Integrator Design Document

Integrator Design Document

Motivation

Identify use cases that exploit:

- Horizontal integration of data sources. Sources provide the same type of information with respect to the global schema. The data may overlap (redundancy) or be complementary (partitioned) and it can be seen as union of homogeneous information. This is typically the case for sources with different spatial jurisdictions for instance.
- Vertical integration of sources. Sources provide different information with respect to the global schema. In this case data of each source adds additional information to an original set. In this case the joins are between different kinds of information and finding their relationships is not always straightforward.
- Integration of different streaming sources. Answer queries including data from 2+ streaming sources.
- Integration of streaming and stored data.
- Integration of streaming data in the past and current streaming data

SemSorGrid4Env flood use case

Integration storyboard. Proposal being refined with EMU/Soton.

1. User has their profile role linked to flood events in the area of South England.
2. User sees a map with data values for tide height. Heights and wind speed registered values are displayed in the user interface continuously generated charts.
    - Values generated by continuous query which merges data from several sources
        - tide height measurements
            - Get tide height measurements from CCO
            - Get tide height measurement from BODC
            - Wave height measurements form WaveNet
        - wind speed measurements
            - Get data from CCO
            - Get from Met Office
            - wunderground RSS feed
3. A Notification window pops up on the user display as a potential flood event in the area of interest has been detected. The pop-up indicates the place of potential danger, e.g. Portsmouth. In addition to the merge query, a query that identifies potential flood events has been configured (data sources as in 2).
    - Query detects if the wave height is higher than some threshold (e.g. See storm prediction).
    - Query combines tide height and wind speed so that we get a flood warning on certain conditions (See Flood prediction under discussion).
4. User clicks on Portsmouth to see more information about the place, so as to assess the risks to infrastructure and services. The user is presented with a list of information

that is available through the registered sources for the endangered area. For example, information about

- Transportation stops, by postcode through the NPTG gazeteer and Naptan
  - bus stations
  - train stations
  - ferry stops
- Schools from data.gov.uk endpoint by location
- Hospitals from data.gov.uk endpoint by location

5. The user selects the additional information they want to see about this particular location.
   - System generates query to the integrator
   - Integrator retrieves and filters data from sources
   - Application tier receives information from integrator and converts into GeoJSON layer
6. An additional layer is added to the user's display presenting the additional information about the endangered area

Storm prediction:

A simple method for storm prediction compares the stream of wave height values against a threshold for that particular location.

Schema

```
locations (location:varchar, storm_threshold:float)
envdata_boscombe (timestamp:timestamp, DateTime:datetime, Lat:float,
      Lon:float, Hs:float, HMax:float, Tp:float, Tz:float, Sprp:float,
      Dirp:float, TSea:float)
envdata_...[1]
```

Query to detect storm events:

```
SELECT l.location, w.Hs, w.HMax
FROM   wave w, locations l
WHERE  w.location = l.locaction AND
      (w.Hs >= l.storm_threshold OR
       w.HMax >= l.storm_threshold)
```

Requires view `wave` to be created.

Flood prediction:

Simple version: Compare significant wave height (Ts) for waves with a period (Tp) < 10 seconds against the storm threshold value

Complex version: Combination of swell peak period > 10 seconds and associated energy > X. These values are being calculated by the CCO and made available as a chart (see example chart). Need to understand what calculation is being performed.

Use case features

From this storyboard we identify:

- Integration of streaming and stored data (storm prediction)

---

[1] There are 24 streams of this type in the CCO.

- Integration of past streaming data (still not defined, perhaps flood prediction?)
- Horizontal integration (CCO+WaveNet). Different complementary sources about wave heights
- Vertical integration: transportation information with the endangered area, storm threshold detection.

SPARQL$_{Stream}$ Language

Design of SPARQL$_{Stream}$ and relationship with C-SPARQL

Language requirements

From the above use case we see the necessity of defining a language capable of:

- Handling RDF streams
- Specifying a clear type system
- Specifying time windows over RDF Streams
- Providing a consistent mechanism and semantics to join streaming and stored data
- Providing window-to-stream operators

Language definitions

Types

- basic types:
  - triples
  - tuples (output type only)
- complex types
  - stream of triples
  - stream of tuples (output type only)
  - stream of triple windows
  - stream of tuple windows (output type only)

Definitions

We adopt the definitions for the SPARQL operators given in Perez et al (2009), and extend these to support streams.
For queries involving aggregates, we support the definition given in the SPARQL1.1 recommendation (Harris and Seaborne, 2010), which can be directly applied to operate over a window of triples as if they were a stored graph.

Definition of RDF Stream

An RDF stream is defined as

$$S = \{(\langle s, p, o\rangle, \tau) \mid \langle s, p, o\rangle \in ((I \cup B) \times I \times (I \cup B \cup L)), \tau \in \mathbb{T}\}$$

where *I* is the set of all IRIs, *B* is the set of all blank nodes, *L* is the set of all RDF literals, and *T* the set of non-decreasing timestamps.

Triples are annotated with timestamp. Timestamps are relating to the sensing time of the triple.

Definition of Time-based Windows

A window over an RDF stream is defined as

$$\omega^{\tau}_{t_s,t_e,\delta}(S) = \{\langle s,p,o\rangle \mid (\langle s,p,o\rangle,\tau_i) \in S, t_s \leq \tau_i \leq t_e\}$$

where τ is the evaluation time of the window, t_s and t_e define the time boundaries of the window, and δ is the window slide.

Note that we do not support the notion of a triple-based window since, in general, the answer to a query requires more information than can be carried by a single triple.

Window-to-Stream Operators

We adopt the SNEEql/CQL operators for converting a window into a stream of triples.

RStream adds a triple to the resulting stream for every triple in the window
IStream adds a triple to the resulting stream only if it was not in the previous window
DStream adds a triple to the resulting stream only if it was on the previous window but not in the current one.

These are defined as:

$$\mathsf{RStream}((\omega^{\tau},\tau)) = \{(\langle s,p,o\rangle,\tau) \mid \langle s,p,o\rangle \in \omega^{\tau}\}$$

$$\mathsf{IStream}((\omega^{\tau},\tau),(\omega^{\tau-\delta},\tau-\delta)) = \{(\langle s,p,o\rangle,\tau) \mid \langle s,p,o\rangle \in \omega^{\tau}, \langle s,p,o\rangle \notin \omega^{\tau-\delta}\}$$

$$\mathsf{DStream}((\omega^{\tau},\tau),(\omega^{\tau-\delta},\tau-\delta)) = \{(\langle s,p,o\rangle,\tau) \mid \langle s,p,o\rangle \notin \omega^{\tau}, \langle s,p,o\rangle \in \omega^{\tau-\delta}\}$$

where τ is the evaluation time of the window-to-stream operator, and δ is the window slide

Note about Time

It is feasible that RDF graphs capture information about when measures are made. For example, consider the following RDF statements

```
:uri1 a wave:WaveHeightMeasurement ;
        wave:hasValue 0.25 ;
        wave:hasLocation loc:BrackleshamBay ;
        wave:measuredAt "27-07-2010 09:00" .
```

Initially, the τ annotation on these 5 RDF triples will be set to "27-07-2010 09:00". However, if the triples are processed into windows and back into triples, the τ annotation of the resulting RDF stream triples would be the time of evaluation of the RStream operation.

Currently we do not support returning the details of the annotation time associated with a stream or a window.

Syntax

In SPARQL named graphs can be referenced using the FROM NAMED keywords. In this way triples from the specific referenced graph (collection of data) can be used in the query.

Streams

An RDF stream of timestamped triples can be specified using the syntax:

```
FROM NAMED STREAM <IRI>
```

Windows

Windows can be applied to a stream, using the following syntax:

```
FROM NAMED STREAM <IRI> [start TO end SLIDE literal]
```

where *start* and *end* are of the form `NOW` or `NOW - literal`, and *literal* represents some number of time unit from `{DAYS,HOURS,MINUTES,SECONDS}`.

Stored Data Scans

For continuous queries, a mechanism must be introduced to coherently join streaming and stored data. The assumption of non-changing stored data is not realistic. We use the notion of scanning stored data, with an interval parameter that dictates the frequency with which the stored data will be scanned. The scan interval is expressed as:

```
FROM <IRI> [SCAN literal]
```

where *literal* represents some time unit. The result is a window that can be evaluated just like the window produced from streams.The stored data has a time associated with it, which it the time when it was read.

Example Queries

Simple Query

Return the wave heights, locations and times where the wave height has exceeded the storm threshold for the last 10 hours. Update the result every 10 minutes, and check the storm thresholds every day.

```
PREFIX wave: <http://...>
SELECT ?loc ?hs ?time
FROM STREAM <virtualWaveStream> [NOW - 10 HOURS TO NOW SLIDE 10 MINUTES]
FROM GRAPH <virtualStormThreshold> [SCAN 1 DAY]
WHERE {
       ?w a wave:WaveHeightMeasurement ;
           wave:hasValue ?hs ;
           wave:hasLocation ?loc ;
           wave:measuredAt ?time .
       ?loc a wave:Location ;
           wave:hasStormThreshold ?limit .
       FILTER (?hs > ?limit) .
}
```

Complex Example

Calculate the fire risk rating according to the method given by (Sharplesa et al, 2009). The danger rating is computed as

where *U* is the wind-speed (km/h), *T* is the temperature (ºC), and *H* is the relative humidity (%). This index uses the fact that certain climatic conditions, i.e. higher temperature, higher wind speed, and lower humidity, lead to increased risk of fire and destructive potential of a fire. The average for each of the temperature, humidity and wind speed variables is first obtained using sub-queries, and then the formula is applied.

```
PREFIX fire: <http://www.semsorgrid4env.eu#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
SELECT RSTREAM ?WindSpeedAvg
FROM STREAM <www.semsorgrid4env.eu/SensorReadings.srdf> [NOW - 10 MINUTES TO NOW SLIDE 1 MINUTE]
FROM STREAM <www.semsorgrid4env.eu/SensorArchiveReadings.srdf> [NOW - 3 HOURS TO NOW -2 HOURS
SLIDE 1 MINUTE]
WHERE {
  {
        SELECT AVG(?speed) AS ?WindSpeedAvg
        WHERE
        {
                GRAPH <www.semsorgrid4env.eu/SensorReadings.srdf> {
                ?WindSpeed a fire:WindSpeedMeasurement;
                        fire:hasSpeed ?speed; }
        } GROUP BY ?WindSpeed
  }
  {
        SELECT AVG(?archivedSpeed) AS ?WindSpeedHistoryAvg
        WHERE
        {
          GRAPH <www.semsorgrid4env.eu/SensorArchiveReadings.srdf>  {
                ?ArchWindSpeed a fire:WindSpeedMeasurement;
                fire:hasSpeed ?archivedSpeed;  }
        } GROUP BY ?ArchWindSpeed
  }
  FILTER (?WindSpeedAvg > ?WindSpeedHistoryAvg)
}
```

Add more examples (fire use case)

Relationship with C-SPARQL

C-SPARQL is a proposal for a streaming query language for RDF streams (Barbieri et al., 2010a). In this section we compare the syntax and semantics of the C-SPARQL language with our own proposal. We postpone the discussion of the execution environment to the next section.

RDF Stream

We have adopted the C-SPARQL approach for an RDF stream, i.e. an annotated sequence of RDF triples. We have also used their mechanism for declaring a stream in the FROM clause. We extend the FROM clause to support aliases for stream graphs. This enables different windows to be specified over same graph, i.e. self-joins between the current and past of a stream graph.

Windows

We follow the C-SPARQL approach of declaring a window over a stream in the FROM clause. However, we have

- extended the expressiveness of the window operator by allowing windows to declare both the start and the end points relative to the evaluation time;
- made explicit the result type of a window operator, i.e. a window of RDF triples/tuples;
- removed the semantically ambiguous triple-based window.

We have also added support for window-to-stream operators.

Stored Data

We have provided an evaluation semantics for relating stored and streaming data by converting stored data into a stream of triple windows using the SCAN operator.

Aggregates

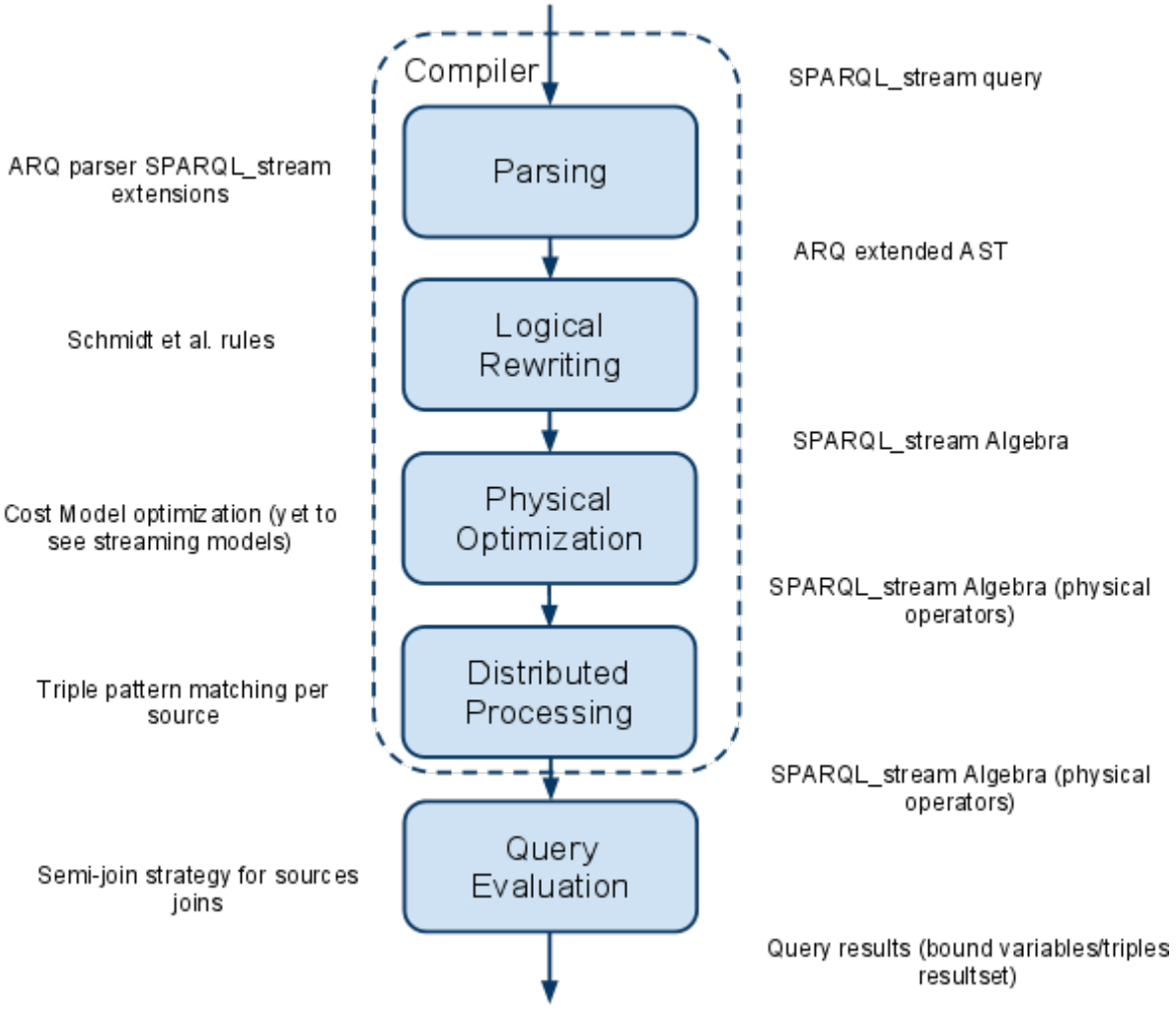
C-SPARQL proposes a form of aggregates that extends, rather than summarising, the data. We

have chose to adopt the semantics proposed in SPARQL1.1 which conform more to the style of SQL aggregates.

Query Optimization

Design of optimizer and relationship with C-SPARQL and DARQ

The distributed query processor consist of a Query Compiler and a Query Evaluator



Query Compiler

Proposal consists of following steps

Parsing

Parsing the SPARQL$_{Stream}$ query into an abstract syntax tree (AST). This AST is an extended version of the ARQ AST already provided.

Extensions are needed for:

- Streams, graphs in FROM clause
- Windows
- Window-to-stream operators
- Aggregates

Logical Rewriting

AST is transformed to a logical algebra expression.

- Provided by ARQ and must be extended.

At this point the rewriting rules of Schmidt et al. (2010) should be used to produce an equivalent logical plan.

Basic rewriting rules must be applied for a first step

- see what simple rules already enforced by ARQ framework
- progressively add features
- Filter Basic Graph Patterns (FBGP) first targeted
    - Clearly identify set of rules. Compare to Perez et al. (2009).

Physical Optimization

Logical plan transformed to physical one and physical operators are introduced at this point.

- include semi-join/bind-join implementation where necessary

The optimization of the physical plan is focused basically on the join optimization using the collected statistics.

Statistics

- Services description updated, using void, scovo
    - determine the exact information needed?
    - cardinality based unfeasible, streaming data requires other mechanisms
    - in the end most likely selectivity estimations, search for methods to calculate
    - Streams rates if available

Capturing the metadata

- still to propose a technique
- for first attempts assume to have it provided?
- must have dynamic means to keep it updated, in stream environment can be highly dynamic

Cost model

- Need to read in detail, proposals in stream world

Distributed Processing

- Triple pattern based capabilities per source. May be the way, as in DARQ (Quilitz and Leser, 2008). Metadata is provided by each source.
- Produce a set of sub-parts of the optimized plan, each to be delivered to a respective source. It must be sent as a serialized query statement.

Query Evaluation

- Query evaluator follows optimized physical plan.
- Queries dispatched to the sources will be executed remotely by the processor of each source and results returned to the DQP.
- At this point DQP will execute its part of the physical plan (typically join operations (semi-join useful here)

Comparison to DARQ

We adopt the approach developed in DARQ (Quilitz and Leser, 2008).
Improvements over DARQ:

- Including Streaming Data sources
- Use of standard SPARQL 1.1 and vocabularies for statistics and metadata
- Incorporate more elaborate cost models
    - Only simplistic models used in DARQ
- Incorporate optimization models for streams
- Incorporate logical optimization rewriting rules

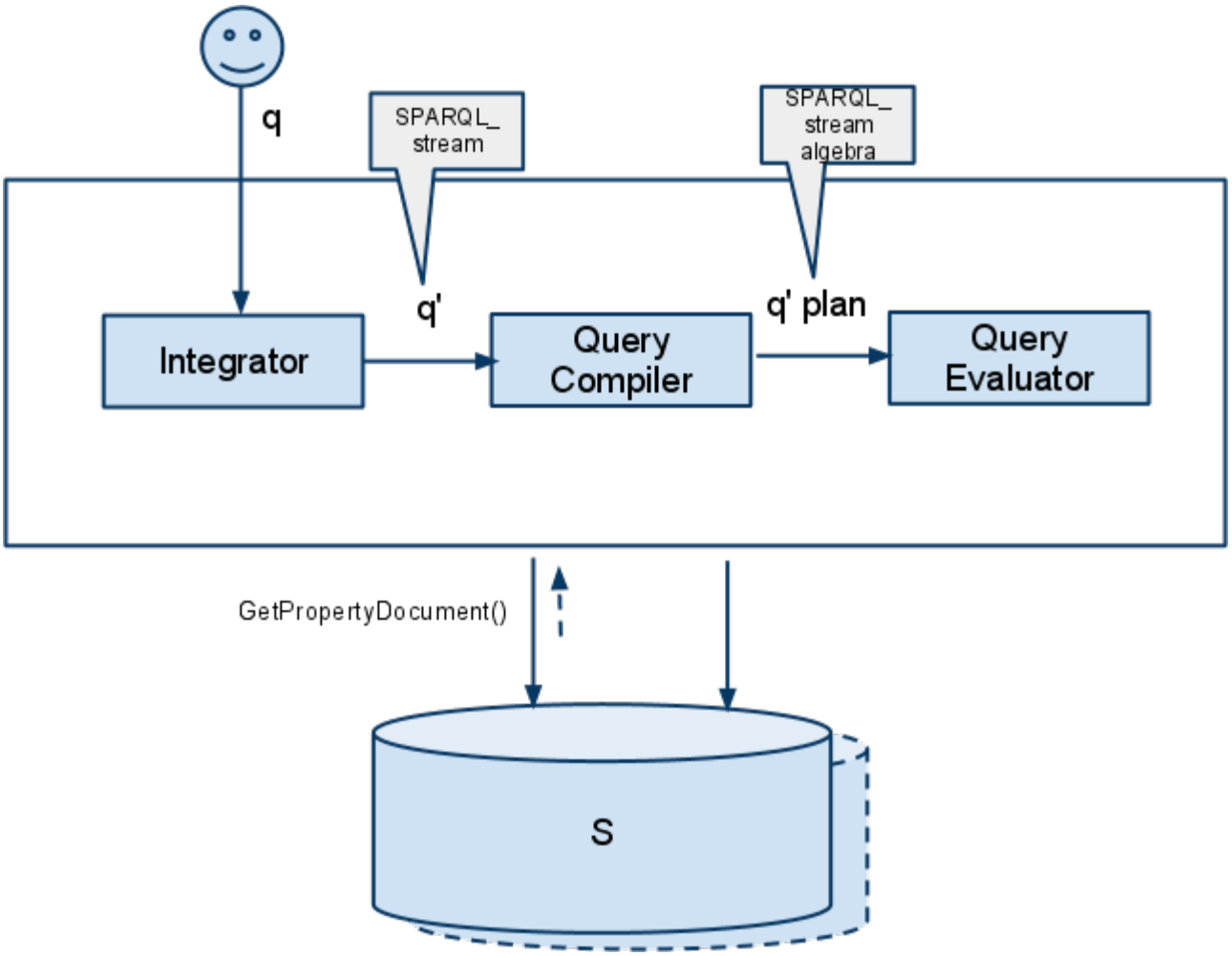
Comparing to C-SPARQL

The C-SPARQL execution environment is described in (Barbieri et al., 2010b).
Differences:

- Distributed evaluation environment
- Logical and Physical optimization, no only push of filters, projections and aggregates
- Consistent integration with stored data, relying on datasource, not loading into TripleStore.

Still a long way to develop....

Semantic Integrator



- Property documents are provided by sources
    - Schema of the data (e.g. relational schema, relational stream schema, ontology schema)
    - Schema provided in RDF format, extensions to Relational.OWL if needed.
    - Schemas annotated semantically to classes/properties in an ontological view
- Ontology-Ontology mappings can be used if the query is posed against a different ontological view than the specified by the source.
- Users pose queries to the ontological view, it is transformed to queries under the internal ontological views. Users are unaware that this is an integrated source, i.e. they do not know which sources are being used.
- Query Compiler eats the query as described in the previous section (parses SPARQL_stream, performs logical and physical optimization and uses the mappings metadata to determine which sources will be used to produce the data. the result is a complete execution plan including the sub-parts to be sent to the sources and the sub plan to integrate the incoming data.
- Query Execution uses the generated plan to send the sub-plans as queries to the correspondent sources, gather the results and perform the merging, join operations remaining to generate the final results sent back to the user.

Some discussion about mappings

Example

stream: envdata_boscome
column: Hs
how is the annotation made?: e.g. --> wave:hasValue
clearly not enough information. It could be wave2:significatWaveHeightValue, but this is not necessarily the case.

the annotation should then be more complete, like:
column Hs --> x? a wave:WaveHeightMeasurement; x? wave:hasValue Hs;
which is essentially a mapping definition (s2o like)

O-O mappings

Out of the scope for the moment, most likely be based on Sharffe et al. work (See here).
Part of Jean-Paul's research at UPM.

Workplan

Identify timescales and deliverables. Assign responsibilities.

1. Parser modifications SPARQL_Stream
2. Reuse & adaptation to ARQ logical algebra
3. Expose property document
4. Complete definition of demo use case
5. Interaction with SNEE-WS
6. Generation of queries for distributed sources
7. Execution, query processor extensions of DARQ for initial use case

06-09-2010: Internal integration deadline with WP5

1. Investigate cost model for streaming
2. Define statistics metadata needed, gathering approach
3. Apply basic cost model, minimal statistics metadata

17-10-2010: Review Y2

1. Apply logical rewriting rules
2. Push interface implementation
3. Refine physical optimization, updated model

28-02-2011: Integrator v2

References

- D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An execution environment for C-SPARQL queries. In Proceedings of 13th International Conference on Extending Database Technology (EDBT2010), pages 441–452, Lausanne, Switzerland, March 2010.
- D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-SPARQL: A continuous query language for RDF data streams. International Journal of Semantic Computing, 2010. To appear.
- S. Harris and A. Seaborne (eds). SPARQL 1.1 query language. Working draft, W3C, 2010. http://www.w3.org/TR/sparql11-query/
- J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. ACM Transactions on Database Systems (TODS), 34(3):1–45, August 2009.
- B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In Proceedings of 5th European Semantic Web Conference (ESWC 2008), volume 5021 of LNCS, pages 524–538, Tenerife, Spain, June 2008. Springer.
- M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In Proceedings of 13th International Conference on Database Theory – ICDT2010, Lausanne, Switzerland, March 2010.
- J.J. Sharplesa, R.H.D. McRaeb, R.O. Webera, and A.M. Gill. A simple index for assessing fire danger rating. Environmental Modelling & Software, 24(6):764–774, June 2009.