Combining and Uniting Business Intelligence with Semantic Technologies
Acronym: CUBIST Project No: 257403
Small or Medium-scale Focused Research Project FP7-ICT-2009-5 Duration: 2010/10/01-2013/09/30

OLAP extensions to SPARQL

Abstract:

Type:	Report
Document ID:	CUBIST D3.1.2
Workpackage:	WP3
Leading partner:	ONTO
Author(s):	Marin Dimitrov (ONTO) Alex Simov (ONTO) Barry Bishop (ONTO)
Dissemination level:	PU
Status:	Final
Date:	21 October 2011
Version:	1.0

Versioning and contribution history

Version	Description	Contributors
0.1	Initial version	Alex Simov
0.2	Introduction and conclusion sections	Marin Dimitrov
0.3	Initial proof reading & review	Marin Dimitrov
1.0	Internal review comments taken into consideration	Alex Simov

Table of contents

1 Introduction

The purpose of this deliverable is to outline – based on the input from the project use cases – extensions to the SPARQL 1.0 language that will make it suitable for use in a Business Intelligence context. At the time of planning this deliverable the plans of the SPARQL Working Group at W3C regarding potential extensions of the first version of the query language were still not clear, so this deliverable was expected to fill a gap between what was available in SPARQL 1.0 and what was needed by the CUBIST use cases. Meanwhile (after the project proposal has been officially submitted), the SPARQL WG resumed its work and outlined a roadmap for extending the query language, which significantly overlapped with the scope and purpose of this deliverable.

Since the new features in the SPARQL extension roadmap were prioritized on a "time permitting" basis (e.g. the WG would scope out some features during its work), we decided to not completely drop D3.1.2 from the CUBIST project plan, but to narrow down its scope, so that we will directly incorporate extensions from the SPARQL WG when available, but also propose extensions which were out of the scope of the SPARQL 1.1 work, but were still needed by the CUBIST use cases. The reported effort for D3.1.2 will also be significantly reduced from the original plan, in order to correctly reflect the fact that some work was not to be done by the CUBIST partners, but would be directly reused from the SPARQL WG.

Deliverable D3.1.1 gathered the use case requirements with respect to potential queries of interest for each use case. We can distinguish between several different types of requirements:

1. Requirements that can be satisfied with SPARQL 1.0 (requirements HWU01-04, HWU06)
2. Requirements that can be satisfied only with SPARQL 1.1 (requirements HWU01-04, HWU06-07, HWU09, SAS03, SAS05, SAS09-12, SAS22, INN03-04)
3. Requirements that can be satisfied with custom OWLIM extensions, but fall out of the current scope of SPARQL 1.0 and 1.1 (SAS09, SAS16, SAS18, SAS21, SAS30-31, INN01-02, INN04-06)
4. Requirements that need new functionality extensions, not available in SPARQL 1.1 and OWLIM at present (HWU05)
5. Requirements which technically are not query language requirements but require some preliminary analytical pre-processing or complex workflows. These fall out of scope for this deliverable (HWU08, HWU10-13, SAS01-02, SAS04, SAS07-08, SAS13-15, SAS17, SAS23-28, SAS32-36, INN07)

Note: some requirements might fall in more than one category, as they either represent several classes of queries (query + negated query) or implementations benefit from different sets of features (expressiveness and efficiency aspects).

This deliverable is organised as follows:

- Section 2 provides a summary of the CUBIST use case requirements (as described in D3.1.1) together with the reference to the corresponding SPARQL query features or extensions that provide support for each requirement
- Section 3 provides a summary of the SPARQL 1.1 features (as defined by the SPARQL WG at W3C), which are overlapping with the initial plan for D3.1.2 and are directly incorporated in the deliverable. As already explained above, no effort from the CUBIST partners is reported for this work, since it was done outside of the project.
- Section 4 provides a summary of extensions to the SPARQL language which fall out of the scope of SPARQL 1.1 (as defined by the SPARQL WG), but are still required for the CUBIST

use cases. We have provided references to such extended functionality available in the OWLIM database (which will be used as the RDF data warehouse in the project). All of these features use extension mechanisms which *do not* break SPARQL standard compatibility.

- Finally, section 5 outlines extensions to the SPARQL language, which are neither available in the 1.1 version of the standard, nor available as custom extensions in the OWLIM database, so these will be implemented as part of Task 3.2 in order to support the respective use case requirements.

2 Summary of the use case requirements

This section is based on the requirements for extending the SPARQL query language, which were provided by the CUBIST use cases within deliverable D3.1.1. Here we provide implementation guidelines, aligned with the latest achievements in the field of triple store query language (a.k.a. SPARQL) and its implementation in OWLIM. Certain requirements which cannot be covered by the query language are met by specific extensions of the implementation (not breaking the general query syntax).

The following sections list the requirements from the use case partners, each having its original ID (from deliverable D3.1.1), description and related section(s) concerning its implementation. Some of the requirements are already satisfied by SPARQL 1.0 (pre CUBIST) and some requirements do not concern the query language but require more complex analytical pre-processing or complex workflows – for such requirements no relevant reference section is provided in this document, since these fall out of the scope of this deliverable.

2.1 Requirements from HWU (WP6)

ID	Requirement Content	Related Section
HWU01	Where is *gene G* (not) detected?	3.2.1 (negation)
HWU02	Where is *gene G* (not) detected in developmental stages *S1* to *S2*?	3.2.1 (negation)
HWU03	What genes are (not) detected in *anatomy term T*?	3.2.1 (negation)
HWU04	What genes are (not) detected in *anatomy term T* in developmental stages *S1* to *S2*?	3.2.1 (negation)
HWU05	What genes are (not) detected in *3D spatial region R* ?	3.2.1 (negation), 5.1 (3D support)
HWU06	Where are the genes involved in *biological process P* (not) detected?	3.2.1 (negation)
HWU07	Detection of possible inconsistencies and errors in the gene expression information	3.2.1 (negation), 3.3 (sub-query), 3.5 (functions)
HWU08	How does the level of expression *for gene G* change from developmental stage S1 to stage S2?	-
HWU09	How does the pattern of expression for gene G change from developmental stage S1 to stage S2?	3.2.1 (negation)
HWU10	Which genes have similar expression patterns in *stages S1* to *S2*?	-
HWU11	What do the genes involved in *biological process P* have in common?	-
HWU12	What pathways does expression pattern cluster C correspond to?	-
HWU13	Based on *expression pattern EP* from *experiment E*, which stage in the pathway was *E* performed at?	-

2.2 Requirements from SAS (WP7)

ID	Requirement Content	Related Section
SAS01	Console Logs: go through all console logs since previous shifts to check for open issues, actions to follow up, general awareness, ...	-

SAS02	TM archive: Check for completeness of the archive	-
SAS03	TM Displays: during periods of ‘data connection’ monitoring of the on-going activities by telemetry monitoring of the housekeeping data of the instruments and payload (temperatures, modes, command schedule tracking, ...)	3.2.1 (negation)
SAS04	ISS ATL: check for possible reboots of the station which requires a special SOLAR configuration	-
SAS05	SOLAR Mission Tool: check for upcoming activities: Files needed are indeed available onboard	3.2.1 (negation)
SAS07	SOLAR Mission Tool: check for upcoming activities: Timeline coherence	-
SAS08	CEFN : Timeline review: check OSTPV and SOLAR Mission Tool, provide input through CEFN where needed	-
SAS09	CEFN : Requests for Powerdown: Check for payload developer’s MEMO for minimal required SOLAR power consumption + provide into CEFN	3.1 (aggregation), 4.2 (fts)
SAS10	CEFN : IPV review: Check whether PODF related to B.USOC Payloads are correctly uplinked into new IPV+ provide approval (or approval with modification)	3.2.1 (negation)
SAS12	CEFN : Open ARs : check AR that may affect the payload, if needed report to COL FCT	3.1 (aggregation), 3.2.1 (negation)
SAS13	CEFN : Ground Segment operations, Check for CEFNs regarding ground segment maintenance or so, check impact on the SOLAR operations and inform Ground Controller	-
SAS14	CEFN : Older CEFNs: check previous inputs	-
SAS15	MDB: check command stack for upcoming activities	-
SAS16	IPV/PODF: retrieve correct PODF for execution activities	4.2 (fts)
SAS17	IOT-AR/IOT-SPR: check open actions on B.USOC side	-
SAS18	SOLAR Documentation: used as background information to successfully operate SOLAR	4.2 (fts)
SAS21	BUSOC Mail: check for PIs input, files that have been provided, requests from other operations	4.2 (fts)
SAS22	Displays: check which parameters were off nominal	3.1 (aggregation), 3.2.1 (negation), 3.5 (functions)
SAS23	Archive: Retrieve specific parameters of the occurrence / period to insert in an Excel file	-
SAS24	Archive: Replay the occurrence using the displays	-
SAS25	MDB/Command History: Check which commands have been sent	-
SAS26	IPV/PODF: check which procedure has been used	-
SAS27	IOT-AR: check whether the anomaly has occurred before and the analysis or CARTs disposition, if so: a) CEFN: If Anomaly has occurred before check recovery procedure used back then b) IOT-SPRs: check the offline analysis if the anomaly has occurred before c) Check Console logs of previous occurrence	-
SAS28	Console Log: Check the logs of all involved parties during the occurrence	-
SAS30	COL DMS: retrieve engineering documentation to support the analysis or recovery actions (Safety DP, User Manuals, ...)	4.2 (fts)

SAS31	SOLAR Documentation: check reference documentation for analysis	4.2 (fts)
SAS32	SOLAR operations Documentation: check reference on the agreed operations concept	-
SAS33	SOLAR shiftplanner Tool (JMW tool) : check which ground controller is on call	-
SAS34	SOLAR POC list: check numbers of PI/Payload Developer/.. to contact them for support	-
SAS35	SOLAR Mission Tool: retrieve current planning to update it	-
SAS36	SOLAR Mission Tool: retrieve filenames to be uploaded	-

2.3 Requirements from Innovantage (WP8)

ID	Requirement Content	Related Section
INN01	Job Vacancy search by title, description, advertiser organization name or type (agency or direct employer), location hierarchy, category hierarchy, date posted, business sector, salary, job type, source(job boards or company website)	4.2 (fts)
INN02	Quick Vacancy search through keywords using Lucene text search facility	4.2 (fts)
INN03	Statistics from search results such as % of jobs from various sources, locations, job categories, industry sector etc.	3.1 (aggregation), 3.4 (complex select) 3.5 (functions)
INN04	Company lookup for the advertisers from 3rd party directories such as Market Location or DNB, jobs associated and various recruitment statistic about an advertiser	3.1 (aggregation), 4.2 (fts)
INN05	Yahoo news articles relevant to a particular advertiser by keyword match	4.2 (fts)
INN06	Radius job search around a town, county or postcode or part of a postcode by number of miles using the geo-spatial extension of SPARQL	4.1 (geo-spatial), 4.2 (fts)
INN07	Skill, qualifications, company information within job description	-

3 Overview of SPARQL 1.1 features

This section focuses on the new features in the SPARQL 1.1 version of the standard, as specified by the latest *W3C SPARQL Working Group* draft[1].

3.1 Aggregation functionality

Aggregates apply expressions over groups of solutions. Grouping may be specified using the *GROUP BY* syntax. Aggregates defined in version 1.1 of SPARQL are:

- *COUNT*
- *SUM*
- *MIN/MAX*
- *AVG*
- *GROUP_CONCAT*
- *SAMPLE*

The aggregated solution sets can be filtered by using the *HAVING* operator (similar to the standard SQL query syntax).

Example:

```
SELECT (AVG(?size) AS ?asize)
WHERE {
  ?x :size ?size
}
GROUP BY ?x
HAVING (AVG(?size) > 10)
```

This query will return average sizes, grouped by the subject, but only where the mean size is greater than 10.

3.2 Negation

The SPARQL 1.1 query language incorporates two styles of negation, one based on filtering results depending on whether a graph pattern does or does not match in the context of the query solution being filtered, and one based on removing solutions related to an additional query pattern.

3.2.1 Detection of (not) matching patterns

Filtering of query solutions is done within a *FILTER* expression using *NOT EXIST* and *EXISTS*. The *(NOT) EXISTS* filter expression tests whether a graph pattern does (not) match the dataset, given the values of variables in-scope. It does not generate any additional bindings.

Example: select all persons without a name provided

```
SELECT ?person
WHERE {
   ?person rdf:type foaf:Person .
   FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

[1] By the time of writing the deliverable, the working draft has a status *Last Call Working Draft*

3.2.2 Removing possible solutions

The other style of negation provided in SPARQL 1.1 is the *MINUS* operator, which evaluates both its arguments, then calculates solutions in the left-hand side that are not compatible with the solutions on the right-hand side.

Example: select persons which have a name other than "Bob"

```
SELECT DISTINCT ?person
WHERE {
   ?person rdf:type foaf:Person .
  MINUS {
     ?person foaf:name "Bob" .
  }
}
```

3.3 Sub-query support

Sub-queries are a way to embed SPARQL queries within other queries, such as limiting the number of results from some sub-expression within the query.

Due to the bottom-up nature of the SPARQL query evaluation, subqueries are evaluated logically *first*, and the results are projected up to the outer query.

Example: return a name (the one with the lowest sort order) for all the people that know Alice and have a name.

```
SELECT ?y ?minName
WHERE {
 :alice :knows ?y .
 {
   SELECT ?y (MIN(?name) AS ?minName)
   WHERE {
     ?y :name ?name .
   }
   GROUP BY ?y
 }
}
```

3.4 Expressions in the SELECT clause

In SPARQL 1.1 the *SELECT* clause can also introduce new variables. The rules of assignment in the *SELECT* expression are the same as for assignment in a *BIND* clause (see Assignment section). The expression combines variable bindings already in the query solution, or defined earlier in the *SELECT* clause, to produce a binding in the query solution.

Example:

```
SELECT  ?title (?p * (1 - ?discount) AS ?price)
WHERE {
  ?x ns:price ?p .
  ?x dc:title ?title .
  ?x ns:discount ?discount
}
```

3.5 An expanded set of functions and operators

A summary of the major improvements of the functions and operator sets in SPARQL 1.1 is provided below:

- Conditional constructs *IF-THEN-ELSE*
- Set test operators (*IN*, *NOT IN*)
- String processing functions (*STRLEN*, *SUBSTR*, *CONTAINS*, etc.)
- Mathematical functions (*ABS*, *ROUND*, *FLOOR*, *CEIL*, *RAND*, etc.)
- Date and time functions
- Hash functions (MD5, SHA1, etc.)

3.6 Assignment

The value of an expression can be added to a solution by binding a new variable to the value of the expression, which is an RDF term. In SPARQL, this binding within a query solution is never changed. The new variable must not already be in-scope in the query at that point it is used. The variable can then be used in the query and also can be returned in results.

Three syntax forms allow this: the *BIND* keyword, expressions in the *SELECT* clause and expressions in the *GROUP BY* clause. The assignment form is (*expression AS ?variable*).

The *BIND* form allows a value to be assigned to a variable in a group graph pattern. The use of *BIND* is a separate element of a group graph pattern and it ends any basic graph pattern.

Example:

```
SELECT  ?title ?price
WHERE {
   ?x ns:price ?p .
   ?x ns:discount ?discount .
   ?x dc:title ?title
   BIND (?p * (1 - ?discount) AS ?price)
   FILTER(?price < 20)
}
```

4 Extended features supported by OWLIM

4.1 Geo-spatial support

OWLIM provides support for querying of 2-dimensional geo-spatial data based on the WGS84 Geo Positioning RDF vocabulary[2] (World Geodetic System 1984).

Special indices can be used for this data that permit the efficient evaluation of special query forms and extension functions that allow:

- locations to be found that are within a certain distance of a point, i.e. within the specified circle on the surface of the sphere (Earth), using the *NEARBY* construct;
- locations which are within rectangles and polygons, where the vertices are defined using spherical polar coordinates, using the *WITHIN* construct.

4.1.1 Implementation details

- Spherical Earth – the current implementation treats the Earth as a perfect sphere
- Only 2 dimensional points are supported, i.e. latitude and longitude but no altitude
- All latitude and longitude values must be specified using decimal degrees, where East and North are positive and the latitude is between (-90, +90) while longitude is between (-180, +180).
- Distances must be in units of kilometres (suffix 'km') or statute miles (suffix 'mi')

4.1.2 Geo-spatial query syntax

The special syntax used to query geo-spatial data makes use of SPARQL's RDF Collections syntax. This syntax uses round brackets as a shorthand for the statements connecting a list of values using *rdf:first* and *rdf:rest* predicates with terminating *rdf:nil*. Statement patterns that use one of the special geo-spatial predicates supported by OWLIM are treated differently by the query engine. The following special syntax is supported when evaluating SPARQL queries (the descriptions all use the namespace *omgeo:<http://www.ontotext.com/owlim/geo#>*):

4.1.2.1 *NEARBY* pattern

Syntax: *?point omgeo:nearby(?lat ?long ?distance)*

This statement pattern will evaluate to true if the following constraints hold:

- ?point geo:lat ?plat .
- ?point geo:long ?plong .
- Shortest great circle distance from (?plat, ?plong) to (?lat, ?long) <= ?distance

Such a construct will use the geo-spatial indices to find bindings for the *?point* variable, which are within the defined circle.

Constants are allowed for any of the parameters (latitude, longitude, and distance), where latitude and longitude are specified in decimal degrees and distance is specified in either kilometres ('km' suffix) or miles ('mi' suffix).

[2] http://www.w3.org/2003/01/geo/wgs84_pos

Example: Find the names of airports that are within 50 miles of Seoul

```
SELECT DISTINCT ?airport_name
WHERE {
  ?city geo-ont:name "Seoul" .
  ?city geo-pos:lat ?latCity .
  ?city geo-pos:long ?longCity .
  ?airport omgeo:nearby(?latCity ?longCity "50mi") .
  ?airport geo-ont:name ?airport_name .
  ?airport geo-ont:featureCode geo-ont:S.AIRP .
}
```

4.1.2.2 *WITHIN* pattern (rectangle)

Syntax: *?point omgeo:within(?lat1 ?long1 ?lat2 ?long2)*

This pattern is used to find points that lie within the rectangle specified by diagonally opposite corners (*?lat1*, *?long1*) and (*?lat2*, *?long2*). The corners of the rectangle must be either constants or bound values.

The pattern expression will evaluate to *true* if the following constraints hold:

- ?point geo:lat ?plat .
- ?point geo:long ?plong .
- ?lat1 <= ?plat <= ?lat2
- ?long1 <= ?plong <= ?long2

Constants are allowed for any of the parameters, where latitude and longitude are specified in decimal degrees. If *?point* is unbound then bindings for all points within the rectangle will be produced.

Example: Find tunnels lying within a rectangle enclosing Tirol, Austria

```
SELECT DISTINCT ?name
WHERE {
   ?tunnel omgeo:within(45.85 9.15 48.61 13.18) .
   ?tunnel geo-ont:featureCode geo-ont:R.TNL .
   ?tunnel geo-ont:name ?name .
}
```

4.1.2.3 *WITHIN* pattern (polygon)

Syntax: *?point omgeo:within(?lat1 ?long1 ... ?latn ?longn)*

This pattern is used to find points within the polygon whose vertices are specified by three or more latitude/longitude pairs. The polygon is closed automatically if the first and last vertices do not coincide.

Example: Find caves lying within the polygon approximating the shape of England:

```
SELECT ?name ?lat ?long
WHERE {
?cave omgeo:within( "51.45" "-2.59" "54.99" "-3.06" "55.81" "-2.03" "52.74"  "1.68" "51.17"  "1.41" ) .
?cave geo-ont:featureCode geo-ont:S.CAVE .
?cave geo-ont:name ?name .
?cave geo-pos:lat ?lat .
?cave geo-pos:long ?long .
}
```

4.1.2.4 DISTANCE function

Syntax: *double omgeo:distance(?lat1, ?long1, ?lat2, ?long2)*

This SPARQL extension function computes the distance between two points (in kilometres) and can be used in *FILTER* and *ORDER BY* clauses.

Example: Find all the airports within 80 miles of Bournemouth and filter out those that are more than 80 kilometres from Brize Norton, order the results with the closest to Brize Norton first:

```
SELECT DISTINCT ?airport_name
WHERE {
  ?city1 geo-ont:name "Bournemouth" .
  ?city1 geo-pos:lat ?lat1 .
  ?city1 geo-pos:long ?long1 .
  ?airport omgeo:nearby(?lat1 ?long1 "80mi" ) .
  ?airport geo-ont:name ?airport_name .
  ?airport geo-ont:featureCode geo-ont:S.AIRP .
  ?airport geo-pos:lat ?lat .
  ?airport geo-pos:long ?long .
  ?city2 geo-ont:name "Brize Norton" .
  ?city2 geo-pos:lat ?lat2 .
  ?city2 geo-pos:long ?long2 .
  FILTER( omgeo:distance(?lat, ?long, ?lat2, ?long2) < 80)
}
ORDER BY ASC( omgeo:distance(?lat, ?long, ?lat2, ?long2) )
```

4.2 Full-text indexing and search

Full-text search (FTS) provides the means for retrieving text documents out of a large collection by keywords or, more generally, by tokens (represented as sequences of characters). Generally, the full-text query represents an unordered set of tokens and the result is set of documents, relevant to the query.

FTS and structured queries, like those in database management systems (DBMS), provide different information access methods based on different query syntax and semantics. FTS and databases usually require different types of indices too. The ability to combine these two types of information access methods is very useful for a wide range of applications.

Many RDBMS provide custom FTS extensions (which are integrated into the standard SQL querying syntax) and maintain additional indices that allow efficient evaluation of FTS constraints. Typically, RDBMS allow the user to define a query, which requires specific tokens to appear in a specific column of a specific table.

In SPARQL 1.1 there is no standard way for the specification of FTS constraints (further versions of the specification may provide such standardization). Nevertheless, several RDF database vendors have already implemented some sort of FTS extensions into their databases. This section describes the FTS supported by the OWLIM database. FTS extensions from other vendors are conceptually similar.

OWLIM provides full text search capabilities based on the popular open source Lucene[3] platform, with a variety of indexing options and the ability to simultaneously use multiple indices in the same query.

In order to use the full-text search in OWLIM a Lucene index must first be created. Each index can be parameterised in a number of ways using SPARQL ASK queries. This provides the ability to:

[3] http://lucene.apache.org/

- select what kinds of nodes in the RDF graph are indexed (URIs/literals/blank-nodes)
- select what is included in the 'molecule' (adjacent nodes in the graph) associated with each node
- select literals with certain language tags
- choose the size of the RDF 'molecule' to index
- select from various content analysers
- select various resultset scorers

To embed a full-text search query within SPARQL query with OWLILM, a special triple pattern is used, where the Lucene index name is the predicate and the Lucene query itself is the object of the statement.

The following query will produce bindings for *?s* from entities in the repository, where the RDF molecule associated with that entity (for the given index) contains terms that begin with "United". Furthermore, the bindings will be ordered by relevance (with any boosting factor):

```
PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
SELECT ?s
WHERE {
   ?s luc:myIndex "United*" .
}
```

4.3 RDF priming

RDF Priming is a technique that selects a subset of available statements for use as the input to query answering. It is based upon the concept of 'spreading activation' as developed in cognitive science. RDF Priming is a scalable and customisable implementation of the popular connectionist method on top of RDF graphs that allows for the "priming" of large datasets with respect to concepts relevant to the context and to the query.

Various parameters of the spreading activation process can be configured according to the specific application or use case needs:

- *Statement weight* (for statements with a specific predicate)
- *Activation threshold* (guarantees that the activation value of a node won't grow beyond a certain value)
- *Filter threshold* (specifies whether a statement is visible depending on the activation level of its Subject, Predicate and Object)
- *Firing threshold* (the threshold above which a node will activate its neighbours)
- *Decay factor* (control how much a node's activation level is transferred to nodes that it affects)
- *Number of cycles* (the number of activation spreading cycles to perform when the process is initiated)
- Default/initial *activation value*, default *activation weight*

The RDF Priming feature and its underlying spreading activation process provide a very powerful way to customize the results returned by a SPARQL query, based on a user-defined context (e.g. important statements, weights and thresholds). Implementation details and usage example are available in OWLIM's advanced features documentation[4].

[4] http://owlim.ontotext.com/display/OWLIMv42/OWLIM-SE+Advanced+Features#OWLIM-SEAdvancedFeatures-RDFPriming

5 Required Extensions

5.1 3D/2D Spatial Querying

Although certain effort has been invested in spatial indexing/querying (see section 4.1) its usage is restricted to the domain of *geo-spatial* querying and indexing, which does not cover all of the use case requirements (in particular the 3D spatial requirements from the HWU use case). In CUBIST, this functionality will be extended to cover broader range of use cases.

The extension should have the following features:

- Support for arbitrary 2D and 3D (orthogonal) coordinate systems
- Indexing locations with respect to certain coordinate system
- Efficient searching within regions

The querying mechanism in SPARQL will not change essentially from what was already described in section 4.1. New statement patterns will be introduced for matching entities in certain distance from a certain location or in regions delimited by lines/planes.

6 Conclusion

This deliverable builds upon the aggregated use case requirements for extending the SPARQL language outlined in D3.1.1. The requirements were analysed and the proper recommendation for satisfying these requirements were provided:

- Features already available in SPARQL 1.0
- Features available in the latest SPARQL 1.1 drafts
- Features which are out of the scope of the current SPARQL standards, but are nonetheless available in the OWLIM database which will provide the infrastructure for the RDF data warehouse in CUBIST
- New features which need to be developed within WP3

Additionally, we have identified several use case requirements which need more complex workflow or analytic pre-processing, e.g. these will not be reflected directly in the query language extensions but will be satisfied by other means.

As already clarified, the scope of this deliverable was narrowed down due to the unforeseen (at the time of the project proposal submission) overlap with activities in the W3C SPARQL Working Group towards extending the SPARQL language. We have directly adopted these W3C extensions in our recommendation and since this work was performed outside of the CUBIST project, the consortium partners have reduced their reported resource spending accordingly.