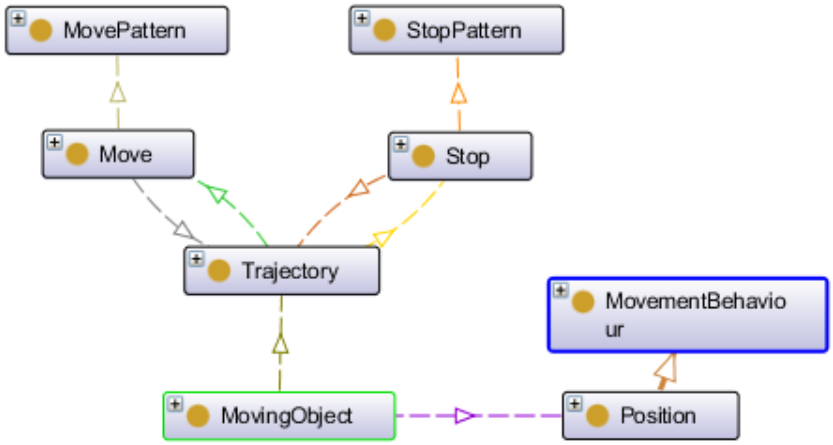


Figure 6.3: RMSAS Movement ontology

6.2.5 Semantic Data Analysis

In this section we describe how RMSAS uses the Semantic Web technologies that we mentioned in the introduction in order to achieve the following goals:

- Transparent integration of different, geospatial and thematic data sources using ontologies.
- Processing of in-house dynamic and static data, enriching them with information already available on the web (linked open data).
- Avoid replicating the same data as much as possible (e.g., materializing data to RDF, storing data from scratch when a SPARQL endpoint for them is already available) using OBDA techniques and federation.
- Visualization of the data and creation of persistent, web accessible maps, with no need to load the datasets or issue the queries again every time we want to populate the existing databases/endpoints with fresh data.

We illustrate the abstract architecture of RMSAS in Figure 6.4. RMSAS uses the OBDA system *Ontop* and *Ontop-spatial* to expose the data we need from the relational databases as SPARQL endpoints. For accessing non-relational data sources, RMSAS first wraps these sources into relational ones by Teiid, and then uses *Ontop* [46, 18] to access them. For federating third party SPARQL endpoints like GeoNames, Sesame is used for the SPARQL 1.1 federated query answering. Finally, Sextant is used for visualizing the results on temporally-enabled maps combining geospatial and temporal results from different (Geo)-SPARQL endpoints.

The relational data in RMSAS can be faithfully mapped to the ontology using the ontology-based data access (OBDA) approach. We use *Ontop* and the its extension *Ontop-spatial* for this purpose. As illustrated in Figure 6.4, *Ontop* allows for querying relational data sources through a conceptual representation of the domain of interest, provided in terms

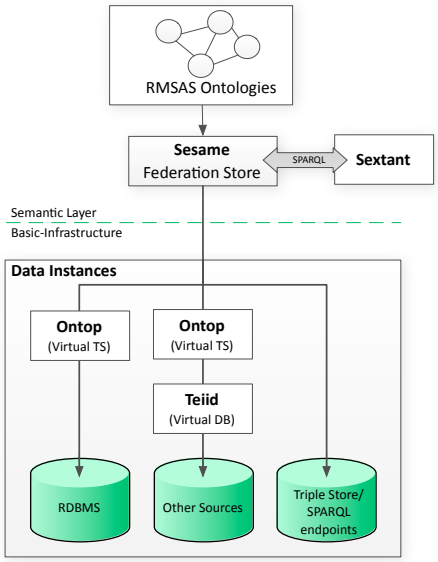


Figure 6.4: Abstract Architecture of RMSAS

of an ontology, to which the data sources are mapped. *Ontop* answers the SPARQL queries by translating them into SQL queries over the database and avoids materializing triples. *Ontop*-spatial is an extension of *Ontop* with geospatial features.

Ontop uses declarative mappings to encode how relational data are mapped to the respective RDF terms. *Ontop* supports W3C R2RML mapping language [30] and its native *Ontop* mapping languages. Here we use the native syntax because it is more compact. An *Ontop* mapping consists of three fields: *mappingId*, *source* and *target*. The *mappingId* is an identifier for mapping; the *source* is an arbitrary SQL query over the database; and the *target* is a triple template written in Turtle syntax that contains placeholders referencing column names mentioned in the source query.

For example, all information about the positions of vessels are stored in a spatially enabled PostGIS database. In Figure 6.5, we present mappings related to vessels in *Ontop* native syntax. The coordinates of the mappings that are stored in the respective columns named `longitude` and `latitude` in the database in textual form are mapped into RDF literals, as objects of the respective virtual triples as indicated in the mapping assertion with mappingId “Position”. The respective geometries that represent the vessels positions are also stored in the well-known binary format (WKB) in a separate column, named `geom`. The mapping assertion `Geometry` indicates how this information is mapped to RDF: The binary geometry of the database is exported as a well-known text literal (WKT), following the OGC GeoSPARQL standard [26].

In the following we present two example SPARQL queries that we used in order to process our data using OBDA technologies and combine them with other sources.

The query described in Figure **??** retrieves geometries of the locations of vessels (ordered by the timestamps) that are stored in binary (WKB) format in the relational database. Objects of this datatype are internally handled by *Ontop-spatial* and are eventually trans-

```
mappingId   Vessel
target      :Vessel-{v.id} a :Vessel ; :hasName {v.name} .
source      SELECT v.id, v.name FROM Vessel v

mappingId   VesselPosition
target      :Vessel-{v.id} :hasLocation :Position-{vp.position_id} .
source      SELECT v.id, vp.position_id FROM Vessel v,
                    Vessel_position vp WHERE v.id=vp.vessel_id

mappingId   Position
target      :Position-{id} a :Position ; :hasLatitude {latitude} ;
                  :hasLongitude {longitude} ; :hasDateTime {ts} ;
                  :hasGeometry geos:Geometry-{id} .
source      SELECT id, latitude, longitude, ts FROM position

mappingId   Geometry
target      geos:Geometry-{id} a geos:Geometry ;
                               geos:asWKT {geom}^^geos:wktLiteral .
source      SELECT id, geom FROM position
```

Figure 6.5: Example mappings in RMSAS

Listing 6.3: SPARQL query retrieving positions of a vessel through time

```
PREFIX : <http://www.rmsas.de/DMARitime#>
PREFIX geos: <http://www.opengis.net/ont/geosparql#>
SELECT DISTINCT ?x ?z ?g ?timestamp
WHERE {
  ?x rdf:type :Vessel.  ?x :hasName "Vesselname"^^xsd:string.
  ?x :hasLocation ?z. ?z :hasDateTime ?timestamp .
  ?z geos:asWKT ?g. }
ORDER BY DESC(?timestamp)
```

Figure 6.6: SPARQL query retrieving positions of a vessel through time

```
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX osm: <http://linkedgeodata.org/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>

SELECT DISTINCT ?lu ?geo
WHERE {
       ?x osm:landUse lgd:port . ?x geo:asWKT ?geo .
       ?x1 geo:asWKT ?geo1 . ?x1 osm:landUse ?lu .
       FILTER (geof:sfIntersects(?geo,?geo1))}
```

Figure 6.7: SPARQL query retrieving locations of ports and land use of intersecting areas

```
PREFIX : <http://www.rmsas.de/DMARitime#>
PREFIX geos: <http://www.opengis.net/ont/geosparql#>

SELECT ?vessel ?location ?geometry ?wkt ?mmsi ?length ?height
WHERE { SERVICE <http://www.rmsas.de/openrdf-sesame/PositionStore>
                { ?vessel rdf:type :Vessel .
    ?vessel :hasLocation ?location .
    ?vessel :hasName "388328333".
    ?location :hasGeometry ?geometry .
    ?geometry geos:asWKT ?wkt .
    OPTIONAL {
      SERVICE <http://www.rmsas.de/openrdf-sesame/ObjectStore>
        {?vessel :hasMMSI ?mmsi ; :hasName "388328333".
        ?vessel :hasLength ?length ; :hasHeight ?height .}}}}
```

Figure 6.8: SPARQL Federation: finding locations of a vessel and their static metadata

formed into RDF literals of WKT datatype, as specified the OGC standard GeoSPARQL and indicated by the mappings that we presented in the previous section. This is the template of the queries we posed to retrieve the locations of ferries to three German islands (Langeoog, Spiekeroog, and Wangerooge).

The query described in Figure 6.7 retrieves the geometries that represent the locations of ports and the land use of areas that they intersect with (e.g., farmyards, commercial/religious areas).

For federating third party SPARQL endpoints like GeoNames, RMSAS relies on the SPARQL 1.1 federated query [64] implemented in Sesame [16]. In the query described in Figure 6.8, we use “SERVICE” function in order to combine information coming from different endpoints exposed by *Ontop*. The first endpoint (PositionStore) contains dynamic data about the locations of vessels stored in a PostGIS database. The second endpoint (ObjectStore) contains static metadata about vessels, such as dimensions, name, etc. The query retrieves all available information about a specific vessels combining both *Ontop* endpoints in a federated store.

Evaluation. The benefits of the approach that we presented in this the maritime awareness use case described in this section are explained below.

Improved data analysis using virtual triples. The data given in this project mainly exists in databases and data streams and is modeled with respect to different data models. The use of OBDA techniques facilitates the process of data analysis as these data are mapped to the ontology that has been created for RMSAS. This allows decision makers to formulate queries against a standardized ontology instead of articulating different queries in different languages against different data sources like the ones described in Section 6.2.5.

Benefits of data integration for maritime decision makers. Compared to the old workflow with respect to information exchange and integration that was identified in the beginning of the EMSec project, where maritime staff had to exchange data often in very traditional ways like email, USB-sticks, mail, paper, or else, the current workflow is significantly improved. With the presented technologies in place, maritime decision makers have all the desired information at hand in near real-time, integrated from different data sources. This increasing having an overview on the maritime security and having a better maritime situational awareness.

Detection of routine traffic and abnormal vessel behavior. In the process on data analysis, SPARQL queries and SWRL rules [42] have been used as a good means (w.r.t. expressivity and efficiency) to detect routine traffic and abnormal vessel behavior. Since we cannot display these rules here for confidentiality reasons, we can state that vessel movements can be easily classified using movement ontologies and that vessel behavior can be classified using the introduced movement pattern. Having combined this with the OBDA approach and with the utilization of (Geo-)SPARQL functionalities, this has strong benefits regarding the detection of routine traffic and abnormal vessel behavior.

6.3 Summary

In this chapter, we described two real-world applications in the context of which the work described in this thesis was applied. The first application belongs to the Earth Observation domain and it aims at accessing Copernicus data through a Rest API, i.e., the OPeNDAP API, as virtual triples on-the-fly, using ontologies and mappings. The second application belongs to the maritime domain awareness and uses Ontop-spatial to integrate relational data about vessels positions with linked geospatial data.

In the next chapter we conclude the thesis and we also present future extensions.

7. CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In the context of this PhD thesis we describe techniques for efficient integration and querying of geospatial and temporal data. We focus in ontology-based data access techniques for creating virtual semantic graphs on top of relational geospatial and temporal databases, avoiding the conversion and materialisation of original data into RDF, using ontologies and mappings. We introduce the first geospatial OBDA system and we demonstrate its efficiency, comparing its performance with state-of-the-art RDF stores. Then, we introduce new temporal features to the temporal dimension of the data model stRDF and the query language stSPARQL, in order to facilitate the support of temporal SPARQL queries in OBDA systems.

The next step was to go beyond relational databases as data sources by extending the OBDA paradigm with the capability to create virtual RDF graphs on top of data that can be accessed via Web APIs, HTML tables, etc. We propose an architecture of a system that implements these techniques and we showcase its functionality using real-world scenarios. We conduct an experimental evaluation of the system and we compare our approach with a related approach offering similar functionality. The outcome of the evaluation proves that our system is more rich in functionality and also more efficient. Last but not least, we present real-world applications in which the approaches described in this thesis were used.

7.2 Future work

Regarding future work, Ontop-spatial could be extended to support distributed GeoSPARQL processing. One possible solution into this direction would be to use a distributed system with geospatial support as back-end, such as SpatialHadoop[1], Hive, GeoSpark[2], etc. However, since Ontop-spatial performs GeoSPARQL-to-SQL translation, a candidate back-end system should also have an SQL API apart from geospatial support. Our recent study [35] presented an evaluation of various geospatial distributed systems with an SQL API. In this paper the system Exareme[3], which is a parallel DBMS built on top of MadIS, is extended with geospatial support.

Another extension of the work described in this thesis could be the further development of the raster support. In the context of the thesis, we did not extend GeoSPARQL with primitives for raster data, however, in DBMSs with raster support, such as the systems described in [27, 78, 6], there is a wide variety of operators supported. GeoSPARQL could be extended to support these operators, which could then be implemented in Ontop-

[1] http://spatialhadoop.cs.umn.edu/

[2] https://datasystemslab.github.io/GeoSpark/

[3] http://madgik.github.io/exareme/

spatial or any other GeoSPARQL query engine. Even in these systems, however, support for relational geospaital operations combining vector and raster data is challenging. For example, RasDaMan [6], a well-known DBMS with raster support, is not able to perform spatial joins between a raster and a vector table. The raster extension, on the other hand, supports it but it is not very effective. It seems that there is still room for research in the area of databases with raster support in order to effectively handle vector and raster data. An extension of SPARQL with support for scientific data was proposed in [4], however as this work is not aligned with GeoSPARQL. The W3C working group “Coverages in Linked Data”[4] was formed to address the challenges of the effective representation and querying of raster data on the Web.

Last but not least, another possible direction could be to introduce support for trajectory data. Of course, trajectories can already be modeled once spatial and temporal support are in place, however, there are systems that natively support dedicated primitives and operators for trajectories, such as the system Hermes [59]. The system Hermes is an extension of an RDBMS with special data stractures for trajectories and dedicated operators. The query language GeoSPARQL could be extended to support similar features and Ontop-spatial could be extended to support systems like Hermes to efficiently query trajectory data.

[4] https://www.w3.org/2015/spatial/wiki/Coverages_in_Linked_Data

ABBREVIATIONS - ACRONYMS

RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
OWL	Web Ontology Language
OGC	Open Geospatial Consortium
OBDA	Ontology-based Data Access
WKT	Well-known text
WKB	Well-known binary
GML	Geography Markup Language
R2RML	RDB-to-RDF Mapping Language
RML	RDF Mapping Language

REFERENCES

[1] Open Geospatial Consortium. OpenGIS Simple Features Specification For SQL. OGC Implementation Standard, 1999.

[2] Towards the integration of maritime surveillance: A common information sharing environment for the EU maritime domain. Technical report, 2009. Available at `http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=URISERV:pe0011&from=EN`.

[3] James F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11), 1983.

[4] Andrej Andrejev and Tore Risch. Scientific SPARQL: semantic web queries over scientific data. In *Workshops Proceedings of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012*, pages 5–10, 2012.

[5] Stuart J. Barnes and Martin Böhringer. Continuance usage intention in microblogging services: The case of twitter. In *17th European Conference on Information Systems, ECIS 2009, Verona, Italy, 2009*, pages 556–567, 2009.

[6] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. The multidimensional database system rasdaman. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 575–577, 1998.

[7] K. Bereta and M. Koubarakis. Creating virtual semantic graphs ontop of big data from space. In *BiDS*, 2017.

[8] Konstantina Bereta, Hervé Caumont, Ulrike Daniels, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus, and Firman Wahyudi. The copernicus app lab project: Easy access to copernicus data. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 501–511, 2019.

[9] Konstantina Bereta, Hervé Caumont, Erwin Goor, Manolis Koubarakis, Despina-Athanasia Pantazi, George Stamoulis, Sam Ubels, Valentijn Venus, and Firman Wahyudi. From copernicus big data to big information and big knowledge: A demo from the copernicus app lab project. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 1911–1914, 2018.

[10] Konstantina Bereta and Manolis Koubarakis. Ontop of Geospatial Databases. In *Proceedings of the 15th International Semantic Web Conference*, 2016.

[11] Konstantina Bereta and Manolis Koubarakis. Ontop of geospatial databases. In Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I*, pages 37–52, Cham, 2016. Springer International Publishing.

[12] Konstantina Bereta, George Papadakis, and Manolis Koubarakis. Sparqling-up the web on-the-fly using ontologies and mappings. In *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018.*, 2018.

[13] Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In *Extended Semantic Web Conference 2013*, volume 7882, pages 259–274. Springer Berlin Heidelberg, 2013.

[14] Konstantina Bereta, George Stamoulis, and Manolis Koubarakis. Ontology-based data access and visualization of big vector and raster data. In *2018 IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2018, Valencia, Spain, July 22-27, 2018*, pages 407–410, 2018.

[15] Michael H. Boelen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in Temporal Databases. *IEEE CS*, 19:35–42, 1996.

[16] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *ISWC-12*, volume 2342 of *LNCS*, pages 54–68. Springer,

2002.

[17] Stefan Bruggemann, Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. *Ontology-Based Data Access for Maritime Security*, pages 741–757. Springer International Publishing, 2016.

[18] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web Journal*, 2016. (to appear).

[19] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web Journal*, 8(3):471–487, 2017.

[20] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

[21] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO System for Ontology-based Data Access. *Semant. Web journal*, 2(1), 2011.

[22] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[23] Alexandros Chortaras and Giorgos Stamou. Mapping diverse data to RDF in practice. In *ISWC*, pages 441–457, 2018.

[24] Yannis Chronis, Yannis Foufoulas, Vaggelis Nikolopoulos, and et al. A Relational Approach to Complex Dataflows. In *EDBT/ICDT Workshops*, 2016.

[25] James Clifford, Curtis Dyreson, Tomas Isakowitz, Christian S. Jensen, and Richard Thomas Snodgrass. On the semantics of now in databases. *ACM TODS*, 22(2):171–214, 1997.

[26] Open Geospatial Consortium. OGC GeoSPARQL - A geographic query language for RDF data. OGC Candidate Implementation Standard, 2012.

[27] Philippe Cudré-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Roman Simakov, Emad Soroush, Pavel Velikhov, Daniel L. Wang, Magdalena Balazinska, Jacek Becla, David J. DeWitt, Bobbi Heath, David Maier, Samuel Madden, Jignesh M. Patel, Michael Stonebraker, and Stanley B. Zdonik. A demonstration of scidb: A science-oriented DBMS. *PVLDB*, 2(2):1534–1537, 2009.

[28] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. W3C Recommendation, W3C, 2014. Available at `https://www.w3.org/TR/rdf11-concepts/`.

[29] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: Rdb to rdf mapping language, 2012. W3C Rec.

[30] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium, 2012. Available at `http://www.w3.org/TR/r2rml/`.

[31] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In *LDOW*, 2014.

[32] MaxJ. Egenhofer. A formal definition of binary topological relationships. In *Foundations of Data Organization and Algorithms*, volume 367 of *Lecture Notes in Computer Science*, pages 457–472. Springer Berlin Heidelberg, 1989.

[33] Thomas Eiter, Thomas Krennwallner, and Patrik Schneider. Lightweight spatial conjunctive query answering using keywords. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 243–258, 2013.

[34] George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A Benchmark for Geospatial RDF stores (long version). volume 8219 of *Lecture Notes in Computer Science*, pages 343–359. Springer, 2013.

[35] Konstantinos Giannousis, Konstantina Bereta, Nikolaos Karalis, and Manolis Koubarakis. Distributed execution of spatial SQL queries. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 528–533, 2018.

[36] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 entailment regimes. W3C Recommendation, W3C, March 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.

[37] Claudio Gutierrez, Carlos Hurtado, and Ro Vaisman. Temporal RDF. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *LNCS*, pages 93–107. Springer, 2005.

[38] Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM TODS*, 25(1):1–42, 2000.

[39] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C Recommendation, W3C, March 2013. Available at `http://www.w3.org/TR/sparql11-query`.

[40] Steven Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C recommendation, March 2013.

[41] John R. Herring. OpenGIS implementation specification for geographic information - simple feature access - part 2: SQL option. OpenGIS Implementation Standard 06-104r4, Open Geospatial Consortium Inc., 2010.

[42] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium, 2004.

[43] Akshay Java, Xiaodan Song, Tim Finin, and Belle L. Tseng. Why we twitter: An analysis of a microblogging community. In *Advances in Web Mining and Web Usage Analysis, 9th International Workshop on Knowledge Discovery on the Web, WebKDD 2007, and 1st International Workshop on Social Networks Analysis, SNA-KDD 2007, San Jose, CA, USA, August 12-15, 2007. Revised Papers*, pages 118–138, 2007.

[44] Mark Kaminski, Egor V. Kostylev, and Bernardo Cuenca Grau. Query nesting, assignment, and aggregation in SPARQL 1.1. *ACM Transactions on Database Systems*, 42(3):17:1–17:46, 2017.

[45] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *ISWC-14*, volume 8796 of *LNCS*, pages 552–567. Springer, 2014.

[46] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of International Semantic Web Conference (ISWC 2014)*, Lecture Notes in Computer Science. Springer, 2014. (Accepted).

[47] Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In Lora Aroyo and et al., editors, *ESWC*, volume 6088 of *LNCS*, pages 425–439. Springer, 2010.

[48] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A Semantic Geospatial DBMS. In Philippe Cudré-Mauroux and et al., editors, *ISWC*, volume 7649 of *LNCS*, pages 295–311. Springer, 2012.

[49] Kostis Kyzirakos, Ioannis Vlachopoulos, Dimitrianos Savva, Stefan Manegold, and Manolis Koubarakis. Geotriples: a tool for publishing geospatial data as RDF graphs using R2RML mappings. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, pages 393–396, 2014.

[50] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *ESWC*, pages 35–50, 2017.

[51] Frank Manola and Eric Mille. RDF primer. W3C Recommendation, World Wide Web Consortium, February 2004. Available at `http://www.w3.org/TR/rdf-primer-20040210/`.

[52] Wouter Maroy, Anastasia Dimou, Dimitris Kontokostas, Ben De Meester, Ruben Verborgh, Jens Lehmann, Erik Mannens, and Sebastian Hellmann. Sustainable Linked Data Generation: The Case of DBpedia. In *ISWC*, pages 297–313, 2017.

[53] Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1-2):343–385, 1996.

[54] Franck Michel, Catherine Faron-Zucker, and Fabien Gandon. Sparql micro-services: Lightweight integration of web apis and linked data. In *Workshop on Linked Data on the Web co-located with 27th International World Wide Web Conference (WWW 2018)*, 2017.

[55] Matthieu Mosser, Fernando Pieressa, Juan L. Reutter, Adrián Soto, and Domagoj Vrgoc. Querying

apis with SPARQL: language and worst-case optimal algorithms. In *ESWC*, pages 639–654, 2018.
[56] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, W3C, December 2012.
[57] Charalampos Nikolaou, Kallirroi Dogani, Konstantina Bereta, George Garbis, Manos Karpathiotakis, Kostis Kyzirakos, and Manolis Koubarakis. Sextant: Visualizing time-evolving linked geospatial data. *J. Web Sem.*, 35:35–52, 2015.
[58] Özgür Lütfü Özçep and Ralf Möller. Scalable geo-thematic query answering. In *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pages 658–673, 2012.
[59] Nikos Pelekis, Elias Frentzos, Nikos Giatrakos, and Yannis Theodoridis. HERMES: A trajectory DB engine for mobility-centric applications. *IJKBO*, 5(2):19–41, 2015.
[60] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):16:1–16:45, 2009.
[61] David Peterson, Shudi (Sandy) Gao, Ashok Malhotra, C. M. Sperberg-McQueen, and Henry S. Thompson. W3C XML schema definition language (XSD) 1.1 part 2: Datatypes. W3C Recommendation, W3C, 2012. Available at `https://www.w3.org/TR/xmlschema11-2/`.
[62] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
[63] Freddy Priyatna, Óscar Corcho, and Juan F. Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In *WWW*, pages 479–490, 2014.
[64] Eric Prud'hommeaux and Carlos Buil-Aranda. SPARQL 1.1 Federated query. W3C Recommendation, World Wide Web Consortium, March 2013. Available at `www.w3.org/TR/sparql11-federated-query/`.
[65] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992.*, pages 165–176, 1992.
[66] Dominique Ritze and Christian Bizer. Matching Web Tables To DBpedia - A Feature Utility Study. In *EDBT*, pages 210–221, 2017.
[67] Mariano Rodríguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *Web Semantics: Science, Services and Agents on the World Wide Web*, 33(1), 2015.
[68] Hassan Saif, Yulan He, and Harith Alani. Semantic Sentiment Analysis of Twitter. In *ISWC*, pages 508–524, 2012.
[69] Juan F. Sequeda and Daniel P. Miranker. Ultrawrap: SPARQL execution on relational data. *J. Web Sem.*, 22:19–39, 2013.
[70] Markus Stocker and Evren Sirin. Pelletspatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, 2009.
[71] Jonas Tappolet and Abraham Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In Lora Aroyo and et al., editors, *ESWC*, volume 5554 of *LNCS*, pages 308–322. Springer-Verlag, 2009.
[72] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *J. Web Sem.*, 37-38:184–206, 2016.
[73] W3C. Sparql 1.1 entailment regimes. Technical report, W3C, March 2013.
[74] W3C OWL Working Group. OWL 2 Web Ontology Language document overview (second edition). W3C Recommendation, W3C, December 2012.
[75] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *IJCAI-ECAI-18 – July 13-19 2018, Stockholm, Sweden*, 2018.
[76] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *IJCAI-ECAI-18 – July 13-19 2018, Stockholm, Sweden*, 2018.
[77] Guohui Xiao, Dag Hovland, Dimitris Bilidas, Martin Rezk, Martin Giese, and Diego Calvanese. Efficient ontology-based data integration with canonical iris. In *ESWC*, volume 10843 of *Lecture Notes in*

Computer Science, pages 697–713. Springer, 2018.

[78] Ying Zhang, Martin L. Kersten, and Stefan Manegold. Sciql: array data processing inside an RDBMS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1049–1052, 2013.