Streaming Linked Data: A Survey on Life Cycle Compliance

Pieter Bonte[a], Riccardo Tommasini[b]

[a]*Ghent University - imec, Belgium*
[b]*INSA Lyon, LIRIS, France*

ARTICLE INFO

Keywords:
Linked Data
RDF Stream Processing
life-cycle

ABSTRACT

Data streams are becoming omnipresent on the Web. The Stream Reasoning (SR) paradigm, which combines Stream Processing with Semantic Web techniques, has been successful in processing these data streams. The progress in SR research has led to several applications in domains such as the Internet of Things, social media analysis, Smart Cities, and many others. Each of these applications produces and consumes data streams, however, there are no fixed guidelines on how to manage data streams on the Web, as there are for their static counterparts. More specifically, there is no fixed life cycle for Streaming Linked Data (SLD) yet. Tommasini et al. [66] introduced an initial proposal for a SLD life cycle, however, it has not been verified if the proposed life cycle captures existing applications and no guidelines were given for each step.

In this paper, we survey existing SR applications and identify if the life cycle proposed by Tommasini et al. fully captures the surveyed applications. Based on our analysis, we found that some of the steps needed reordering or being split up. This paper proposes an update of the life cycle and surveys the existing literature for each life cycle step while proposing a number of guidelines and best practices. Compared to the initial proposal by Tommasini et al., we drill down into the details of the processing step which was previously neglected. The updated life cycle and guidelines serves as a blueprint for future SR applications. A life cycle for SLD that allows to efficiently manage data streams on the web, brings us a step closer to the realization of the SR vision.

ORCID(s):

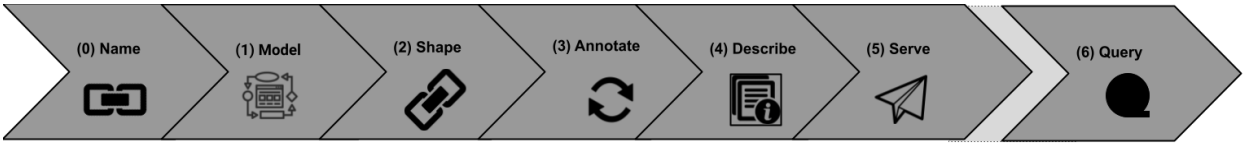


Figure 1: Streaming Linked Data life cycle.

1. Introduction

The Semantic Web community has witnessed a growing interest in streaming data over the last decade. Under the Stream Reasoning (SR) umbrella, Semantic Web technologies were combined with Stream Processing ones to answer the research questions *is it possible to make sense, in real-time, of heterogeneous, vast, incomplete, and noisy data streams coming from complex domains?* This research question spans a number of application domains, including social media analytics, the Internet of Things for smart cities [52, 31], and cluster management [56].

The research outcomes of SR include but are not limited to, findings on Continuous Querying over RDF Data, Incremental Reasoning, and Complex Event Recognition [21]. In particular, the research around RDF Stream Processing (RSP) has been incredibly active. Indeed, the related literature shows evidence of query languages and systems architectures but also working prototypes, benchmarks, and applications [43]. RSP proposes several extensions[1] of the semantic Web stack depicted in Figure 2, i.e., an extension of RDF to represent infinite streams of data, i.e. *RDF Streams*, continuous extensions of SPARQL, and engines capable of ingesting RDF streams and evaluating queries under continuous semantics defined in *RSP-QL* [22].

Recent achievements, e.g., RSP4J [64] and the Stream Reasoning Playground [55] attempt to push the community boundaries by lowering the cost of approaching the field. Moreover, the growing availability of data streams re-opened the debate around publishing dynamic data on the Web. In particular, there is a growing need for guidelines on how to produce and consume data streams in a sustainable manner.

Similar guidelines exist for non-streaming data, e.g., the *Linked Data Principles*, which support the original vision of Sir Tim Barners-Lee, and the *FAIR initiative*, which provides principles to make data Findable, Accessible, Interoperable and Reusable. Together they inspired the first attempt to define a Streaming Linked Data (SLD) life cycle [66]. Such a proposal identifies the life cycle steps starting from three situations a practitioner may find when dealing with SLD. Three possible starting points are identified, i.e., Web Data published in batches; Linked Data published in batches; and Web Data published as streams. Although realistic, the work presented in [66] elicits the life cycle, neglecting an application perspective. In practice, a lot of attention is given to how to provision data, neglecting the important role of querying.

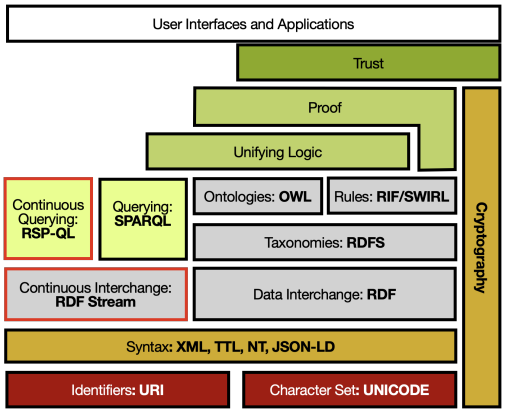


Figure 2: RSP extension proposal for the Semantic Web Stack.

Despite RSP leading to the vision of SLD, best practices for sharing highly dynamic datasets on the Web are still missing. The existing proposal [66] does not fully capture the complexity of stream reasoning applications. Therefore, this paper presents a novel life cycle for SLD, which extends the one in [66] with best practices elicited from state-of-the-art SR/SLD applications.

The new life cycle is depicted in Figure 1. It consists of seven steps and aims at making data streams findable, accessible, reusable, and interoperable [70]. In particular, the life cycle details how to publish SLD, indicating how to (i) *identify* and name data streams as Web resources, (ii) *model* data and metadata, (iii) *shape* the data, (iv) *annotate* the data into RDF, (v) *describe* the data to improve stream discovery, and (vi) *serve* the streaming data. Differently from [66], the *description* step (4) is postponed after the conversion that, in turn, is split into *shaping* and *annotation*. Like in [66], the paper highlights the available resources to be used within each step ultimately helping practitioners in the maintenance. For example, the paper discusses ontologies [67, 27], systems [44, 64], and describes how to use them

[1] Non-Standard

in practice with reference to the seven steps. Compared to [66], this paper goes into much more detail when discussing each step, proving the reader with ample background to make the correct decisions when publishing SLD. Finally, the life cycle terminates in a *query*ing step, which is discussed in more depth than in [66] and includes the sub-task we identified while studying the surveyed projects. In summary, the paper contributions are:

- we survey existing stream reasoning/streaming linked data projects and we highlight their characteristics.
- we criticize the existing linked data lifecycle proposal [66]. In particular, we show that it does not fully capture how existing SR applications were designed. Therefore, we justify two amendments, i.e., we postpone the description step and we split the conversion step into two, i.e., shaping and annotation.
- We drill down into the details of the querying step that was previously neglected. In particular, we distinguish the essential operational components of application pipelines and we present them using the RSP-QL reference model.
- we introduce a running example of the novel life cycle using a well-known stream reasoning task, i.e., the DEBS Grand Challenge 2015. We use such an application to walk through the new life cycle as was done in [63],
- We provide a more comprehensive description of the resources required at every step than in [63]. In particular, we present the alternatives and discuss the pros and cons.
- Finally, we discuss the level of maturity of the state of the art for each step, we elicit best practices, and we draw potential future research directions for the SR/SLD community.

We highlight that this paper focuses on surveying SR/SLD projects and not on Semantic Web of Things (SWoT) projects in general. The difference resides in the fact that the former has much more dedicated focus on how to handle streams on the Web, while the latter has its focus on the Thing, i.e. how to discover the Thing, how to represent the Thing, how to communicate with the Thing and how to make various Things interoperable. Even though the Things might produce streams, the discovery, description and general focus is on the Things and not on the streams as in SLD. There is some overlap between SLD and SWoT, i.e. some SWoT projects fall within the SLD paradigm, however, SLD also has applications outside of the SWoT, such as social media analysis.

Outline. Section 2 describes the survey methodology while Section 3 introduces the selected Stream Reasoning Applications. Section 4 discusses the limitation of the existing proposal wrt. the selected SR applications. Section 5 introduces a running example used to explain the different steps of the lifecycle. Section 6 presents the updated lifecycle in details, and Section 7 positions the survey in the state of the art. Section 8 concludes the paper by highlighting best practices and drawing guidelines for future research.

2. Survey Methodology

In particular, we aim at answering the following research questions: *Are existing stream reasoning applications compliant with the proposed SLD lifecycle?* Intuitively, in case of a negative response, we are interested to understand the reason for such a mismatch.

Our analysis of the state-of-the-art follows the guidelines for systematic mapping studies proposed in [12], which were successfully applied to other surveys on the semantic web. In particular, we follow the same research protocol applied in [12]. Reyero-Lobo et al. first collected relevant studies following a keyword-based search then applied different filters, i.e., a source-based filter that discards non-relevant research areas; a metadata-based filter that inspects

Figure 3: Selecting relevant work on Streaming Linked Data.

Projet	Year	Deployment	Domain	Task
SpitFire	2011		Smart Building	Enabling services for Semantic Web of Things
Bottari	2012	Seul, Korea	Social Media	Augmented reality application for personalized and localized restaurant recommendations,
SLD	2013	London, UK Milan, Italy	Event Management	Collect analyse of data streams and visualise the results in dashboards .
StarCity	2014	Dublin, Ireland	Smart City	Spatio-temporal diagnosis of traffic conditions; Spatio-Temporal exploration of traffic contexts; Traffic Status Prediction.
OpenIoT	2015		Internet of Things	Integrate, communicate and enable interoperability between IoT sensors
CityPulse	2016	Aarhus, Denmark	Smart City	Smart city service creation
AgriIoT	2016		Smart Farming	Fertility management of dairy cows; soil fertility for crop cultivation
Optique	2017	Munich, Germany Stanger, Norway	Manufacturing, Oil Extraction	Monitoring of qualist of drills and turbine
OpenSense2	2017	Lausanne, Switzerland	Smart City	Estimating air quality
StreamingMASSIF	2018	Ghent, Belgium	Smart City	Detection of traffic events
CitySensing	2019	Milan, Italy	Smart City	Visual story telling of activity in the city

Table 1
Summary of the Selected Projects.

the title, abstract, venues, and years; a content-based filter which inspects the paper in details. An enrichment step (aka snowballing) concludes the selection, where additional papers are added inspecting citations and related works of those resulting from the filtering. Figure 3 visualizes these different steps and indicates the number of identified works in each step.

For our search, we used DBLP, ACM library, and Google Scholar. The latter was also used for enrichment. The keywords we selected are *stream reasoning*, *streaming linked data*, *dynamic linked data*, *rdf stream processing*, *Linked Streams* and *Linked Stream Data*. Moreover, we refine the selected papers by searching for keywords that indicate the phase of the proposed life cycle, i.e., *naming, modeling, converting, serving, querying*.

According to [12], the paper collection process should follow *Inclusion Criteria*, which simultaneously defines the scope and allow one to deterministically decide if a paper needs to be considered. The criteria applied during the metadata-based filtering are:

IC1 Papers written in English

IC2 Papers published between 2010 and 2022

IC3 Papers subject to peer review.

Moreover, for content-based filtering, we focus on a paper that describes a stream reasoning system within the context of a use case. In particular, we exclude papers that focus only on the definition of a query language, an execution engine, or a benchmark to prioritize work that approaches stream reasoning as a full-stack problem.

3. Summary of Selected Projects

To verify if the steps of the SLD life cycle in [66] map to real SR/SLD applications, we investigated various successful SLD frameworks and applications from the literature. For these frameworks and applications, we investigate if they are compliant to the SLD life cycle in[66] or if the life cycle needs updating to better capture real-life SR/SLD applications.

BOTTARI [4] is a streaming analytic application designed to make sense of social media using inductive and inductive stream reasoning methods. The platform performs analyses of the activities of monitored influencers around the points of interest (POIs) of a given area. The analysis is window-based, spanning from a few seconds to months. The social media streams are gathered from the Web (in particular from Twitter) and converted into an RDF stream using the proprietary crawling and sentiment mining infrastructure of Saltlux. BOTTARI, which employs augmented reality

applications for personalized and localized restaurant recommendations, was experimentally deployed in the Insadong district of Seoul. In BOTTARI, data is *modelled* using the SIOC and WGS-84 ontology. In order to convert the tweets to RDF, a hard-coded solution for *annotation* is used. Once annotated, the RDF data is *served* using websockets and *queried* using the C-SPARQL [8] RSP engine.

The **CityPulse** project [52] handles a typical Smart City use case. The framework allows the development of applications that can provide a continuous and dynamic view of a city, making sense of social and sensor streams. To this extent, CityPulse employs semantic discovery, data analytics, and large-scale reasoning in real-time. CityPulse is built in a service-oriented manner combining RDF Stream Processing and Complex Event Processing. The framework was demonstrated using live data from the city of Aarhus, Denmark. In CityPulse, the sensor data is *annotated* using a custom hard-coded solution, resulting in a stream producing graph *shaped* time-annotated events. The events themselves have been *identified* using random identifiers, allowing to differentiate between events. The SSN, PROV-O and OWL-S ontology are used to *model* the event data, while the SAO ontology is used the *describe* the streams themselves. The CQELS [38] RSP engine is used to *query* the RDF streams.

SLD (which stands for Streaming Linked Data) [7] is a framework to collect, annotate, and analyse data streams. To this aim, SLD uses semantic technologies like RDF and SPARQL as well as techniques for sentiment mining. The framework follows the publication method proposed in [9]. SLD was successfully used to monitor the London Olympic Games 2012 and the Milano Design Week 2013 and 2016. In SDL, the idea of instantaneous Graphs (iGraphs) is used to *identify* a set of triples that have the same timestamp. The data itself is *modeled* using the SIOC ontology[2] and a custom hard-coded mapper is used to *annotate* the data to RDF. The C-SPARQL engine is used to *query* the data.

STAR-CITY [41], which stands for Semantic Traffic Analytics and Reasoning for CITY, is a system for streaming data integration and analysis focusing on traffic data management. STAR-CITY is capable of interpreting the semantics of contextual information and then deriving innovative and easy-to-explore insights. In practice, it computes the spatio-temporal similarities of traffic congestion and calculates accurate traffic forecasting using recent theoretical research work in contextual predictive reasoning. STAR-CITY was developed in collaboration with IBM Dublin, Ireland, where it was also applied. STAR-CITY also uses a custom hard coded mapper to *annotate* the data to RDF and uses the Time[3] and Geo Ontology [4] to *model* the events. Rest APIs are used to provide and *serve* the data that is being *queried* by the Jena SPARQL engine.

The **SpitFire** project [49] enables the creation of SWoT services by providing specialized vocabularies, i.e. the SpitFire Ontology, abstracting sensors, semi-automatic generation of annotations, and efficient search of sensors. It uses custom code to *annotate* the sensor data. The streams themselves are not *described*, but rather the sensor (or things) that produce the sensor streams. Once a certain sensor has been identified as being of interest, the CQELS engine is used to *query* the data.

The **OpenIoT** project [59] is an open-source IoT platform aiming at semantic interoperability of IoT services. It is designed to enable and facilitate SWoT projects and has showcased its capabilities through agriculture, smart city, smart building, or IoT-enabled communication [1] applications. OpenIoT uses the X-GSN [16] sensor middleware to collect data streams from virtual sensors or physical devices. X-GSN has custom hard-coded mapping functionality to *annotate* the sensor data to the SSN ontology. As OpenIoT is a SWoT project, the streams themselves are not described, but rather the sensor (or things) that produce the sensor streams. OpenIoT uses the Linked Stream Middleware [40] for this purpose and allows *querying* using the CQELS RSP engine.

Agri-IoT [31] is a highly customizable online platform for IoT-based data analytic in the context of smart farming. It is capable of large-scale data querying, and automatic reasoning based on data streams of data coming from a variety of sources, e.g., sensory systems, surveillance cameras, hyperspectral images from drones, weather forecasting services, and social media. Agri-IoT aims at helping farmers in more informed decision-making in real-time and fast reaction to changes and unpredictable events. In Agri-IoT is *annotated* using a custom hard coded mapper and the events are *modeled* using the Agri-IoT ontology, which extends the SSN [5] and OWL-S ontology[6]. The SAO ontology[7] is used to *describe* the streams themselves, such that they can be *queried* by RSP engines such as CQELS and C-SPARQL.

[2] https://www.w3.org/Submission/sioc-spec/
[3] https://www.w3.org/TR/owl-time/
[4] https://www.w3.org/2005/Incubator/geo/XGR-geo-ont-20071023/
[5] https://www.w3.org/TR/vocab-ssn/
[6] https://www.w3.org/Submission/OWL-S/
[7] http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao

The **Optique** [33, 35] project handles several Big Data scenarios in the context of energy production. The case of Siemens Energy (Munich, Germany) aims at monitoring a number of service centers for power plants, the main task was monitoring remotely and in real-time thousands of appliances like gas and steam turbines, generators, and compressors installed in plants. The case of Statoil (Stavanger, Norway) aims at improving the data gathering and analysis routines of Statoil geologists, who need IT specialists in order to make sense of multiple complex and large data sources. The Optique platform employs semantic technologies for enabling ontology-based data access for the aforementioned scenarios. Logical reasoning is used in form of query rewriting to efficiently analyze data from heterogeneous data sources. In the Optique project, R2RML [19] is used to *annotate* the data which much more flexible instead of a hard-coded solution. Incorporating the *annotation* allows performing rewriting from the RDF Streams to the underlying data to improve efficiency. The data is *modeled* using a turbine ontology and *queryed* using the STARQL engine.

The **OpenSense2** project aims at integrating air quality measurements captured by various mobile and crowd-sensing data sources and sensors, intending to understand the impact of urban air pollution exposure on the health of the citizens. OpenSense2 also uses X-GSN and allows to use its custom mapping *annotation* functionality or use augmented CSV as a common format for describing observation as RDF can be too verbose. This augmented CSV follows descriptions that follow the specifications of the CSV on the Web Working Group[8]. The sensor data is modeled using the SSN ontology. OpenSense2 allows to use Complex Event Processing (CEP) or RSP engines such as CSPARQL and CQELS to *query* the data.

StreamingMassif [10] is an extension of the MASSIF platform for cascading stream reasoning. It combines several layers of processing that include RSP, Description Logic reasoning, complex event recognition, and aggregations. Streaming Massif was studied to overcome MASSIF limitations in terms of throughput and latency. Its layered structure eases the deployment of expressive components for advanced analytics. Streaming Massif uses a simplified version of RML [24] to *annotate* the data while identifying the time annotated graphs using the idea of iGraphs. It allows the use of multiple ontologies to *model* the data and WebSockets are used to *serve* the data to the engine. By exploiting the *shape* of the data, it is able to push some of the processing closer to the source of the stream.

CitySensing combines various social media streams and anonymous call data records during large city-scale events, such as the Milan Design Week. It is mainly used for visual storytelling, giving an overview of activity hotspots in a visual interface. CitySensing semantically annotates the stream through the FraPPE ontology, which takes its inspiration from the digital image domain. The ontology allows to split up the view and events in the city in pixels and frames to provide a spatial unit of aggregated information. Doing this allows zooming in and out on the events happening throughout the city. CitySensing uses an ontology-based access approach while *annotating* the data with the FraPPE ontology and integrating it with more static resources. Data are ingested from streaming data sources, i.e., Instagram, Twitter, and TelecomItalia's Call Data Records, and several static sources for places and events. Input data are converted using triple templates into the activity stream 2.0 format. The *description* of streaming data follows sGraph approach, but it is not fully detailed in the work, nor is the *serving* protocol. Finally, a system called Natron semantically *augments* data using a custom Named Entity Recognition and evaluates a continuous query every 15min. The result of this continuous query is a stream modeled in FraPPE.

4. Renewing the SLD life cycle

We can now look at the presented projects, in terms of the original life cycle steps, i.e., identify, model, describe, convert, serve, and query. Table 2 classifies the projects above, according to the six steps of the original SLD life cycle. Two problems can be identified by comparing how existing SLD applications were designed and how to how to original life cycle models its various steps. A first problem arises when looking at the position of the *Convert* steps in the original life cycle. Conversion from potential raw data to RDF happens after the description step, making it impossible the describe the conversion step in the stream meta-data in the *Describe* step. Incorporating this information allows to exploit how the data is shaped to improve querying, e.g. in Streaming MASSIF and OpenIoT, or to enable rewriting techniques, e.g. in Optique and CitySensing. The former brings us to the second problem, the *Convert* step does not investigate the *shape* of the items in the stream. *Shaping* events differently can have an impact on querying performance. Based on these insights, and the investigation of how the above projects provisioned their RDF Streams, we propose an improved SLD life cycle by 1) reordering the steps and placing the *Describe* step before the *Serve* step instead of directly after the *Model* step and 2) splitting up the *Convert* step in both a *Shape* and *Annotate* step. The new life cycle is depicted in Figure 1 and contains the following steps:

[8] https://www.w3.org/TR/csv2rdf/

Projet	Domain	Identify	Model	Describe	Convert	Serve	Query
SpiteFire	Smart City		custom	DSO	REST		CQELS
Bottari	Social Media	source (tw)	SIOC,WGS-84		custom	websocket	C-SPARQL
CityPulse	Smart City	random	SSN,PROV-O, OWL-S	SAO	custom	REST?	CQELS
SLD	Event M	ts and igraphs	SIOC	sGraph	custom		CSPARQL
StarCity	Smart City		Time and Geo Ontology		custom	REST	SPARQL
OpenIoT	Crowd M		SSN		custom		CQELS
Agri-IoT	Smart Farming		DSO	SAO			CSPARQL, CQELS
Optique	Energy M	source (db)	DSO		R2RML		STARQL
OpenSense2	Smart City		SSN		custom, CSV2RDF		CEP, CSPARQL, CQELS
Streaming MASSIF	Healthcare	igraphs	multiple		RML	websocket	YASPER, CSPRITE, OBEP
CitySensing	Smart City	source(ig,tw)	Activity Streams 2.0 FrApPE	sGraph	Triple Template		Natron

Table 2
Classification of SLD projects according to the original life cycle. Legend: [M]management; [T]time [S]tamp; [D]domain [S]specific [O]ntology

- *Identify*, which focuses on the assignment of an IRI to identify data streams as Web resources.
- *Model*, which necessitates knowledge representation abilities to represent both the data stream metadata and the data itself, while accounting for their ephemeral nature.
- *Shape*, which discusses the method of the data model of choice, i.e., how to represent the smallest unit of data. RDF does, in fact, lead to a variety of design decisions that have an impact on querying performance.
- *Annotate*, which focuses on converting raw streaming data into RDF streams. This optional step emphasizes the need for having an interoperable data format for data streams, too.
- *Describe* highlights the need for extensive and interoperable metadata that enables stream discovery.
- *Serve* focuses on the format, protocol, and services needed for data sharing in practice.
- *Query* concludes the life cycle indicating the remaining processing steps which are summarized in the following.

5. Running Example

As a running example, we will use the taxi dataset made available by ACM DEBS 2015 Grand Challenge[9]. The goal of the running example is to provide tangible examples for each of the steps of the updated lifecycle. The DEBS challenge consists of a taxi route analysis scenario in the city of New Year. The dataset consists of two streams: A *ride* stream that represents the route of a taxi rides in terms of (i) taxi description, (ii) pick-up and drop-off information (e.g., geographical coordinates of the place and time of the event), and (iii) the number of passengers. The *fare* stream describes the ride payment information (e.g., tip, payment type and total amount). In more detail, Table 3 describes the fields contained in the *rides* stream, while Table 4 describes the fields in the *fare* stream. Note that the *rideId*, *taxiId* and *driverId* are contained in both streams.

The dataset was used in the DEBS challenges to solve two queries: 1) finding the *frequent routes* and 2) detecting the most *profitable areas*. For the first query, the goal was to find the top 10 most frequent routes during the last 30 minutes. The second query aimed to inditify areas that are, at a certain time, most profitable for taxi drivers. In the remainder of this paper, we will use the taxi dataset as a running example to detail the various steps of the life cycle.

[9] http://www.debs2015.org/call-grand-challenge.html

field	description
rideId	the unique ride id
taxiId	the unique id for the taxi itself
driverId	the unique id of the taxi driver
isStart	indicates if the ride has started or ended
eventTime	timestamp of the event
startLon	the longitude where the ride started
startLat	the latitude where the ride started
endLon	the longitude where the ride ended
endLat	the latitude where the ride ended
passengerCnt	the number of passengers

Table 3
Description of the taxi *ride* stream data

field	description
rideId	the unique ride id
taxiId	the unique id for the taxi itself
driverId	the unique id of the taxi driver
startTime	the time the ride started
paymentType	the type of payment, either *cash* or *card*
tip	the tip amount for the ride
tolls	the amount of tolls payed for this ride
totalFare	the total fare

Table 4
Description of the taxi *fare* stream data

6. The New Lifecycle in Pratice

The following section will go through each of the life cycle steps in more detail. For each step, state-of-the-art solutions are described and compared. The running example is used to give ample examples for each step.

6.1. Identify

The lifecycle's identification step aims to distinguish relevant resources and design IRIs to identify them. When discussing streaming data on the Web, two types of resources are critical: the stream itself and the elements that the stream contains. Indeed, data streams, like datasets, represent a collection of data points, each of which can be independently identified. Some important aspects of the data stream are highlighted by Tommasini et al. [66] as they affect how data should be managed and identified, i.e., they are unbounded and ordered [61]:

- *Unboundedness* refers to the stream's inability to be stored in its entirety.
- *Order* refers to how data is consumed, i.e., sequentially as soon as it arrives.

Definition 6.1. A Web data stream is a Web Resource that identifies an unbounded ordered collection of pairs (o, i), where o is a Web resource, such as a named graph, and i is a metadatum, e.g. a timestamp, that can be used to build an ordering relation.

Definition 6.1, as originally proposed by Tommasini et al. [66], makes no assumption on the data that will be received. Instead, the definition decouples the identification between the data stream and the resource it contains. The use of HTTP IRIs to identify Web resources is recommended by Linked Data best practices for good IRIs design. While these recommended practices apply to the Web Stream resource as well, it's worth noting that for data access it is not the same. Streaming data does, in fact, necessitate an always-on connection (e.g., WebSocket). Hereafter, the section introduces two methods for solving identification, one from Sequeda et Corcho [57] and the other from Barbieri et Della Valle [9]. Notably, both methods require human intervention to design the URI scheme as shown in Figure 10.

```
hrs:1 a s:Sensor  ;
      s:measures [
        _measurement a hr:HeartRateMonitor   ] .
hrs:1 s:measures hrs:1/2022-07-15 17:00:00 .
      rdf:type hr:HeartRateMonitor;
      hr:heartRate "74";
      hr:timestamp "2022-07-15 17:00:00"^^xsd:dateTime
hrs:3 s:measures hrs:1/2022-07-15 17:05:00 .
      rdf:type hr:HeartRateMonitor;
      hr:heartRate "58";
      hr:timestamp "2022-07-15 17:05:00"^^xsd:dateTime
```

Listing 1: Sensor and Observation from Sequeda et Corcho [57].

Sequeda and Corcho suggested three new IRI approaches for identifying sensors and their observations [57]. The URI scheme below identifies a window with *start* and *end times*:

`http://linkeddata.stream/sensor/name/%start time%,%end time%`

Listing 1 shows an example from [57] that specifies a sensor as a streaming data source, and an observation as a streaming element. Sequeda et Corcho's vision includes spatio-temporal metadata in addition to streaming data. Sensor data are frequently linked to location metadata.[10].

```
:sgraph1 sld:lastUpdate "τ_{i+1}"^^xsd:dataTime ;
    sld:expires "τ_{i+2}"^^xsd:dataTime ;
    sld:windowType sld:logicalTumbling ;
    sld:windowSize "PT1H"^^xsd:duration .

:igraph1 :receivedAt "τ_i"^^xsd:dataTime ; :rdfs:seeAlso :sgraph1.
:igraph2 :receivedAt "τ_{i+1}"^^xsd:dataTime ; :rdfs:seeAlso :sgraph1.

:igraph1  { # I-Graph at τ_i using TriX Syntax
    :broker1 :does [ :tr1 :with "$ 1000"  ] }
:igraph2  { # I-Graph at τ_{i+1} using TriX Syntax
    :broker1 :does [ :tr2 :with "$ 3000" ] .
    :broker2 :does [ :tr3 :with "$ 2000" ] . }
```

Listing 2: sGraph and iGraphs from Barbieri et Della Valle [9].

Barbieri et Della Valle suggested to identify streams using IRIs that resolve a named graph comprising all relevant metadata (called sGraph) and IRIs to identify a single element in a stream (called iGraphs) via time-stamping. iGraphs and sGraph are linked to each other using the *rdfs:seeAlso* property, while the *:receivedAt* [9] property attaches a timestamp to the iGraphs using *xsd* literals. For example, in lines 1-4 in Listing 2 the sGraph is the result of resolving *http://stockex.org/transactions*. The iGraphs in Listing 2 lines 6-8 and 12-19 are the result of resolving *http://stockex.org/transactions/urlenconde(τ_i)*, and *http://stockex.org/transactions/urlenconde(τ_{i+1})*. Barbieri et Della Valle propose two URIs schemes:

`http://linkeddata.stream/%stream-name%`

`http://linkeddata.stream/%stream-name%/urlenconde(%timestamp%)`

In practice, unboundedness implies the ephemerality of stream elements. Indeed, clients subscribing to the stream will receive data chronologically, starting from the beginning of the connection. If past data are not explicitly stored, they will be inaccessible. Barbieri et Della Valle bypass this difficulty by linking iGraphs to the origin sGraph, which is always referentiable. Sequeda et Corcho propose that such dependencies be encoded inside the IRI schema. Extending the IRI schema though, may lead to misinterpretation, since each resource becomes independent of the stream. Rather than that, hash IRIs and fragment identifiers enable the binding of stream elements to stream resources without the requirement for an additional triple or the use of a special IRI scheme, i.e.

`http://linkeddata.stream/{stream-name}#{igraph-id}`.

Example 6.1. *(cont'd) Carrying on the running example for the identification step, the taxi streams can be identified by the base URL* `http://linkeddata.stream`*. Moreover, the following URI schemas will be used to identify Web Resources that are relevant for the paper:*

1. `http://linkeddata.stream/ontologies/taxi-ontology`
2. `http://linkeddata.stream/resource/ride-stream`
3. `http://linkeddata.stream/resource/fare-stream`
4. `http://linkeddata.stream/resource/ride-stream#ride-id`
5. `http://linkeddata.stream/resource/fare-stream#ride-id`.

[10]The interested reader can consult the book [42]

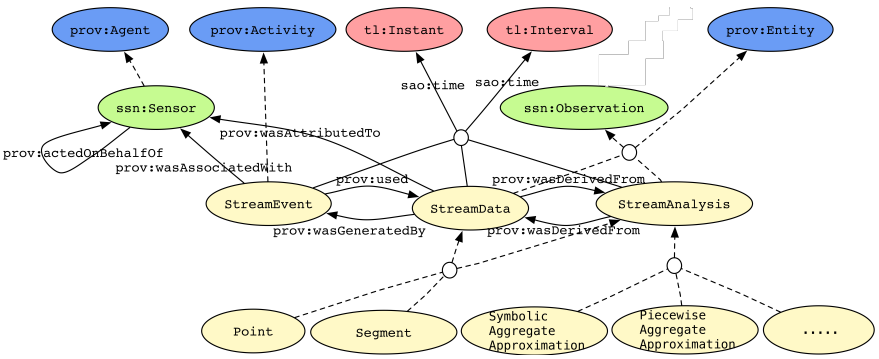


Figure 4: SAO Schema.

6.2. Model

This life cycle stage is concerned with defining the application domain from which data arrive. Towards this end, ontologies and models are designed using formal languages such as RDFS or OWL, which allow describing domain knowledge in a machine-readable fashion. At the time of writing, no mechanism for representing stream-specific knowledge exists, nor exists tools to support such a specific knowledge representation effort. Although the task remains experimental and essentially manual (cf Figure 10), some ontologies are well-known in the stream reasoning literature, like SIOC [13], Event Ontology [53], Activity Streams [11]. Moreover, ontologies like FrAPPE [6] and SAO [27] were designed specifically for stream reasoning applications. Therefore, their conceptualizations present peculiar characteristics that can guide future modelling efforts for stream reasoning. Notably, the role of a knowledge engineer is essential to the step completion.

There is widespread consensus in the stream processing literature regarding the most important abstractions. Specifically, streaming data is typically classified as *instantaneous* or *time-varying*. The former classifies data items as valid at a certain point in time, for example, sensor observations in a IoT stream; the latter classifies data that are changing over time, for example, the people that are in a room. Arasu et al. [2] introduced such data dichotomy to the extent of clarifying relational continuous queries, while Dell'Aglio et al. [22] extended it to RSP in order to operate on RDF graphs. Time-Varying abstractions are the consequence of a *continuous* stateful computation being applied to instantaneous data. While strictly static data are excluded from the formalization, they are frequently employed in practice. Indeed, the formalization can be extended to describe permanent facts [43]. The section will show how SAO and FrAPPE differentiate between such notions. Additionally, the section builds on the previous examples by modeling the color domain using the same abstractions. The comparison of the two ontologies to the abstract concepts used in stream processing is summarized in Table 5.

```
<sensor>  a  ssn:Sensor ;
        ssn:observes  qoi:Freshness .
:sensorRec1 rdf:type sao:Point ;
      prov:wasAttributedTo <sensor> .
:sensorRec2 rdf:type sao:Point ;
      prov:wasAttributedTo <sensor> .
:traffic-sensor-recording-619 rdf:type sao:StreamEvent ;
    prov:wasAsscoatedWith :government ;
  prov:used [ rdf:type :sensorRec1, :sensorRec2 ] ;
  sao:time  [ rdf:type  tl:Interval ;
          tl:at "2014-02-13T08:25:00"^^xsd:dateTime ;
          tl:duration "PT15H30M"^^xsd:duration  ] .
:freshness-traffic-619 rdf:type qoi:Freshness ;
  qoi:value "2014-02-13T08:25:00"^^xsd:dateTime .
:sax_AverageSpeedSample rdf:type  sao:SymbolicAggregateApproximation;
  rdfs:label "The sax representation of the traffic sensor recording obtained from Aarhus City.";
  sao:value "bbbbacdd";
  sao:alphabetsize "4"^^xsd:int ;
  sao:segmentsize "8"^^xsd:int ;
  prov:wasGeneratedBy :traffic-sensor-recording-619;
  qoi:hasQoI :freshness-traffic-619 .
```

Listing 3: SAO Example.

[11] https://www.w3.org/TR/activitystreams-core/

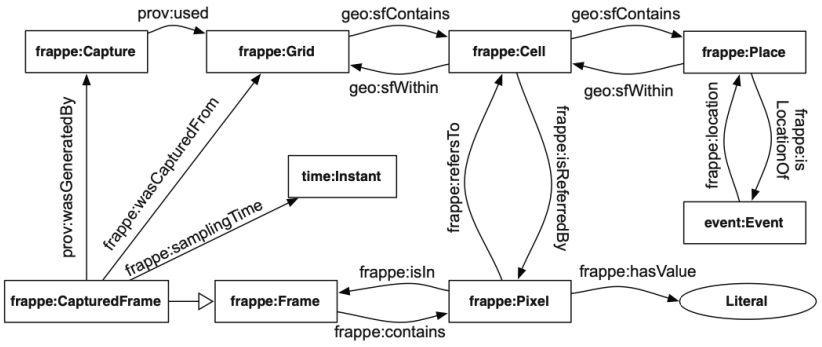


Figure 5: Frappe Schema.

The **Stream Annotation Ontology (SAO)**[12] enables publishing IoT-derived data streams. The vocabulary allows the in depth description of the stream transformations. PROV-O is used by SAO to track the provenance and OWL-Time is used for the temporal annotations [36]. To represent instantaneous data, SAO adopts two classes: `sao:Point` and `sao:StreamEvent`. The former represents one single data point in a stream of sensor observation (see Listing 3 at line 3.), while the latter models an artificial classification of the elements in the stream (see Listing 3 at line 7.). Moreover, SAO includes two Time-Varying concepts, i.e., `sao:StreamData` that models data points which are the results of aggregation, and `sao:Segments`, which allow representing a specific portion of a stream. The class `sao:StreamAnalysis` in SAO is used to represent a continuous computation (see line 15, Listing 3). As mentioned above, the result of the continuous computation is a time-varying concept. Finally, in SAO, the conceptualization is limited to the `sao:Sensor` class that represents the source of the streaming data (see line 2, Listing 3).

Frappe[13] is an OWL 2 QL vocabulary which is designed for spatio-temporal data analytics. It borrows its conceptualization from the photography domain [6]. As in photography, Frappe represents the world as a sequence of frames and events occurring within a spatio-temporal context. Listing 4 shows how Frappe can be use to model the dataset from the running example. Frappe borrows the instantaneous concept *Event* from the EventOntology. The *Event* concept represents something that happened in the real world at a given time. Notably, both SAO and Frappe make use of the OWL Time ontology to represent temporal concepts (see Listing 4 at line 17). Similarly to SAO, Frappe identifies time-varying concepts `frp:Pixel` and `frp:Frame`, which respectively correspond to the results of continuous computations over the *Events* occurring in a *Cell* or *Grid*. (see Listing 4 at lines 26 and 29.) Frappe represents the continuous computations through the concepts `frp:Capture` and `frp:Synthesize` (see Listing 4 at line 34). Furthermore, the classes *Grid*, *Cell*, and *Place*, which are used to represent spatial context, are assumed to be static (see Listing 4 at line 2). Note that data are treated as static when they do not change during the query execution. Notably, the static data represent different concepts which are not subject to time like such as units of measurement.

```
:Grid_1 gs:sfContains :Cell_1, :Cell_2 .
:Cell_1 a fr:Cell ;
  rdfs:label "39460"^^xsd:long ;
  fr:isReferredBy :1356995100000_39460 ;
  gs:sfContains :place1 ;
  gs:sfWithin :Grid_1 .
:place1 a fr:Point ;
  gs:asWKT "POINT( 40.715008 -73.96244 )"^^gs:wktLiteral ;
  gs:sfWithin :Cell_1 .

:E_A a :PickUpEvent ;    a event:Event ;
  event:time [ a time:Instant ;
   time:inXSDDateTime "2013-01-01T00:00:00"^^xsd:dateTime ];
  fr:location :place1 ;
  :hackLicense "E7750A37CAB07D0DFAF7E3573AC141"^^xsd:string;
  :medallion "07290D3599E7A0D6209346EFCC1FB5"^^xsd:string .
:E_B a :DropOffEvent ;  a event:Event ;
  event:time [ a time:Instant ;
```

[12] http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao
[13] http://lov.okfn.org/dataset/lov/vocabs/frapp

Concept	**SAO**	**FrAPPE**	Taxi Example
Instantaneous	Point,StreamEvent	Event	Ride/Fare
Time-Varying	StreamData,Segment	Pixel, Frame	FrequentRoute/ProfitableArea
Continuous	ColorSynthesis	Synthesize	RouteComposition/FareComposition
Static	StreamAnalysis	Grid, Cell, and Place	CityMap/Road/POI/...

Table 5
Concepts and Classes for SAO, Frappe, and Taxi Example.

```
   time:inXSDDateTime "2013-01-01T00:02:00"^^xsd:dateTime ];
  fr:location :B ; :connected :E_A ;
  :paymentType "CSH"^^xsd:string;
  :fareAmount "3.5"^^xsd:double;
  :totalAmount "4.5"^^xsd:double;
  :tripDistance "0.44"^^xsd:long; :tripTime "120"^^xsd:long.

:1356995100000_39460 a fr:Pixel ;
  fr:isIn :1356995100000 ;
  fr:refers :Cell_1 .
:1356995100000 a fr:CapturedFrame ;
  fr:contains :1356995100000_39460, :1356995100000_39461 ;
  fr:wasCapturedFrom :Grid_1 ;
  prov:wasGeneratedBy :1356995100000 ;
  fr:samplingTime [ a time:Instant ;
   time:inXSDDateTime "2013-01-01T00:05:00"^^xsd:dateTime].
```

Listing 4: Frappe DEBS Example.

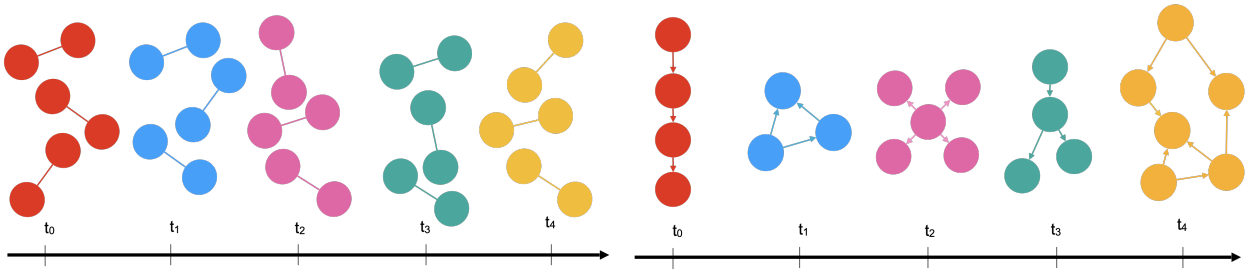
Example 6.2. *As indicated above, it is important to differentiate between instantaneous and time-varying concepts. These concepts are essential for the formulation of information needs and to present them as continuous transformations. Furthermore, if any static data abstraction emerges it should be included in the modeling. In our running example, the observed taxi events (both rides as fares) capture well the idea of the instantaneous concept as it is only valid at a certain point in time.*

In our taxi ontology, the class `:Ride` *identifies instantaneous concepts, while* `:FrequentRoute` *models time-varying ones resulting from the co-occurrence of multiple rides. In this taxi stream example, the acts of computing frequent routes or profitable areas are continuous, i.e., they are repeatedly evaluated over a stream of input and their output is also a stream. In particular, the result of the continuous computation that takes care of the ride composition through the* `:RouteComposition` *concept, is a stream of derived routes from the ride events. Finally, it is worth noting that the association between ride and the geographical areas does not depend on any temporal annotation. Thus, location information themselves, e.g. data regarding the roads, are represented as static knowledge that can be joined with the stream of rides and fares.*

6.3. Shape

The shaping step of the lifecycle is concerned with the structuring of stream data items in order to facilitate their consumption and processing [21]. There are two types of data streams in the stream processing literature: *structured* and *semantic*. The former denotes a data stream whose contents are restricted by a schema. For instance, in relational stream processing, each element of a data stream must adhere to a single, unique set of attributes. The latter is a data stream whose immediate elements can be interpreted in terms of a particular model. In practice, the model is frequently expressed using a knowledge representation language (e.g., RDFS or OWL), and the interpretation is guided by conventional deductive reasoning tasks. Typically, the schema for structured streams is determined in advance. Thus, the shaping process is reduced to identifying a subset of the schema to publish or expanding the schema through the use of numerous data sources. By contrast, semantic streams often use a schema-free data paradigm, such as RDF. Thus, shaping is reduced to determining the smallest structural unit that makes up the stream data item. Shaping is distinct from modeling in that the conception is already specialized in an ontological schema at this point. Rather than that, the amount of granularity at which data is provided has yet to be established. To determine the final shape, the following critical questions must be addressed:

- What is minimal unit of information that is useful for the processing?
- What are the requirements in terms of throughput-latency?



(a) Triple-based. (b) Graph-based.

Figure 6: RDF Stream.

In the stream reasoning literature, two approaches are popular for shaping streaming data, i.e., **triple-based** RDF streams, which use individually timestamped triples to punctuate the data stream, and **graph-based** RDF streams, which use timestamped (named) graphs to punctuate the data streams. From a formal standpoint, the two techniques are equivalent [22]. Indeed, by grouping the items by time, it is always possible to construct a graph-based stream from a triple-based one. To convert a graph-based stream to a triple-based stream, on the other hand, requires a stateless operation that flattens the stream's content (e.g. flatMap).

The two models, however, have a distinct effect on processing performance. In particular, triple-based streams (see Figures 6a) minimize latency, as ingestion occurs concurrently with input parsing (which happens node by node, triple by triple, and graph by graph). However, they impact redundancy as equivalent nodes are repeated multiple times. On the other hand, graph-based streams (see Figure 6a and 6b) optimize better for throughput since they are a de-facto micro-batching mechanics, but they require a more sophisticated parsing process. The finer-grained punctuation makes the triple-based streams more subject to noise like late arrivals, while graph-based molecules have a higher risk of information loss. Finally, triple-base streams are quite generic and suitable for any processing task, including incremental maintenance (i.e., they should be further interpreted as addition and deletions). Instead, graph-based streams pave the road to query-driven stream shaping.

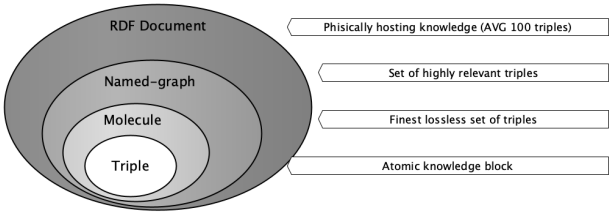


Figure 7: RDF Molecule from [25].

Other options are ERI Streams, which are sequences of blocks with the same subjects triples [26]. The solution then employs an encoding procedure akin to that used by the Efficient XML Interchange (EXI) format but adapted for RDF. Each group is multiplexed into lists of items with comparable values that are well-suited for compression using common techniques. ERI Stream makes use of Ding et al [25] notion of RDF's Molecule to determine groups. In practice, the notion can aid in generalizing the concept of shaping RDF streams. RDF molecules, according to Definition 6.2, are significantly more informative than individual RDF triples but are not classified as named graphs (see Figure 7). Notably, Ding's molecules are not identified as a-priori and they are meant for provenance tracking. An agreement between the stream molecule and the shape of continuous queries might allow performance optimisation. Figure 6b presents a graph stream in which every graph follows one of the most common SPARQL query shapes that could inspire specific molecular structures.

Definition 6.2. ***RDF Molecule***. *Given an RDF graph G, an RDF molecule $M \subset$ is a finest set of triples $\{t_1, t_2, ..t_n\}$ that decomposes the graph without loss of information [25].*

Notably, although methods for asserting the data validly based on shapes exist, e.g., SHACL [18] and Shex [60]; to the best of our knowledge they were never applied to the streaming scenario.

6.4. Annotate

The annotation step of the lifecycle focuses on converting streaming data into a machine-readable format. As for Linked Data, RDF is the data model of choice for publishing data on the Web. However, data streams are not always

rideId	taxiId	driverId	startTime	paymentType	tip	tolls	totalFare
1305	2013008681	2013007085	2013-01-01T00:02:00	CSH	1.0	0	4.5
1306	2013002109	2013005429	2013-12-01 00:02:01	CRD	0.5	0	7.5
1307	2013002083	2013002080	2013-12-01 00:02:11	CRD	2.87	0	14.37

Table 6
Example of event in the *fare* stream.

produced directly in RDF. For instance, in the IoT, sensors usually push observations using a tabular format like CSV or document formats such as JSON. In many applications, data are also compressed to save bandwidth and reduce transfer costs. To tackle these problems, a conversion mechanism must be set up that can transform the data stream into a machine-readable format. In order to design an adequate conversion pipeline, the following questions should be answered first:

- What is the model of the input data and what is the expected output?
- Is there any contextual information that can be used during the conversion?
- What mapping language is best suited for the conversion?
- How can the streaming temporal dimension be treated during the annotation?

The domain ontologies designed during the modeling step should be used in the conversion pipeline to annotate the raw data to the RDF model. Additionally, streaming data may be combined with more contextual domain knowledge, which allows capturing the domain information collected in an ontological model. The result of the annotation step should shape the original stream as an RDF stream, according to the decision made at the previous step of the lifecycle. Thus, the RDF molecule described in the *Shape* step, also plays an important role in the annotation process.

Technologies like R2RML [19] and RML [24] are adequate to set up static data conversion pipelines, but they are suited to deal with data streams.[14] In particular, due to the infinite nature of streaming data, the conversion mechanism does not terminate if it operates under the same assumptions used for static dataset annotation. In the literature, the most common workaround for this problem consists of annotating the stream one element at a time. Alternatively, one can use a window-based Stream Processing engine to transform the input stream by means of a continuous query. The latter approach has however only been described theoretically, yet never implemented. A notable exception is the Morph$_{stream}$ [15] engine, which extends the mapping language to include the windowing semantics. However, rather than annotating the data to RDF, it uses the mapping to enable ontology-based streaming data access [34].

Example 6.3. *Continuing the taxi stream running example, the annotation step should make use of the taxi ontology in Listing 4. Table 6 depicts a number of observed fare event. In practice, event data are often represented as JSON or CSV. For simplicity, we assume the latter.*

For the transformation of the CSV stream into an RDF Stream, the conversion pipeline must apply a transformation rule that can be specified using the RML mapping language. RML is preferred over R2RML W3C recommendation since RML has support for processing CSV directly as well as a number of other formats and input protocols. Listing 5 shows a subset of the transformation rules that transforms the fare event. Similar rules are deployed for the ride events. Note that the mapping represented here uses the more readable YARRML syntax [29], which is more concise than RML. Finally, Figure 8 depicts the mapping process and the resulting RDF stream denoting a sample occurrence of a taxi fare. Streaming conversion pipelines can implemented using CARML[15] or RMLStreamer[16] [28], which support streaming-like annotation given a standard way for iterating over the input data, i.e., line-by-line for CSV or document-by-document for JSON.

[14] Note that RML is more generic and stronger theoretical foundations than CSV2RDF used in the OpenSense2 project.
[15] https://github.com/carml/carml
[16] https://github.com/RMLio/RMLStreamer

```
prefixes:
 taxi: "http://linkeddata.stream/ontologies/taxi-ontology#"

mappings:
  fares:
    sources:
      - ['source_1.csv~csv']
    s: taxi:fare/$(rideId)
    po:
      - [a, taxi:DropOffEvent]
      - [taxi:paymentType , $(paymentType)]
      - [taxi:totalAmount , $(totalFare)]
      ...
```

Listing 5: An RML Mapping.

6.5. Describe

The *Describe* step of the life cycle aims at providing useful representations of the data streams to be consumed by humans and/or Web agents. In practice, metadata must be written using formal language while sharing the used terminology. In the case of the description of static datasets, the use of standard vocabularies for metadata management, e.g., DCAT or VOID, is established. Public data catalogues such as DataHub, Zenodo, or Google's dataset search then use the metadata for indexing purposes. The WebDataCommons indexes more than two million datasets that are annotated using schema.org. Although machine-readable metadata can be generated or extracted from the data, the curation work remains substantially manual.

In addition, streaming data calls for domain-specific metadata that captures their time-varying nature. Open data catalogue search capabilities struggle to interpret data semantics correctly, often falling into pitfalls that hinder the user's ability to discover data streams. To avoid such issues, two important questions should be answered:

- How can we capture the semantics of streaming data?
- What metadata are essential for streaming data?

Link in the modelling step, metadata curation for data streams is mostly human-centred (cf Figure 10). In this context, prominent vocabularies are emerging to address the problem of describing streaming data. The **Vocabulary for Cataloging Linked Streams (VoCaLS)** is an ontology to enable the interoperability between data streams and streaming services on the web [67]. It presents three modules for 1) publishing of streaming data following the Linked Data principles, 2) description of the streaming services that process the streams, and 3) tracking the provenance of stream processing [67].

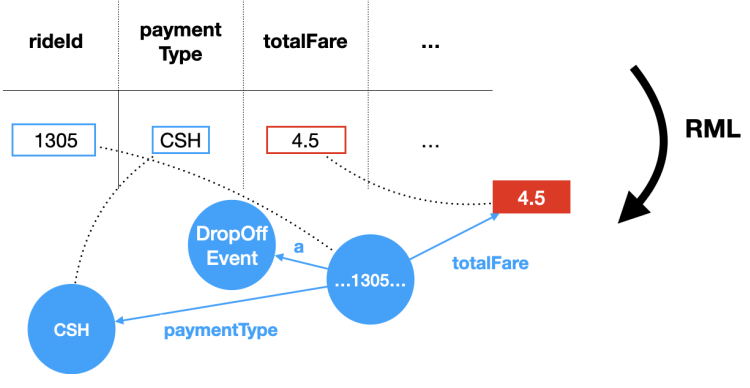


Figure 8: Illustrating the annotation process from a table stream to an RDF Stream through RML.

The **Stream Annotation Ontology (SAO)** allows the publishing of derived data about IoT streams. SAO can represent both raw and aggregated data. The vocabulary allows the description of the aggregation transformations in depth. SAO relies on PROV-O to track the aggregation provenance and OWL-Time for the temporal annotations [36].

Linked Data Event Stream (LDES)[17] defines a collection of immutable objects that evolve over time. It can describe both historical and real-time updates. *LDES* uses the TREE specification[18] for modeling the collections and data fragmentation purposes when the size of the collections becomes too large for a single HTTP response. TREE defines a collection of objects that adhere to a certain SHACL shape, and how these collections can be fragmented and interlinked using multi-dimensional HTTP pagination [37].

Lastly, **Schema.org (SORG)** includes two concepts that are relevant for stream representation, i.e., DataFeed[19] and DataFeedItem[20].

The FAIR initiative[21] highlights that metadata should be generous and extensive, including *descriptive* and *contextual* information about the data, as well as indications of data quality. However, the initiative gives some indication of what metadata is more relevant. The remainder of this section discusses the most important metadata with respect to SLD. In particular, the section shows how the aforementioned ontologies allow to (i) include descriptive information in natural language, e.g., the title; (ii) include information about streaming data quality, e.g., the stream rate and schema violations; (iii) present contextual domain knowledge, e.g., related ontologies; (iv) semantically annotate the publishing service; (v) present proper licensing (vi) include provenance metadata, e.g., generating pipeline or R2RML mappings.

Before going into details, we give a summary of the FAIR initiative, which aims to improve the way (scientific) data are shared and foster interoperability, using four pillars, i.e., Findable, Accessible, Interoperable, and Reusable. We will use these pillars to further go into details of the *Discribe* step of the SLD life cycle:

Findable. (F1) data should be assigned unique and persistent identifiers, e.g., DOI or URIs. (F2) data should be assigned metadata that includes descriptive information, data quality and context. (F3) metadata should explicitly name the persistent identifier since they often come in a separate file. (F4) Identifiers and metadata should be indexed or searchable.

Accessible. (A1) Data and metadata should be accessible via (a) free and (b) open-sourced, and (c) standard communication protocols, e.g., HTTP or FTP. Nonetheless, authorization and authentication are possible. (A2) metadata should be accessible even when data are no longer available.

Interoperable. (I1) Data and metadata must be written using formal languages and shared vocabularies that are accessible to a broad audience. (I2) Such vocabularies should also fulfill FAIR principles. (I3) Data and metadata should use qualified references to other (meta)data.

Reusable. (R1) Data should adopt an explicit license for access and usage. (R2) Data provenance should be documented and accessible. (R3) Data and metadata should comply with community standards.

Table 7 summarizes the analysis presented below. The table also summarizes the explicit support for RDF Streams (I1) and Stream Descriptor (F3,A2). Moreover, the tables do not include evidence for I2 and I3. Indeed, none of the presented vocabularies fulfil I2, as they all rely on at least one non-FAIR import. On the other hand, all the ontologies support I3, since they all reuse concepts from other vocabularies.

Metadata. (F2) *VoCALS* does not allow to include of any information on data quality but limits its support to descriptive information about the resources, e.g., name and owner, and contextual information, e.g., the vocabulary used to annotate the stream content. *SAO* supports all the three metadata annotations. It is worth noticing the presence of specific classes and properties for annotating data quality by extending *QOI*[22]. *LDES* explicitly supports only contextual metadata as it directly relies on the TREE specification.

Service. (A1) The FAIR prescription for serving data and metadata relies on standard protocols. While on the Web this usually means HTTP, it does not directly apply to streaming data that call for specific protocols. Except *LDES*, which inherits the HTTP access assumption from *TREE*, the other ontologies include a specific abstraction that aims at generalizing the access to the streaming data, i.e., `voc:StreamEndpoint`, `iots:Service`, `ces:EventService`.

[17] `https://w3id.org/ldes/specification`
[18] `https://w3id.org/tree/specification`
[19] `https://schema.org/DataFeed`
[20] `https://schema.org/DataFeedItem`
[21] `http://go-fair.org`
[22] `https://mobcom.ecs.hs-osnabrueck.de/cp_quality/`.

Ontology	D	Q	C	RDF Stream (I1)	Service (A1)	Descriptor (F3,A2)	LICENSE (R1)	Prov. (R2)
VoCALS	✓		✓	✓	✓	✓	Apache 2	✓
LDES			✓	≃		✓	CC	
SAO	✓	✓	✓		✓		CC	✓
SORG	✓	✓	✓		✓		CC	✓

Table 7
Summary of the thirty-thousand foot view analysis. Legend: ✓=supported, ≃=partially supported, [D]escriptive, Data [Q]uality, [C]ontextual, [C]reative [C]ommons.

License. (R1) All the selected ontologies have an explicit license. *VoCaLS* and *LDES* explicitly suggest associating a license with the annotated data streams.

Provenance. (R2) Finally, tracking the provenance of the shared data is an encouraging practice from the FAIR initiative. In these regards, all the ontologies includes dedicated classes and properties that allow to represent the analysis performed on the streaming data, i.e., `voc:Query` based on RSP-QL; CES `ces:EventPattern` for complex event recognition, `sao:StreamAnalysis` and `iots:Analytics` for continuous analysis of the data streams.

Example 6.4. *(cont'd)*

Carrying on the running example for the description step, Listing 6 describes the taxi stream using one of the presented vocabularies, i.e., VoCaLS and DCAT Service metadata are listed at line 15 Provenance is limited to the publisher, see line 13, and could be extended for example by describing the annotation process. Finally, service metadata are listed at line 15. More details are given in Listing 7, which is discussed in the next section.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX frmt: <http://www.w3.org/ns/formats/>
PREFIX vocals: <http://w3id.org/rsp/vocals#>
PREFIX vsd: <http://w3id.org/rsp/vocals-sd#>
<> a vocals:StreamDescriptor .

:taxiRidestream a vocals:RDFStream ;
 dcat:title "Taxi Ride Stream"^^xsd:string ;
 dcat:description "Stream of taxi rides"^^xsd:string ;
 dcat:license <https://creativecommons.org/licenses/by-nc/4.0/> ;
 dcat:publisher "https://linkeddata.stream" ;
 dcat:landingPage <http://linkeddata.stream/page/taxistream> ;
 vocals:hasEndpoint :TaxiRideEndpoint  .
```

Listing 6: Publishing taxi stream with Vocals and RSP-QL.

6.6. Serve

The fifth step of the lifecycle aims at making the streaming data accessible on the Web. This step aims to provide the data to the audience of interest, i.e., making them available for consumption. To this extent, three important questions should be answered:

- What data should we provision?
- How should we provide such data?
- Who (or what) is responsible for the provisioning?

Looking back at the beginning of the life cycle, and in particular at the identification step, it emerges that two kinds of resources are relevant when publishing streaming data, i.e., *the stream itself and the resource it contains.* This idea, which was originally introduced by Barbieri et al. [9], suggests decoupling the sharing of streaming data and metadata. The former should be accessible in a continuous and reactive way, to promote streaming data access and the development of streaming applications. The latter, instead, should be accessible via HTTP for backward compatibility with the Web infrastructure. Although the metadata document should be standalone, it should be easy to link it to the actual streaming data, which are provisioned using a different yet more adequate protocol. To this extent, the stream descriptor should incorporate service metadata that describes the streaming data access.

Regarding specialized protocols, Van de Vyvere et al. wrote a comprehensive comparison between push and pull-based protocols for Web data [20]. We briefly summarize below the most common protocols used in processing Web Streams [46]:

- **HTTP long polling** reduces the amount of requests by instructing the server to return responses to clients upon the occurrence of an update. However, it requires stateful resource management to maintain the connections open. HTTP Long-polling shows Websockets-like performance when the underlying network latency is lower than half the data measurement rate [51].
- **Server-Sent Events** (SSE) extends, like HTTP long polling, maintains an open connection open for every client, but allows pushing multiple updates instead of one. SSE works best on HTTP/2, which multiplex all requests and responses over one connection
- Last but not least, **Websocket** is the most common protocol for real-time communication. It provides a bidirectional communication channel over one TCP connection for every client, after an HTTP handshake between the client and server. Websocket has a similar performance as SSE, but a lower transmission latency when the server needs to send large messages above 7.5 kilobytes. Also, the client-to-server communication provides Websockets with a lower transmission latency than using HTTP [58].

In addition, several streaming-oriented protocols exist (MQTT, CoAP [47], or STOMP[23].) with as many advanced features, e.g., pub/sub via topic management. However, they were not extensively used in the context of Web Stream publishing.

Provisioning data does not only concern the choice of a *protocol* but also the *data format*. The recommendation is to follow the standard guidelines to share the stream metadata (e.g., any suitable RDF serialization). On the other hand, for streaming data, the choice of serialization might hugely impact the performance [26]. Moreover, Fernandez et al. [26] clarified the need for a form of punctuation for the semantic stream that is called a molecule. Due to the strict latency constraints of stream processing applications, it is of paramount importance that the RDF stream serialization agrees with the molecule structure.

In absence of a standardized RDF stream serialization, existing works in the stream reasoning context seem to agree on adopting RDF formats that simplify the temporal annotation. In particular, the most common choice results in N-Quads and JSON-LD. Intuitively, both formats allow graph-based shaping to punctuate the stream. JSON-LD better fits semantically richer streams whose unit of information goes beyond the single triple. Nonetheless, the more expressive syntax allows contextualizing (which can also be referenced to avoid repeating information). N-Quads facilitate the parsing but require rebuilding of the graph unit in memory, ultimately impacting the latency of complex processing.

Finally, in terms of systems, the research activity about data stream provisioning can count on a number of solutions: RSP Engines like C-SPARQL [8] or CQELS [38] support the provisioning of query results; TripleWave [44] allows sharing streaming data via WebSockets. Finally, the RSP Services [5] allows to manage the stream resulting from a query either via WebSockets or asynchronous REST endpoint.

Example 6.5. *(cont'd) Carrying on the running example for the serving step, Listing 7 completes the stream descriptor presented in Listing 6 with the specification of the corresponding publishing service and access point. At line 4 the endpoint specifies the protocol, while at line 2 it specifies the format. In practice, multiple endpoints that provision alternative serialization of the stream is possible, following traditional content negotiation. Moreover, different protocols can be enabled to satisfy the needs of different clients. Listings 8 and 9 show the difference between the N-Quads and JSON-lD data formats formats, respectively.*

```
:TaxiRideEndpoint a vocals:StreamEndpoint ;
                  dcat:format frmt:JSONLD ;
                  dcat:accessURL "ws://taxiridestream:8080" ;
                  dcat:protocol "WebSocket" .
```

Listing 7: An access point to the taxi stream.

```
<../64a93e56abb0> <../rdf-syntax-ns#type> <../taxi#PickUpEvent> <../taxistream#1588684149> .
<../842483d5c22f> <../rdf-syntax-ns#type> <../taxi#DropOffEvent> <../taxistream#1588684150> .
<../g46842483ds5> <../rdf-syntax-ns#type> <../taxi#PickUpEvent> <../taxistream#1588684152> .
```

[23] https://stomp.github.io, Graph-QL Subscriptions

t]

```
{   "@id":"http://linkeddata.stream/streams/taxistream#1588684149",
    "@context":{
        "taxi":"http://linkeddata.stream/ontologies/taxi#",
        "rdf":"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs":"http://www.w3.org/2000/01/rdf-schema#" },
    "@graph":{  "@id":"64a93e56abb0", "@type":"taxi:PickUpEvent" } },
{ "@id":"http://linkeddata.stream/streams/taxistream#1588684150",
    "@context":{...},
    "@graph":{  "@id":"842483d5c22f", "@type":"taxi:DropOffEvent" } },
{ "@id":"http://linkeddata.stream/streams/taxistream#1588684152",
    "@context":{...},
    "@graph":{  "@id":"f0e58ce1c485",  "@type":"taxi:PickUpEvent" } }
```

Listing 9: The RDF stream of taxi rides in JSON-LD.

Listing 8: The RDF stream of taxi rides in N-Quads.

Finally, streaming data are exchanged across actors that expose, provision, and manipulate them. In traditional Web architecture, the notion of Agent or Service was dedicated to identifying the actors interested in or responsible for given resources. In the content of SLD, a Web Stream Processing Service or Agent (if they employ some AI technique [65]) is a special kind of Web service that manipulates Web streams and, in particular, RDF Streams.

Definition 6.3. *A Web Stream Processing Agent/Service (WSP) is a special kind of Web service that manipulates Web streams and, in particular, RDF Streams.*

Three main types of Web stream services are relevant for the SLD lifecycle: (i) *Catalogs* that provide metadata about streams, their content, query endpoints and more. (ii) *Publishers* that publish RDF streams, possibly following a Linked Data compliant scheme (e.g. TripleWave in Listing 10). (iii) *Processors*, which model a Stream Processing service that performs any kind of transformation on streaming data, e.g. querying.

Intuitively, the first two services are relevant within streaming data publication, while the latter is relevant for querying and, thus, will be discussed in detail in the next section. **Catalogs** are a common abstraction in the Web of Data due to the widespread need for dataset search. Nonetheless, existing data catalogues are not capable to capture the data stream semantics. Specialized RSP Catalogs can interpret the semantic annotations (see Annotation Step) and enable streaming data discovery. The adoption of vocabularies like those mentioned above are steps towards solving this issue. **Publishers** deployed to provide the streaming data using a protocol suitable for continuous and/or reactive data access. As servers, Publishers should implement content negotiation. Thus, publishing interest in serving RDF streams take over the conversion process. As discussed alongside the section, stream descriptors have a critical role in making the stream findable. To this extent, TripleWave [44] is a reusable and generic tool that enables the publication of RDF streams on the Web. It can be invoked through both pull- and push-based mechanisms, thus enabling RSP engines to automatically register and receive data from TripleWave.

Example 6.6. *(cont'd) Carrying on the running example, Listing 10 adds additional metadata for the stream publisher originally included in Listing 6. represent a bridge between publishers and catalog. Figure 9 shows how to carry on the lifecycle consuming the taxi streams, converting it, and at the same time making the stream descriptor available.*

```
<http://linkeddata.stream> a vsd:PublishingService ;
                           vsd:hasFeature vsd:transforming .
```

Listing 10: The Publisher of the Taxi Stream.

6.7. Query

The last step of the lifecycle details how the data, that has been *Served* using the previous steps in the lifecycle, can be queried. Looking at the literature on applied stream reasoning, one can notice that the querying steps tend to repeat. Although the community has not yet identified best practices, it is possible to enumerate a number of frequently used streaming data transformations:

Application	System	Filter	Enrich	Lift	Merge	Synthesize
BOTTARI [4]	C-SPARQL/SUN	σ,TP	POIs,Areas		Splice	Aggregates
SLD [7]	C-SPARQL	σ,TP	Sentiment		Splice	Aggregates
STAR-CITY [41]	custom	SPARQL	Map,Weather, Traffic History	OWL 2 EL	Splice	Aggregates, Prediction
CityPulse [52]	CQELS	σ,TP	Sensors	ASP	Cross	Aggregates, Traffic Events
Agri-IoT [31]	CQELS/C-SPARQL	σ,TP	Sensors	RDFS	Splice	Aggregates
Optique [33, 35]	custom/STARQL	σ, π	Sensors Configurations	DL-Lite$_A$	Cross, Splice	Aggregates

Table 8
Summary of Platforms Querying Features. Legend: [T]riple [P]attern.

- *Filter*, i.e., remove from the input streams the sub-portion that is not relevant for the analysis;
- *Enrich*, i.e., add to the input streams contextual data that make the analysis more informative;
- *Lift*, i.e., raise the input stream's abstraction level by means of background knowledge;
- *Merge*, i.e., combine two or more input streams to perform joint analyses;
- *Synthesize*, i.e., reduce the input streams to a summary by means of aggregations such as count, min, max, average, quartiles, or analytical function, e.g., Pearson correlation.

Table 8 summarizes the analysis for the selected projects that provided detailed information regarding their *Query* step. Finally, to aid comprehension, the section provides a visual representation using the taxi example.

Data streams are collected very close to their source at high-frequency with limited focus on data quality. Therefore, they are often filled with redundant, noisy, and non-relevant data. Therefore, data cleansing is a necessary step for most applications. In the mentioned frameworks, filtering is a stateless operation, i.e., it operates on a single stream element at a time. In these cases, filtering is possible using traditional relational selection (σ) or Triple pattern matching. More sophisticated forms of filtering are possible, e.g., STAR-CITY employs the full expressive of SPARQL to select data from a view over the input sources. An example of filtering from CityPulse requires removing observations regarding parking with no vacancy or selecting locations with low congestion levels;

Example 6.7. *Listing 11 presents the filtering practice for the taxi examples. From the stream of taxi rides, the filter removes non-Dropoff events and outputs a DropOffEvent stream.*

```
PREFIX taxi: <http://linkeddata.stream/ontologies/taxi#>
PREFIX :     <http://linkeddata.stream/resource/>
```

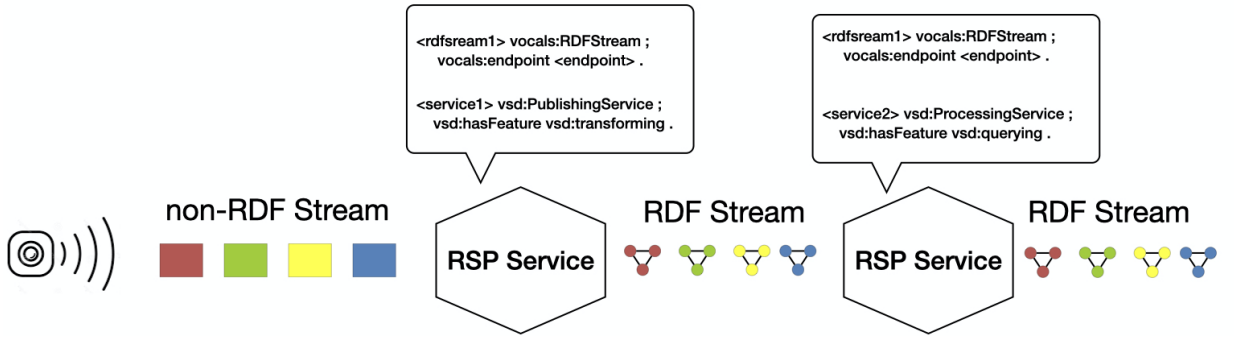


Figure 9: Converting and publishing the sensor stream with TripleWave.

```
CONSTRUCT { ?d a taxi:DropOffEvent .}
FROM NAMED WINDOW <w> ON :taxistream [RANGE PT15S STEP PT5S]
WHERE {
  WINDOW <w> { ?d a taxi:DropOffEvent .}
}
```

Listing 11: An example of Stream Filtering that selects all Dropoff events.

Due to strict latency requirements, it is often a good idea to reduce the information to the minimum when publishing data streams. However, sometimes is necessary to put items in context by enriching streaming data with static ones. Indeed, static and slowly-evolving data should not be repeatedly streamed but instead made available with the stream's metadata. In CityPulse, Agri-IoT, and Optique, sensors' positions and rates evolve much slower than sensors' observations. Therefore, it is reasonable to keep such metadata separate from the sensors' measurements and assume them as static. Sensor metadata enriches streaming observations with geo-spatial context, measure precision, or units of measure. Moreover, Optique also includes configuration regarding the platform as static metadata. Finally, BOTTARI and STAR-City show enrichment cases with sensor data joined with POIs, traffic, and weather history.

Example 6.8. *One way to enrich the taxi ride stream is by linking the drop-off or pick-up locations with the static city map, which allows linking the Points of Interest near the drop-off or pick-up locations. The RSP-QL query to perform this enrichment is shown in Listing 12. The enrichment is done through the variable ?place that occurs both in the ride stream and in the static city map graph.*

```
PREFIX : <http://linkeddata.stream/ontologies/taxi-ontology#>
PREFIX fr: <http://linkeddata.stream/ontologies/frappe#>
REGISTER RSTREAM : enrichedtaxistream AS
CONSTRUCT{  ?rideEvent a :DropOffEvent; :nearPoI ?poi }
FROM NAMED <citymap.rdf>
FROM NAMED WINDOW <rw>  ON :ridestream [RANGE PT30S STEP PT5S]
WHERE {
  GRAPH <citymap.rdf> { ?place :hasPoI ?poi . }
  WINDOW <rw> {
      ?rideEvent a :DropOffEvent;
                 fr:location ?place.
  }
}
```

Listing 12: Enriching the ride stream with the PoI from the city map.

Similar to enrichment, lifting allows to leverage an external context for enhancing the analysis. In particular, logical reasoning can abstract the data in the stream to higher-level concepts defined in the domain knowledge. This can be useful for integration purposes. Considering the aforementioned frameworks, neither BOTTARI nor SLD performs any lifting. On the other hand, STAR-CITY and Agri-IoT adopt standard knowledge representation languages, i.e., OWL 2 EL and RDFS, to define knowledge-based. CityPulse uses Answer Set Programming to perform *lifting*. Domain knowledge is captured in form of rules that are leveraged to raise the abstraction level of streaming data and operate on more complex concepts. Also, Streaming MASSIF employs Description Logics to abstract event types and combines them using complex event recognition rules. Last but not least, the Optique project employs reasoning to perform query rewriting rather than lifting, which is more efficient in a streaming scenario due to the permanent nature of continuous queries. Nonetheless, the final result is comparable.

Example 6.9. *In the taxi ride example, reasoning can be used to abstract the rides based on their location of pick-up or drop-off. So for example, after enrichment, lifting can be used to abstract the rides to either CityCenterDropOff events or SuburbDropOff events. Listing 13 shows the corresponding RSP-QL query that exploits the taxi ontology, containing rules such as:*

(R_1) *CityCenterDropOff(?x)* $\leftarrow$ *DropOffEvent(?x), hasPoI(?x, ?poi), CityCenterPoI(?poi)*

(R_2) *CityCenterPickUp(?x)*$\leftarrow$ *PickUpEvent(?x), hasPoI(?x, ?poi), CityCenterPoI(?poi)*

(R_3) *SuburbDropOff(?x)*← *DropOffEvent(?x), hasPoI(?x, ?poi), SuburbPoI(?poi)*

(R_4) *SuburbPickUp(?x)*← *PickUpEvent(?x), hasPoI(?x, ?poi), SuburbPoI(?poi)*

```
PREFIX : <http://linkeddata.stream/ontologies/taxi-ontology#>
PREFIX :      <http://linkeddata.stream/resource/> .
SELECT ?cityDropOff
FROM NAMED WINDOW <ew> ON :enrichedtaxistream [RANGE PT15S STEP PT5S]
WHERE {
  WINDOW <ew> { ?cityDropOff a :CityCenterDropOff. }
}
```

Listing 13: Enriching City drop offs.

All the aforementioned examples present multi-streams scenarios. Indeed, merging streams together is a standard practice in most commercial streaming applications. Without going in-depth about streaming join algorithms, this section distinguishes two main approaches to merging, i.e., Splice and Cross. The former consists of the point-wise union of two-stream without any temporal evaluation. This approach is common when multiple sources are annotated and converted locally, while data are rerouted within a mixed factory. Multiple streams can be combined to extend the analysis expressivity. Merging include both splice, i.e., the element-wise union of two RDF streams, and crossing, i.e., time-based windowed joins. For example, streams from different parking lots can be *spliced* into one and crossed with one or more streams that monitor the streets in the last 15 minutes. BOTTARI, SLD, STAR-CITY, and Agri-IoT focus on splicing, as they first merge all the input streams into a unified RDF stream. On the other hand, CityPulse and Optique allow performing finer grain analysis crossing only the most recent portions of the stream of interest.

Example 6.10. *Listing 14 shows the RSP-QL query for crossing ride and fare streams.*

```
PREFIX taxi: <http://linkeddata.stream/ontologies/taxi-ontology#>
PREFIX :      <http://linkeddata.stream/resource/> .
CONSTRUCT {  ?g a :RideFareEvent; ?pr ?or; ?pf of.}
FROM NAMED WINDOW <rw> ON :bluestream [RANGE PT30S STEP PT5S]
FROM NAMED WINDOW <fw> ON :yellowstream [RANGE PT15S STEP PT5S]
WHERE {
  WINDOW <rw> { ?r a :RideEvent . ?r :hasrideID ?id; ?pr ?or}
  WINDOW <fw> { ?f a :FareEvent . ?f :hasRideID ?id; ?pf ?of.}
  BIND( UUID() as ?g )
}
```

Listing 14: Crossing ride and fare streams.

Finally, to synthesize the content of a stream, RSP allows the use of *Aggregations*, which can consist of counting several occurrences, computing averages, computing minimum/maximum values, etc. Common aggregations are time-scoped for efficiency reasons, e.g., the average car passing by a certain section within 15 minutes. Moreover, CityPulse suggests the reporting of composite events. In CityPulse, aggregations are used to compute the average vehicle speed over the past 5 minutes to enable route planning. *Aggregations* are an important querying step in RSP, as it allows to summarize the data that is captured inside a certain window.

Example 6.11. *Listing 15 shows an aggregation example for the ride example.*

```
  PREFIX taxi: <http://linkeddata.stream/ontologies/taxi#>
  PREFIX :      <http://linkeddata.stream/resource/>
  SELECT (COUNT(?d) AS ?num_dropOff)
  FROM NAMED WINDOW <w> ON :taxiStream [RANGE PT1H STEP PT5M]
  WHERE {
  WINDOW <w> { ?d a taxi:DropOffEvent .}
  }
```

Listing 15: An example of Stream Aggregation that counts all Blue occurrences.

In summary, the query step of the lifecycle is the one with the highest maturity from a system standpoint: BOTTARI and SLD use the C-SPARQL engine to implement the querying step of the lifecycle. At the same time, AgriIoT, SpitFire, OpenIot and CityPulse also include a second RSP engine, i.e., CQELS engine, for optimizing latency. Finally, STAR-CITY and Optique implement custom solutions. The former relies on existing reasoners for ontology stream reasoning, while the latter leverages ontology-based data access technologies for translating the queries that later developed into STARQL [32]. Although each project organizes the pipeline independently, we can notice how they all agree on the design and implementation.

7. Related Work

In this section, we discuss the related work positioning our survey in the stream reasoning literature. At the center of our investigation, there is the life cycle of stream reasoning applications. Nonetheless, as the title suggests, the proposal of [66] is the first attempt to formalize the steps. On the other hand, there have been a number of proposals for non-SLD life cycles.

The closest proposal to [66] is the one by Hyland et Wood [30]. Indeed, Tommasini et al. explicitly reuse the same steps to stress the generality of the SLD approach. The main difference, which appears in all the traditional Linked Data life cycles, lies in the continuous nature of each step. Despite Hyland and Wood taking the problem of maintenance into account while facing dataset updates, the published resources are assumed to be finite. This aspect emerges clearly in those steps that involve algorithmic work, e.g., Conversion.

Steps of the Life Cycle (underlying the relevant ones): *Identify*, *Model*, *Name*, *Describe*, *Convert*, *Publish*, *Maintain*.

Auer et al. [3] describe LOD2, i.e., a tool that integrates different resources to support a circular life cycle. Their proposal contains eight steps. However, the authors recommend that such steps shall not be taken in complete isolation. The steps, which are listed below, are tailored for handling data that change slowly or not at all. Indeed, the proposal includes a storage step. On the other hand, both authoring and evolution/repair hint at the possibility of time-varying linked data. Nonetheless, neither LOD2 nor the described process suggests any continuous processing approach. Notably, a more detailed introduction to such a life cycle was provided in [48]. However, none of the changes concerns the steps, but just the resources involved.

Steps of the Life Cycle (underlying the relevant ones): *Extraction*, *Storage*, *Authoring*, *Interlinking*, *Classification*, *Quality*, *Evolution/Repair*, *Search/Browsing/Exploration*.

Villazón-Terrazas et al. [68] describe a life cycle and a set of guidelines for the publication of governmental data. Similar to Auer et al., their proposal is a circular life cycle. However, their selected steps are more abstract and further elaborated in while described. It is worth noticing the presence of a unique step, i.e., data cleansing within the transformation context, which is mentioned to remove the huge amount of noise. Neither in [66] nor in our updated proposal such step is considered. Indeed, to the best of our knowledge cleansing for Streaming Linked Data has not been investigated yet. Although the life cycle is tailored for handling static or slowly changing data, a further similarity with our approach lies in the exploitation step. Indeed, Villazón-Terrazas et al. discuss the role of applications consuming such data and stress the importance of data discovery.

Steps of the Life Cycle (underlying the relevant ones): *Specification*, *Modeling*, *Generation* (Transformation and Cleansing), *Publication*, *Exploitation*.

Furthermore, we position our work with reference to other surveys in the Stream Reasoning/RDF Stream Processing context. In particular, to the best of our knowledge, three surveys were published on the topic of SLD over the years.

Margara et al. [43] surveys existing Stream Reasoning processing proposals, starting from concrete application scenarios to extract the requirements for Stream Reasoning. The survey focuses mainly on different proposals and provides a research agenda to move forward. It does not provide any details on the other steps of the life cycle, except for the processing step.

Mileo et al. [45] give an overview of the Stream Reasoning paradigm, focusing on expressivity, scalability, distribution, and benchmarking of existing processing approaches. The paper also identifies the key application domains for Stream Reasoning, such as Smart Cities, Web Of Things, and many others. The paper does not provide

Step	Maturity	Assumption	Contributions	Best Practices
Identify	High	A1	[9, 57]	Use Hash IRIs with fragment identifiers
Model	Medium	A2	[17, 13, 6, 52]	Distinguish Instantaneous and Time-Varying Concepts Mind the Static-Streaming Gap Represent Continuous Computations
Shape	Low	A2		Align Shape and Instantaneous Concept
Annotate	Medium	A2	[24, 19]	Use Semantic Streams Avoid Temporal Data Annotations
Describe	Medium	A1;A2	[67, 52]	Use Specialized Vocabularies Use HTTP for metadata and Continuous Protocol for Streams
Serve	High	A1;A2	[44, 5, 9]	Use RDF Serialization that simplifies time annotation Describe known Services that are related to stream
Query	Very High		[11, 8, 38, 34, 22, 64]	Filter as early as possible Keep Static Data Small Little Semantics goes the long way Opt for Crossing when streams are not synchronized Use time-based windows to scope advanced practices

Table 9
Discussion Summary.

any details of the other life cycle steps discussed here in the paper, however, it identifies the lack of unified foundations in Stream Reasoning, hindering its adoption. We believe that a unified life cycle for SLD will increase the adoption of Stream Reasoning in general.

Dell'Aglio et al. [23] provide a complete overview of the Stream Reasoning domain, comparing different approaches for entailment and highlighting many open problems and challenges for the future. Interestingly, the survey touches on the topic of shaping, annotating, and describing streams as open challenges. It does not provide an overview of solutions that were available at the time of writing (if any) or a broader view of a complete life cycle.

8. Discussion

In this section, we present the observations we collected while analyzing the life-cycle compliance of existing projects. In particular, we report best practices for the life cycle steps and we highlight the research gaps. Figure 10 depicts the extended proposal once again by colouring each step according to its *maturity level* and *human involvement* in each of the steps.

8.1. Maturity of the lifecycle steps

In our investigation, we discern the presence of two underlying assumptions

- A1: the Web architecture and its extensions are adequate for the distribution and consumption of data streams;
- A2: Semantic technologies, *except those related to querying*, are sufficient to implement the life cycle steps.

Such assumptions influenced the progress of research on Streaming Linked Data and, thus, the formulation of research questions and the emergence of best practices. Below we discuss our findings according to A1 and A2.

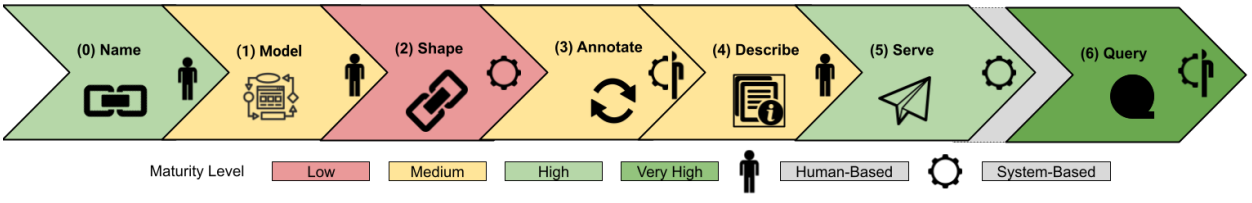


Figure 10: Streaming Linked Data life cycle.

Identify - Maturity: High (green). The identification step focuses on naming data streams as Web resources, i.e., assigning an IRI. As a consequence of A1, HTTP IRIs are adopted for stream resource identification. This emerges also in state-of-the-art proposals [9, 57]. In these regards, the best practice is to *Use Hash IRIs with fragment identifiers*, which ensures that the stream content representation is directly associated with the stream resource itself. We believe that relaxing A1 would not substantially impact such steps, except for extended URI schemes. Therefore, we consider the maturity level high.

Model - Maturity: Medium (yellow). The modelling step focuses on the knowledge representation efforts concerning SR/SLD applications. A2 directly impacts such a step, which relies on knowledge representation languages like RDFS and OWL, and recently temporal logics [69]. We identified the following best practices when A2 holds: (i) *Distinguish Instantaneous and Time-Varying Concepts:* Instantaneous concepts are valid at a specific point in time, e.g., a sensor observation. On the other hand, time-varying concepts change over time. They represent the two conceptual sides of a streaming application that operate on the elements in a stream. (ii) *Mind the Static-Streaming Gap:* When modeling the application domain it is important to take into account links between static and streaming data. Such links should be time-agnostic and ease tasks like streaming data enrichment and augmentation. (iii) *Representing Continuous Computations:* Identifying continuous computations allows for characterizing the streaming application workflow. Moreover, they link instantaneous and time-varying concepts, enabling provenance tracking for streaming analyses and transformations. Relaxing A2 implies the investigation of continuous knowledge representation languages that are able to incorporate continuous computations in the language semantics. To the best of our knowledge, such languages were not yet proposed. Nor a knowledge engineering method that is suitable for representing ephemeral knowledge. Therefore, we consider the maturity level medium.

Shape - Maturity: Low (red). The shaping step concerns the choice of a data model. Currently, little work has been done which directly derives from A1, i.e., RDF objects like triples, graphs, and datasets are used within the SLD context. We recommend to *Align Shape and Instantaneous Concept*. Notably, the stream shape directly impacts different aspects of the ingesting, e.g., parsing and filtering. Aligning it with the instantaneous concepts simplifies the design of efficient querying steps. Without relaxing A2, the introduction of recent work such as SHACL are being investigating by the SR community. Therefore, we consider the maturity level low.

Annotate - Maturity: medium (yellow). The annotation step focuses on converting non-RDF streams into RDF ones to increase interoperability. On the one hand, such a step partially relaxes A2 as a minimal extension of RDF is adopted for SLD. However, such an extension does not impact RDF semantics but introduces a simple form of punctuation to order RDF objects. In addition, as for Step 1, mapping languages like R2RML/RML are designed for static data conversion. Therefore, we identified the following best practices when A2 holds: (i) *Use Semantic Streams:* like for Linked Data, the recommendation is to adopt semantic streams to foster data sharing and integration. (ii) *Avoid Temporal Data Annotation:* due to the limitations of existing mapping languages, it is recommended to annotate element by element or adopt a micro-batching approach to ensure the termination of the conversion process. Relaxing A2 would call for novel mapping languages that capture the continuous aspect of the annotation process. A step towards this direction is the work of Calbimonte et al. [15] that extends the mapping language to incorporate windows. However, they propose a syntactic extension tailored for rewriting. Thus, we rank the maturity level as medium.

Describe - Maturity: medium (yellow). The description step covers the need for extensive and interoperable metadata. Both assumptions directly impact such a step, as the access to data stream metadata follows traditional interaction patterns of the Web (client/server). Moreover, the focus has been on static stream metadata (e.g., licensing and access). Nonetheless, some stream metadata are time-varying, e.g., stream rate in the last 5 minutes. Under the aforementioned assumptions, it emerges the recommendation to *Use Specialized Vocabularies*, e.g., VoCaLS or SAO. Stream-specific metadata allows to correctly interpret the streaming data semantics, improving discovery. Moreover, they simplify the automation that concerns streaming linked data access. Relaxing the assumptions would allow imagining alternative Web representations for streaming data. Currently, this speculation is beyond the scope of this survey. Moreover, research in this direction is active. In summary, the maturity level is medium.

Serve - Maturity: high (green). The serving step discusses how streaming data sharing, including their metadata, is done in practice. The emerging assumptions directly influence such a step. Indeed, the existing proposals adopt a client/server interaction pattern, except for web-socket-based communication (A1). Moreover, standard data formats are adopted for sharing data streams on the web. Under such assumptions, we identified three best practices: (i) *Use HTTP for metadata and Continuous Protocol for Streams:* When publishing Web streams, it is essential to guarantee the reactivity of the stream content consumption as well as the backward compatibility of metadata retrieval. (ii) *Use RDF Serialization that simplifies time annotation:* in the absence of a standard RDF serialization, adopt those that simplify

Step	Actionable	Decentralization	Distribution	Dependency (LW)	Interoperability (RW)
Name (S0)	Human	Yes	NotApp	High	High
Model (S1)	Human	Yes	NotApp	High	Medium
Shape (S2)	System	Possible	No	Medium	Low
Annotate (S3)	Human + System	Possible	No	Medium	Medium
Describe (S4)	System + Human	Possible	NotApp	Low	High
Serve (S5)	System	Yes	Yes	High	Medium
Query (S6)	Human + System	Envisioned	Yes	Low	High

Table 10
Summary of System/Human perspective of the Lifecycle steps. Legend: [L]eft-[Wise]; [R]ight-[Wise]; [Not App]licable.

the temporal annotation, i.e., JSON-LD and N-Quads. (iii) *Describe known Services that are related to stream*: Adding metadata about the services that are related to the stream helps contextualize the origin of the provided information and track the data's provenance. Relaxing assumption A1 suggests rethinking an interaction pattern tailored for streaming resources. For instance, content negotiation could be extended to obtain a streaming representation of a certain resource. Relaxing A2 stresses the work on data formats. Exciting work is ongoing in the stream processing community, e.g., Avro[24], which hasn't been discussed for SLD. Despite this possibility, the work on serving Web streams is quite mature. Thus, we categorize the maturity of such a step as high.

Query - Maturity: very high (dark green). The last step of the life cycle focuses on the consumption and manipulation of the published RDF streams. As indicated above, research in this area does not proceed under A2. Indeed, most existing proposals for processing RDF Streams include an extension of SPARQL or other semantic technologies. Therefore, we identified best practices that apply despite the holding of A2. (a) *Filter as early as possible:* Filtering should precede other transformations to reduce the amount of data to transfer. (b) *Keep Static Data Small:* To avoid performance loss in case of enrichment, keep the static data small and, whenever necessary, apply a cleansing technique to avoid errors. (c) *Little Semantics goes the long way:* Query rewriting is possible under certain conditions that limit the expressivity of the knowledge representation language. However, it gives substantial advantages in the streaming context, due to the long-lasting nature of continuous queries. (d) *Opt for Crossing when streams are not synchronized:* When the streams have different rates window-based merging is preferred as it regularizes the output. (e) *Use time-based windows to scope advanced practices:* When it comes to Synthesizing, using window-based computation can reduce the computational effort. In practice, window-based processing trades latency for throughput, reducing the performance stress on the system.

8.2. Human involvement, decentralization and distribution

Table 10 summarizes some horizontal aspects of the lifecycle; in particular, it helps us elaborate on the nature of the step, i.e., if it is either human-based or system-based, the role of decentralization and distribution.

The first column of Table 10 summarizes the **actionability** of each step. As represented by the human/gear icons in Figure 10, Step S5 is the only completely automated step, as it relies on internet protocols. Steps S0, S1, S4 and S6 require some human intervention. While S0 and S1 are completely manual, in S3, S4, and S6, human intervention enables the computation. For instance, in steps S3 and S6, the mappings and the queries shall be written by an engineer. Moreover, in step S4, metadata can be first extracted, but their integration requires human-based analysis. Regarding **decentralization**, Steps S0, S1, and S5 directly depend on the Web technological stack that guarantees decentralized approaches. At the same time, decentralized approaches for RSP have been envisioned in the stream reasoning context [65, 62, 14], for, Shaping, Annotation, and Description steps we can argue that is feasible yet not evident from the analysed literature. Regarding **distribution**, it does not seem applicable to human-based steps S0, S1, and S4. At the same time, we could not find evidence for Steps S2 and S2. Instead, S5, given its often related to IoT protocols, is known to work within a distributed environment; similarly, systems for distributed RSP query exist, e.g., CQELS Cloud [50] and Strider [54], but they were not directly involved in the selected projects.

Table 10 also summarises our findings regarding the dependency and interoperability between neighbour steps. Notably, given the *cyclic* nature of any lifecycle, we consider S6 and S1 neighbour steps too.

[24] https://avro.apache.org/

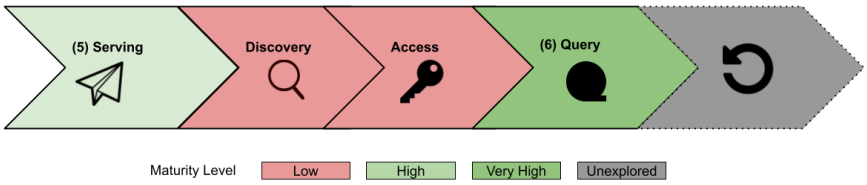


Figure 11: Discovery, Access, and Reboot.

8.3. Interoperability and dependencies between the lifecycle steps

For what concerns interoperability between the lifecycle steps, we study pairs of the lifecycle steps right-wise: **(S0-S1)**: The naming step (S0), with the possible exception of an extended protocols scheme like a WebSockets, interoperates well with the technologies used in the modelling steps (S1), which are part of the Semantic Web stack. **(S1-S2)**: as we presented in Section 6.3, if the streams have a schema/model, the shaping step is reduced to identifying the subset to publish. Instead, if the streams are schema-free, shaping requires determining the smallest stream unit, i.e., RDF molecules. Although this integration requires human integration, it is currently feasible. **(S2-S3)**: Being a subset of the streaming model, RDF molecules are implicit in the definition of the annotation rules. However, we are unaware of a study investigating them in depth, for example, relying on shape validation languages. Therefore, we cannot justify the compatibility more than low. **(S3-S4)**: Although none require the annotation phase of the data sharing principles, we observe a medium interoperability level between annotation languages and tools and description vocabularies, e.g., both VoCaLS and SAO have properties to link mapping files for provenance tracking. **(S4-S5)**: From our analysis emerges that S4 and S5 are highly interoperable. Indeed, metadata are essential for consumers to obtain the information to access the data, which are well covered by existing vocabularies (SAO and Vocals) and APIs such as RSP4J and RSP Services. **(S5-S6)**: The seminal works like on TripleWave granted high interoperability between serving and querying. Moreover, the recent advancement in the Web stack with reactive protocols increases interoperability even further. **(S6-S0)**: While tools like the RSP Sevices allow the integration of provisioning protocols and query answering, they are not universally used to connect RSP systems (medium). However, every RSP engine respects the naming convention for sharing the data results. This is essential to guarantee the continuity of the life cycle.

For what concerns dependency between the steps, we navigate the lifecycle left-wise, e.g., from S6 to S0. **(S6-S5)**: Arguably, query answering does not directly depend on the protocols used to provide the data. Although the protocol choice may substantially impact performance, under assumption A1, the Web architecture strives for decentralization, not optimisation. Therefore, the steps remain loosely coupled. **(S5-S4)**: The description and serving steps are tightly coupled. In particular, S5 highly depends on S4 as shown by Barbieri et Della Valle [9] and by Fernandez et al [26]. Indeed, both works show how providing access-related metadata can enable data consumption. **(S4-S3)**: Although there is not a strong dependency between S3 and S4, as the annotation step remains optional, we remark that the medium interoperability level between annotation languages and tools and description vocabularies may change such a situation in the future. **(S3-S2)**: although the two steps are tightly-coupled, at the current stage we cannot observe a strong dependency of the annotation step on the shaping one. Indeed, the latter is often implicit, which makes it difficult to carry out such an analysis. **(S2-S1)**: as we previously discussed, the shaping step is currently focusing on the minimal transmittable unit. This is fairly dependent on the presence of a schema/model for the data streams, as they set the vocabulary to use in the definition of the RDF molecule. **(S1-S0)**: the modelling step depends on URI for implementing the unique name assumption. Thus, it is tightly coupled to S0. **(S0-S6)**: Finally, we conclude the cycle by highlighting that the query results should be named accordingly with the conventions. Therefore, the naming step depends on the query one when results are shared. This is possible using additional interfaces like RSP Services or with custom implementations.

8.4. Gaps in the lifecycle

Last but not least, the gap between serving and querying in Figure 10 is worth noticing. Figure 11 zooms into the life cycle to provide a better perspective on the grey area in the last two steps. Indeed, data stream *discovery* and *Access* are two intermediate steps that separate publication and querying. Despite their importance within the SLD life cycle, most of the research that has been conducted in these two areas focuses on the publisher's perspective (see *Describe* and *Serve* steps of the life cycle). In practice, data stream discovery and access are in their infancy and still require extensive

ad-hoc work for each use case. For these reasons, they are out of the scope of this paper, which aims at presenting only consolidated research work. On the one hand, the step following data querying remains undefined. Indeed, querying often leads to analyses powered by sophisticated data visualization. The literature on this topic is extensive, and other surveys have discussed best practices. Thus, they are also out of scope. On the other hand, publications' best practices like those by Auer et al. [3] stress on the "cyclic" aspect of the life cycle. Indeed, applications processing Streaming Linked Data shall also publish their analysis results. Such a vision appeared few times in the literature [65, 14, 62]. Stuckenschmidt et al. [62] propose, among other approaches, to push the expressivity level of SR using networks of stream reasoners. Calbimonte et al. [14] do a step toward such an idea by leveraging Linked Data Notification as a means for P2P communication across RSP engines. Finally, Tommasini et al. [65] lift from the notion of an actor to the one of an intelligent agent, closer to the original vision of the Semantic Web. However, none of such proposals has been currently integrated into a stream reasoning application.

We note that the new lifecycle results from a bottom-up investigation of the existing projects, exposing some of the limitations of the original lifecycle (which resulted from a top-down approach). The new lifecycle, therefore, better matches and represents the existing projects. Depending on the maturity of each step, the evidence is reflected in the various surveyed projects. For example, the new lifecycle changes the order of *Describe* and *Annotate* steps, for which we see evidence in the projects that have data on both steps. The introduction of the *Shap*ing step, which is low in maturity, is backed by projects that shape the data to speed up querying and processing, such as the OpenSense2 project. By surveying the projects and defining the new lifecycle based on the results, i.e. a bottom-up approach, the new lifecycle is grounded in reality yet opens opportunities for further research in the SLD/SR community by identifying the maturity of each step.

9. Conclusion and Future work

In this paper, we propose an updated version of the life cycle for SLD. A complete life cycle for SLD is necessary as the publication of dynamic data on the Web is rapidly growing. The need for guidelines on sustainably producing and efficiently consuming data streams is increasing. An initial version of such a life cycle has been proposed in previous research [66] and further updated in this paper. This update of the original life cycle has been proposed by surveying existing SLD applications and investigating how they aligned with the initial life cycle. Identifying that the initial life cycle did not fully capture the design of existing applications led to two crucial changes: the reordering of the stream *Description* step and splitting up the *Convert* step in a *Shape* and *Annotate* step. Next to the updates on the life cycle, this paper discussed each step in detail, explaining state-of-the-art solutions and best practices for each step. In particular, we drill down into the details of the querying step, which was neglected in the initial life cycle. Furthermore, we introduced a running example based on the DEBS Grand Challenge 2015 that is used to exemplify the best practices of each step in the life cycle. The life cycle and guidelines can serve as a blueprint for future SR/SLD applications. In this paper, we propose *Querying* as the last step of the lifecycle. However, the interest in processing steps beyond querying is growing, e.g. learning approaches [39]. In future work, we also wish to research the inclusion of these inductive processes in the lifecycle.

In conclusion, we encourage the SR community to further investigate the *Stream Discovery* and *Access* steps, which were left out in this survey, as research in these areas is still in its infancy. This will enable to fully realize our vision of an SLD life cycle. Adopting a life cycle for SLD brings us closer to a unified method to share and consume data streams on the Web and realize the Stream Reasoning vision.

Acknowledgments:

Pieter Bonte is funded by a postdoctoral fellowship of Research Foundation Flanders (FWO) (1266521N). Riccardo Tommasini acknowledges the support from the European Social Fund via IT Academy programme.

References

[1] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Zia Ush Shamszaman, Thu-Le Pham, Feng Gao, Keith Griffin, and Alessandra Mileo. Real-time data analytics and event detection for iot-enabled communication systems. *Journal of Web Semantics*, 42:19–37, 2017.

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2), 2006.

[3] Sören Auer, Lorenz Bühmann, Christian Dirschl, Orri Erling, Michael Hausenblas, Robert Isele, Jens Lehmann, Michael Martin, Pablo N Mendes, Bert van Nuffelen, et al. Managing the life-cycle of linked data with the lod2 stack. In *International semantic Web conference*, pages 1–16. Springer, 2012.

[4] Marco Balduini, Irene Celino, Daniele Dell'Aglio, Emanuele Della Valle, Yi Huang, Tony Lee, Seon-Ho Kim, and Volker Tresp. Bottari: An augmented reality mobile application to deliver personalized and location-based recommendations by continuous analysis of social media streams. *Journal of Web Semantics*, 16:33–41, 2012.

[5] Marco Balduini and Emanuele Della Valle. A restful interface for RDF stream processors. In *International Semantic Web Conference (Posters & Demos)*, volume 1035 of *CEUR Workshop Proceedings*, pages 209–212. CEUR-WS.org, 2013.

[6] Marco Balduini and Emanuele Della Valle. Frappe: A vocabulary to represent heterogeneous spatio-temporal data to support visual analytics. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 321–328. Springer, 2015.

[7] Marco Balduini, Emanuele Della Valle, Daniele Dell'Aglio, Mikalai Tsytsarau, Themis Palpanas, and Cristian Confalonieri. Social listening of city scale events using the streaming linked data framework. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013.

[8] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: a continuous query language for RDF data streams. *Int. J. Semantic Computing*, 4(1):3–25, 2010.

[9] Davide Francesco Barbieri and Emanuele Della Valle. A proposal for publishing data streams as linked data - A position paper. In *LDOW*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[10] Pieter Bonte, Riccardo Tommasini, Emanuele Della Valle, Filip De Turck, and Femke Ongenae. Streaming MASSIF: cascading reasoning for efficient processing of iot data streams. *Sensors*, 18(11):3832, 2018.

[11] Pieter Bonte, Riccardo Tommasini, Filip De Turck, Femke Ongenae, and Emanuele Della Valle. C-sprite: Efficient hierarchical reasoning for rapid RDF stream processing. In *DEBS*, pages 103–114. ACM, 2019.

[12] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

[13] John G Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. Sioc: an approach to connect web-based communities. *Intl J of Web Based Communities*, 2(2):133–142, 2006.

[14] Jean-Paul Calbimonte. Linked data notifications for rdf streams. In *Proc. of the Web Stream Processing (WSP) Workshop at ISWC*, pages 66–73, 2017.

[15] Jean-Paul Calbimonte, Óscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2010.

[16] Jean-Paul Calbimonte, Sofiane Sarni, Julien Eberle, and Karl Aberer. Xgsn: An open-source semantic sensing middleware for the web of things. In *TC/SSN@ ISWC*, pages 51–66, 2014.

[17] Michael Compton, Payam M. Barnaghi, Luis Bermudez, Raul Garcia-Castro, Óscar Corcho, Simon J. D. Cox, John Graybeal, Manfred Hauswirth, Cory A. Henson, Arthur Herzog, Vincent A. Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin R. Page, Alexandre Passant, Amit P. Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *J. Web Sem.*, 17:25–32, 2012.

[18] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. Semantics and validation of recursive SHACL. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, volume 11136 of *Lecture Notes in Computer Science*, pages 318–336. Springer, 2018.

[19] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2rml: RDB to RDF Mapping Language. W3c Recommendation, W3C, 2012.

[20] Brecht Van de Vyvere, Pieter Colpaert, and Ruben Verborgh. Comparing a polling and push-based approach for live open data interfaces. In Mária Bieliková, Tommi Mikkonen, and Cesare Pautasso, editors, *Web Engineering - 20th International Conference, ICWE 2020, Helsinki, Finland, June 9-12, 2020, Proceedings*, volume 12128 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2020.

[21] Daniele Dell'Aglio, Emanuele Della Valle , Frank van Harmelen, and Abraham Bernstein. Stream reasoning: A survey and outlook. *Data Sci.*, 1(1-2):59–83, 2017.

[22] Daniele Dell'Aglio, Emanuele Della Valle, Jean-Paul Calbimonte, and Óscar Corcho. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *Int. J. Semantic Web Inf. Syst.*, 10(4), 2014.

[23] Daniele Dell'Aglio, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. Stream reasoning: A survey and outlook. *Data Sci.*, 1(1-2):59–83, 2017.

[24] Anastasia Dimou, Miel Vander Sande, Jason Slepicka, Pedro A. Szekely, Erik Mannens, Craig A. Knoblock, and Rik Van de Walle. Mapping hierarchical sources into RDF using the RML mapping language. In *2014 IEEE International Conference on Semantic Computing, Newport Beach, CA, USA, June 16-18, 2014*, pages 151–158, 2014.

[25] Li Ding, Yun Peng, Paulo Pinheiro da Silva, Deborah L McGuinness, et al. Tracking rdf graph provenance using rdf molecules. *TR-CS-05-06*, 2005.

[26] Javier D. Fernández, Alejandro Llaves, and Óscar Corcho. Efficient RDF interchange (ERI) format for RDF data streams. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, volume 8797 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2014.

[27] Feng Gao, Muhammad Intizar Ali, and Alessandra Mileo. Semantic discovery and integration of urban data streams. In *Proceedings of the Fifth Workshop on Semantics for Smarter Cities a Workshop at the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 19, 2014.*, pages 15–30, 2014.

[28] Gerald Haesendonck, Wouter Maroy, Pieter Heyvaert, Ruben Verborgh, and Anastasia Dimou. Parallel rdf generation from heterogeneous big data. In *Proceedings of the International Workshop on Semantic Big Data*, pages 1–6, 2019.

[29] Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *European Semantic Web Conference*, pages 213–217. Springer, 2018.

[30] Bernadette Hyland and David Wood. The joy of data-a cookbook for publishing linked government data on the web. In *Linking government data*, pages 3–26. Springer, 2011.

[31] Andreas Kamilaris, Feng Gao, Francesc X. Prenafeta-Boldu, and Muhammad Intizar Ali. Agri-iot: A semantic framework for internet of things-enabled smart farming applications. In *3rd IEEE World Forum on Internet of Things, WF-IoT 2016, Reston, VA, USA, December 12-14, 2016*, pages 442–447. IEEE Computer Society, 2016.

[32] Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez-Ruiz, Davide Lanti, Hallstein Lie, Christoph Pinkel, Martin Rezk, Martin G Skjæveland, Evgenij Thorstensen, Guohui Xiao, et al. Ontology based access to exploration data at statoil. In *International Semantic Web Conference*, pages 93–112. Springer, 2015.

[33] Evgeny Kharlamov, Dag Hovland, Martin G. Skjæveland, Dimitris Bilidas, Ernesto Jiménez-Ruiz, Guohui Xiao, Ahmet Soylu, Davide Lanti, Martin Rezk, Dmitriy Zheleznyakov, Martin Giese, Hallstein Lie, Yannis E. Ioannidis, Yannis Kotidis, Manolis Koubarakis, and Arild Waaler. Ontology based data access in statoil. *J. Web Semant.*, 44:3–36, 2017.

[34] Evgeny Kharlamov, Yannis Kotidis, Theofilos Mailis, Christian Neuenstadt, Charalampos Nikolaou, Özgür Özcep, Christoforos Svingos, Dmitriy Zheleznyakov, Sebastian Brandt, Ian Horrocks, et al. Towards analytics aware ontology based access to static and streaming data. In *International Semantic Web Conference*, pages 344–362. Springer, 2016.

[35] Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, Özgür L. Özçep, Mikhail Roshchin, Nina Solomakhina, Ahmet Soylu, Christoforos Svingos, Sebastian Brandt, Martin Giese, Yannis E. Ioannidis, Steffen Lamparter, Ralf Möller, Yannis Kotidis, and Arild Waaler. Semantic access to streaming and static data at siemens. *J. Web Semant.*, 44:54–74, 2017.

[36] Sefki Kolozali, María Bermúdez-Edo, Daniel Puschmann, Frieder Ganz, and Payam M. Barnaghi. A knowledge-based approach for real-time iot data stream annotation and processing. In *2014 IEEE International Conference on Internet of Things, Taipei, Taiwan, September 1-3, 2014*, pages 215–222, 2014.

[37] Dwight Van Lancker, Pieter Colpaert, Harm Delva, Brecht Van de Vyvere, Julián Andrés Rojas Meléndez, Ruben Dedecker, Philippe Michiels, Raf Buyle, Annelies De Craene, and Ruben Verborgh. Publishing base registries as linked data event streams. In Marco Brambilla, Richard Chbeir, Flavius Frasincar, and Ioana Manolescu, editors, *Web Engineering - 21st International Conference, ICWE 2021, Biarritz, France, May 18-21, 2021, Proceedings*, volume 12706 of *Lecture Notes in Computer Science*, pages 28–36. Springer, 2021.

[38] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *International Semantic Web Conference*, pages 370–388. Springer, 2011.

[39] Danh Le-Phuoc, Thomas Eiter, and Anh Le-Tuan. A scalable reasoning and learning approach for neural-symbolic stream fusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4996–5005, 2021.

[40] Danh Le-Phuoc, Hoan Quoc Nguyen-Mau, Josiane Xavier Parreira, and Manfred Hauswirth. A middleware framework for scalable management of linked streams. *Journal of Web Semantics*, 16:42–51, 2012.

[41] Freddy Lécué, Simone Tallevi-Diotallevi, Jer Hayes, Robert Tucker, Veli Bicer, Marco Luca Sbodio, and Pierpaolo Tommasi. Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city. *J. Web Semant.*, 27-28:26–33, 2014.

[42] Jens Lehmann, Spiros Athanasiou, Andreas Both, Alejandra García-Rojas, Giorgos Giannopoulos, Daniel Hladky, Jon Jay Le Grange, Axel-Cyrille Ngonga Ngomo, Mohamed Ahmed Sherif, Claus Stadler, Matthias Wauer, Patrick Westphal, and Vadim Zaslawski. Managing geospatial linked data in the geoknow project. In Tom Narock and Peter Fox, editors, *The Semantic Web in Earth and Space Science. Current Status and Future Directions*, volume 20 of *Studies on the Semantic Web*, pages 51–78. IOS Press, 2015.

[43] Alessandro Margara, Jacopo Urbani, Frank van Harmelen, and Henri E. Bal. Streaming the web: Reasoning over dynamic data. *J. Web Sem.*, 25:24–44, 2014.

[44] Andrea Mauri, Jean-Paul Calbimonte, Daniele Dell'Aglio, Marco Balduini, Marco Brambilla, Emanuele Della Valle, and Karl Aberer. Triplewave: Spreading RDF streams on the web. In *ISWC*, 2016.

[45] Alessandra Mileo, Minh Dao-Tran, Thomas Eiter, and Michael Fink. Stream reasoning. 2017.

[46] Paul Murley, Zane Ma, Joshua Mason, Michael Bailey, and Amin Kharraz. Websocket adoption and the landscape of the real-time web. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1192–1203. ACM / IW3C2, 2021.

[47] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017.

[48] Axel-Cyrille Ngodnga Ngomo, Sören Auer, Jens Lehmann, and Amrapali Zaveri. Introduction to linked data and its lifecycle on the web. In Manolis Koubarakis, Giorgos B. Stamou, Giorgos Stoilos, Ian Horrocks, Phokion G. Kolaitis, Georg Lausen, and Gerhard Weikum, editors, *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, volume 8714 of *Lecture Notes in Computer Science*, pages 1–99. Springer, 2014.

[49] Dennis Pfisterer, Kay Romer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, et al. Spitfire: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, 2011.

[50] Danh Le Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 2013.

[51] Victoria Pimentel and Bradford G. Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Comput.*, 16(4):45–53, 2012.

[52] Dan Puiu, Payam M. Barnaghi, Ralf Toenjes, Daniel Kuemper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Xavier Parreira, Marten Fischer, Sefki Kolozali, Nazli FarajiDavar, Feng Gao, Thorben Iggena, Thu-Le Pham, Cosmin-Septimiu Nechifor, Daniel Puschmann, and João Fernandes. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4:1086–1108, 2016.

[53] Yves Raimond and Samer Abdallah. The event ontology, 2007.

[54] Xiangnan Ren and Olivier Curé. Strider: A hybrid adaptive distributed rdf stream processing engine. In *International Semantic Web Conference*, pages 559–576. Springer, 2017.

[55] Patrik Schneider, Daniel Alvarez-Coello, Anh Le-Tuan, Manh Nguyen Duc, and Danh Le Phuoc. Stream reasoning playground. In Paul Groth, Maria-Esther Vidal, Fabian M. Suchanek, Pedro A. Szekely, Pavan Kapanipathi, Catia Pesquita, Hala Skaf-Molli, and Minna Tamper, editors, *The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, volume 13261 of *Lecture Notes in Computer Science*, pages 406–424. Springer, 2022.

[56] Mario Scrocca, Riccardo Tommasini, Alessandro Margara, Emanuele Della Valle, and Sherif Sakr. The kaiju project: enabling event-driven observability. In Julien Gascon-Samson, Kaiwen Zhang, Khuzaima Daudjee, and Bettina Kemme, editors, *DEBS '20: The 14th ACM International Conference on Distributed and Event-based Systems, Montreal, Quebec, Canada, July 13-17, 2020*, pages 85–96. ACM, 2020.

[57] Juan F. Sequeda and Óscar Corcho. Linked stream data: A position paper. In *SSN*, volume 522 of *CEUR Workshop Proceedings*, pages 148–157. CEUR-WS.org, 2009.

[58] Wojciech Slodziak and Ziemowit Nowak. Performance analysis of web systems based on xmlhttprequest, server-sent events and websocket. In Adam Grzech, Leszek Borzemski, Jerzy Swiatek, and Zofia Wilimowska, editors, *Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology - ISAT 2015 - Part II, Karpacz, Poland, September 20-22, 2015*, volume 430 of *Advances in Intelligent Systems and Computing*, pages 71–83. Springer, 2015.

[59] John Soldatos, Nikos Kefalakis, Manfred Hauswirth, Martin Serrano, Jean-Paul Calbimonte, Mehdi Riahi, Karl Aberer, Prem Prakash Jayaraman, Arkady Zaslavsky, Ivana Podnar Žarko, et al. Openiot: Open source internet-of-things in the cloud. In *Interoperability and open-source solutions for the internet of things*, pages 13–25. Springer, 2015.

[60] Slawek Staworko, Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, and Harold R. Solbrig. Complexity and expressiveness of shex for RDF. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*, pages 195–211. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[61] Michael Stonebraker, Ugur Çetintemel, and Stanley B. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

[62] Heiner Stuckenschmidt, Stefano Ceri, Emanuele Della Valle, and Frank van Harmelen. Towards expressive stream reasoning. In Karl Aberer, Avigdor Gal, Manfred Hauswirth, Kai-Uwe Sattler, and Amit P. Sheth, editors, *Semantic Challenges in Sensor Networks, 24.01. - 29.01.2010*, volume 10042 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.

[63] Riccardo Tommasini. Velocity on the web - a phd symposium. In *WWW (Companion Volume)*, pages 56–60. ACM, 2019.

[64] Riccardo Tommasini, Pieter Bonte, Femke Ongenae, and Emanuele Della Valle. RSP4J: an API for RDF stream processing. In Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Óscar Corcho, Petar Ristoski, and Mehwish Alam, editors, *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 565–581. Springer, 2021.

[65] Riccardo Tommasini, Davide Calvaresi, and Jean-Paul Calbimonte. Stream reasoning agents: Blue sky ideas track. In *AAMAS*, pages 1664–1680. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[66] Riccardo Tommasini, Mohamed Ragab, Alessandro Falcetta, Emanuele Della Valle, and Sherif Sakr. A first step towards a streaming linked data life-cycle. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, pages 634–650. Springer, 2020.

[67] Riccardo Tommasini, Yehia Abo Sedira, Daniele Dell'Aglio, Marco Balduini, Muhammad Intizar Ali, Danh Le Phuoc, Emanuele Della Valle, and Jean-Paul Calbimonte. Vocals: Vocabulary and catalog of linked streams. In *International Semantic Web Conference (2)*, volume 11137 of *Lecture Notes in Computer Science*, pages 256–272. Springer, 2018.

[68] Boris Villazón-Terrazas, Luis M Vilches-Blázquez, Oscar Corcho, and Asunción Gómez-Pérez. Methodological guidelines for publishing government linked data. In *Linking government data*, pages 27–49. Springer, 2011.

[69] Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Datalogmtl: Computational complexity and expressive power. In *IJCAI*, pages 1886–1892. ijcai.org, 2019.

[70] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, 2016.