TripleWave: Spreading RDF Streams on the Web

Andrea Mauri[1], Jean-Paul Calbimonte[2 3], Daniele Dell'Aglio[1], Marco Balduini[1], Marco Brambilla[1], Emanuele Della Valle[1], and Karl Aberer[2]

[1] DEIB, Politecnico di Milano, Italy {name.surname}@polimi.it
[2] EPFL, Switzerland {name.surname}@epfl.ch
[3] HES-SO Valais, Switzerland {name.surname}@hevs.ch

Abstract. Processing data streams is increasingly gaining momentum, given the need to process these flows of information in real-time and at Web scale. In this context, RDF Stream Processing (RSP) and Stream Reasoning (SR) have emerged as solutions to combine semantic technologies with stream and event processing techniques. Research in these areas has proposed an ecosystem of solutions to query, reason and perform real-time processing over heterogeneous and distributed data streams on the Web. However, so far one basic building block has been missing: a mechanism to disseminate and exchange RDF streams on the Web. In this work we close this gap, proposing TripleWave, a reusable and generic tool that enables the publication of RDF streams on the Web. The features of TripleWave were selected based on requirements of real use-cases, and support a diverse set of scenarios, independent of any specific RSP implementation. TripleWave can be fed with existing Web streams (e.g. Twitter and Wikipedia streams) or time-annotated RDF datasets (e.g. the Linked Sensor Data dataset). It can be invoked through both pull- and push-based mechanisms, thus enabling RSP engines to automatically register and receive data from TripleWave.

1 Introduction

Semantic streams represent flows of knowledge over time, in diverse domains including social networks, health monitoring, financial markets or environmental monitoring, to name only a few. The Semantic Web community has studied the problems associated with the processing of and reasoning over these complex streams of data, leading to the emergence of *RDF Stream Processing* and *Stream Reasoning* techniques.

The Web is a natural context for semantic streams, due to the quantity of dynamic data it contains, generated for example by social networks or the Web of Things [11]. RDF streams emerged as a model to realize semantic streams on the Web: they are (potentially infinite) sequences of time-annotated RDF items ordered chronologically.

While several definitions of RDF streams have been proposed in the past, thanks to the efforts of the W3C RSP Community Group[4] these are converging towards a general formalization based on time-annotated RDF graphs. However, the model is not the only aspect about RDF streams that needs agreement in this community. Standard protocols and mechanisms for RDF stream exchange are currently missing, therefore limiting the adoption and spread of RSP technologies on the Web.

Existing systems such as C-SPARQL [3], CQELS [9] and EP-SPARQL [1] do not tackle directly this problem, but delegate the task of managing the stream publication

[4] Cf. `https://www.w3.org/community/rsp/`

and ingestion to the developer. Other approaches have proposed to create RDF datasets fed from unstructured streams [8, 16], to lift streaming data as Linked Data [2, 10], or to provide virtual RDF Streams [6]. To improve scalability, systems like Ztreamy [7] are designed for efficient transmission of compressed data streams, although they do not address the heterogeneity of data sources, declarative transformation and consumption modes. Nevertheless, so far there is still a need for a generic and flexible solution for making RDF streams available on the Web. Such a solution needs to follow Semantic Web standards and best practices, and to allow different data source configurations and data access modes.

In this work, we propose TripleWave[5], an open-source framework for *creating RDF streams and publishing them over the Web.* Triplewave facilitates the dissemination and consumption of RDF streams, in a similar manner as is already common for static RDF datasets with RDF graphs and datasets. In order to do so, we first elicit a set of requirements (Section 2) identified from real scenarios and use cases reported in the literature and the W3C RSP group. Then, we extend the prototype in [13]to address the new requirements. These include a flexible configuration of different types of data sources, and different stream generation modes, including transformation (via mappings) of Web streams and replay of existing RDF datasets and RDF sub-streams. Moreover, TripleWave offers a hybrid consumption mechanism that allows both pull-based consumption of RDF streams [2], and push communication through WebSockets.

2 Requirements

To elicit the requirements for TripleWave, we have taken into account a set of scenarios based on real-world use cases[6]. In the following, we highlight the most important of these requirements, organized along four main axes: data sources, data models, data provisioning, and management of contextual data (schema and metadata).

Data Sources. The first set of requirements focuses on the data sources that TripleWave should support to ensure wide adoption and reuse of the tool.
[R1] *TripleWave may use streams available on the Web as input*. Examples of this kind of data may be found in Twitter, Wikipedia, etc. Twitter supplies data through its streaming API[7]; similarly, Wikipedia publishes a change stream through an IRC-based or Websocket API[8]. Online streaming data is not the only kind of data that TripleWave may support. In the context of testing and benchmarking, system developers & designers have an intrinsic need to feed the engines in a reproducible and repeatable way. That means, they aim at streaming previously generated data, one or multiple times, to assess the behavior of the system.
[R2] *TripleWave shall be able to process existing time-aware (RDF) datasets*, which could or could not be formatted as streams.
[R3] *TripleWave shall provide format conversion mechanisms towards RDF streams*, in case the input is not formatted as a stream. A typical usage scenario where these features

[5] TripleWave: `http://streamreasoning.github.io/TripleWave/`
[6] Cf. `https://www.w3.org/community/rsp/wiki/Use_cases`.
[7] Cf. `https://dev.twitter.com/streaming/overview`
[8] Cf. `https://www.mediawiki.org/wiki/API:Recent_changes_stream`

are needed is testing. In this context, sharing the test data may not be enough, as the time dimension plays a key role and streaming the same data in different ways may influence the behavior of the engines. An open, reusable tool to stream data is therefore needed to enable a fair and reproducible execution of the tests.

Data Models. Although recommendations on data models and serialization formats for RDF streams are still under specification[9], it is important to identify and reuse formats that adhere as much as possible to existing recommendations and standards.
[R4] *TripleWave should adopt a data format compatible with RDF*, since RDF streams are heavily based on the RDF building blocks. In this way, it would be possible to increase the potential data reuse and tool usage itself in a wider set of scenarios.

Data Provisioning. This category of requirements describes the different ways of consuming RDF streams by RSP client applications.
[R5] *TripleWave shall be capable of pro-actively supplying streaming data to processing engines*. Indeed, stream processing applications are usually designed to be fed with streaming data. For instance, the SLD framework [2] receives and analyzes real-time data from social networks or sensor networks, while Star-City [12] is fed with Dublin public transportation data to compute urban analyses.
[R6] *TripleWave shall offer the data accordingly to existing W3C recommendations*. In particular, RDF streams should be accessible not only for stream processing and reasoning engines, but also for other applications based on Semantic Web technologies (e.g. SPARQL and Linked Data).
Requirements [R4] and [R6] ensure the compatibility with tools and frameworks already developed and available to process RDF data.

Contextual Data Management. In the stream processing context, continuous execution models are often adopted.
[R7] *TripleWave shall be able to publish the schema and metadata about the stream independently from the actual transmission of the stream itself.* Indeed, in the case of continuous query evaluation, the steps of query registration, schema provisioning, and metadata provisioning can be performed separately from the streaming itself.

3 The TripleWave Approach

In this section we describe how TripleWave enables the publication and consumption of RDF streams on the Web, following the requirements listed in Section 2.

Figure 1 represents a high-level architectural view of our solution. As we saw previously, RDF stream consumers may have different requirements on how to ingest the incoming data. According to [R1] and [R2], in TripleWave we consider two main types of data sources: (i) Non-RDF live streams on the Web, and (ii) RDF datasets with time-annotations. While the former mainly requires a conversion of existing streams to RDF, the latter is focused on streaming RDF data, provided that it has timestamped data elements. When performing the transformation to RDF streams, TripleWave makes use of R2RML mappings in order to allow customizing the shape of the resulting stream. In the case of streaming time-annotated RDF datasets, TripleWave also re-arranges the data

[9] W3C RSP Design Principles draft `http://streamreasoning.github.io/RSP-QL/RSP_Requirements_Design_Document`

if necessary, so that it is structured as a sequence of timestamped RDF graphs, following the W3C RSP Group design principles.

As output, TripleWave produces a JSON stream in the JSON-LD format: each stream element is described by an RDF graph and the time annotation is modeled as an annotation over the graph[10]. Using this format compliant with existing standards, TripleWave enables processing RDF streams not only through specialized RSP engines, but also with existing frameworks and techniques for standard RDF processing.

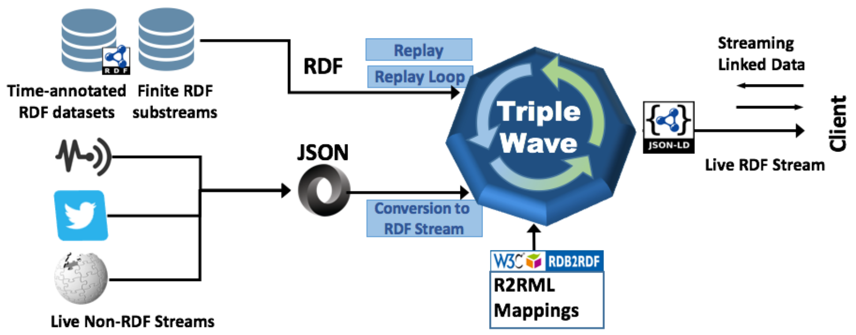

Fig. 1. The architecture of TripleWave: Generating RDF Streams from non-RDF data sources and time-annotated datasets. R2RML mappings allow customizing the transformation from non-RDF streams. The RDF stream output can be pushed or pulled toward the client as a JSON-LD dataset.

3.1 Running Modes

In order to address requirements [R1, R2, R3], TripleWave supports a flexible set of data sources, namely non-RDF streams from the Web, as well as timestamped RDF datasets. TripleWave also provides different use modes for RDF stream generation, detailed below.

Converting Web streams. Existing streams can be consumed through the TripleWave JSON and CSV connectors. Extensions of these can be easily incorporated in order to support additional formats. These feeds or streams (e.g. Twitter, earthquakes, live weather, Wikipedia updates, etc.) can be directly plugged to the TripleWave pipeline, which then uses R2RML mappings in order to construct RDF triples that will be output as part of an RDF stream. The mappings can be customized to produce RDF triples of arbitrary structure, and using any ontology.

```
{"type":"Feature",
 "properties":{
   "time":1388620046000,
   "url":"http://earthquake.usgs.gov/earthquakes/eventpage/ak10992887",
   "mag":1.1, "magType":"ml",
   "type":"earthquake",
   "title":"M 1.1 - 117km NW of Talkeetna, Alaska"},
 "geometry":{"type":"Point", "coordinates":[-151.6458,63.102,14.1]},
 "id":"ak10992887" }
```

Listing 1.1. Example of GeoJSON feed retrieved from USGS API

[10] The time annotation is stored in the default graph, as in `http://www.w3.org/TR/json-ld#named-graphs`, Example 49

As an example of input, consider the following GeoJSON feed item from the USGS earthquake API[11]. It contains information about the last reported earthquakes around the world, including the magnitude, location, type and other observed annotations.

Replaying RDF Datasets. RDF data is commonly available as archives and Linked Data endpoints, which may contain timestamp annotations and that can be replayed as a stream. Examples of these include sensor data archives, event datasets, transportation logs, update feeds, etc. These datasets typically contain a time-annotation within the data triples, and one or more other triples are connected to this timestamp. Replaying such datasets means converting an otherwise static dataset into a continuous flow of RDF data, which can then be used by an RDF Stream Processing engine. Common use cases include evaluation, testing, and benchmarking applications, as well as simulation systems. As an example consider the example air temperature observation extracted from the Linked Sensor Data [14] dataset. Each observation is associated to a particular instant, represented as an XSD `dateTime` literal. Using TripleWave we can replay the contents of this dataset as a stream, and the original timestamps can be tuned so that they can meet any test or benchmarking requirements.

```
ssw:Observation_AirTemperature_JEMC1_2003-04-02T06:00:00.0 sobs:samplingTime
     ssw:Instant_2003-04-02T06:00:00.0 .
ssw:Instant_2003-04-02T06:00:00.0 time:inXSDDateTime "2003-04-02 06:00:00.0"^^xsd:dateTime .
ssw:Observation_AirTemperature_JEMC1_2003-04-02T06:00:00.0 sobs:result ssw:
     MeasureData_AirTemperature_JEMC1_2003-04-02T06:00:00.0
ssw:MeasureData_AirTemperature_JEMC1_2003-04-02T06:00:00.0 sobs:uom weather:fahrenheit
ssw:MeasureData_AirTemperature_JEMC1_2003-04-02T06:00:00.0 sobs:floatValue 7.0
ssw:MeasureData_AirTemperature_JEMC1_2003-04-02T06:00:00.0 rdf:type sobs:MeasureData
```

Listing 1.2. Example of an observation contained in the Linked Sensor Data dataset

Replay Loop. In certain cases, the replay of RDF datasets as streams can be set up in a way that the data is re-fed to the system after it has been entirely consumed. This is common in testing and benchmarking scenarios where data needs to be endlessly available until a break point, or in simulation use-cases where an infinite data stream is required [15]. Similar to the previous scenario, the original RDF dataset is pre-processed in order to structure the stream as a sequence of annotated graphs, and then it is continuously streamed through TripleWave as a JSON-LD RDF stream. The main difference is that the timestamps are cyclically incremented, when the dataset is replayed, so that they provide the impression of an endless stream.

3.2 R2RML to generate RDF streams

Streams on the Web are available in a large variety of formats, so in order to adapt and transform them into RDF streams we use a generic transformation process that is specified as R2RML[12] mappings. Although these mappings were originally conceived for relational database inputs, we use light extensions that support other formats such as CSV or JSON (as in RML extensions[13]).

The example in Listing 1.3 specifies how earthquake stream data items can be mapped to a graph of an RDF stream[14]. This mapping defines first a triple that indicates that the

[11] Cf. `http://earthquake.usgs.gov/earthquakes/feed/v1.0`
[12] R2RML W3C Recommendation: `http://www.w3.org/TR/r2rml/`
[13] `http://rml.io`
[14] We use schema.org as the vocabulary in the example.

generated subject is of type `ex:Earthquake`. The `predicateObjectMap` clauses add two more triples, one specifying the URL of the earthquake (e.g. the reference USGS page) and its description.

```
:earthquakeMap a rr:TriplesMap; rr:logicalTable :quakestream;
 rr:subjectMap [rr:template "http://streamreasoning.org/TripleWave/{id}"; rr:class ex:Earthquake;];
 rr:predicateObjectMap [rr:predicate schema:url; rr:objectMap [ rr:column "url" ]];
 rr:predicateObjectMap [rr:predicate schema:description; rr:objectMap [ rr:column "title"]];
 rr:predicateObjectMap [rr:predicate schema:location; rr:objectMap [ rr:parentTriplesMap :locMap]].
```

Listing 1.3. Example of R2RML mapping

A snippet of the resulting RDF Stream graph, serialized in JSON-LD, is shown in in Listing 1.4. As can be observed, a stream element is contained in a timestamped graph, using the `generatedAtTime` property of the PROV ontology[15].

```
{"http://www.w3.org/ns/prov#generatedAtTime": "2015-06-30T16:44:59.587Z",
 "@id": "http://streamreasoning.org/TripleWave/ak10992887",
 "@graph": [
   { "@id": "http://streamreasoning.org/TripleWavee/ak10992887",
     "@type": "http://example.org/onto/earth#Earthquake",
     "url": "http://earthquake.usgs.gov/earthquakes/eventpage/ak10992887",
     "location": {"@id": "http://streamreasoning.org/TripleWave/ak10992887Location"},
     "description": "M 1.1 - 117km NW of Talkeetna, Alaska" },
   { "@id": "http://streamreasoning.org/TripleWave/ak10992887Location",
     "@type": "https://schema.org/Place",
     "longitude": "-151.6458",
     "latitude":  "63.102" } ],
 "@context": "https://schema.org/"  }
```

Listing 1.4. Portion of the timestamped element in the RDF stream.

3.3 Consuming TripleWave RDF Streams

TripleWave is implemented in Node.js and produces the output RDF stream using HTTP with chunked transfer encoding by default, or alternatively through WebSockets. Consumers can register to a TripleWave endpoint and receive the data following a push paradigm. In cases where consumers may want to pull the data, TripleWave allows publishing the data according to the Linked Data principles [5]. Given that the stream supplies data that changes very frequently, data is only temporarily available for consumption, assuming that recent stream elements are more relevant. We describe both cases below.

Publishing stream elements as Linked Data. TripleWave allows consuming RDF Streams following the Linked Data principles, extending the framework proposed in [4]. According to this scheme, for each RDF Stream TripleWave distinguishes between two kinds of Named Graphs: the Stream Graph (*sGraph*) and Instantaneous Graphs (*iGraphs*). Intuitively, an iGraph represents one stream element, while the sGraph contains the descriptions of the iGraphs, e.g. their timestamps.

As an example, the sGraph in Listing 1.5 describes the current content that can be retrieved from a TripleWave RDF stream. The ordered list of iGraphs is modeled as an `rdf:list` with the most recent iGraph as the first element, and with each iGraph having its relative timestamp annotation. By accessing the sGraph, consumers discover which are the stream elements (identified by iGraphs) available at the current time instants.

[15] Cf. `https://www.w3.org/TR/prov-o/`

Next, the consumer can access the iGraphs dereferencing the iGraph URL address. The annotations on the sGraph use a dedicated vocabulary[16].

```
{"@context": {
   "sld": "http://streamreasoning.org/ontologies/SLD4TripleWave#",
   "generatedAt": { "@id": "http://www.w3.org/ns/prov#generatedAtTime",
                   "@type": "http://www.w3.org/2001/XMLSchema#dateTime" }},
 "@type": "sld:sGraph",
 "sld:streamLocation": "ws://localhost:8101/TripleWave/replay",
 "sld:tBoxLocation": {"@id":"http://purl.oclc.org/NET/ssnx/ssn"},
 "sld:contains": {"@list": [
    { "generatedAt": "2016-04-21T13:01:18.663Z", "@id": "tr:1461243678663" },
    { "generatedAt": "2016-04-21T13:01:19.784Z", "@id": "tr:1461243679784" } ]},
 "sld:lastUpdated": "2016-04-21T13:02:06.575Z" }
```

Listing 1.5. The sGraph pointing to the iGraph described in Listing 1.4.

RDF Stream Push. An RSP engine can consume an RDF stream from TripleWave, extending the rsp-services framework[17] as follows (with C-SPARQL as a sample RSP): (1) the client identifies the stream by its IRI (which is the URL of the sGraph). (2) rsp-services registers the new stream in the C-SPARQL engine. (3) rsp-services looks at the sGraph URL, parses it and gets the information regarding the TBox and WebSocket. (4) The TBox is associated to the stream. (5) A WebSocket connection is established and the data flows into C-SPARQL. (6) The user registers a new query for the registered stream. (7) The TBox is loaded into the reasoner (if available) associated to the query. (8) The query is performed on the flowing data.

4 Conclusion

In this work we have described TripleWave, an open-source framework for publishing and sharing RDF streams on the Web. This work fills an important gap in RDF stream processing as it provides flexible mechanisms for plugging in diverse Web data sources, and for consuming streams in both push and pull mode. TripleWave covers a set of crucial requirements for the stream reasoning community and the semantic Web community at large, including: reusing available streams on the Web [R1], as well as time annotated RDF datasets [R2], which are transformed to follow a homogenized RDF stream structure [R3]. TripleWave adopts a stream format compatible with Semantic Web standards, including RDF for data modeling, and Linked Data principles for publishing [R4] [R6]. The proposed tool also provides pull and push data access to client applications and RSP engines [R5], as well as context information about the stream [R7].

The inherent flexibility of TripleWave makes it suitable for reuse in a wide range of streaming data applications, and it has the potential of enabling the integration of RSP query engines, stream reasoners, RDF stream filters, semantic complex event processors, benchmark platforms, and stored RDF data sources. This versatility, combined with a standards-driven design, and aligned with the requirements and design principles discussed in the W3C RSP Group, can help spreading the adoption of RDF for streaming data scenarios and applications.

Show Cases. We developed two show cases in order to illustrate the capabilities of TripleWave. In the first case we set up TripleWave for converting Web streams and

[16] Cf. `http://streamreasoning.org/ontologies/SLD4TripleWave#`
[17] Cf. `https://github.com/streamreasoning/rsp-services`

we configured it to transform the stream generated by the changes in Wikipedia. We developed the component to listen to the Wikipedia endpoint and the R2RML mapping.

In the second case we started another instance of TripleWave and we configured it to endlessly replay as a stream the Linked Sensor Data [14] dataset as a stream. Furthermore for this scenario we also set up a instance of the C-SPARQL engine to consume the data produced by TripleWave. Links to both the show cases are available on the project website[18].

Availability. TripleWave is available under the Apache 2.0 license[19], its code is accessible on Github[20], and accompanied by user and developer guides. It is maintained and supported by the Stream Reasoning initiative[21].

References

1. D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, pages 635–644. ACM, 2011.
2. M. Balduini, E. Della Valle, D. DellAglio, M. Tsytsarau, T. Palpanas, and C. Confalonieri. Social listening of city scale events using the streaming linked data framework. In *ISWC*, pages 1–16. 2013.
3. D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-sparql: a continuous query language for rdf data streams. *Intl. J. Semantic Computing*, 4(01):3–25, 2010.
4. D. F. Barbieri and E. Della Valle. A proposal for publishing data streams as linked data - A position paper. In *LDOW*, 2010.
5. T. Berners-Lee, C. Bizer, and T. Heath. Linked data-the story so far. *IJSWIS*, 5(3):1–22, 2009.
6. J.-P. Calbimonte, H. Jeung, O. Corcho, and K. Aberer. Enabling query technologies for the semantic sensor web. *Int. J. Semantic Web Inf. Syst.*, 8:43–63, 2012.
7. J. A. Fisteus, N. F. Garcia, L. S. Fernandez, and D. Fuentes-Lorenzo. Ztreamy: A middleware for publishing semantic streams on the web. *J. Web Semantics*, 25:16–23, 2014.
8. D. Gerber, S. Hellmann, L. Bühmann, T. Soru, R. Usbeck, and A.-C. N. Ngomo. Real-time rdf extraction from unstructured data streams. In *ISWC*, pages 135–150. 2013.
9. D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC*, pages 370–388. 2011.
10. D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth. A middleware framework for scalable management of linked streams. *J. Web Semantics*, 16:42–51, 2012.
11. D. Le-Phuoc, H. N. M. Quoc, H. N. Quoc, T. T. Nhat, and M. Hauswirth. The graph of things: A step towards the live knowledge graph of connected things. *J. Web Semantics*, 2016.
12. F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio, and P. Tommasi. Star-city: semantic traffic analytics and reasoning for city. In *ACM IUI*, pages 179–188, 2014.
13. A. Mauri, J.-P. Calbimonte, D. Dell'Aglio, M. Balduini, E. Della Valle, and K. Aberer. Where are the rdf streams?: Deploying rdf streams on the web of data with triplewave. In *Poster Proc. of ISWC*, 2015.
14. H. Patni, C. Henson, and A. Sheth. Linked sensor data. In *IEEE CTS*, pages 362–370, 2010.
15. T. Scharrenbach, J. Urbani, A. Margara, E. Della Valle, and A. Bernstein. Seven commandments for benchmarking semantic flow processing systems. In *ESWC*. 2013.
16. T.-D. Trinh, P. Wetz, B.-L. Do, A. Anjomshoaa, E. Kiesling, and A. M. Tjoa. A web-based platform for dynamic integration of heterogeneous data. In *IIWAS*, pages 253–261, 2014.

[18] Cf. `http://streamreasoning.github.io/TripleWave/`
[19] Cf. `https://www.apache.org/licenses/LICENSE-2.0.html`
[20] Cf. `https://github.com/streamreasoning/TripleWave`
[21] Cf. `http://streamreasoning.org`