Rules and RDF Streams - A Position Paper

James Anderson[1], Tara Athan[2] and Adrian Paschke[2]

[1] Datagraph, GmbH.
james@dydra.com,
WWW home page: http://dydra.com/
[2] Corporate Semantic Web, Freie Universität Berlin, Germany
[taraathan|paschke]@inf.fu-berlin.de,
http://www.corporate-semantic-web.de/

Abstract. We propose a minor extension of the Graph Store Protocol and the SPARQL syntax that should be sufficient to enable the application of rules to some kinds of RDF Streams (as defined by the the RDF Stream Processing W3C Community Group) to generate new RDF streams. The IRI identifying an RDF stream is extended with a query string field-value pair defining a sequence of windows of the stream. The extended IRI is passed to a data concentrator, which recasts the stream as a dynamic RDF dataset. A SPARQL query may then be applied to this dynamic dataset whenever it is updated, i.e. when the content of the next window as been fully received. The SPARQL query uses the CONSTRUCT form to generate a new element of the resultant RDF stream. The approach is illustrated with a prototypical example from the healthcare domain.

1 Business Case

RDF Graph Stores are increasingly used in the private and public sector to manage knowledge. In practice, although these Graph Stores can be dynamic, most individual transactions involve either complete replacement or small changes. When transactions are made by many users simultaneously or through streaming from data sources, the rate of change can be quite fast, relative to the timescale of query processing. Effectively delivering an up-to-date view (i.e. results of a stored query) of a dynamic Graph Store in real time requires a different approach to query processing than that implemented in most RDF query engines currently available.

The RDF standards have not addressed dynamics to a significant extent, other than to informally propose a name: "We informally use the term RDF source to refer to a persistent yet mutable source or container of RDF graphs. An RDF source is a resource that may be said to have a state that can change over time. A snapshot of the state can be expressed as an RDF graph." [1]

The RDF Stream Processing Community Group[3] is taking on a specialized case of dynamic RDF called an RDF Stream, corresponding to the case of a

[3] https://www.w3.org/community/rsp/

sequence of RDF datasets called "timestamped RDF graphs", where a particular triple in the default graph has been designated as the timestamp triple. A sequence of overlapping windows (contiguous subsequences) can, in general, be considered as snapshots of a dynamic Graph Store[4] such that whenever the window shifts to a new position, a few stream elements expire from and a few new stream elements are added to the dynamic Graph Store.

1.1 Business Drivers

Autonomous, distributed sources will eventually yield data in quantities limited only by the aggregate bandwidth of wireless transmission media. It has been estimated, the market for devices and services in support of this processing will reach a value in excess of twelve digits by 2020 [2]. Based on the relative expenditures by established telecommunication firms - such as AT&T or Deutsche Telekom, where the software budget constitutes roughly one percent of their gross income [3, 4], one can expect the software costs to reach ten figures, of which database services constitute an undisclosed, but substantial portion.

The task to purposefully process this data promises to be formidable. A service which leverages the advantages of RDF data processing, to provide a mechanism for rule based integration, validation and evaluation of diverse streaming data sources, will offer significant advantage.

Particular drivers in this setting are the following:

- develop a mechanism to augment assertions during query processing of dynamic RDF Graph Stores, at basically the complexity of SPARQL construct
- create means to express actions that should happen when a specified integrity constraint is violated by a dynamic RDF Graph Store
- implement real-time processing of such features with good scaling properties in size and rate of change of the Graph Store
- enable linked data to fullfil compliance requirements [5]
- expand the query capabilities available to dynamic RDF Graph Store users, by extending standard query capabilities over streams.
- enhance the management capabilities of dynamic RDF Graph Store publishers, by automating both internal updates and their propagation.
- increase user and owner confidence in published RDF sources
- increase dynamic RDF Graph Store customer satisfaction through minimizing latency associated with automatic updates and integrity-constraint checking

2 Technological Challenges

RDFS and OWL are the traditional technologies for expressing *conceptual* models for RDF data, and can in theory be used to augment assertions during query processing, e.g. SPARQL under the RDFS- or OWL-entailment regime.

[4] Because blank nodes are shared between elements of an RDF stream, then the dynamic Graph Store must also share blank nodes between versions if this correspondence is to hold.

Due to the Open World Assumption, RDFS and OWL models cannot be used for validation of RDF according to a *data* model, except for the extreme case that results in inconsistency.

SPARQL queries express validation and integrity constraints well, but the queries for these tasks are not intuitive, and thus are difficult to properly construct and maintain. It is imperative that extensions to incorporate rules and process data streams avoid compounding these difficulties.

The challenge is to develop a language with the optimum satisfaction of the following characteristics:

- syntax that is both intuitive for the audience (close to RDF) and sufficiently expressive
- has well-defined execution semantics
- can be integrated into remote service control and data flows.
- permits real-time performance, which requires efficient target data-set generation (whether update diffs or inherent stream structure) and an activation network implementation. [6] [7] [8]
- supports named graphs and RDF Datasets as static or dynamic sources
- supports modularization
- permits descriptions of reactions (e.g. to integrity constraint violations)

3 RDF-Stream Options

Numerous alternatives (see Table 1) have been proposed to query dynamic RDF data. These include proposals for both streaming and versioned data. In order to provide a service which supports queries of adequate complexity, we expect we will need to provide the following combinations and variations:

- treat queries as autonomous views in order to promote reuse;
- separate query expressions from temporal attributes in order to permit combinations;
- permit temporal attributes to be computed as an aspect of query processing;
- remain compatible with standard SPARQL in order to permit sharing and simplify development
- remain compatible with standard RDF data models
- permit named graphs in order to fulfil application requirements and permit compatibility with standard encodings
- restrict the processing model to just essential components in order to limit the deployment and management complexity

This perspective eliminates those proposals which either reify statements in the data model or add a temporal term. It also argues against a processing model which involves query re-writing and delegation. It also eliminates language extensions which encode temporal attributes as static values in special clauses. It leads to the proposal for Dydra, to combine `REVISION` and `WINDOW` specifications for temporal properties to designate the target dataset, and to make the

temporal state available through the function `THEN` (e.g. returns the time when the processor had received all the elements in a stream window) and permit the window or revision expression to contain variables, but otherwise to conform to SPARQL 1.1.

4 Rule-based Options

There are several existing languages for applying rules to RDF.

SPARQL Inferencing Notation (SPIN), also called SPARQL Rules, is a W3C Member Submission[5] as well as a "defacto industry standard"[6] implemented by Top Braid. The Member Submission defines the SPIN syntax via a schema, but does not provide semantics. In particular, although the SPIN submission is claimed to be an alternative syntax for SPARQL, the transformation from SPIN to SPARQL has not been published. Further, although SPIN uses an RDF-based syntax, this alone is not sufficient to achieve rule semantics; a sufficiently expressive entailment regime must be defined (e.g. based on the as-yet unpublished transformation to SPARQL) that asserts the constructed source jointly with the queried sources.

Shape Expressions Language (ShEx) is specified in a W3C member submission[7]. It is intended to fill the same role for RDF graphs that the schema language Relax NG fills for XML, and its design is inspired by Relax NG, particular in its foundation on regular expressions. ShEx has both a compact and an RDF-based syntax, in the same way that Relax NG has a compact and an XML-based syntax. The execution semantics of ShEx is well-defined, but is of limited expressivity. In particular, ShEx does not support named graphs or RDF Datasets. ShEx precedents are strictly targeted REST [19]; it is not clear whether ShEx is applicable to streams. ShEx does not support modularization, but this extension could be easily incorporated following the approach of Relax NG, or, better, a monotonic restriction of that approach [20]. ShEx has no constructive capability, and so can only perform validation.

Shapes Constraint Language (SHACL) is intended as an successor and extension of ShEx developed by the RDF Shapes Working Group.[8]. SHACL does not support named graphs or RDF Datasets directly, but has an advanced syntax that allows the use of SPARQL. SHACL does not support modularization; it is not obvious how to make this extension. Like ShEx, SHACL has no constructive capability, and so can only perform validation.

Semantic Web Rule Language (SWRL) is a W3C Member Submission[9] with well-defined semantics having a logical expressivity that exceeds SPARQL.

[5] SPIN - Overview and Motivation `https://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/`

[6] `http://spinrdf.org/`

[7] Shape Expressions 1.0 Definition: `https://www.w3.org/Submission/shex-defn/`

[8] `https://www.w3.org/2014/data-shapes/wiki/Main_Page`

[9] `https://www.w3.org/Submission/SWRL/`

```
[[56]] GraphPatternNotTriples ::=
    OptionalGraphPattern | GroupOrUnionGraphPattern |
    MinusGraphPattern | GraphGraphPattern |
    RevisionGraphPattern | WindowsGraphPattern | ServiceGraphPattern
[[60a]] RevisionGraphPattern ::=
    'REVISION' ( Var | Revision | String ) GroupGraphPattern
[[60b]] WindowsGraphPattern ::=
    'WINDOWS' WindowRef GroupGraphPattern
[[60c]] WindowRef ::= ( 'R' Integer ? '/' ) ? WindowRelation
[[60d]] WindowRelation ::=
    Revision ( '/' ( Revision | XPathDuration )
                ( '/' ( XPathDuration | Integer ) )? )?
[[60e]] Revision ::= UUID | /HEAD(^[0-9]+)?/ | XPathDateTime | Integer

[[111b]] NullOperator ::= 'RAND' | 'NOW' | 'UUID' | 'STRUUID' | 'THEN'
```

Fig. 1. SPARQL grammar extension

Due to the coverage of e.g. relations and functions with arbitrary arity, the syntax is not close to RDF triples.

Rule Interchange Format [21] (RIF) is a W3C Recommendation. It includes a standard mapping from RDF triples to RIF frames[10], allowing rules to be applied to RDF triples. This approach requires recasting RDF into RIF, violating the requirement to remain close to RDF. Also it is not clear if RIF is applicable to RDF datasets and Graph Stores, as the mapping of RDF triples to RIF frames does not address this form.

Each of the existing languages described above has drawbacks that make it unsatisfactory relative to the evaluation criteria of the business case. Instead, we consider a minor extension to SPARQL and the Graph Store protocol. Figure 1 describes the grammar extensions to support elementary dataset revisions, dataset interval revisions, and repeated interval datasets (aka windows of streams).

Figures 3 and 4 illustrates a query which combines static base data with a temporal clause - in this case in the "validity time" domain, with a clause for which the dataset constitutes a stream.

Figure 2 depicts how the generated solution propagation graph is amenable to a static temporality analysis, based upon which the process control is tailored to the dataset form.

In Figure 3, we show a query supporting the "connected patient" scenario [22]; the patient is monitored with physiology (including heart rate) and activity sensors. The objective is to notify a designated responder with an SMS message when there is a combination of sensor readings suggesting an abnormally elevated heart rate that is not explained by vigorous physical activity. A sequence of windows of the heart rate and activity sensors are defined by strings of the form `'?WINDOWS=R/HEAD/PT05M/PT01M'`, which define a sequence of window functions that are applied to the stream. Following ISO-8601 [23], the text `R/` means a

[10] RIF-RDF Compatibility `https://www.w3.org/TR/rif-rdf-owl/#RDF_Compatibility`

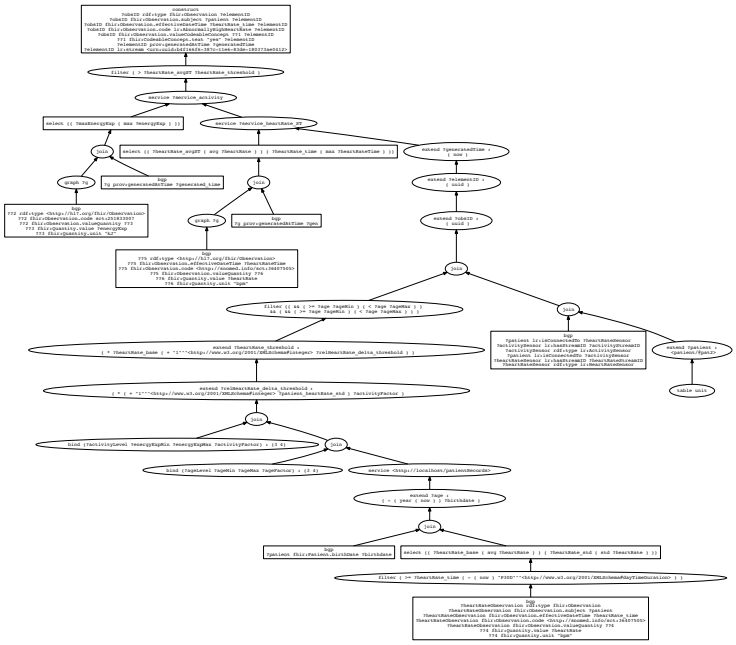

Fig. 2. Data Propagation Graph

recurring sequence of temporal intervals. The text `HEAD/` is an indexical telling the processor, a data concentrator component, to start the windowing operation as soon as possible. The text `PT05M/` specifies the duration of the temporal interval of the time-based window function, while `PT01M/` specifies the duration of the interval between start points of adjacent window functions. The query, written in the syntax of the proposed SPARQL extension, constructs the content of the SMS message.

The approach illustrated in Figures 3, 4 relies on certain properties of the RDF stream that do not hold in general, but hold within the RDF Distinct Time-Series profile[11] In particular, the timestamps of the stream elements are of datatype xsd:dateTimeStamp, and no two elements in the stream have the same timestamp, giving a total order to the stream elements. This permits a unique interpretation of `HEAD` as the most recent window of the stream.

For simplicity, in the concatenation above, the IRI of the RDF stream is assumed to not already have a query string. It is straight-forward to accommodate this possibility in the query.

[11] RDF Distinct Time-Series profile `http://streamreasoning.github.io/RSP-QL/Abstract%20Syntax%20and%20Semantics%20Document/#distinct-time-series-profile`

5 Implementation Approach

The proposed approach, to express temporal specifications in a SPARQL dataset designator, has its origin in the unique architecture of Dydra[12], Datagraph's SPARQL query service. This comprises two independent components, the RDF data store and the SPARQL query processor. These communicate through a limited interface, which allows the processor just to associate a specific dataset revision with an access transaction as the target and to use statement patterns to perform count, match and scan operations on this target.

The following changes will suffice to support streams based on this architecture:

- Extend the RDF store SPARQL Graph Store Protocol implementation to interpret intra-request state to indicate transaction boundaries. This can be HTTP chunking boundaries, RDF graph boundaries, or some other marker. Based on current import rates, the store should be able to accept 10^5 statements per second per stream with the effective rate for sensor readings dependent on the number of statements per reading.
- Extend the dataset designators to designate the dataset state which corresponds to some specific set of transactions. The specification will permit combinations of temporal and cardinality constraints.
- Extend the query processor control model from one which performs a single reduction of an algebra graph to one in which individual branches can be reiterated according to their temporal attributes, while unchanged intermediate results are cached and re-used.
- Extend the response encoding logic to permit alternative stream packaging options, among them, repeated http header/content encoding and http chunking with and without boundary significance.

6 Conclusion and Future Work

We have presented the grammar for a SPARQL extension and described a related Graph Store protocol extension that enables a sliding window operation on an RDF stream to be interpreted as a dynamic dataset. We illustrated the usage of this extension to describe a continuous query, with rules given in the SPARQL CONSTRUCT/WHERE form, on an RDF Stream in the distinct time-series profile with a sample query and a data propagation diagram.

Future work includes generalizing the SPARQL and Graph Store protocol extensions to cover a larger class of RDF streams, and demonstrating the feasibility and performance of the extension with a prototype implementation.

Acknowledgement

This work has been partially supported by the "InnoProfile-Transfer Corporate Smart Content" project funded by the German Federal Ministry of Education

[12] `http://dydra.com`

and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions. The authors wish to thank Davide Sottara of Arizona State University for assistance developing the connected patient query, and the reviewers for helpful suggestions.

References

1. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax
2. Norton, S.: Internet of things market to reach 1.7 trillion by 2020: IDC. http://blogs.wsj.com/cio/2015/06/02/internet-of-things-market-to-reach-1-7-trillion-by-2020-idc/ (June 2015)
3. AT&T, Inc.: mobilizing your world: AT&T inc. 2015 annual report. http://www.att.com/Investor/ATT_Annual/2015/downloads/att_ar2015_completeannualreport.pdf (2015)
4. Deutsche Telekom: Answers for the digital future: The 2015 financial year. http://www.telekom.com/static/-/298764/9/160225-q4-allinone-si (2015)
5. Solbrig, H.R., Prudhommeaux, E., Sharma, D.K., Chute, C.G., Jiang, G.: Feasibility of modeling hl7 fhir profiles using rdf shape expressions language. In: SWAT4LS 2015; Semantic Web Applications and Tools for Life Sciences. Volume 1296 of CEUR. CEUR-WS.org/Vol-1546/ 208–209
6. Rinne, M., Nuutila, E., Trm, S.: INSTANS: High-performance event processing with standard RDF and SPARQL. In: 11th International Semantic Web Conference ISWC 2012, CEUR-WS.org/Vol-914/ 101
7. Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing heterogeneous RDF events with standing sparql update rules. In: On the Move to Meaningful Internet Systems: OTM 2012. Springer (2012) 797–806
8. Rinne, M., Törmä, S., Nuutila, E.: Sparql-based applications for rdf-encoded sensor data. In: Semantic Sensor Networks. Volume 904. CEUR-WS.org/Vol-904/ (2012) 81–96
9. Gutierrez, C., Hurtado, C., Vaisman, A.: Temporal RDF. In: European Semantic Web Conference, Springer (2005) 93–107
10. Sande, M., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., de Walle, R.: R&Wbase: git for triples. In: Linked Data on the Web Workshop. (2013)
11. Graube, M., Hensel, S., Urbas, L.: R43ples: Revisions for triples. Proc. of LDQ (2014)
12. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. ACM SIGMOD Record **39**(1) (2010) 20–26
13. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: International Semantic Web Conference, Springer (2011) 370–388
14. Calbimonte, J.P., Corcho, O., Gray, A.J.: Enabling ontology-based access to streaming data sources. In: International Semantic Web Conference, Springer (2010) 96–111
15. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: a unified language for event processing and stream reasoning. In: Proceedings of the 20th international conference on World wide web, ACM (2011) 635–644

16. Rinne, M., Nuutila, E., Törmä, S.: Instans: High-performance event processing with standard RDF and sparql. In: Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914, CEUR-WS. org (2012) 101–104
17. Meimaris, M., Papastefanatos, G., Viglas, S., Stavrakas, Y., Pateritsas, C., Anagnostopoulos, I.: A query language for multi-version data web archives. arXiv preprint arXiv:1504.01891 (2015)
18. Anderson, J., Bendiken, A.: Transaction-time queries in dydra. In: Joint Proceedings of the 2nd Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW 2016) and the 3rd Workshop on Linked Data Quality (LDQ 2016) co-located with 13th European Semantic Web Conference (ESWC 2016): MEPDaW-LDQ 2016. Volume 1585. CEUR-WS.org/Vol-1585/ (2016) 11–19
19. Gayo, J.E.L., Prud'hommeaux, E., Solbrig, H.R., Rodrguez, J.M..: Validating and describing linked data portals using RDF shape expressions. In: LDQ 2014: Linked Data Quality, CEUR-WS.org/Vol-1215/
20. Athan, T., Boley, H.: The MYNG 1.01 suite for deliberation RuleML 1.01: Taming the language lattice. In: Rule Challenge @ RuleML 2014. Volume 1296 of CEUR. CEUR-WS.org/Vol-1296/ 14
21. Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A., Reynolds, D.: RIF core dialect (February 2013) W3C Recommendation. http://www.w3.org/TR/rif-core/.
22. Athan, T., Bell, R., Kendall, E., Paschke, A., Sottara, D.: Api4kp metamodel: A meta-api for heterogeneous knowledge platforms. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D., eds.: Rule Technologies: Foundations, Tools, and Applications: 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings, Cham, Springer International Publishing (2015) 144–160
23. International Organization for Standardization: Data elements and interchange formats – Information interchange – Representation of dates and times, ISO 8601:2000. (December 2000)

Table 1. SPARQL alternatives for revisions and streams. In the data models T is a temporal literal, R a revision identifier and G a graph name. Temporality indicates whether temporal properties are specified through variables bound and constrained in a BGP and/or through a dataset clause.

Feature / Proposal	Data Model	Temporality	Process Model	SPARQL 1.1 compatible	application graphs
temporal RDF [9]	$((S \times P \times O) \times T)$	BGP	algebra reduction	no	yes
`TriplePattern ::= (Subject Predicate Object ) : [T]`					
R&W Base [10]	$((S \times P \times O) \times R)$	BGP	rewrite, generate dataset, delegate	yes	"virtual", graph $\equiv R$
`Datatset ::= FROM VersionIri`					
R34ples [11]	$((S \times P \times O) \times R)$	Dataset	rewrite, generate dataset, algebra reduction	yes	no, graph $\equiv R$
`Datatset ::= FROM Graph REVISION VersionIdentifier`					
C-SPARQL [12]	$((S \times P \times O) \times T)$	BGP	algebra reduction	no	no, graph $\equiv T$

`Datatset ::= FROM Iri Window?`
`Window ::= ( RANGE Number TimeUnit ( ( STEP Number TimeUnit ) | TUMBLING ) )`
`| ( TRIPLES Number )`					

CQELS [13]	$((S \times P \times O) \times T) * +$ $((S \times P \times O) * \times T)$	Dataset	rewrite (SQL), delegate	no	yes

`GraphPatternNotTriples ::= ... | StreamGraphPattern`
`StreamGraphPattern ::= RANGE [ Window ] VarOrIRIReference TriplesTemplate`
`Window ::= ( RANGE Number TimeUnit ( SLIDE Number TimeUnit )? )`
`| ( TRIPLES Number ) | NOW | ALL`					

SPARQL Stream [14]	$((S \times P \times O) \times T)$	Dataset	rewrite (DSMS), delegate	no	yes

`Datatset ::= FROM STREAM Iri Window ?`
`Window ::= FROM NOW ( - Number TimeUnit )?`
`TO NOW ( - Number TimeUnit )?`
`( STEP Number TimeUnit ) ?`					

EP-SPARQL [15]	$((S \times P \times O) \times T \times T)$	BGP	rewrite (PROLOG), delegate	yes	yes

`GraphPatternNotTriples ::= ... | SeqGraphPattern | EqualsGraphPattern`
`| OptionalSeqGraphPattern | OptionalEqualsGraphPattern`
`NullOperator ::= ... | getDURATION | getENDTIME | getSTARTTIME`					

INSTANS [16]	$(S \times P \times O \times G)$	BGP	RETE	no	yes
`Verb ::= ... | :hasWindow`					
DIACHRON [17]	$((S \times P \times O) \times R)$	federated metadata	rewrite, delegate	no	no

`SourceClause ::= ... | ( FROM DATASET URI ( AT VERSION URI )? ) | ( FROM CHANGES URI ( BEFORE VERSION URI )? )`
`| ( AFTER VERSION URI ) | ( BETWEEN VERSIONS URI URI )`
`SourcePattern ::= ... | (DATASET URI ( AT VERSION URI )? ) | ( CHANGES URI ( BEFORE VERSION URI )? )`
`| ( AFTER VERSION URI ) | ( BETWEEN VERSIONS URI URI )`					

Dydra [18]	$(S \times P \times O \times G)$	BGP Dataset	algebra reduction	yes	yes
`RevisionGraphPattern ::= 'REVISION' ( Var | Revision | String ) GroupGraphPattern` `WindowsGraphPattern ::= 'WINDOWS' WindowRef GroupGraphPattern`					

```
PREFIX fhir: <http://hl7/org/fhir/>
PREFIX obs: <http://hl7/org/fhir/Observation.>
PREFIX sct: <http://snomed.info/id/>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX lr: <http://localhost/local-records#>

CONSTRUCT {
  GRAPH ?elementID {
    ?obsID  a <http://hl7/org/fhir/Observation>;
        obs:subject ?patient ;
        obs:effectiveDateTime ?heartRate_time;
        obs:code <http://localhost/local-records#AbnormallyHighHeartRate> ;
        obs:valueCodeableConcept [ fhir:CodeableConcept.text 'yes' ] .
    }
  ?elementID prov:generatedAtTime ?generatedTime;
    # this would likely be provided with the request as a protocol parameter
    lr:stream <urn:uuid:b4f166f6-387c-11e6-83de-180373ae0412> .
}
# determine heart rate exceeding some limit
# based on history, activity and institutional factors
WHERE {
  # this would likely be provided with the request as a protocol parameter
  BIND ( <patient/@pat2> AS ?patient )
  ?patient lr:isConnectedTo ?heartRateSensor.
  ?heartRateSensor lr:hasStreamID ?heartRateStreamID;
                   a lr:HeartRateSensor .
  ?patient lr:isConnectedTo ?activitySensor.
  ?activitySensor lr:hasStreamID ?activityStreamID;
                   a lr:ActivitySensor .

  BIND (IRI(CONCAT(?activityStreamID, '?WINDOWS=R/HEAD/PT05M/PT01M'))
        AS ?service_activity)
  SERVICE ?service_activity {
    SELECT (MAX(?energyExp) AS ?maxEnergyExp)
    WHERE {
      ?g   prov:generatedAtTime ?generated_time .
      GRAPH ?g {
        [] a <http://hl7.org/fhir/Observation> ;
           obs:code <http://snomed.info/id/251833007> ; # energy expenditure
           obs:valueQuantity [ fhir:Quantity.value ?energyExp;
                               fhir:Quantity.unit 'kJ'          ] .
  } } }

  # establish heart rate baseline through valid-time aggregation
  SERVICE <http://localhost/patientRecords> {
    { SELECT (AVG(?heartRate) AS ?heartRate_base)
             (STD(?heartRate) AS ?heartRate_std)
      WHERE {
        ?heartRateObservation a fhir:Observation ;
          obs:subject ?patient ;
          obs:effectiveDateTime ?heartRate_time ;
          obs:code <http://snomed.info/sct:36407505> ; # heart rate
          obs:valueQuantity [ fhir:Quantity.value ?heartRate;
                              fhir:Quantity.unit 'bpm'        ] .
        FILTER ( ?heartRate_time >= (now() - 'P30D'^^xsd:dayTimeDuration) )
    } }
    ?patient fhir:Patient.birthDate ?birthdate .
    BIND ( (YEAR(NOW()) - ?birthdate) AS ?age )
  }
```

Fig. 3. Combined base-data, temporal and streaming query. See Figure 4 for the continuation.

```
  VALUES ( ?ageLevel ?ageMin ?ageMax ?ageFactor) {
    ( lr:elderly 70 120 0.7 )
    ( lr:midaged 30 70 1.0 )
    ( lr:young 0 30 1.3 )
  }
  FILTER ( ?age >= ?ageMin && ?age < ?ageMax )
  VALUES (?activityLevel ?energyExpMin ?energyExpMax ?activityFactor) {
    ( lr:high 25 100 2.0)
    ( lr:medium 11 25 1.0)
    ( lr:low 0 10 0.7 )
  }
  FILTER ( ?maxEnergyExp >= ?energyExpMin * ?ageFactor &&
               ?maxEnergyExp < ?energyExpMax * ?ageFactor )
  BIND ( ( (1 + ?patient_heartRate_std) * ?activityFactor )
         AS ?relHeartRate_delta_threshold )
  BIND ( (?heartRate_base * (1 + ?relHeartRate_delta_threshold))
         AS ?heartRate_threshold)
  # determine current pulse averages from sensors
  BIND ( IRI(CONCAT(STR(?heartRateStreamID), '?WINDOWS=R/HEAD/PT05M/PT30S'))
         AS ?service_heartRate_ST )

  SERVICE ?service_heartRate_ST
  { SELECT (AVG(?heartRate) AS ?heartRate_avgST)
           # if the transaction timestamp, instead
           # (THEN() AS ?heartRate_time)
           (MAX(?heartRateTime) AS ?heartRate_time)
    WHERE {
      ?g prov:generatedAtTime ?gen .
      { GRAPH ?g {
        [] a <http://hl7.org/fhir/Observation> ;
           obs:effectiveDateTime ?heartRateTime ;
           obs:code <http://snomed.info/sct:36407505> ; # heart rate
           obs:valueQuantity [ fhir:Quantity.value ?heartRate;
                               fhir:Quantity.unit 'bpm'        ] .
  } } } }

  # test limit
  FILTER ( ?heartRate_avgST > ?heartRate_threshold )
  # iff it gets this far, it will generate a result observation
  BIND ( UUID() AS ?obsID)
  BIND ( UUID() AS ?elementID)
  BIND ( NOW() AS ?generatedTime )
}
```

Fig. 4. Combined base-data, temporal and streaming query, continued