POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

ON THE CONTINUOUS AND REACTIVE ANALYSIS OF A VARIETY OF SPATIO-TEMPORAL DATA

Doctoral Dissertation of:
Marco Balduini

Supervisor:
Prof. Emanuele Della Valle

Tutor:
Prof. Stefano Ceri

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2018 – XXX Cycle

Abstract

In recent years, an increasing number of situations call for reactive decisions making process based on a heterogeneous streaming data. In this context, the urban environment results particularly relevant, there is a dense network of interactions between people and urban spaces that produces a great amount of spatio-temporal fast evolving data. Moreover, in a modern city there is a multitude of stakeholders who are interested in reactive decisions for urban planning, commuters, tourists, etc. The growing usage of location-based social networks, and, in general, the diffusion of mobile devices improved the ability to create an accurate and up-to-date representation of reality (a.k.a. Digital footprint or Digital reflection or Digital twin). When I started this thesis the state of the art was exploiting either social media or mobile phones data as a single source. However, better decisions can result from the analyses of multiple data sources simultaneously. Multiple heterogeneous data sources, and its simultaneous usage, offer a more accurate digital reflection of the reality. In this context, we investigate the problem of how to create an holistic conceptual model to represent multiple heterogeneous spatio-temporal data and how to develop a streaming computational model to enable reactive decisions. The main outcomes of this research are FraPPE conceptual model and RI≻EЯ streaming computational model with its implementations.

FraPPE is a conceptual model, more precisely an ontology, that exploits digital image processing terms to model spatio-temporal data and to enable space, time, and content analysis. It uses image processing common terms to bridge the gap between the data engineer perspective and visual data analysis perspective and to enable visual analytics on spatio-temporal data. During my PhD, we first formalize the spatial and temporal concepts in FraPPE 1.0, and then we add concepts related to the provenance and the content in FraPPE 2.0. We check the adherence of both versions of FraPPE to the five Tom Gruber's principles, and demonstrate the validity of the conceptual model in real world use cases.

RI≻EЯ is a streaming computational inspired by two principles: **(P1)** *everything is a data stream* – a variety-proof stream processing engine must indifferently ingest data with different velocities from any sources and of any size –, and **(P2)** *Continuous Ingestion* – the data in input is continuously captured by the system and, once arrived, it is marked with an increasing timestamp. Most of the stream processing engine in the

state of the art transforms and adapts data at ingestion time. Contrariwise, RI⋗EЯ is built around the idea of *Lazy Transformation*. So, a system that implements RI⋗EЯ postpones data transformations until it can really benefits from them, in order to save time and resources. RI⋗EЯ relies on two main concepts, the Generic Data Stream – S⟨T⟩ – and the Generic Time-Varying Collection – C⟨T⟩ – and it proposes five different operators in order to ingest, process and emit data. The IN⟨T⟩ operator is the door of the system, it takes an external data flow and inject the items into the system creating a new S⟨T⟩. The S2C⟨T⟩, C2C⟨T, T′⟩ and C2S⟨T⟩ in RI⋗EЯ, inspired to the CQL processing model, allows to move from S⟨T⟩ to C⟨T⟩ and vice-versa. The OUT⟨T⟩ operator transform an S⟨T⟩ into a new external data flow. Exploiting the Pipeline Definition Language (PDL) – a graphical language to abstract the operators' implementation complexity –, RI⋗EЯ allows users to define computational plans, in the form of pipelines.

In this thesis, we propose different implementations of RI⋗EЯ: Natron – a single-threaded vertically scalable implementation –, rvr@Spark and rvr@Hive – two horizontally scalable implementations based on distributed technologies (Spark and Hive). In order to prove the validity of the *Lazy Transformation* approach, we first evaluate Natron against our Streaming Linked Data (SLD) engine that performs the data transformation at ingestion time. The result of this evaluation shows that Natron is cheaper – it consumes less resources in terms of memory and CPU load – and better approximates the correct answer under stress conditions. Moreover, we evaluate the cost effectiveness of Natron against rvr@Spark to prove that a distributed solution does not pay in all the situations. The results of those evaluations demonstrate the validity of the concepts that underpin RI⋗EЯ streaming computational model.

In order to prove the feasibility and the effectiveness of FraPPE and RI⋗EЯ in enabling reactive decision-making processes on heterogeneous streaming spatio-temporal data, we present five real world use cases in Milan and Como. Moreover, during those case studies, we propose the data visualizations to different audiences (public users and stakeholders) in order to prove the guessability of such visual interfaces.

Finally, we reflect on limitations and state the future directions of this research work. In particular, those reflections involve the reasoning capabilities enabled by FraPPE, the future evaluations of RI⋗EЯ and the evolution of the Pipeline Definition Language (PDL).

Summary

SUMMARY goes here.

Contents

CHAPTER 1

Introduction

FINAL - TO BE READ ONE LAST TIME

1.1 Relevancy

In an increasing number of situations, a decision must be reactive[1] and must be based on a variety of streaming data. In the electricity management domain, a reactive anomaly detection system, which processes the consumption data, must react in *seconds* to avoid network problems. In oil and gas extraction sites, the analyses of sensors' readings from the wells have at most *minutes* for the reactive detection of dangerous situations.

The urban environment is particularly relevant when talking about reactive decisions. In modern cities, a dense network of interactions between people and urban spaces produces a great amount of spatio-temporal fast evolving data [1] and a multitude of stakeholders are interested in reactive decisions. Tourists would value information about the *current* top rated and less crowded attractions around the city [2]. Commuters would like to know the *current* busiest roads to choose the fastest way home [3]. Public safety agencies would like to learn about over-crowded area *during* a public event.

In the mid 2000s, the growing use of location-based social networks via mobile devices, improved the ability to capture the people's interests, habits, and preferences in a privacy-preserving manner and enabled innovative scenarios. It became possible to create an accurate and up-to-date representation of reality (a.k.a. Digital footprint or Digital reflection or Digital twin) exploiting either social media or mobile phones data, i.e. Call Data Records (CDR). For instance, analyzing social media Cho et al. [4] were able to identify mobility patterns, while we built a location-based recommendation engine for restaurant in Korea [5]. Parallel works exploited CDR to create models to estimate the

[1] Deciding an action in response to a stimulus before new incoming information makes the planned action ineffective.

density of crowds and vehicles [6–8].

However, better decisions can result from the analyses of multiple data sources simultaneously. The growing availability of new urban data sources (e.g., IoT and WI-FI logs) stimulated the research of a holistic conceptual model to manage data variety in a comprehensive way. The current interest is for solutions that fuse streaming heterogeneous data to enable reactive decisions.

1.2 Problem Statement and Research Question

In the first years of the 2010s the interest of the Semantic Web community for the heterogeneous streaming urban data was growing [4,9]. We started investigating the modeling and the analysis of streaming data from social media [5, 10] exploiting Stream Reasoning [11] and state-of-the-art techniques based on RDF Stream Processing (RSP) [12], named entity recognition and linking, and machine learning [10, 13].

Reflecting on those results, we identified two main findings from previous research: (i) when dealing with data streams, a Continuous Ingestion mechanism avoids data losses, but continuous analysis is not always needed; an analysis can be reactive even if postponed. (ii) Ontologies are an adequate knowledge representation technique for modeling data characterized by high variety. However Stream Reasoning researchers count on two Assumptions:

A ontologies (adequate to model a domain) are available or they can be obtained with minimal effort by extending existing ones. For instance, SMA [2], an ontology that we created to represent location-based social media data, was defined starting from SIOC[2] by adding only few axioms.

B Data streams can be RDF-ized at a negligible cost. For instance, in our previous works we used social media streams. Social media APIs return statuses in JSON that can be easily transformed in JSON-LD[3] exploiting standard format, such as Activity Stream[4].

Aiming at continuously and reactively analyzing a variety of spatio-temporal data, we develop the research question with the Macro, Mezzo and Micro method [14].

At Macro level we focus on relevancy and formulate the question: *Is it possible to support reactive decisions by managing data characterized by velocity and variety without forgetting volume?*

At Mezzo level, we focus the attention on a question for which we can find a viable solution. We concentrate our effort on spatio-temporal streaming data and, focusing on previous findings, we characterize the way to support reactive decisions, i.e. visually making sense of data. So, the Mezzo level question is: *Is it possible to visually making sense of a variety of spatio-temporal streaming data by enabling continuous ingestion and reactive analysis?*

Finally, at Micro level, we formalize a question that can be evaluated. We concentrate our effort on the streaming urban data and we specify a way to exploit the visual analytics instrument to support reactive decision making, i.e. find emerging patterns and data

[2] http://sioc-project.org
[3] https://json-ld.org
[4] http://activitystrea.ms

dynamics. As a result, the research question of this PhD thesis is: *Is it possible to continuously ingest and reactively analyses a variety of streaming urban data in order to visualize emerging patterns and their dynamics?*

Within the scope of this research question, the assumptions A does not hold. In our previous work, we focused on social media data. Social media data is semi-structured: only time and space information is presented in a structured way; the content is unstructured, e.g. free texts or images. On the contrary in this work we aim at integrate IoT data, WI-FI logs, CDRs, which are structured. While the integration of semi-structured data is generally based on the content analysis (e.g. named entity recognition and linking), the integration of structured data requires other methods, e.g., Ontology Based Data Integration (OBDI) [15]. So, a first problem emerges:

Rp.1 Defining a conceptual model to represent a variety of streaming data.

Moreover, also Assumption B holds only for social media data. Therefore, we need to face two problems :

Rp.2 Defining a streaming computational model to enable analysis on a variety of data.

Rp.3 Defining appropriate technical instantiations of the computational model in Rp.2.

Last, but not least, to verify and validate the solutions proposed to solve the problems above, we need to:

Rp.4 Assess, in real world scenarios, the feasibility and the effectiveness of the instantiations developed addressing Rp.3 using the models developed in solving Rp.1 and Rp.2.

The three levels of the research question are strictly correlated to the research problems. In fact, they aim at probing the validity (Rp.4) of the conceptual model (Rp.1), of the computational model for streaming heterogeneous data (Rp.2) and of its technical instantiations (Rp.3).

In answering to the Micro level question, we are directly contributing to answer the Mezzo level question, and, indirectly, to cast some light on the Macro level question.

1.3 Approach

Inspired by OBDI methods, we approach the research problems in a modular way by relaxing, in parallel, the two original assumptions presented in Section 1.2. This modularity reflects the research problem structure and allows performing a continuous evaluation.

On the one hand, relaxing Assumption A, we create a conceptual model in the form of an ontology by following the METHONTOLOGY [16] methodology and evaluate the result using Tom Gruber's principles [17] (See Chapter 4).

On the other hand, relaxing Assumption B, we develop a computational model that enables continuous ingestion, wrangling and reactive analysis of heterogeneous data streams. We implement such a computational model using different technologies, i.e. single-threaded and distributed, in order to prove its adequacy in different work conditions (see Chapter 5). We, then, evaluate those implementations against already existing system (SLD [5]) and one against the other (see Section 6.3). In particular, inspired by

COST [18], we evaluated the cost-effectiveness of the single-threaded system against the distributed one (see Section 6.4).

We, finally, put at work a complete system, composed by an implementation of the computational model that exploits the conceptual model, in different scenario (see Chapter 7).

1.4 Outline

The thesis is structured as follows:

- Chapter 2 offers an overview on the relevant background concepts used by the Semantic Web community to tame velocity, variety and both of them in a single system. It, then, defines the basic concepts of RSP Middleware systems and offers an overview of the benchmarking principles.
- Chapter 3 offers an overview on the urban data analysis by setting the main characteristics of urban data, by reviewing the state of the art, and ,finally, by offering significant examples of application of RDF stream processing in the field.
- Chapter 4 introduces FraPPE ontology, the conceptual model we proposed to tame with the problem Rp.1. The chapter presents the motivation, the genesis and a first evaluation of the original FraPPE 1.0. Moreover, it offers an overview of the FraPPE 2.0 extension while casting some light on how the proposed conceptual model helps the user in spotting emerging patterns and understanding data dynamics in the urban data analysis field.
- Chapter 5 introduces RI≻EЯ the streaming computational model we proposed to face the research problems Rp.2 and Rp.3. In this chapter we present an overview of the principles at the basis of RI≻EЯ, its genesis, its internals, its implementations and different evaluations based on performance and cost-effectiveness metrics.
- Chapter 7 presents real world use-cases where we put at work FraPPE and the different implementations of RI≻EЯ in order to verify the solution to the research problem Rp.4 and, consequently, validate the solutions proposed for the problems Rp.1, Rp.2, Rp.3.
- Chapter 8 concludes the dissertation with an overall review of the contributions and with a discussion of the future work based on the limits of the actual solutions.

1.5 Publications

This thesis is based on the articles [19–26]

- Fabrizio Antonelli, Matteo Azzi, Marco Balduini, Paolo Ciuccarelli, Emanuele Della Valle, Roberto Larcher: "City sensing: visualising mobile and social data about a city scale event". AVI 2014: 337-338
- Marco Balduini, Emanuele Della Valle: "FraPPE: A Vocabulary to Represent Heterogeneous Spatio-temporal Data to Support Visual Analytics". International Semantic Web Conference (2) 2015: 321-328

- Emanuele Della Valle, Marco Balduini: "Listening to and Visualising the Pulse of Our Cities Using Social Media and Call Data Records". BIS (Workshops) 2015: 3-14
- Marco Balduini, Emanuele Della Valle, Riccardo Tommasini: "SLD Revolution: A Cheaper, Faster yet More Accurate Streaming Linked Data Framework". ESWC (Satellite Events) 2017: 263-279
- Marco Balduini, Emanuele Della Valle, Matteo Azzi, Roberto Larcher, Fabrizio Antonelli, Paolo Ciuccarelli: "CitySensing: Fusing City Data for Visual Storytelling". IEEE MultiMedia 22(3): 44-53 (2015)
- Marco Balduini, Sivam Pasupathipillai, Emanuele Della Valle: "Cost-Aware Streaming Data Analysis: Distributed vs Single-Thread". DEBS 2018: 160-170
- Marco Balduini, Marco Brambilla, Emanuele Della Valle, Christian Marazzi, Tahereh Arabghalizi, Behnam Rahdari, and Michele Vescovi: "Models and practices in urban data science at scale". Big Data Research, 2018: In Press
- Marco Balduini: "On the continuous and reactive analysis of a variety of Spatio-Temporal Data". ISWC-DC 2018

CHAPTER 2

Preliminary Concepts: Taming Velocity and Variety

FINAL - TO BE READ ONE LAST TIME

The transient nature of streaming data often requires to treat it differently from persistent data, which can be stored and queried on demand. Data streams should often be consumed on the fly by continuous queries. Such a paradigmatic change has been largely investigated in the last decade by the database community [27], and, more recently, by the Semantic Web community [11]. Several independent groups have proposed extension of RDF and SPARQL [28] for continuous querying [29–31] and reasoning [12,32].

The development of these solutions needs to deal with the nature of data streams and with the user needs. The input information always changes over time *(Velocity)*, the sources are different and offer data that vary in syntax, structure and semantics *(Variety)*. The data continuously flows into the system and, even what looks like *static data*, e.g. a city street grid, is not immutable over time. It slowly evolves.

In the next sections, we present an overview on (i) the theoretical and practical concepts developed for managing data characterized by velocity and variety, (ii) the solutions to easy the creation of system based on RDF Stream Processors, and (iii) the benchmarking principles and techniques. In the Section 2.1, we present the theoretical models, the languages and the architectures for taming velocity, and we conclude with an overview of the extended solutions we exploited during the research work. Section 2.2 presents an overview on the concepts and the architectures for managing the variety. In Section 2.3, we present an overview on the attempts for taming both velocity and variety in a single architecture. In Section 2.4, we present the fundamental principles of an RSP middleware and the implementations we exploited as term of comparison during this work. Finally, Section 2.5 presents the benchmarking principles and the benchmarking techniques and metrics we exploited during this research work.

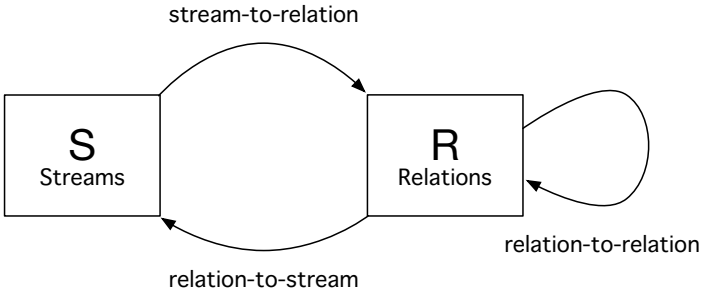


Figure 2.1: *The CQL processing model.*

2.1 Velocity

Section 2.1.1 presents an overview of the models and languages to manage the data velocity. Sections 2.1.2 and 2.1.3 then present two different architecture for a velocity-first system, and the solutions that inspired our research work.

2.1.1 Models and Languages

Stream Processing

In order to introduce the concepts related to the analysis of streaming data, we need to introduce the concepts of Time and Time Instant.

Definition 2.1.1. *(Time) The time $\mathcal{T}$ is an infinite, discrete, ordered sequence of time instants $(\tau_1, \tau_2, ..., \tau_n)$, where $\tau_i \in \mathbb{N}$. A time unit is the difference between two consecutive time instants $(\tau_{i+1} - \tau_i)$ and it is constant.*

Definition 2.1.2. *(Time Instant) A time instant (or simply instant) is any value from T.*

Now that we have formally defined the Time concept, we can introduce the Stream.

Definition 2.1.3. *(Stream) A stream S is a bag (multiset) of elements $\langle s, \tau \rangle$, where s is a tuple belonging to the schema of S and $\tau \in \mathcal{T}$ is the timestamp of the element.*

This definition of stream does not consider the time as a part of the data schema. At any time instant $\tau \in \mathcal{T}$, zero or more elements could be in the stream and, consequently, share the timestamp.

We finally present a formal definition of Relation, an unordered bag of tuples at any time instant $\tau \in \mathcal{T}$, or simply $R(\tau)$.

Definition 2.1.4. *(Relation) A relation R is a mapping from $\mathcal{T}$ to a finite but unbounded bag of tuples belonging to the schema of R.*

CQL

The Continuous Query Language(CQL) [33] represents both an expressive SQL-based declarative language for managing continuous queries against streams and updatable relations, and a processing model. It was originally proposed by the DB group of the Stanford university.

Figure 2.1 depicts the three operators defined by the CQL processing model.

(i) The stream-to-relation. It takes a stream S as input and produces a relation R as output, maintaining the schema. At any instant τ, $R(\tau)$ should be computable from S.

(ii) The relation-to-relation operator. It takes one or more relations $R_1, ..., R_n$ as input and produces a relation R as output. At any instant τ, $R(\tau)$ should be computable from $R_1(\tau), ..., R_n(\tau)$.

(iii) The relation-to-stream operator. It takes a relation R as input and produces a stream S as output maintaining the schema. At any instant τ, S at τ should be computable from R up to τ.

CQL, also, define an abstract semantics for the data management:

Definition 2.1.5. *(Continuous semantics) Consider a query Q that is a composition of the three basic CQL operators. The inputs to the operators operators of Q are streams* $S_1, ..., S_n (n \geq 1)$ *and relations* $R_1, ..., R_m (m \geq 0)$*. The result of continuous query Q at a time* τ *when all inputs are "available" can be defined as:*

1. *If the top operator in Q is relation-to-stream and produces the stream S, the result of Q at time* τ *is S up to* τ*, produced by recursively applying the operators comprising Q to streams* $S_1, ..., S_n$ *up to* τ *and relations* $R_1, ..., R_m$ *up to* τ*.*

2. *If the top operator in Q is stream-to-relation or relation-to-relation and produces the relation R, the result of Q at time* τ *is* $R(\tau)$*, produced by recursively applying the operators comprising Q to streams* $S_1, ..., S_n$ *up to* τ *and relations* $R_1, ..., R_m$ *up to* τ*.*

Stream-to-relation operators The stream-to-relation operators in CQL are based on the sliding window concept (see Section 2.3.1). CQL exploits the concepts of window to define three classes of sliding window: time-based, tuple-based and partitioned.

Definition 2.1.6. *(Window) A window* $W(S)$ *is a set of elements extracted from a stream S.*

Time-based sliding window operator's output is defined by sliding an interval of T time units over the stream S.

Definition 2.1.7. *(Time-based sliding window) A time-based sliding window on a stream S takes a time-interval* Ti *as a parameter and is specified by following S in the query with [Range* Ti*]. The output relation R of S[Range* Ti*] is defined as:*

$$R(\tau) = \{s \mid \langle s, \tau' \rangle \in S \wedge (\tau' \leq \tau) \wedge (\tau' \geq \max\{\tau - \mathrm{Ti}, 0\})\}$$

Tuple-based sliding window operator's output is defined by sliding a window of size N tuples over the stream S.

Definition 2.1.8. *(Tuple-based sliding window) A tuple-based sliding window takes a positive integer N as a parameter and is specified by following S in the query with [Rows N]. The relation R of S[Rows N],* $R(\tau)$*, consists of tuples obtained from the N elements with the largest timestamps in S no greater than* τ*.*

Partitioned sliding window logically partitions S into different sub-streams based on equality of attributes $A_1, ..., A_k$, computes a tuple-based sliding window of size N on each sub-stream, then the output relation is the union of these sub-windows.

Definition 2.1.9. *(Partitioned sliding window) A partitioned sliding window on a stream S takes a positive integer N and a subset $\{A_1, ..., A_k\}$ of S attributes as parameters. It is specified by following S in the query with [Partition By $A_1, ..., A_k$ Rows N]. Formally, a tuple s with values $a_1, ..., a_k$ for attributes $A_1, ..., A_k$ occurs in output instantaneous relation $R(\tau)$ iff exists an element $\langle s, \tau' \rangle \in S$ such that $\tau' \leq \tau$ is among the N largest timestamps among elements whose tuples have values $a_1, ..., a_k$ for attributes $A_1, ..., A_k$*

Relation-to-relation operators The relation-to-relation operators operators transform relations in other relations. They are often derived from typical relational queries, by applying the semantic mapping to time-varying relations. Relational algebraic expressions are a well-known cases of this class of operators.

Relation-to-stream operators Starting from the concepts of stream and relation, CQL defines three classes of relation-to-stream operators: Istream, Dstream, and Rstream.

Definition 2.1.10. *(Istream) The insert stream applied to relation R contains an element $\langle s, \tau \rangle$ iff the tuple s is in $R(\tau) - R(\tau - 1)$:*

$$Istream(R) = \bigcup_{\tau \geq 0}((R(\tau) - R(\tau - 1)) \times \{\tau\}).$$

Definition 2.1.11. *(Dstream) The delete stream applied to relation R contains an element $\langle s, \tau \rangle$ iff the tuple s is in $R(\tau - 1) - R(\tau)$:*

$$Dstream(R) = \bigcup_{\tau \geq 0}((R(\tau - 1) - R(\tau)) \times \{\tau\}).$$

Definition 2.1.12. *(Rstream) The relation stream applied to relation R contains an element $\langle s, \tau \rangle$ iff the tuple s is in R at time τ:*

$$Rstream(R) = \bigcup_{\tau \geq 0}(R(\tau) \times \{\tau\}).$$

The concepts introduced by CQL represent a fundamental theoretical base for the development of the stream processors, see Section 2.3.2. We exploited these constructs during the development of our Streaming Computational Model, see Chapter 5.

SECRET

In 2000s, different systems try to implement a streaming processing model (see Section 2.3.2). Despite they are based on common concepts, they present significant differences in the way they manage data and queries. In order to explain the differences in the behavior of window operators in the existing stream processing engines, Botan et el. present SECRET [34]. Differently from CQL, it assigns two time instants to each stream item: (i) the application and (ii) the system time. The former, already defined by CQL processing model, refers to the instant related to the event represented by the

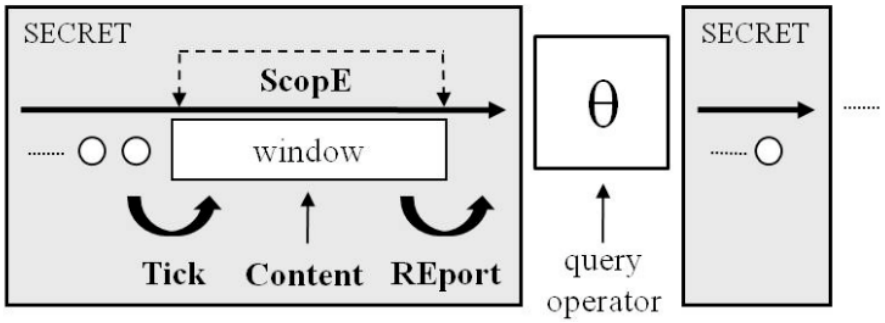


Figure 2.2: *SECRET of a query plan (source [34]).*

element in the stream. It is not unique (contemporaneity is allowed) and defines a partial order among the stream elements. The latter must be unique and introduces a total order in the stream. From a conceptual point of view, the application time represents the most relevant information, but the system time is also important to understand the correct behavior of the stream engine.

As depicted in Figure 2.2, the SECRET framework introduces the notions of scope, content, report and tick to explain the window operator.

The *Scope* function associates an evaluation time instant t to the active window time interval. The computation of the scope relies on the t_0 parameter, the first active window start timestamp.

The *Content* identifies the set of items of S in the active window. This function is influenced by both the application and the system time.

The *Report* function defines the conditions under which the relation-to-relation operators can access the window content for additional query evaluation and result reporting. SECRET identifies four reporting strategies: (i) Content change – the system reports if the content changes –, (ii) Window close – the system reports if the active window closes –, (iii) Non-empty content – the system reports if the active window is not empty – and, finally, (iv) Periodic – the system reports only at regular intervals.

The *Tick* is a function that defines under which conditions input can enter the window and, consequently, can be processed by the engine. SECRET defines tuple-driven and time-driven strategies. Systems that adopt the former strategy add the tuple to the window operator as soon as they arrive, contrariwise, systems that adopt the latter strategy, add tuple to the window at each application time instant.

The key concepts formalized by SECRETS result useful to guarantee a comparable behavior of all the different implementations of our Conceptual Model presented in Chapter 5.

2.1.2 Information Flow Processing and Architectures

Cugola et al. in [35] proposed the Information Flow Processing (IFP) as an application domain in which users need to collect information produced by multiple, distributed sources for processing it in a timely way in order to extract new knowledge as soon as the relevant information is collected.

From an high-level point of view, an IFP takes data flows from multiple sources as input, processes them and produces other information flows as output. This output is

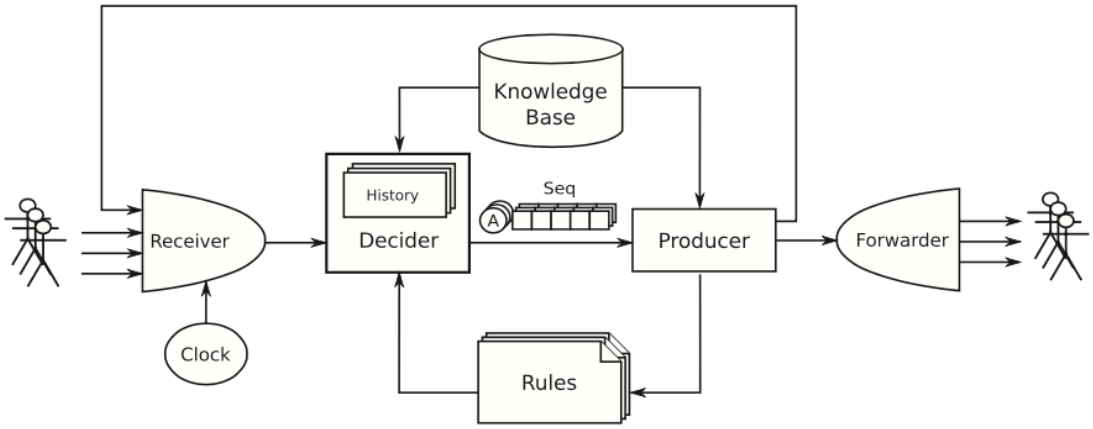


Figure 2.3: *The functional architecture of an IFP system (source [35]).*

then directed toward a set of sinks.

Figure 2.3 shows the components of a generic IFP system. The *receiver* implements the transport protocol to move data over the network and manages the connection between the sources and the IFP engine. Moreover, it is also connected to the *clock* – an architectural element that produces special data items that hold the current time. Then, the data items, from external sources or from the clock, enter the processing pipeline that elaborates the data according to the rules stored into the rule store. A rule is logically composed by a condition part (C) and an action part (A). C specifies the condition that has to be satisfied by the information to trigger the rule in the IFP, while A specifies what to do. The logical disjunction produces a physical disjunction, the processing are splitted in two different phases: (i) the detection – realized by the *decider* that checks the condition (C) on each incoming item –, and (ii) the production – realized by the *producer* that triggers the actions (A). The *knowledge base* represents a read only-memory[1] that contains useful information for the decider and the producer. Finally, the *forwarder* is in charge to deliver the information to the output sinks.

In the early 2010s, Nathan Marz coined the term λ architecture describing a generic, scalable and fault-tolerant data processing architecture that was very successful in distributed environment. The IFP functional model are at the basis of this architecture, formalizes by Marz et al. in [36].

In a system implementing a λ architecture (see Figure 2.4), a separation between the batch processing pipeline (a.k.a. Batch Layer) and the real-time processing pipeline (a.k.a. Real-time Layer) is easily identifiable. This clear separation helps to isolate and localize the complexity of data update. The Serving Layer offers a mechanism to combine Batch Layer results with Real-time Layer results in order to offer the latest information to the user. The three layers are depicted in Figure 2.4. The Batch Layer is in charge of storing the immutable, constantly growing master dataset and of computing views from the stored dataset. The computation of the views is a periodic operation, the new data is aggregated once arrived and the views are incrementally computed on the entire dataset every time. The Batch Layer operations could take hours to be completed, depending on the size of the cluster and of the data. The Speed Layer compensates the

[1]The knowledge base is read-only from the IFP engine perspective, but can be modified by external systems

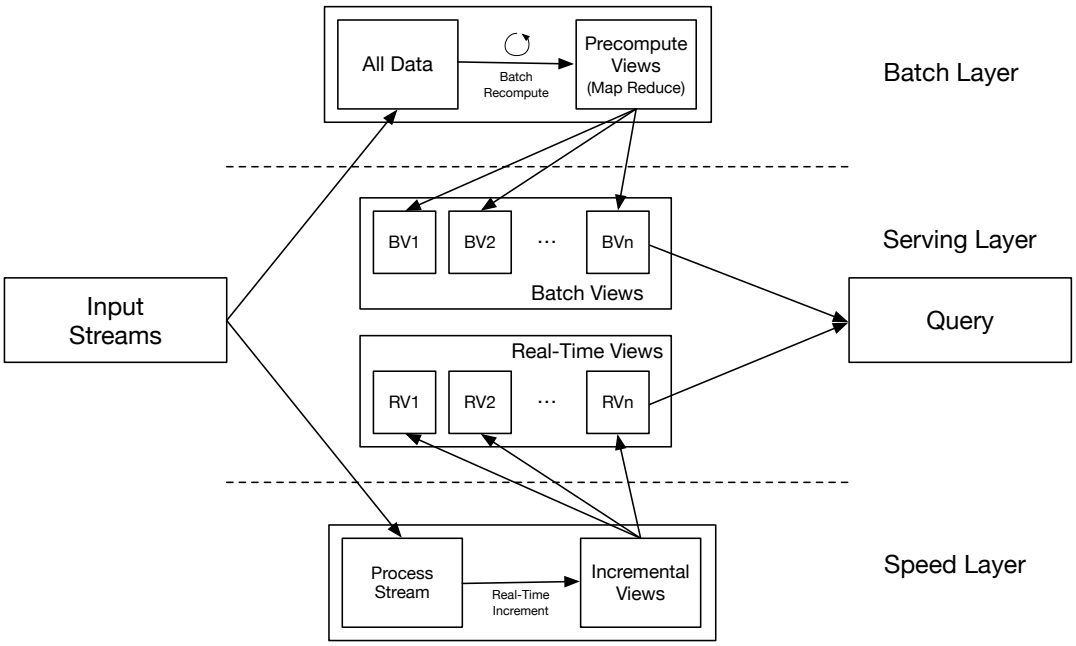


Figure 2.4: *The λ architecture.*

high latency of the Batch Layer. It, normally, computes real-time views on the most fresh data. The views, computed by the Speed Layer, contain only the delta results to supplement the ones computed by the Batch Layer. The Speed Layer continuously computes real-time views. That views are transient, once the information propagates through the Batch and Serving Layers the corresponding results in the real-time views lost its validity. The Serving Layer is responsible for merging, indexing and exposing the views in order to make them available for query operations.

In recent years the complexity and the maintenance cost of λ architectures were criticized[2] and Jay Kreps proposed the κ architecture[3], a stream-only architecture.

Figure 2.5 depicts the main components of the κ architecture, that aims at exploiting a single stream processing engine to handle real-time data processing and continuous reprocessing. In this architecture the Speed Layer are in charge of computing real-time views on the most fresh data from the Input streams. The Real-Time views are incrementally computed when new data enters the system. The most recent views are available to the Serving Layer for querying operation. In a system based on a κ architecture, the canonical data store is an append-only immutable log.

Both κ and λ architectures inspired our system presented in Section 2.1.3 and our Computational Model presented in Chapter 5.

2.1.3 Open Source Solutions

In this section, we present open-source solutions for taming velocity. All the proposed systems represent implementations of a λ or a κ Architecture. We choose to work

[2] `https://www.oreilly.com/ideas/questioning-the-lambda-architecture`
[3] `http://milinda.pathirage.org/kappa-architecture.com/`

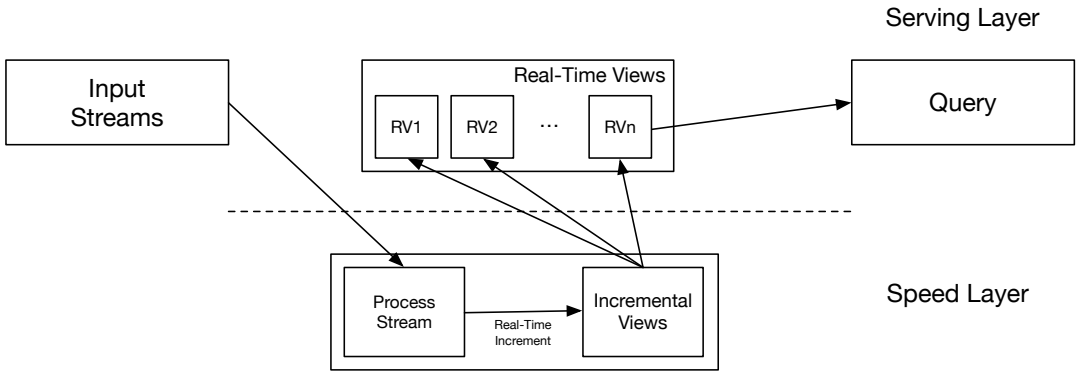


Figure 2.5: *The κ architecture.*

with open-source software in order to explore and modify the internals to cope with our needs. The section proposes one vertical scalable system (ESPER) and four different horizontally scalable systems with different characteristics (Kafka, Spark, Hive and Flink). The proposed solutions inspired our streaming Computational Model proposed in the Chapter 5.

ESPER & EPL

Esper[4] is an open-source system for complex event processing (CEP) and streaming analytics. Esper exploits in-memory processing to address the requirements of applications that analyze high volume of fast data (between 1,000 to 100k messages per second in input), and must promptly react to events (from a few milliseconds to a few seconds of latency) by applying complex computations (e.g., pattern detection, filter, aggregation, etc.)

The Event Processing Language (EPL) offers SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses and is compliant to the SQL-92 standard. In the EPL logic, streams replace tables as primary source of data and, instead of rows, events become the basic information unit. As for rows in the relational environment, events are composed by data. EPL allows the definition of windows over stream of data to define a subset of the data to be analyzed. Such windows can be time-based or event-based (see Section 2.1.1) and can be combined applying intersection or union operators. Moreover, EPL provides the concept of named window, a data windows that can be used in multiple statements via the FROM clause, in a join or in a sub-query.

Together, Esper and EPL, provide a powerful and extendable environment for stream processing and implements the basic concepts of a κ Architecture (see Section 2.1.2). Esper is a key components of the C-SPARQL Engine (see Section 2.3.2).

Apache Kafka & KSQL

Apache Kafka [37] is a distributed message broker with stream processing capabilities. Kafka organizes data into *topics*. Each topic is made up of one or several *partitions*. Each partition is assigned to a node in the Kafka cluster.

[4] `http://www.espertech.com/esper/`

The Kafka APIs are based on the *producer* and *consumer* components. The producer is responsible for transferring data from an external source to a Kafka cluster. Conversely, the consumer is responsible for reading data from a Kafka cluster and sending it to an external sink. By instantiating and using these components, an application can integrate Kafka as its storage solution. Kafka is designed to enable high-throughput applications, and it supports at-least-, at-most-, and exactly-once message delivery.

KSQL[5] is a streaming SQL engine for real-time data processing against Apache Kafka. KSQL offers SQL-like language and ensures scalability and fault-tolerancy while enabling commons streaming operations (e.g. window, filter, aggregations, etc).

A system based on Kafka and KSQL enables all the commons operators described in CQL (see Section 2.1.1) and represents an implementation of a λ Architecture (see Section 2.1.2). Kafka is a key component of the infrastructure we used to test our work, see Section 6.4

Apache Spark

Apache Spark [38] is a distributed processing engine which improves the Apache Hadoop [39] cluster computing paradigm for processing massive amounts of data in parallel. The main advantage over Apache Hadoop is that intermediate results can be stored into main memory, thus reducing disk I/O operations.

Spark environment consists of several components, which communicate with each other via the network. The highest level components are the *master* and the *workers*. The master is responsible for coordinating the execution of a Spark application and presenting its results. The workers are responsible for managing the execution of the distributed application code. There can be more than one worker, and each physical machine can host several workers. Both the master and the workers are implemented as separate processes running in the JVM.

Apache Spark is based on the Resilient Distributed Dataset (RDD) abstraction. An RDD represents an immutable dataset distributed over a cluster of machines. Each fragment of the dataset is termed a *partition*. A Spark application consists of a sequence of transformations on a collection of RDDs. During execution, these transformations run in parallel on each partition. When an aggregated result is needed, e.g. COUNT after GROUP BY, Spark performs a shuffle operation by transferring partitions over the network between workers. Each worker spawns several subprocesses known as *executors*. Executors run the distributed application code. The atomic unit of parallel execution is called a *task*. At runtime each task is assigned to an executor.

Spark Streaming [40] is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant and real-time processing of data. It offers adapters for various data sources (e.g. Kafka, Flume, etc.). The key abstraction behind Spark Streaming is the DStream (Discretized Stream), a potentially infinite flow of small batches. DStream are built on RDDs. Spark Streaming represents an attempt to enable streaming, interactive, and batch queries in a single engine that supports: (i) continuous aggregations, (ii) windowing operations, (iii) stateful stream aggregations, and (iv) stream watermark operations.

Structured Streaming [41] is a new declarative streaming API available starting from Apache Spark 2.0 to support continuous applications. It is a higher-level API than the

[5] `https://www.confluent.io/product/ksql/`

one offered by Spark Streaming and it is integrated into Dataset and DataFrame API. Structured Streaming treats all the input data as an unbounded input table, each new items is appended once arrived. The queries see the input as a static table and the system compute the results incrementally. Structured Streaming represents streams as DataFrames or Datasets with the isStreaming property set to true, therefore, the creation of an application with both stream and batch operations results very simple. A developer has just to describe the query at higher level, with few information about input, output and other details, and the system runs the query incrementally supporting consistency and recovery operations.

We exploited Spark Structured Streaming to create a distributed implementation of our streaming Computational Model presented in the Chapter 5.

Apache Hive

Apache Hive [42] is an open-source data ware-housing solution built on top of Apache Hadoop [39]. It offers a SQL-like declarative language, namely HiveQL, that supports the insertion of custom map-reduce scripts directly into queries.

Hive data is organized into: (i) the Database – the counterpart of the relational databases –, (ii) the Table – an abstraction of the classic relational tables, it corresponds to an HDFS directory that contains the serialized data –, (iii) the Partition – it represents the organization of the data in the sub-directories tree –, (iv) the Bucket – a division of the data within a partition, a single bucket is represented as a single file. Hive has a limited support to streaming data. Hive Streaming API allows a system to continuously ingest information in small batches into an existing Hive partition or table. Once the flowing information is committed, it becomes immediately available to all Hive queries.

HiveQL supports primitive data-types within a table, but the underlying IO libraries can be extended to access data in custom formats. Hive includes a system catalog, the Hive-Metastore. It contains schemas and statistics to be used during the data exploration phases. HiveQL support a simple window operator[6] that can partially simulate the CQL window operator (see Section 2.1.1).

Hive implements typical batch operator and offers a minimal support for the streaming operations (i.e. window operator). We exploited Hive windows to implement a distributed version of Our streaming Computational Model presented in the Chapter 5.

Apache Flink

Apache Flink [43] is a distributed platform for streaming data (DataStream API) and batch data (DataSet API). The dataflow engine, the core of the platform, guarantees the fault tolerance during the distributed computations. Flink's provides an event-at-a-time processing model throw a dataflow of streams and transformations. The DataStream API enables classic stream processing operations (e.g., filters, aggregations, window functions) on bounded or unbounded streams of data, the DataSet API enables transformations (e.g. filters, mapping, joining, grouping) only on finite dataset.

Flink offers a relational abstraction of the data, the Table, that can be created from external data sources or from existing DataStreams and DataSets. The table can be accessed via: (i) the Tables API, a SQL-like expression language that supports relational

[6] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+WindowingAndAnalytics

operators (e.g., selection, aggregation, joins); and via (ii) regular SQL. Both of the access methods offer equivalent functionalities and can be mixed in the programming flow.

Flink can natively manage both stream and batch of data. A system based on it, represents an implementation of a λ architecture (see Section 2.1.3).

2.2 Variety

In the next sections, we present an overview of the Models, Languages and Methodologies to tame the data variety. After an overview on the main concepts behind RDF, SPARQL and R2RML mapping language, we go through the ontological world. Gruber in [44] define an ontology as "an explicit specification of a conceptualization". A conceptualization is defined as "the objects, the concepts and other entities that are assumed to exist in some area of interest and relationships that hole among them" [45]. An ontology, with its capability to offer a common representation of heterogeneous data, offers a good starting point in facing the variety problem. The following sections present OWL, an overview of two ontology engineer methodologies (METHONTOLOGY and NEON) and OBDI as an example of integration methods based on ontologies. Finally, we present the open source solutions that we exploited during our research work.

2.2.1 Models, Languages and Methodologies

RDF & SPARQL

The Resource Description Framework (RDF) is a W3C standard for data interchange on the Web [46]. The main data structure in RDF is the directed labeled graph, made of nodes, which represent the resources and of edges which represent the relations between them. The atomic RDF element is the triple that consists of subject, predicate, and object. We identify with I, B and L respectively the sets of IRIs, blank nodes, and literals. We define an RDF term as an element of the set $I \cup B \cup L$.

Definition 2.2.1. *(RDF statement and RDF graph). An RDF statement is a triple* $(s, p, o) \in (I \cup B) \times (I) \times (I \cup B \cup L)$*, while a set of RDF statements is called an RDF graph, which is a directed, labeled graph that represents Web resources.*

The SPARQL Protocol and RDF Query Language (SPARQL) [47] is W3C Recommendation and enables data retrieve, data manipulation and query operation on data stored in RDF format. A SPARQL query typically contains one or more triple patterns called a basic graph pattern, it is similar to an RDF triple except for the possible presence of variables instead of resources.

Definition 2.2.2. *(Triple pattern and Basic Graph Pattern). A triple pattern tp is a triple* (sp, pp, op) *such that*

$$(sp, pp, op) \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V),$$

where V is the infinite set of variables. A basic graph pattern is a set of triple patterns. Graph patterns in a SPARQL query can include basic graph patterns and other compound expressions defined recursively as:

1. A set of triple patterns is a basic graph pattern;

2. *If P_1 and P_2 are graph patterns, then (P_1 AND P_2), (P_1 OPT P_2) and (P_1 UNION P_2) are graph patterns;*
3. *If P is a graph pattern and u is a symbol in $I \cup V$, (GRAPH u P) and (SERVICE u P) are graph patterns;*
4. *If P is a graph pattern and R is a SPARQL built-in condition, then (P FILTER R) is a graph pattern.*

A SPARQL built-in condition consists of the elements of the set $(I \cup L \cup V)$ and constants, logical connectives $(\neg, \vee, \wedge)$, the binary equality symbol $(=)$, ordering symbols $(<, \leq, \geq, >)$, and unary predicates such as $bound$, $isBlank$, $isIRI$.

To introduce the evaluation semantics of a SPARQL query, we define solution mapping as detailed in [47, 48].

Definition 2.2.3. *(Solution mappings). A solution mapping μ is a partial function $\mu : V \rightarrow I \cup B \cup L$. It maps a set of variables to a set of RDF terms. A mapping has a domain $dom(\mu)$ which is the subset of V over which it is defined. We denote as $\mu(x)$ the RDF term resulting by applying the solution mapping to variable x. We denote as ω a multiset of solution mappings, and as ψ a sequence of solution mappings. Typical relational algebraic operators can be applied to multiset of solution mappings:*

$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge \mu_1 \sim \mu_2\}$
$\Omega_1 \cup \Omega_2 = \{\mu | \mu_1 \in \Omega_1 \vee \mu_2 \in \Omega_2\}$
$\Omega_1 \backslash \Omega_2 = \{\mu | \mu \in \Omega_1 \wedge \nexists \mu_1 \in \Omega_2 : \mu \sim \mu_1\}$
$\Omega_1 \leftthreetimes \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \backslash \Omega_2)$

RDF datasets is a collections of one or more RDF graphs and represents the format of input data.

Definition 2.2.4. *(RDF dataset). An RDF dataset DS is a set:*

$$DS = G0, (u1, G1), (u2, G2), ...(un, Gn)$$

where G0 and Gi are RDF graphs, and each corresponding ui is a distinct IRI. G0 is called the default graph, while the others are called named graphs. During the evaluation of a query, the graph from the dataset used for matching the graph pattern is called active graph. Multiple graphs can become active during the evaluation, but only one at time.

SPARQL defines four query forms: SELECT – which produces a result of variable bindings matching the graph pattern –, CONSTRUCT, which produces a new RDF graph with the query solutions –, ASK – which produces a boolean value that is true if at least a solution exists –, and DESCRIBE – which produces an RDF description of resources in the solution. A query can also contain solution modifiers (e.g., LIMIT, DISTINCT, ORDER BY) that are applied after pattern matching. A SPARQL query [48] can be defined as:

Definition 2.2.5. *(SPARQL Query). A SPARQL query is defined as a tuple (E, DS, QF), where E is a SPARQL algebraic expression, DS is an RDF dataset, and QF is a query form.*

A query solution is a bag of solution mappings that assign RDF triples to variables of the query. The evaluation semantics of a SPARQL query algebraic expression w.r.t.

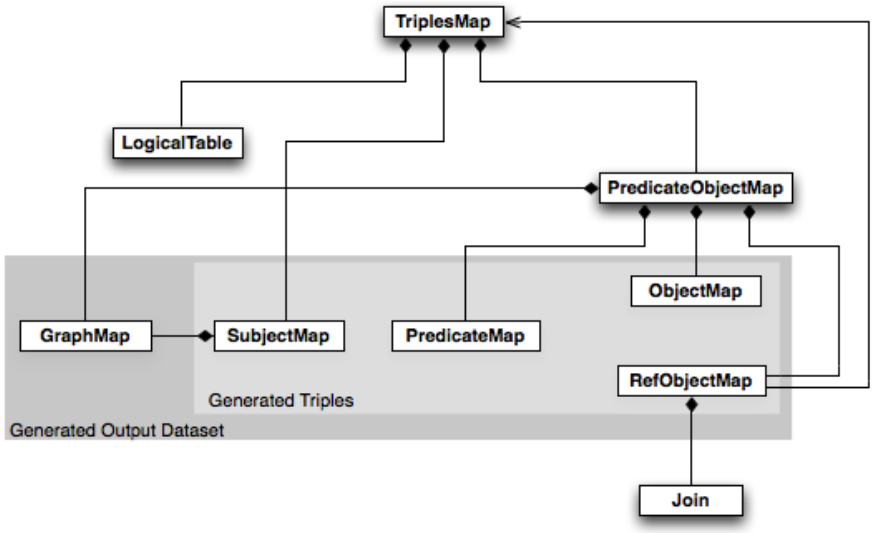


Figure 2.6: *R2RML mapping process (source: https://www.w3.org/TR/r2rml).*

an RDF dataset is defined for every operator of the algebra, and it is expressed through an evaluation function.

Definition 2.2.6. *(SPARQL evaluation semantics). The SPARQL evaluation semantics of an algebraic expression E is denoted as ED(G), where DS(G) is the dataset DS with active graph G.*

RDF and SPARQL represent a fundamental concepts in the data variety management, both in static and streaming environment (see Section 2.3).

R2RML

The RDB to RDF mapping language[7] (R2RML) is a language to create custom mappings to transform relational data into RDF. R2RML mappings file presents itself as an RDF graphs and, unlike Direct Mapping[8] (DM), allows user to define highly customized views over relational data sources. The R2RML conceptual mapping is tailored to a specific database schema and target vocabulary (i.e., the input database must conform to the presented schema) and produces an RDF dataset (see Section 2.2.1) that conforms to the target vocabulary. The R2RML processors can work as a virtual access layer to the relational data or can materialize the output data.

Figure 2.6 depicts the overview of the mapping process. R2RML mapping accesses to the relational input data through logical tables. A logical table can be a base table in the input database, a view or a valid SQL query (R2RML view). Each logical table is then mapped to RDF using a triple map, a set of rules that allows the system to transform each row in the logical table into one or more RDF triples. The rules are composed by a subject map that creates the subject for all the triples generated by a single row and by multiple predicate-object maps that consist of predicate maps and object maps.

[7] http://www.w3.org/TR/r2rml/
[8] https://www.w3.org/TR/rdb-direct-mapping/

OWL

The Web Ontology Language (OWL)[9] is a W3C Semantic Web language designed for creating ontologies. OWL is part of the W3C's Semantic Web technology stack. It is a computational logic-based declarative language characterized by formal semantics. The knowledge expressed in OWL can be reasoned to verify the consistency of that knowledge or to make implicit knowledge explicit. The fundamental notions exploited by OWL to represent knowledge are: (i) the Axioms – the basic statement of an ontology –, (ii) the Entities – the representation of a real world object –, and (iii) the Expressions – a combination of entities to describe a complex object.

The three concepts presented above allow OWL to create a human-like knowledge representation with the concept of consequence. When a statement is a consequence of another statement, it is true whenever the other statements are. In OWL, a set of statements *S* entails a statement *s* if in any state of affairs wherein all statements from *S* are true, also *s* is true. A set of statements may be consistent (there is a possible state in which all the statements in the set are jointly true) or inconsistent (there is no such state). The formal semantics of OWL specifies for which condition a particular set of OWL statements is true.

The W3C-endorsed OWL specification includes the definition of three variants of OWL: OWL Lite, OWL DL and OWL Full, presented in order of expressiveness. OWL2[10] represents the latest specification of the Web Ontology Language. It is dated to 2009 and introduces three additional profiles: OWL2EL, a fragment with polynomial time reasoning complexity; OWL2QL, a language to enable easier access and query to data stored in databases; and OWL2RL, a rule subset of OWL2.

The Conceptual Model presented in the Chapter 4 is formalized with OWL2.

Ontology Engineering

We report a briefly overview of METHONTOLOGY, an ontology development methodology we exploited in the definition of the Conceptual Model presented in Chapter 4, and NeOn, an evolution of METHONTOLOGY, that supports the reuse of already available ontologies.

METHONTOLOGY [16] is a methodology for creating ontologies from scratch, by reusing other ontologies or by re-engineering them. The framework enables the construction of ontologies at the "knowledge level".

It includes: (i) the formal identification of the development process and its phases (i.e., scheduling, control, quality assurance, specification, knowledge acquisition, conceptualization, integration, formalization, implementation, evaluation, maintenance, documentation and configuration management); (ii) an evolving prototypes based lifecycle (see Figure 2.7) that identifies the phases the ontology passes during its lifetime and the interdependencies with the lifecycle of the connected ontologies; (iii) the specific techniques to perform each activity, the output of each phases and the evaluation methods.

The **NeOn** [49] methodology encourage the reuse of other ontologies as well as of non-ontological resources during the engineering process. The framework provides

[9] https://www.w3.org/TR/owl-features/
[10] https://www.w3.org/TR/owl2-overview/

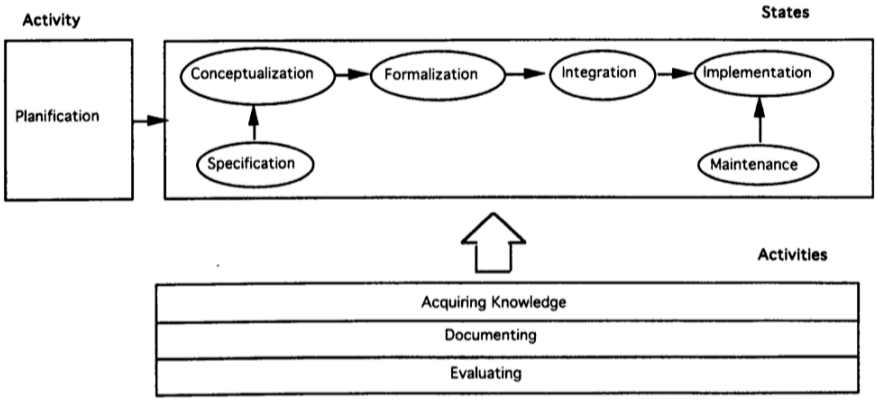


Figure 2.7: *Ontology lifecycle (source [16]).*

strong guidelines for the execution of the development activities (e.g., the usage of ontology design patterns).

OBDI & OBDA

The data integration problem consists of combining data from heterogeneous sources, and offering the user a unified view of the information. Due to the growing number of data sources (e.g., smartphones, sensors, etc), the problem of designing data integration systems are becoming more and more important in the real-world. Moreover, companies are shifting from centralized and self-contained way to manage their data (i.e., a central database or a data warehouse) to a distributed, interactive and multi-source world where the information becomes the exchange good. The volume and the heterogeneity of the data are constantly growing and companies are now focusing on finding the right information. Most of the available sources is characterized by syntactic, structural and semantic heterogeneities.

In a generic data integration system [15], the sources contain the real data, while a global schema offers an integrated and reconciled overview of such sources. Modeling the relation (namely, the mapping) between the sources and the global schema represents an important aspects of the data integration process. The data integration issue have been faced following three different approaches: (i) the Global-As-View (GAV) approach – the most used form of mappings, where the data schema is expressed in terms of the data sources –, (ii) the Local-As-View (LAV) approach – where the data schema is specified independently from the sources –, and a hybrid approach (GLAV) – the most general form of mapping. Independently from the approach, in a data integration system, the query process requires a reformulation step where the query over the global schema is reformulated in terms of a set of queries over the sources.

In several domains, such as Enterprise Application Integration, in the Semantic Web and, in particular, in the data integration [15], ontologies are considered as the ideal formal tool to provide a common conceptualization of the domain, and Description Logics (DLs) are widely considered appropriate for expressing ontologies. DLs are also at the basis of the OWL Language (see Section 2.2.1).

Consequently, an Ontology-Based Data Integration (OBDI) system consists of three

main components: (i) an ontology – the formal description of the considered domain –, (ii) a set of data sources – the repositories where data are stored –, and (iii) the mapping – a specification of the correspondences between the data and the ontology elements. Such system allows users to access the data using the ontology components as a predicates. Differently from the traditional data integration, OBDI system offers a semantically rich description of the relevant concepts in the domain of interest and of the relationships between such concepts, in addition to a separation between the conceptual level (presented to the client through the ontology), and the logical/physical level of the information system (the one stored in the sources).

Intuitively, in the simplest technique for enabling query answering, the system first retrieves the concepts and the roles instances from the data sources through the mapping and then, exploiting the ontology axioms, it "expands" such a set of (stated and inferred) instances deriving and materializing all the logically entailed concepts and roles assertions. In this way, the queries can be evaluated on the complete set of instances. Unfortunately, the set of entailed instance assertions may be infinite and, consequently, such a techniques is not feasible.

Since 2000s, Ontology-Based Data Access (OBDA) [50] has become a popular approach to enable users to access data sources through an ontology. Moreover, a GAV approach based on a data model expressed in OWL 2 QL (based on the DL-Lite family of description logics), ensures the effectiveness of query answering operation with LOGSPACE complexity (more precisely, AC^0).

In a classic OBDA framework, query rewriting starts from the computation of the perfect rewriting in order to enable the query evaluation on the data sources. The creation of the rewriting can be modularized into a first phase of of query rewriting related to the ontology and in a second phase of query rewriting related to the mapping.

During our research work we exploited OBDI and OBDA techniques in many applications (see Chapter 7) using our Conceptual Model presented in Chapter 4 as the ontology to mediate between the user queries and the data.

2.2.2 Open Source Solutions

In this section, we present the solutions we used over the years in our research work. As usual in this research, we have privileged open source solution in order to access and exploits the internals.

We start from Jena and Sesame, frameworks to manage RDF data and to build Semantic Web applications. **Apache Jena**[11] is written in Java. The main data structure in Jena is the Model, an abstract representation of an RDF graph. A model can be accessed and queried via SPARQL and its source can be external file, databases, URLs or a combination of them. Jena offers different components to support the creation of application, it provides the TDB, an RDF database that supports SPARQL 1.1 query, and Fuseki, a database server that support the access via the standard SPARQL protocols[12]. Jena is a core component of the C-SPARQL Engine (see Section 2.3.2).

Sesame [51] is an open-source framework for storing, querying and analyzing RDF data. It also offers support for RDFS inferencing and querying. Sesame implements an in-memory/on-disk triplestore and offers a Servlet packages to manage and access

[11] https://jena.apache.org/

[12] http://www.w3.org/TR/sparql11-protocol/

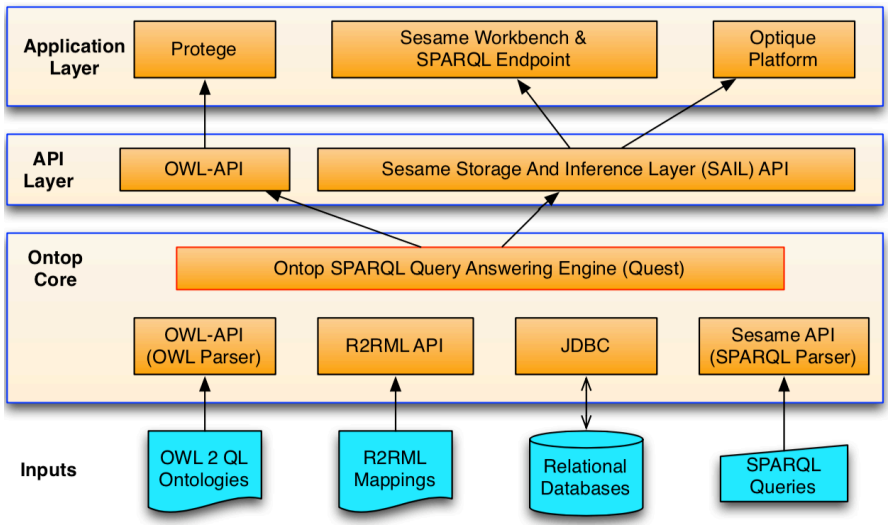


Figure 2.8: *Architecture of the Ontop system (Source [56]).*

the stored data on a permanent server. Sesame supports concurrency control, export of RDF and RDFS information and a query engine for RQL, SPARQL and SeRQL. In May 2016, Sesame officially forked into an Eclipse project called RDF4J[13] that provides functionality for efficient and scalable storage, querying, and reasoning with RDF data, and a vendor-neutral access API for RDF databases.

To explore the basic reasoning techniques we worked with Hermit and RDFox. **HermiT** [52] is a Description Logic reasoner based on a "hypertableau" calculus techniques, an innovative and efficient reasoning algorithm. It also incorporates the "anywhere blocking" strategy, which limits the sizes of constructed models. HermiT uses direct semantics and is conformant to all OWL 2 tests for direct semantics reasoners. **RDFox** [53] is a main-memory, scalable, centralized datalog engine. It supports materialization-based parallel reasoning, SPARQL query answering and implements Backward/Forward algorithm proposed in [54]. The innovative intuition is the combination of backward and forward reasoning to limit the recomputation performed by the traditional DRed [55].

As OBDA framework, we used **OnTop** [56], an open-source Ontology-Based Data Access (OBDA) system that offers a solid theoretical base, a virtual OBDA approach implemented with query rewriting technique, its compliance to W3C recommendations and its support for all major relational databases.

Figure 2.8 presents the four levels of a system based on Ontop: (i) the input, such as queries, database, ontology and mapping; (ii) the Ontop core needed to rewrite SPARQL queries SQL queries; (iii) the API for accessing system services; and (iv) the applications for the end-user to execute SPQARL queries over relational data.

[13] `http://rdf4j.org`

2.3 Velocity and Variety

In the next sections, we present the solution to manage the data variety in a streaming fashion. We start from the formal definition of the concepts behind Stream Reasoning and RDF Stream Processing (RSP). We present RSP-QL, an attempt to unify the different RSP query languages developed over the years by the RSP community and, finally, we present an overview of the currently available solutions.

2.3.1 Models and Languages

Stream Reasoning & RDF Stream Processing

In 2009, Della Valle et al. [11] proposed to start to investigate on how to represent, manage, and reason on heterogeneous continuously flowing data in the presence of expressive domain models.

In a scenario, where an ontology offers a conceptual view over autonomous data sources, a reasoner can play a key role in finding answers that are not syntactically present in the data sources, but are derivable from the data and the ontology (see Section 2.2.1 on OBDI and OBDA). RDF (see Section 2.2.1) represent the dominant data model in the field of reasoning for data integration and RDF streaming languages (see Section 2.3.2) bridge the gap between stream processing and OBDI. Reasoning in a streaming fashion looks conceptually simple, but it is hard to be efficiently performed. Della Valle et al. [57] showed how continuous DL reasoning can be reduced to periodic repetition of reasoning over a windowed ontology stream. Barbieri et al. [58] presented an optimization of DRed algorithm when deletion becomes predictable. In parallel Komazec et al. [59] implemented an extension of the RETE algorithm, Ren at al. [60] approached the problem via truth maintenance systems and Motik et al. [54] represents the state of the art in this research field. Still along this DL-centric line, more recently, Calbimonte et al. [31] exploited OBDA principles by rewriting continuous ontological queries to a stream processing system. In parallel to this line, at the beginning of 2000s Heintz et al. [61] proposes a first approach to middleware for knowledge processing and in [62] an implementation of this approach. More recently, De Leng et al. [63] proposed logic-based spatio-temporal stream reasoning focusing on run-time verification to guarantee the safety of autonomous systems and Anicic et al. [64] developed a system that processes in logic programming both stream reasoning and complex event processing.

RDF Stream Processing (RSP) represents the sub-area of stream reasoning that concentrates on the Semantic Web [65].

In this section, we introduce the concept of RDF data item and, consequently, Timestamped RDF data item. The RDF data item represents the minimal informative unit in the RDF stream. It is a generic concept and the existing RSP implementations (see Section 2.3.2) consider it in two different ways: RDF statements and RDF graphs, as defined in Section 2.2.1. In the most intuitive case a RDF stream is composed of RDF statements [57]. Due to the limited amount of information carried by a single statement, in order to ease the task of model real world use case, in 2010 Barbieri et al. [66] proposes to use the RDF graphs. In 2013 we were the first to implement this concept in [5].

The definition of a Timestamped RDF data item can be formalized as:

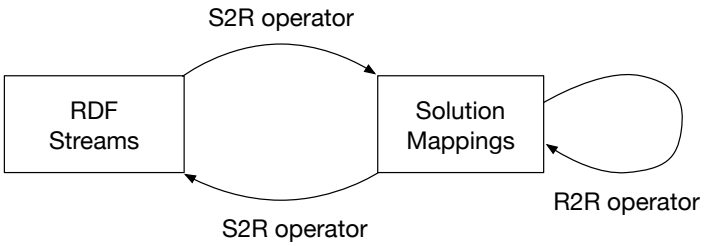


Figure 2.9: *The CQL processing model adapted for RSP.*

Definition 2.3.1. *(Timestamped RDF data item) A timestamped RDF data items is a pair (d,τ), where d is an RDF graph and $\tau \in \mathcal{T}$ is a time instant.*

Once defined the possible nature of the RDF data item, we can now define a RDF stream.

Definition 2.3.2. *(RDF stream) An RDF stream S is an unbounded sequence of timestamped RDF data items in non-decreasing time order:*

$$S = ((d_1, \tau_1), (d_2, \tau_2), ..., (d_{n-1}, \tau_{n-1}), (d_n, \tau_n), ...)$$

where, for every $i > 0$, d_i is a timestamped RDF item and $\tau_i <= \tau_{i+1}$.

A time-varying RDF graph capture the evolution of graph content over time, contrariwise, instantaneous graph represents the content of the graph at a fixed time instant

Definition 2.3.3. *(Time-varying Graph) A time-varying graph G is a function that relates time instants $\tau \in \mathcal{T}$ to RDF graphs:*

$$G : \mathcal{T} \rightarrow g | g \, is \, an \, RDF \, graph$$

Definition 2.3.4. *An instantaneous RDF graph G(t) is the RDF graph identified by the time-varying graph G at the given time instant t.*

The RDF stream processing model is depicted in Figure 2.9. It is directly derived from the CQL one (see Section 2.1.1). The stream and the relation concepts are mapped to RDF streams and to set of mappings (using the SPARQL algebra terminology), respectively. To highlight the similarity of the RSP operators [67] to the CQL ones, the same names (**S2R**, **R2R** and **R2S**) are used to indicate the stream-to-relation, relation-to-relation and relation-to-stream operators.

The concepts described in this section are all implemented in our Streaming Linked Data (SLD) framework (see Section 2.4.1) and inspired the definition of our generic streaming Computational Model presented in Chapter 5.

RSP-QL

RDF Stream Processor Query Language (RSP-QL) [67] is an extension of SPARQL created to unify existing RSP query languages (i.e, C-SPARQL [12], CQELS [30] and $\text{SPARQL}_{\text{stream}}$ [64]). RSP-QL enables user to register continuous queries. The queries

are registered once over streams of data and continuously evaluated. Due to the continuous evaluation semantic, a query produces multiple results over time and the instantaneous answer is a composition of the results of each iteration. RSP-QL is designed following two main requirements: (i) every evaluation of a query over input data produces a unique solution, (ii) RSP-QL, inspired by SECRET(see Section 2.1.1), captures the operational semantics of C-SPARQL engine, CQELS and $\text{Morph}_{\text{stream}}$ (see Section 2.3.2). RSP-QL implements the basic concepts of the extensions of SPARQL concepts presented in the Section 2.2.1. We start presenting the core concept of the RSP-QL language, the RSP-QL query.

Definition 2.3.5. *(RSP-QL query) RSP-QL query Q is defined by the tuple (SE,SDS,ET,QF) where*

- *SE is an RSP-QL algebraic expression*
- *SDS is an RSP-QL dataset*
- *ET is the sequence of time instants on which the evaluation occurs*
- *QF is the Query Form*

The presence of the time dimension calls for a new notion of RDF dataset, the input data of the RSP-QL query.

Definition 2.3.6. *(RSP-QL dataset) An RSP-QL dataset SDS is a set composed by an (optional) default graph, n ($n \geq 0$) named graphs and m ($m \geq 0$) named time-varying graphs obtained by the application of time-based sliding windows over $o \leq m$ streams:*

$$\begin{aligned} SDS = \{&G_0, (u_1, G_1), ..., (u_n, G_n), \\ &(w_1, W_1(S_1)), ..., (w_j, W_j(S_1)), \\ &(w_{j+1}, W_{j+1}(S_2)), ..., (w_k, W_k(S_2)), ..., \\ &(w_l, W_l(S_o)), ..., (w_m, W_m(S_o))\} \end{aligned}$$

with:

- G_0 *is the default time-varying graph*
- u_p, w_q *are IRIs* ($u_p, w_q \in \mathbb{I}$) *for each* $p \in [1, n]$ *and* $q \in [1, m]$
- (u_p, G_p) *identifies a time-varying named graph, for each* $p \in [1, n]$
- $(w_q, W_q(S_r))$ *identifies a named time-based sliding window over an RDF stream, for each* $q \in [1, m]$ *and* $r \in [1, o]$

In a continuous environment a definition o a time-varying sequence of solution mappings is needed.

Definition 2.3.7. *(Time-varying sequence of solution mappings) A time-varying sequence of solution mappings ψ maps time instants $\tau \in \mathcal{T}$ to the set of solution mapping sequences:*

$$\psi : \mathcal{T} \rightarrow \{\Psi | \psi \; is \, a \, sequence \, of \; solution \, mappings\}$$

The RSP-QL evaluation semantic is an evolution of SPARQL evaluation semantic defined taking into account the time dimension.

Definition 2.3.8. *(RSP-QL evaluation semantic) Given an RSP-QL dataset SDS, an algebraic expression SE and an evaluation time instant τ, we define*

$$eval(SDS(G), SE, \tau)$$

as the evaluation of SE at time τ with respect to the RSP-QL dataset SDS having active time-varying graph G.

2.3.2 Open Source Solutions

In the next sections, we present the principal solutions proposed by the Semantic Web community for taming velocity for heterogeneous data. In particular, we present vertical RDF Stream Processors developed before the definition of RSP-QL with their own query languages and a couple of distributed implementations. The proposed solution inspired the streaming Computational Model presented in the Chapter 5.

C-SPARQL Engine Continuous SPARQL (C-SPARQL) [12] is a language to express continuous queries over flowing data in RDF format. It extends SPARQL 1.1 by defining Data Stream Processing operators, including the possibility of defining window over streams of data. The C-SPARQL engine[14] is an open-source software that exploits the C-SPARQL language to enable the registration of continuous queries to be executed over RDF streams. Its core is based on two sub-components: Esper (see Section 2.1.3) and Jena (see Section 2.2.2) . The former is responsible of executing continuous operations on the stream, e.g. sliding window to produce RDF graph. The latter periodically executes standard SPARQL query on the RDF stream fragments to produce continuous results.

CQELS CQELS-QL [30] is a declarative query language that extends SPARQL 1.1 grammar with operators to deal with streaming data. Continuous Query Evaluation over Linked Streams (CQELS) interpret queries in CQELS-QL and, differently from C-SPARQL, supports only the Istream relation-to-stream operator (see Section 2.1.1). CQELS offers a flexible framework for the query operations with a dynamic adapting processors that continuously reorders operators to improve query execution in terms of delay and complexity. It natively implements the query operators in order to limit the overhead and the limitations related to the usage of other engines (e.g. C-SPARQL engine relies on Esper and Jena).

Morph$_{\text{stream}}$ SPARQL$_{\text{stream}}$ [31] extends SPARQL to support all the stream operators (e.g. sliding window). SPARQL$_{\text{stream}}$ is implemented in Morph$_{\text{stream}}$ engine that exploits Ontology-Based Data Access techniques [68]. Morph$_{\text{stream}}$ execution is based on R2RML[15] mappings between ontologies and data streams. The queries are first rewritten in a relational algebra expression with time window extension and then translated in the Data Stream Processing target language.

[14] `http://streamreasoning.org/resources/c-sparql`
[15] `https://www.w3.org/TR/r2rml/`

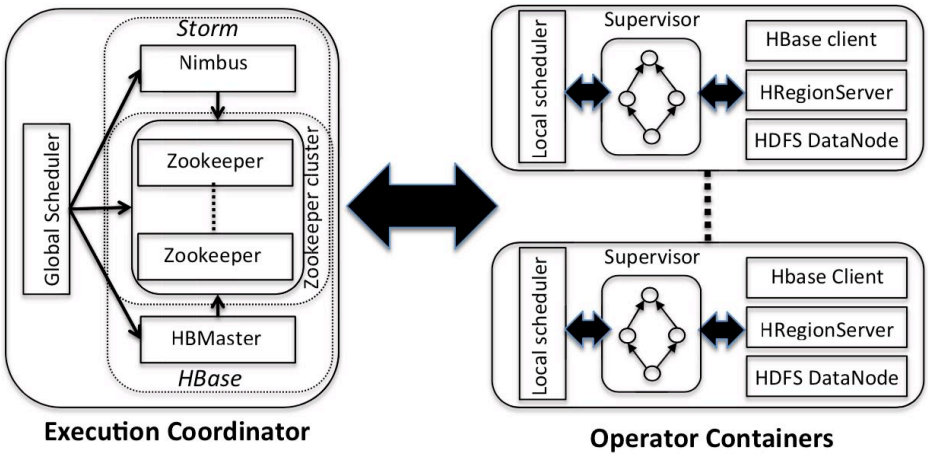


Figure 2.10: *CQELS Cloud architecture (source [70]).*

INSTANS Incremental eNgine for STANding Sparql (INSTANS) [69] enables users to create a flow of multiple SPARQL 1.1 queries that represents a single task. The engine is in charge of continuously evaluates the incoming data and store intermediate results. INSTANS approaches RDF stream processing from a different perspective and does not require a continuous extensions to RDF or SPARQL.

ETALIS and EP-SPARQL Event TrAnsaction Logic Inference System (ETALIS) [64] is an RDF stream processing engine that takes in account two time-stamps during the data processing. It is a pluggable system that can use multiple prolog engines. An internal ETALIS task can be specified using two different languages, Event Processing SPARQL (EP-SPARQL) and ETALIS Language for Events (ELE). Both of them allow users to derive complex events through deductive prolog rules. The engine supplies the results as soon as they are available and supports three different policies during the execution of the task: (i) unrestricted, all the input items are used for matching the declared patterns; (ii) chronological, the earliest matchable input are selected for matching the event patterns, the next evaluations will ignore the already selected data; (iii) recent, the latest matchable input are selected for matching the event patterns, as for chronological policy, they are then ignored.

CQELS Cloud CQELS Cloud [70] presents an elastic and distributed environment that exploits an extension of CQELS to enable a network of processing nodes to parallelize tasks. The input of the CQELS Cloud execution model is a set of CQELS-QL queries to be executed against a set of RDF input streams and the output is a set of output streams (in RDF or relational format). The input queries are compiled to produce a logical query network that defines the algebras for each input stream. The logical query network is, then, mapped on a processing network made of processing nodes, namely the Operator Containers (OCs). The tasks are distributed among the OCs by the Global Scheduler.

Figure 2.10 depicts the general architecture of the system. CQELS Cloud implements the elastic execution model and the parallel algorithms exploiting ZooKeeper [71],

Storm[16] and HBase[17].

Strider Strider [72] is a distributed and adaptive RDF Stream Processing engine that optimizes logical query plan according to the data stream evolution developed to guarantee scalability, availability, fault-tolerance, high throughput and acceptable latency.

Strider relies on well known distributed technology and is composed by two main components: (i) the data flow management and (ii) the Computing core. The former is based on Apache Kafka, it categorizes the input RDF streams into different message topics which represent the different family of RDF events. The latter is based on the Spark Streaming framework, it creates a pipeline to perform parallel processing on the messages emitted by Kafka.

Strider proposes static and adaptive optimization components based, respectively, on heuristic rules and (stream-based) statistics and two strategies for the Adaptive Query Processing (AQP): backward (B-AQP) and forward (F-AQP) that mainly differ on the query plan computation time (i.e. at the previous or current window).

2.4 RSP Middleware

RSP Middlewares ease the task of deploying the RSP Engine in real-world applications by offering extensible means for collecting data in real-time, for publishing, accessing and querying collected information as Linked Data, and for visualizing query results.

In the next sections we present two different implementations of an RSP Middleware (the Streaming Linked Data (SLD) framework [5] and the Linked Stream Middleware [73]). They approach the problem in different ways. The SLD framework adopts a data driven in-memory approach for the processing of RDF streams with limited support for static information, the Linked Stream Middleware is a cloud-based infrastructure to integrate time-dependent data with other Linked Data sources.

2.4.1 Streaming Linked Data (SLD)

The Streaming Linked Data (SLD) framework is a general-purpose, pluggable system that supports the development of applications that continuously analyse RDF streams. SLD is designed to address five different requirements:

(R1) *every input is an RDF data stream*. The system must indifferently ingest data with different velocities from any sources. All the incoming information is modeled as an RDF data stream.

(R2) *Continuous Ingestion*. The continuous nature of data streams requires a continuously capture phase. The data, once arrived, is marked with an increasing timestamp. The system could handle data arriving with its own time mark.

(R3) *publish/subscribe*. SLD enable a publish/subscribe logic for its components. A senders, the publishers, publish timestamped RDF triples into RDF streams, and receivers, the subscribers, listen to one or more RDF streams, and only receive timestamped RDF triples that are of their interest. Publisher and subscribers do not have to know each other.

[16] http://storm-project.net/
[17] http://hbase.apache.org/

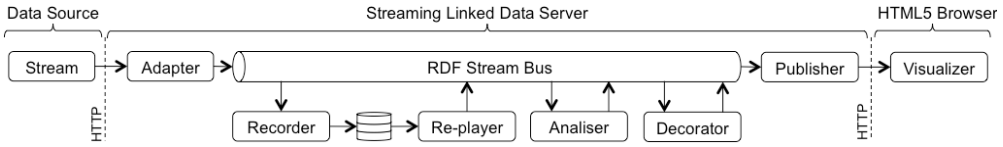


Figure 2.11: *The architecture of the Streaming Linked Data framework.*

(R4) *reliable message-passing*. SLD implements a logically reliable message-passing system that guarantees timestamped RDF triples to be delivered in order.

(R5) *minimizes latency by using main memory*. SLD minimizes latency by using main memory and avoiding disk I/O bottlenecks.

Figure 2.11 illustrates the architecture of the SLD framework. The leftmost column logically contains the *streaming data sources*, the central one the SLD server, and the rightmost one the visual widgets to be embedded in a dashboard.

The *streaming data sources* are assumed to be distributed across the Web and accessible via HTTP.

The core of the framework is *SLD Server*. It includes components for accessing data stream sources, internally streaming data, registering and replaying portion of data streams, decorating and analysing time-boxed portion of the stream, and publishing the results.

The adapters allow to access data stream resources, possibly delegating filtering operations to the data source, and to translate data items in the stream into set of timestamped RDF triples. SLD framework includes adapters for different social networks (e.g., Twitter, Instagram, foursquare) and for several sensor networks. For instance, the Twitter adapter allows to push to Twitter either geo-spatial filters, which ask Twitter to stream to SLD only tweets posted from given locations, or keyword-based filters, which ask Twitter to stream to SLD only tweets containing one or more of such key-words. Each tweet is internally represented using the extension of SIOC ontology presented in [2].

The publishers make available on the Web the content of chosen RDF stream following the Linked Data principles [74] in the Streaming Linked Data format proposed in [66]. The format is based on two types of named RDF graphs: instantaneous Graphs (iGraphs), which contain a set triples having the same timestamp, and stream graphs (sGraphs), which contains triples that point to one or more timestamped iGraphs. The number of iGraphs pointed by an sGraph and their time interval of validity can be configured when instantiating the publisher.

The recorders are special types of publishers that allow for persistently storing a part of an RDF stream. As format, we used an extension of the Streaming Linked Data format based on iGraphs and recording graphs (rGraphs). The latter are similar to sGraphs, but they include pointers to all the iGraph recorded and such pointers do not have a time interval of validity. The re-players can inject in an RDF stream what recorded in an rGraph.

The analysers continuously observe the timestamped triples that flow in one or more RDF stream, perform analyses on them and generate a continuous stream of answers. SLD framework includes a built-in engine that executes C-SPARQL queries, but any

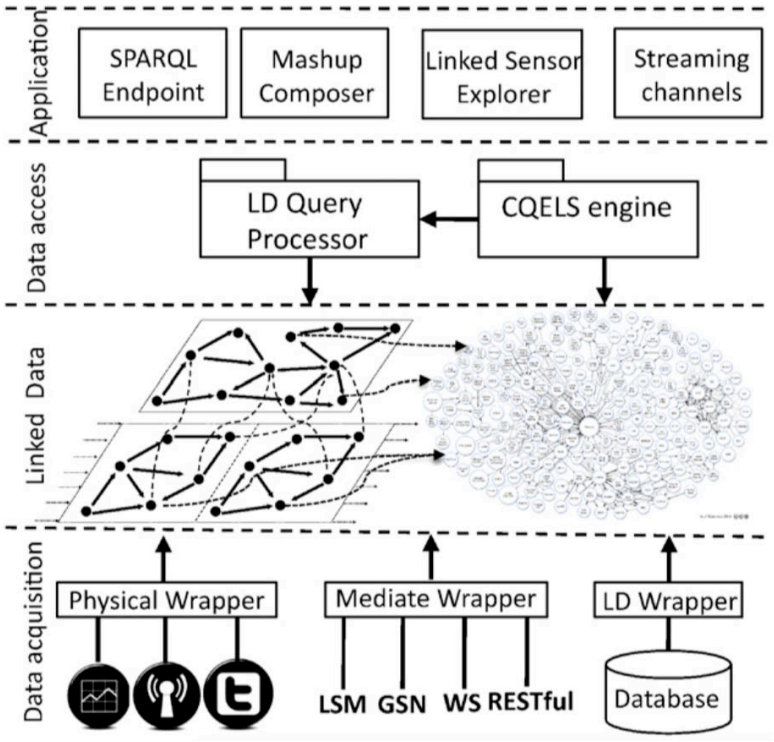


Figure 2.12: *Linked Stream Middleware architecture (source [73].)*

of the aforementioned continuous extensions of SPARQL (see Section 2.3.2) can be plugged in SLD server and used for the analysis.

The decorators are special types of analysers that look for a pattern of triples in a RDF stream. When the pattern matches, the decorators run a computation of the matching and add new triples to the stream.

SLD represents the first attempts to create a streaming computational model with the definition of generic principles related to the RDF stream processing world. SLD inspires the generic streaming computational model presented in the Chapter 5.

2.4.2 Linked Stream Middleware

Le-Phuoc et al. [73] proposes Linked Stream Middleware (LSM), a platform to create a bridge between data streams and Semantic Web. LSM offers: (i) a wide range of extendable wrappers to access streaming sources and transform the raw data into Linked Stream Data [75], (ii) components for annotating and visualizing data through a Web interface and live querying functionalities over unified Linked Stream Data and data coming from the Linked Open Data cloud exploiting a standard SPARQL query processor and CQELS.

The LSM architecture is layered to increase scalability and flexibility, Figure 2.12 presents an overview of LSM middleware.

The *Data Acquisition layer* is in charge of collecting data from streaming data sources via a wide range of wrappers. The different characteristics of the proposed wrappers allows the system to cover a broad range of input format. The output of the

wrappers conforms to format described in the Data Access Layer. The system allows users to develop their own wrapper in order to guarantee the flexibility of the entire system. The *Linked Data layer*, following the Linked Data publishing principles presented in [76], adds a global identifier to the data to ensure the data composability and exploits an ontology to represent data in a triple-based format. In particular LSM use the Semantic Sensor Network (SSN) Ontology[18]. The *Data Access layer* enable declarative query answering on top of the Linked Data layer in a pull-based or push-based fashion. This layer enables the storage of stream data and metadata in a triple format to ease the access of historic data. The *Application layer* offers support for developing applications to exploit query processing capabilities of the Data Access layer. The layer offers a SPARQL Endpoint, a Linked Sensor Explorer, a Mashup Composer and Notifications/Stream channels and enables query operation on historical sensor data. Continuous queries can be registered into the system to populate user-defined output streams.

2.5 Benchmarking

In this section, we propose an overview the benchmarking principles. In particular, Section 2.5.1 introduces *domain-specific* benchmarks. Section 2.5.2, then, concentrates on the benchmarking of velocity oriented systems, i.e., systems created to deal with continuously flowing data (e.g., RSP). Finally, Section 2.5.3 casts some light on the innovative concept of Configuration that Outperforms a Single Thread (COST), showing that a distributed solution, to be effective, must outperform a single-threaded one.

2.5.1 Domain-Specific Benchmarks

Nowadays, the variety of application of computer systems are growing faster than ever. A single metric cannot measure and evaluate the performance of a computer system in all applications and domains. A computer system is generally designed to face few problem domains and the performance in performing other tasks can be very poor. Jim Gray [77] proposes the *domain-specific* benchmarks concept as a response to the heterogeneity of computer system use.

A benchmark, built according to this concept, specifies a synthetic workload shaped to evaluate a system while it performs a typical application for which it was designed. A *domain-specific* benchmarks must follow four criteria. It must be:

- *Relevant*: given the problem domain, the benchmark must measure the peak performance and price/performance of the system during a typical domain-specific operation.
- *Portable*: the benchmark should be easy to be implemented on different systems and architectures.
- *Scalable*: the benchmark should apply to small and large computer systems, it should be also possible to test system on parallel architecture.
- *Simple*: to ensure the credibility of the system, it must be understandable.

[18] `http://purl.oclc.org/NET/ssnx/ssn`

As stated in the principles, Gray [77] highlights the importance of price/performance metrics for system benchmarking. The next sections will highlight the existing works on benchmarking streaming systems and the cost-aware approach.

2.5.2 Benchmarking Velocity Oriented Systems

The most relevant benchmark effort related to data stream management systems is the Linear Road benchmark [78]. In this work the authors present a benchmark specification for data management system with continuous querying capabilities. The benchmark is based on a traffic simulation use case. The system under test must answers queries from the vehicles on the road network in simulated real-time. The specification includes I/O, queries, expected results and relevant metrics for the system under test.

In many contexts, performance is very relevant. One of the first benchmarks for distributed streaming data processing is the Yahoo Streaming Benchmark [79]. This benchmark is based on a realistic use case. Also, a complete pipeline is implemented, from data ingestion to results. It is focused on performance metrics, namely latency and throughput, and it does not consider cost in its comparison. The benchmark code is available online under the Apache 2.0 license[19].

A more recent work on comparing popular distributed processing engines can be found in [80]. This work presents a performance comparison for Apache Storm [81], Apache Flink [82] and Apache Spark [38]. The comparison is based on realistic industrial use cases, and it focuses on throughput and latency.

The correctness theme pushed the discussion on what is really important in a dsb for Stream Reasoning [83]. Dell'Aglio et al. [84] propose a formal characterization of the operational semantics of different RSP (i.e. C-SPARQL Engine, CQELS, $\text{SPARQL}_{\text{stream}}$ and EP-SPARQL) exploiting CQL and SECRET. This formalization allows to determine a concept of correctness in RSP domain and to develop CSRBench [84] an extension of SRBench [85] to address query result correctness verification using an automatic method. Along the same line, Ali et al. [86] and Kolchin et al. [87] propose workloads and software environments to run experiments and perform measurement of quality criteria. A recent work of Tommasini et al. [88] proposes an open-source test-stand to ensure the reproducibility of experiments.

The performance metrics plays a key role in the evaluation (see Section 6.3) of the implementations of our streaming Computational Model presented in Chapter 5

2.5.3 Cost-Aware approach

Recently, in the Distributed System community, the concept of Configuration that Outperforms a Single Thread, namely COST, is gaining importance. McSherry et al. [18] introduce the COST metric and demonstrate that a single-threaded implementation of popular graph algorithms outperforms all distributed graph processing engines by orders of magnitude, at a fraction of the cost. Similarly, Boden et al. [89] compare single-threaded implementations of common machine learning algorithms with their distributed counterparts. Those are implemented using the most popular distributed machine learning libraries.

[19] https://github.com/yahoo/streaming-benchmarks

The concept of COST plays a crucial role in the evaluation (see Chapter 6.4) of the different implementations of the Computational Model proposed in Chapter 5

CHAPTER 3

Urban Data analysis

FINAL - TO BE READ ONE LAST TIME - ADD CONCLUSION IF HAVE TIME

In this chapter, we introduce the state-of-the-art on urban data analysis. We justify the choice of urban data as a privileged form of spatio-temporal data useful for a PhD thesis like this one, due to its variety and availability. Section 3.1 presents relevance and motivations of the urban data analysis with an overview of the most important research works in the field. In Section 3.2, we present the different dimensions of urban data analysis, related to content, time and space (and a combination of them). Section 3.3 presents an overview of relevant examples of implementations of RDF stream processing concepts at work on urban data. RSP represents a suitable solution because of the variety and velocity of the data sources and because of the need to fuse static (slowly evolving) and streaming data. The first examples (see Section 3.3.1 and Section 3.3.2) exploit data from different urban sources and the capability of RSP to abstract the real nature of the data through OBDA (see Section 2.2.1) that helps the machinery and the final user to extract insights from data. The two examples in Section 3.3.4 and in Section 3.3.3, mainly exploit data from Twitter. The data from social network has an intrinsic graph nature and was extracted using the official APIs that return the data in JSON format. Therefore, it is easy to transform it in JSON-LD[1], a format compatible with Semantic Web technologies. The usage of an RDF stream processor in this scenario is natural.

3.1 Relevance and Motivation

The relevance of urban computing, or urban informatics, has been recognized since long. A recent survey on urban sensing [90] clearly shows the value of mobile phone

[1] https://json-ld.org/

data to get insights of urban dynamics and human activities.

Studies like Gonzalez et al. [91] focus on using the mobile phones data, namely the call data records (CDRs), to track individual motion patterns characterizing each individual using a time-independent travel distance parameter and a significant probability to return to a few highly frequented locations. Candia et al. [92] investigate patterns of calling activity at the individual level and show that the inter-event time of consecutive calls is heavy-tailed. In recent years, the analysis of telco data in such an individual way rises serious privacy issues.

However, it is also possible to use telco data in privacy preserving way. A common applications is to use CDRs to estimate the density of crowds and vehicles in different urban regions [93, 94]. Another example involves the detection of people habits. Ratti et al. [95] present the Mobile Landscape project, one of the first urban analysis based on the geographical mapping of cell phone usage at different times of the day in the metropolitan area of Milan. Becker et al. [7] capture key mobility patterns within Morristown, NJ, by identifying users' home and work locations from CDRs. This information is particularly useful for urban managers and authorities that are responsible for efficient public transportation systems. Also De Nadai et al. [96] test the four Jacobs conditions[2] that promote life in cities by using CDRs. Wesolowski et al. [98] combine CDRs with other cellphone-related logs (e.g., tower pings, cellular handovers) in order to compare human mobility patterns derived from CDRs.

Although mobile phones data is a priceless source to gather underlying patterns of cities and their citizens, they have got some limitations since they cannot reveal any information about people interests and thoughts. Social media data represents an opportunity to access information at individual level without violating privacy. This data is, indeed, made public by the user through a self determination process. Social media streams are a powerful mean to explore people opinions and preferences with regard to specific venues and events. For instance, Hristova et al. [9] analyze temporal, spatial, and microeconomic patterns of sport game attendees to understand the users' dynamics, while Lee et al. [99] and Cho et al. [4] use social networks and cell phone location data, to identify humans' mobility patterns. Psyllidis et al. [100] concentrate their effort on wide range of urban data and create a web-based platform to support city planning and decision-making. In particular, Singh et al. [101] introduce the concept of *social pixel* that aggregates social interests of users about particular themes and locations. This notion plays a key role during the development of our conceptual model presented in Chapter 4.

Moreover, urban data from different sources (e.g., CDRs, social data, IoT, etc.) can be merged to reveal even more interesting insights on city dynamics and urban monitoring. The platform built by Calabrese et al. [8] combines the users' mobile phones' data with the real-time location of buses and taxis to model the car traffic in Rome. Botta et al. [102] try to quantify the dimension of the crowd by exploiting a combination of CDR and social media data. Quercia and his colleagues [103] use CDRs to study human mobility related to special planned events in Boston. Calabrese et al. [104] show that there is a high correlation between the kind of event, e.g., sport, theater, music, family events, and the home location area of its attendees. Quercia et al. [105] build a

[2]The four conditions exposed by Jane Jacobs in [97]: mixed land uses, small blocks, buildings diversity in terms of age and form and sufficient dense concentration of people and buildings

recommendation system for social events and find out that the most effective algorithm recommends those events that are popular among local residents.

Sentiment analysis covers a wide range of applications in cities. Authors in [106] propose a city sensing architecture from Twitter data to monitor user opinions about events and topics. Hawelka et al. [107] and Calabrese et al. [108] analyze geo-located Twitter messages and geographical preferences in order to predict global patterns of human mobility.

One of the features of interest for policy makers and cities managers [109] is the extremely diversified composition of the language mix, or multilingualism. This interest is motivated by the increasing immigration flows towards cities [110], which result in rapidly changing population density [111]. Multilingualism has also a broad scope in academia. In particular, different papers approach the issue of multilingualism from a historical perspective. Leimgruber in [112], for example, analyses the city of Singapore, Garcia et al. [113] the city of New York, Extra et al. [114] develop a cross-linguistic perspective on Gothenburg, Hamburg, The Hague, Brussels, Lyon and Madrid. Moreover, Tasse et al. [115], Arnaboldi et al. [116] and Bokanyi et al. [117] characterize cities and their neighborhoods from different aspects namely safety, culture and demographics through social media networks. Quaggiotto et al. [118] present a tool called City Murmur with the aim at showing how different media describe the urban space through the attention that is payed on each street of a city. It wants to build a time-based narration, an historical archive of media coverage of the urban space which is able to reveal some hidden dynamics useful for city policy support, critical media analysis, and sociocultural research.

The interest around the exploitation of urban data are growing, however the joined use of multiple data sources has not yet been fully explored. In the next sections, using technologies meant to tame velocity and variety simultaneously (see Section 2.4), we present different use cases of urban streaming data analysis.

3.2 Urban Data Analysis Dimensions

The urban data analysis can be developed on three different dimensions: **space**, **time**, and **content**. In this section, we summarize the characteristic of each analysis dimension. See Section 4.4 for more information about how our conceptual model (proposed in Chapter 4) enables the described analysis dimensions

3.2.1 Content Analysis

The content can be associated to an event and thus indirectly to the time and space of the happening, and carries information that represents a measure of intensity of a tracked event. During a content based analysis of urban data, a stakeholder can be interested about the contextual and behavioral knowledge about what and how users share about an event.

The content analysis can be approached in two different ways: (i) using the *Original* content, or (ii) creating *Augmented* content. In the former approach, the original content is analyzed as is and used for profiling social media users who are engaged in events. In the latter one, the augmented content can be created by using concept and feature

extraction techniques from the original content for the purpose of more complex analysis about the events and their attendees.

The augmented content could consist of different media types including text, image, video, etc. that also contain low-level information about events like locations, time tables, related social users and so forth. From such content, **low-level features** such as color schema for images or n-gram distribution for text, can be extracted. For instance, a system can use the main color schema in photos related to an event to verify the correctness of the estimated location of that event. Furthermore, **high-level features** like number of people and their demographics in a photo, list of existing concepts that are represented in a photo or a video (using deep learning techniques) or semantic entities from text using ontology-based matching, can be extracted.

3.2.2 Spatial Analysis

Another dimension of interest in city analyses is space. Therefore, a urban analytics system must focus on analyzing events, people presence and flow, content and opinion sharing, or any other type of phenomena (like electrical consumption, traffic, economical value) with respect to the spatial distribution and spreading, also considering its dynamics in time. The spatial dimension is more complex to deal with than one can expect. Indeed, in smartcity context, the data sources may vary a lot: some information may refer to specific geographical points (geo-coordinates), some others may refer to venues or locations (restaurants or other public or private spaces), while others can provide information referring to broad areas, possibly with different size and shape. Any analysis considering two or more different data sources need to keep this into account.

Interesting types of relevant analysis categories can be:

1. *Dispersion*: studying the spatial distribution of locations of events, in particular with respect to the deviation from purely random configuration. This can be achieved with measures such as the Gini coefficient [119].

2. *Distance and relation to places*: studying the spatial relation of events with respect to a set of given locations (e.g., stores or venues for specific happenings such as fashion shows, see Section 7.2). This is covered by simple measures such as the average Euclidean distance between event and location, or the average Manhattan distance over an artificial grid of cells or travel distance over the road network.

3. *Correlation*: studying the relevant correlations between different signals along the space dimension (e.g, within and across administrative boundaries).

4. *Prediction*: defining predictive analytics along the space dimension, by analyzing historic series of data and comparing it with the most fresh information from given geographical area.

3.2.3 Temporal Analysis

Temporal analysis focuses on the study of the evolution and spreading of signals over time (e.g., measuring how fast information about an event propagates). The goals of temporal analysis can be diverse. We identify the following types of relevant analysis categories:

1. *Description*: consisting in defining the signal as a time series in order to have a view of the evolution of information flows over time.
2. *Correlation*: studying the temporal correlation between different time series and infer common behaviors and dynamics.
3. *Prediction*: allowing generating temporal prediction over observed or correlated phenomena.
4. *Anomaly detection*: identifying discrepancies between expected temporal behaviors and actual happenings.
5. *Causality*: determining possible causality relations between different events.

3.2.4 Combined Time and Space Analysis

Given the basic space and time analysis aspects described above, the subsequent level of interest is the combined analysis along both directions. A system can combine techniques described in the previous sections for running analysis across time and space.

Furthermore, one can define time series of values that are aggregated or calculated on geographical basis. For instance, a system can define the time series of the values of the Gini Index or of the average distance of events from a set of given venues, and then analyze them along the temporal axis.

3.3 Existing Semantic Web-Based Solutions

The next sections present four different solutions for urban data analysis based on Semantic Web technologies. Section 3.3.1 and Section 3.3.2 present respectively STAR-CITY and Traffic LarKC, two works that face the problem, from architectural point of view. Both of the presented solutions perform *spatial* and *temporal* analysis on the collected data to offer the best route to the user based on weather, traffic and other multiple external sources. In Section 3.3.3 we propose an overview of the infrastructure we created for monitoring crowds movement during the 2012 Olympic Games in London. The solution exploits data from social network and perform *content*, *spatial* and *temporal* analysis to spot emerging pattern and mobility dynamics and enables the creation of complex data visualizations. Finally, Section 3.3.4 presents BOTTARI, a mobile application that performs *content*, *spatial* and *temporal* analysis on social network and urban data to recommend restaurants to the user.

3.3.1 Monitoring Traffic Using Semantic and Stream Technologies

In recent years, public administrations and governments are embracing Open Data in the attempt to made available information to increase transparency and improve accountability of public services. Many cities are offering data regarding transportation, environment, energy and planning. Web sources offer an abundance of information (e.g., weather information, bike sharing usage, etc.) and non-public data can also be accessed (e.g., current location and state of public transportation, CCTV images, etc.). Lecue et al. [3] proposes Semantic Traffic Analytics and Reasoning for CITY (STAR-CITY), a solution designed to ease the integration of data characterized by variety (structured and

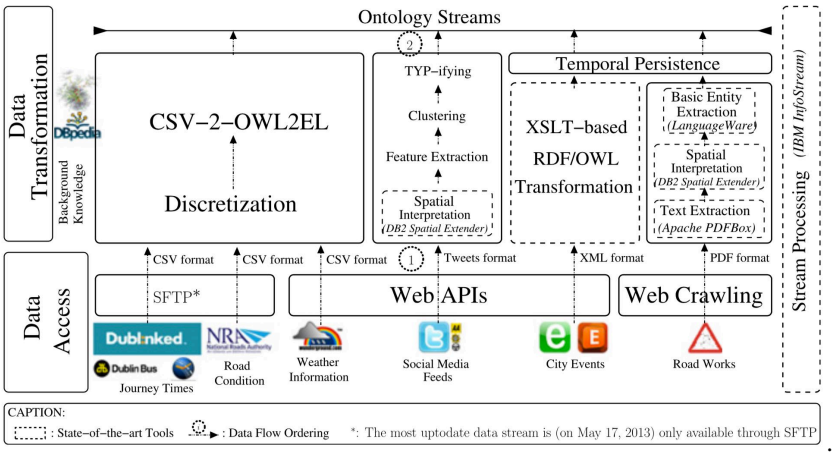


Figure 3.1: *The STAR-CITY semantic stream enrichment (source [3]).*

unstructured), velocity (static and real time streaming data) and volume (large amount of historical data).

STAR-CITY exploits the W3C Semantic Web stack to represent the semantics of information and to elaborate the outcomes through a combination of reasoning techniques (Figure 3.1 shows STAR-CITY semantic stream enrichment architecture). The solution was mainly designed to perform *spatial* and *temporal* analysis on heterogeneous data, on order to provide insights on historical and real-time traffic conditions. The traffic scenario was chosen because most of the industrial countries are suffering of traffic congestion and transportation issues that can reduce the health of citizenship and can interfere with the passage of emergency vehicles. The system was successfully tested in various scenario involving different cities (Dublin, Bologna, Miami and Rio).

3.3.2 Semantic Traffic-Aware Routing

In the late 2000s, the increasing usage of mobile technologies to get directions and information about the surrounding presented various challenges. Those challenges were only partially solved by the existing research: operation research solved the routing problem, machine learning addressed traffic forecasting, and semantic technologies managed data integration and information retrieval. Therefore, the research of location-based comprehensive solution was still an open problem.

Traffic LarKC [120] is a fist attempt to build a comprehensive system able to tame of those challenges simultaneously. It offers a mix of conceptual query answering, machine learning, and operations research. It can answer questions like "What Asian restaurant can I reach in less than 15 minutes if I get into my car at 6 p.m.?".

Figure 3.2 depicts the Traffic LarKC workflows. The service exploits the LarkC platform [121] to offer a comprehensive solution to periodically perform *spatial* and *temporal* analysis on the data and compute the best route taking in account weather, traffic and other data from external dataset.

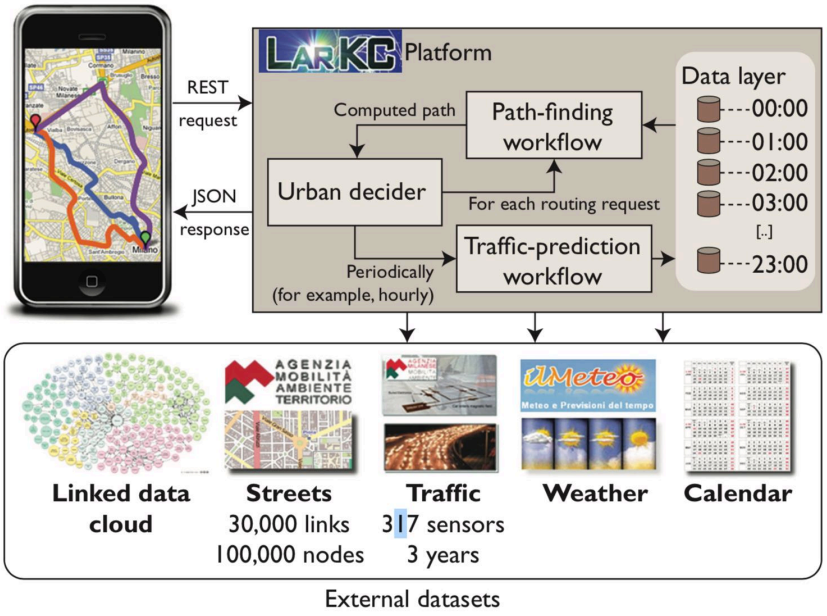


Figure 3.2: *The Traffic LarKC workflows and external datasets (source [120])*

3.3.3 Monitoring Crowd Movement During London 2012 Olympics Games

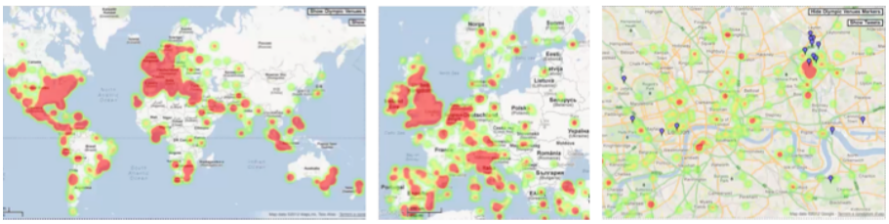
The work presented in this section shows how to track the movements of the crowds in big events exploiting the analysis of geo-tagged tweets. Previous work shows that the data from social network is incomplete and inconsistent (see Section 2.3.2) and the proposed system must deal with these data characteristics.

The presented system exploits the C-SPARQL Engine [12] within SLD (see Section 2.4.1) framework in order to perform *content*, *spatial* and *temporal* analysis on Social Media streams (i.e., Twitter). It models the data in a convenient format and exploits OBDA to extract the position on the interesting geo-tagged tweets. Being public not only the position of the tweet but also the content, the system can track the people attention and select only the tweets with content related to the event.

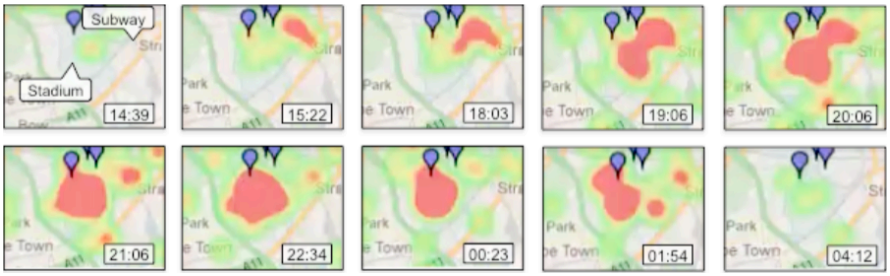
Figure 3.3(a) shows the attention of the people at different space granularity, i.e. World, Europe, City of London, during the London 2012 Open Ceremony. The information was extracted from Tweets selected exploiting keywords present in text of the content. Figure 3.3(b) shows movements of the crowd before, during and after the Olympics open ceremony. Exploiting the Tweets position the system can clearly shows people arriving at the Olympic Stadium, entering the Olympic Stadium and leaving the Olympic Stadium. Figure 3.3(a) and Figure 3.3(b) shows how the infrastructure can enable visual analytics. Both of the figure are based on the same data, but, while the former enables the observation of world-wide attention pattern, the latter offers insights of crowds' movements in a given area.

3.3.4 Bottari

In 2011, an average of three million tweets per day was posted in Seoul and many of that carry the user's live opinion about restaurants, bars, cafes, and many other semi-public points of interest (POIs) in the city. The stream of data, which we continuously col-



(a)



(b)

Figure 3.3: *The Figure (a) shows the people's interest at different zoom level during the Open ceremony of Olympic Games. Figure (b) shows the movement of the crowd in the surrounding of the Olympic Stadium at different time around the ceremony (source [5]).*

lected, results (i) incomplete, i.e. only 41% of users rated at least the same POI, and (ii) inconsistent, i.e. many users rates a POI several times in different way. BOTTARI [2] exploits inductive and deductive stream reasoning (see Section 2.3.1) to continuously perform *content*, *spatial* and *temporal* analysis on Social Media streams (i.e., Twitter) in order to understand how the Social Media users collectively perceive the POIs in a given area (e.g., Insadong's restaurants) and build a recommendation engine.

BOTTARI is designed following an OBDI architecture (see Section 2.2.1). The infrastructure behind BOTTARI application is pluggable and is based on SLD framework (see Section 2.4.1). A first component in SLD pipeline casts in RDF the data item in the stream. A second downstream component performs the analysis exploiting the C-SPARQL Engine (see Section 2.3.2) as deductive stream reasoner A last downstream component, based on the Statistical Unit Node Set (SUNS) [122, 123] approach, acts as inductive stream reasoner. The data was continuously collected and modeled using BOTTARI ontology, an extension of SIOC vocabulary, to cope with the variety.

Figure 3.4 shows the various visualization offered by the augmented reality Android application that returns BOTTARI results to the end user. Such an application guides the user to the POI choice. The presented POIs are based on the deductive/inductive stream reasoning results and are personalized for the user.

Figure 3.4: *BOTTARI visualizations (source [5]).*

CHAPTER 4

Conceptual Model

FINAL - TO BE READ ONE LAST TIME.

In this chapter, we propose a conceptual model named FraPPE [20]. FraPPE is a vocabulary that bridge the gap between the data engineer and visual interface designers for enabling visual analytics for the detection, the understanding and the interpretation of spatio-temporal data. We introduce the problem, solved by FraPPE, in Section 4.1. In Section 4.2, we present an overview of the first version of FraPPE. In Section 4.3, we propose FraPPE 2.0, an extension of the original vocabulary where we expand the provenance fragment and add a content related fragment. Finally, Section 4.4 presents an overview of the analysis enabled by FraPPE 2.0.

4.1 Introduction and Problem Statement

Since the last decade, the rapid increase of sources, which expose geo-located time-varying data, has been drawing attention of those who are looking for data-driven decision making. The availability of social media, telecommunication, traffic and weather data improved the ability to capture peoples' interests, habits and preferences.

In particular, the growing availability of urban data sources (see Chapter 3) stimulated the research of a holistic conceptual model to manage data variety in a comprehensive way. The current interest is for solutions that enable the fusion of streaming heterogeneous data to enable reactive decisions.

One of the main assumptions of any smart-city approach is to work upon a layer of data collected from the city itself that describes its dynamics. The city evolution can span multiple layers, from architecture to urban design, from population composition and migrations to citizen behaviors and interests. Each of these layers has a different dynamics and speed of change. Therefore, it should be monitored collecting data from

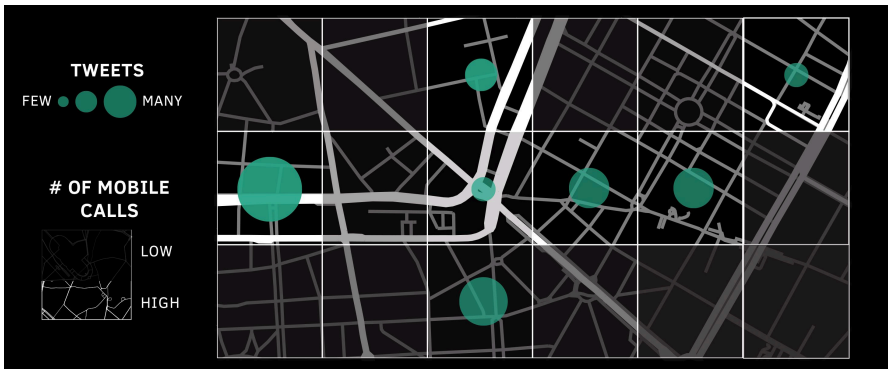


Figure 4.1: *A real-world example of visual analytics of two heterogeneous datasets. (source [20]).*

different sources and using multiple analysis techniques (see Section 3.2).

In particular, our research question, presented in Section 1.2, refers to visual analytics[1] as a way for making sense of heterogeneous spatio-temporal streaming data. For instance, Figure 4.1 illustrates a real case of visual analytics for a general audience we experimented during the Milano Design Week events (see Section 7.1.2). The Figure presents a grid of 6x3 cells overlaid to a city street map. Green circles can appear in each cell. They visually represent the number of tweets posted in a time interval from each cell. The fill color opacity value of each cell is, instead, mapped to the number of mobile calls from each cell. As we showed in Section 7.1.2, people without specific expertise in data analytics can easily guess the cells where the two signals are correlated.

Unfortunately, data is not often ready for visual analytics. A gap exists between the terms used by the designers who create the visual analytics interface and those used by the computer scientists who prepare the data. The designers expect data aggregated over time and space, while the data engineers talk about fine-grained geo-located time-varying data. For instance, in 2012, during the BOTTARI experiment (see Section 3.3.4) in the city of Seoul, we model data using SMA ontology (see Figure 4.2). SMA extends the SIOC vocabulary[2] adding the spatial aspect. To do so, it exploits terms from W3C WGS-84 vocabulary[3]. SMA poses the attention only on the raw data representation. It does not include any specific concept to represent data aggregations or data abstractions to enable advanced analytics, which is one of the goal of this thesis.

During years of collaboration with the Design Department of Politecnico di Milano[4], we discover that image processing terms like Pixel or Frame are common among both designers and data engineers. They represent a bridge between who prepare the data and who create the interfaces for visualizing it.

With this observation in mind, in order to answer the research questions, we formulated the hypotheses Hp.1.1 and Hp.1.2

Hp.1.1 A conceptual model containing terms from the image processing domain can represent spatio-temporal data in an extendable and coherent way with a minimal encoding bias and a minimal ontological commitment.

[1] The science of analytical reasoning facilitated by interactive visual interfaces [124].
[2] http://sioc-project.org/
[3] https://www.w3.org/2003/01/geo/
[4] https://densitydesign.org/

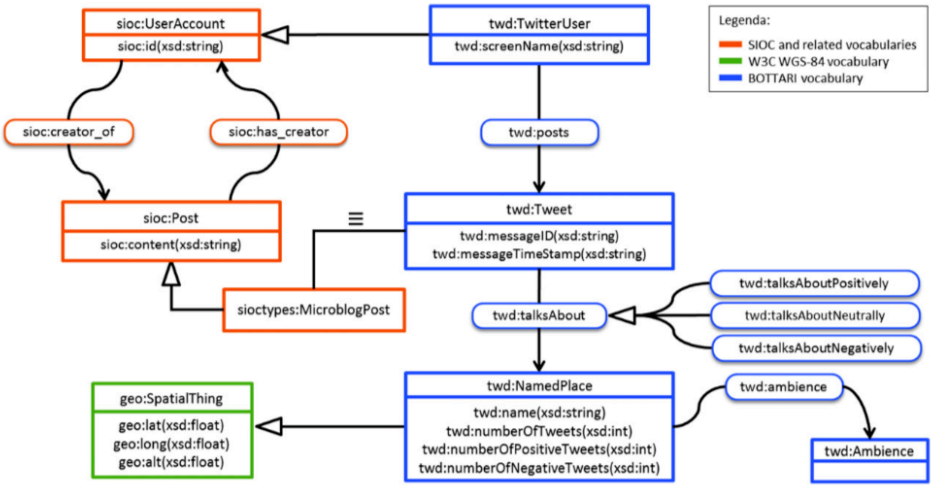


Figure 4.2: *Overview of the SMA ontology used in the BOTTARI project to model the data (source [2]).*

Hp.1.2 Visual analytics interfaces built directly on data represented with the conceptual model of Hp.1.1 are guessable[5].

4.2 FraPPE 1.0

In this section, we propose our first attempt to create a conceptual model to represent spatio-temporal data. Section 4.2.1 exposes the main concepts behind FraPPE and the development methodology. In Section 4.2.2, we present an evaluation of FraPPE 1.0 based on its adherence to the Tom Gruber's principles [44]. Finally, Section 4.2.3 presents a working example, where we use FraPPE 1.0 to model the data of the DEBS Grand Challenge 2015[6].

4.2.1 The Conceptual Model

In order to verify the hypotheses Hp.1.1 and Hp.1.2, we proposed the conceptual model FraPPE. It is named out of its four main concepts: Frame, Pixel, Place and Event. In this section, we refer to FraPPE 1.0 as FraPPE.

FraPPE ontology is formalized using the version 2 of the Web Ontology Language (see Section 2.2.1). It reuses GeoSparql [126] as geographical data model, the Time [127] and Event ontologies [128]) as time/event vocabularies, and PROV- O [129] as provenance ontology.

FraPPE enables an OBDA approach for the data analysis by exploiting the terms imported from GeoSparql, Time, Event and PROV-O ontologies without any axiomatization. This is because OBDA requires OWL2-QL ontologies, while FraPPE with the imported ontologies is in OWL2 Full.

FraPPE offers a high level view of the detection, the understanding, and the interpretation of geo-spatial time-varying data. It uses a digital image processing metaphor

[5]The guessability is defined as the measure of the cost to the user involved in using an interface to perform a new task for the first time. The lower the cost, the higher the guessability [125]. The cost can be measured in terms of time, errors, or effort.

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

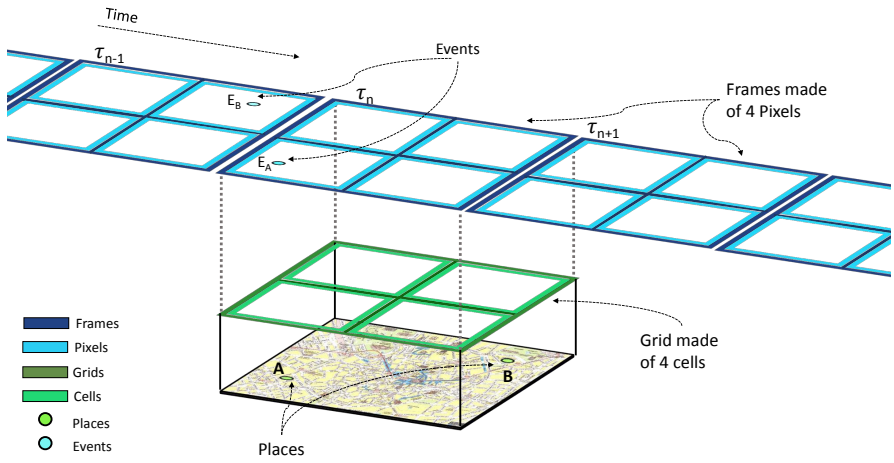


Figure 4.3: *A high-level view of FraPPE including 3 Frames made of 4 Pixels containing the Places where the Events happen.*

(see Figure 4.3) to track the three main dimensions of analysis introduced in Section 3.2: space, time, and content.

FraPPE assumes that the real world can be described as a bi-dimensional space, where Events happen in Places over time. For instance, a user making a check-in on a geo-located social network generates an event in a place. A taxi ride generates a sequence of two events (a pick-up and a drop-off) in two distinct places. A garbage collector truck generates a sequence of events around the city in different points in time, one for each trash bin it cleans up.

Figure 4.4 depicts all the main concepts in the model. FraPPE is organized in three interconnected parts: the geographical, the time-varying and the provenance fragments. It proposes to capture the digital footprints of what happens in the real-world as a sequence of Frames. A Grid sits between the physical world and the frames of the film. It decomposes the physical world in Cells. Each frame is, therefore, decomposed accordingly in Pixels.

More formally, Place, Cell and Grid belong to the geographical fragment and reuse terms from the GeoSparql vocabulary [126]. Event, Pixel and Frame are in the time-varying part. The Event term is borrowed from the Event ontology [128]. The provenance part includes the activities Capture and Synthesize and reuses the PROV Ontology [129] (PROV-O).

An Event has a location in a Place in a Cell – the basic spatial unit of aggregation of information in FraPPE – which, in turn, is in a Grid.

A Pixel is the time-varying representation of a Cell. It is the only element in FraPPE 1.0 that carries information through the hasValue data property. As in image processing, this value represents a measure of intensity of some phenomenon in the real world. For instance, it can represent the number of micro-posts posted in a given time interval in a certain Cell. Each Pixel refersTo a single Cell, contrariwise a Cell could be referredBy many different Pixels that captures different information associated to the same Cell, e.g., the already mentioned number of micro-posts, but also the number of mobile phone calls or the number of goods' pick-ups.

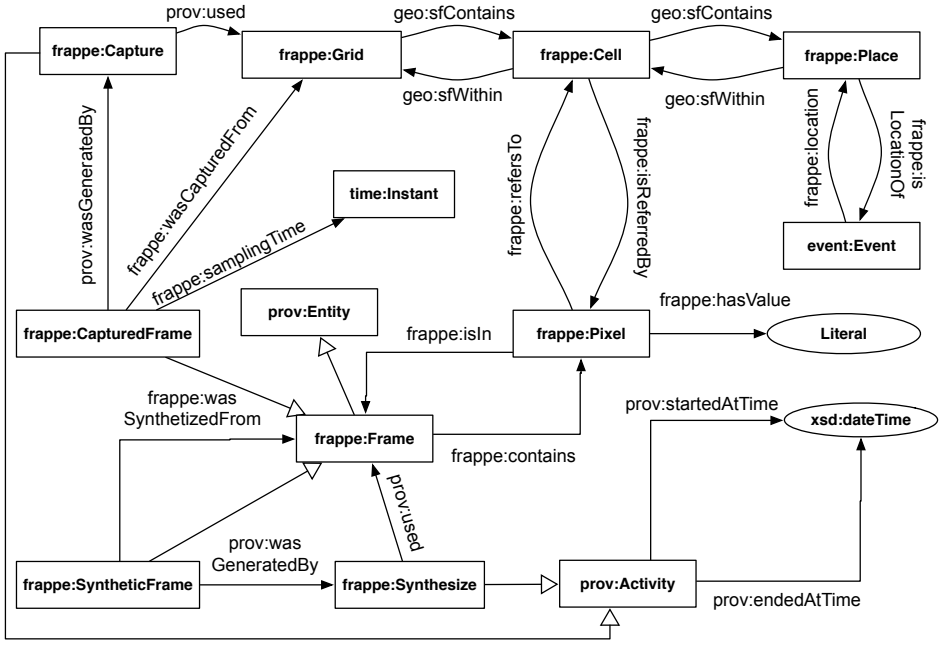


Figure 4.4: *The UML representation of the original version of the FraPPE ontology*

Similarly, a Frame is the time-varying counterpart of a Grid and a sequence of Frames composes the film of the evolution of a physical portion of the world over time. FraPPE distinguishes between two specializations of Frames: the CapturedFrames and the SyntheticFrames. Frame and Grid are *Entity* in PROV-O. This is because FraPPE proposes the ternary relationships Capture and Synthesize as specializations of the relationship *Activity* of PROV-O.

A CapturedFrame *wasGeneratedBy* a Capture *Activity startedAtTime* τ_i and *endedAtTime* τ_j that *used* a given Grid. It contains a Pixel for every Cell in the Grid it wasCapturedFrom. Different Frames represent different images of the observed phenomena at the same samplingTime (e.g., a frame captured the volume of the social activity while another one captured the volume of the mobile phone calls at 12.00). The object property wasCapturedFrom is the result of the chaining of the two *wasGeneratedBy* and *used* object properties. Moreover, the value of the samplingTime data property, which describes the CapturedFrame, is the one assigned to the *startedAtTime* data property that describes the captured activity.

Similarly, a SyntheticFrame *wasGeneratedBy* by a Synthesize *Activity* that *used* one or more Frames. The idea is to derive a Frame from one or more others. The Synthesize operation can be a filter applied to the values of the pixels, or an aggregation of values of Pixels across Frames or the difference between the values associated to the Pixels of two different Frames.

We develop FraPPE using METHONTOLOGY [16] (see Section 2.2.1) methodology by following the whole ontology life cycle. We started from the specification phase following, in the scoping activity, a middle-out approach (see Section 4.1 in [16]). We exploited terms from image processing and spatial glossaries in order to find the primary concepts of FraPPE and to verify, at the earliest possible stage, the conciseness and completeness of our specification. We then conceptualized the specification and produced

a complete Glossary of Terms (GT). Most of the concepts in FraPPE are directly integrated from other domain specific ontologies (e.g., Instant from Time ontology [127], Activity from PROV-O ontology [129], etc.)

4.2.2 Adherence to the Tom Gruber's Principles

We evaluated FraPPE, by checking if it adhere the five principles of Tom Gruber [44]: clarity, coherence, minimal encoding bias, minimal ontological commitment and extendibility.

FraPPE satisfies the *clarity* principle because all definitions are documented in natural language (see the version of FraPPE published on github[7]). The terms proposed in FraPPE are: (i) common terms in spatial-related vocabularies (e.g., Place, Cell, Grid); (ii) well known terms of the image processing domain (e.g., Pixel, Frame, Capture, or Synthesize); and (iii) terms defined in other ontologies (e.g., Event, Instant, Entity, or Activity).

FraPPE has a *minimal encoding bias* because it is encoded in OWL2. Moreover, we explicitly avoided adding cardinality restrictions, because, in our works (see Chapter 7), we use FraPPE to integrate data following an Ontology-Based Data Access approach which requires OWL2 profile that does not include cardinality restrictions.

FraPPE requires a *minimal ontological commitment*, meaning that, as Tom Gruber recommended, FraPPE makes as few claims as possible about the geo-located time-varying data being modeled allowing who uses FraPPE to specialize and instantiate it as needed.

We tested in details that FraPPE is *extendable* by successfully modeling the dataset made available by ACM DEBS 2015 Grand Challenge[8], for all the details, see Section 4.2.3. Moreover, we check the extendibility of FraPPE while using it in the experiences reported in the Chapter 7. For further details see Section 7.4.

Last but not least, FraPPE is *coherent*, i.e., all FraPPE inferences at T-box level are consistent with the definitions and in modeling A-boxes containing social, telecommunication, environment, traffic, and energy consumption data, we never inferred inconsistent or meaningless data.

4.2.3 Working Example

The ACM DEBS 2015 Grand Challenge proposes a taxi route analysis scenario based on a grid of 150x150 Kms with cells of 500x500 m. A stream of data 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) ride information (e.g., tip, payment type and total amount). In Listing 4.1, we report a subset of the information representing a single taxi ride in FraPPE. The pick-up Event represents the start of the ride and contains the taxi id. The drop-off Event represents the end of the trip and it is connected to all the information about the ride. The fragment models the geographical part of the ride using two Places within two different Cells of a single Grid. Moreover, it models the time varying-part of the ride using two Events captured in two Pixels of a single Frame along with the provenance part through the Capture activity. Indeed, we use all FraPPE concepts, we specialize Event in PickUpEvent and

[7] https://github.com/streamreasoning/FraPPE.git
[8] http://www.debs2015.org/call-grand-challenge.html

Listing 4.1: *Fraction of the model representing ACM DEBS Grand Challenge 2015 Data*

```
@prefix frGrid: <http://streamreasoning.org/debsGC/Grids/> .
@prefix frCell: <http://streamreasoning.org/debsGC/Cells/> .
@prefix frPixel: <http://streamreasoning.org/debsGC/Pixels/> .
@prefix frPlace: http://streamreasoning.org/debsGC/Places/:> .
@prefix frEvent: <http://streamreasoning.org/debsGC/Events/> .
@prefix frFrame: <http://streamreasoning.org/debsGC/Frames/> .
@prefix frCapture: <http://streamreasoning.org/debsGC/Captures/> .

frGrid:Grid_1 gs:sfContains frCell:Cell_1, frCell:Cell_2 .

frCell:Cell_1 a fr:Cell ;
    rdfs:label "39460"^^xsd:long ;
    fr:isReferredBy frPixel:1356995100000_39460 ;
    gs:sfContains frPlace:A ;
    gs:sfWithin frGrid:Grid_1 .

frPlace:A a sf:Point ;
    fr:isLocationOf frEvent:E_B ;
    gs:asWKT "POINT( 40.715008 -73.96244 )"^^gs:wktLiteral ;
    gs:sfWithin frCell:Cell_1 .

frEvent:E_A a fr4d:PickUpEvent ;
    a event:Event ;
    event:time [ a time:Instant ;
       time:inXSDDateTime "2013-01-01T00:00:00"^^xsd:dateTime ] ;
    fr:location frPlace:A> ;
    fr4d:hackLicense "E7750A37CAB07D0DFF0AF7E3573AC141"^^xsd:string ;
    fr4d:medallion "07290D3599E7A0D62097A346EFCC1FB5"^^xsd:string .

frEvent:E_B a fr4d:DropOffEvent ;
    a event:Event ;
    event:time [ a time:Instant ;
       time:inXSDDateTime "2013-01-01T00:02:00"^^xsd:dateTime ] ;
    fr:location frPlace:B ;
    fr4d:connected frEvent:E_A ;
    fr4d:fareAmount "3.5"^^xsd:double ;
    fr4d:mtaTax "5.0"^^xsd:double ;
    fr4d:paymentType "CSH"^^xsd:string ;
    fr4d:surcharge "5.0"^^xsd:double ;
    fr4d:totalAmount "4.5"^^xsd:double ;
    fr4d:tripDistance "0.44"^^xsd:long ;
    fr4d:tripTime "120"^^xsd:long .

frPixel:1356995100000_39460 a fr:Pixel ;
    fr:isIn frFrame:1356995100000 ;
    fr:refers frCell:Cell_1 .

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

Listing 4.2: *Sparql query to create the* SYTHETICFRAME*s containing the* PIXEL*s with the profitability value.*

```
PREFIX   rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX   xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX   time:<http://www.w3.org/2006/time#>
PREFIX   f4d:<http://streamreasoning.org/ontologies/frappe4debs#>
PREFIX   prov:<http://www.w3.org/ns/prov#>
PREFIX   event:<http://purl.org/NET/c4dm/event.owl#>
PREFIX   geos:<http://www.opengis.net/ont/geosparql#>
PREFIX   rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX   fr:<http://streamreasoning.org/ontologies/frappe#>
PREFIX   sf:<http://www.opengis.net/ont/sf#>

CONSTRUCT{
    concat("frFrame:",now()") a fr:SyntheticFrame ;
    fr:contains ?pixel ;
    fr:samplingTime [ a time:Instant ;
        time:inXSDDateTime ?time ] ;
    fr:wasSynthesizedFrom ?frame .
    ...
}
WHERE {
  ?pixel f:isIn ?frame ;
  fr:refers ?cell .
  ?cell geos:sfContains ?place ;
  rdfs:label ?cellLabel .
  ?place f:isLocationOf ?e .
  ?e a f4d:DropOffEvent ;
  f4d:totalAmount ?t .
  ?frame time:hasBeginning ?beginning .
  ?beginning time:inXSDDateTime ?time .
  FILTER(?time >= \"2013-01-01T00:10:00\"^^xsd:dateTime
    && ?time <= \"2013-01-01T00:15:00\"^^xsd:dateTime)"
}
GROUP BY ?pixel ?cell ?cellLabel
```

DropOffEvent, and we extend the vocabulary adding two attributes (e.g., tripTime, and totalAmount) and an object property (i.e., connected) specific of the taxi ride domain.

Synthetic frames are also important in representing the data of the challenge. One of the problems, assigned to the challengers, asks to compute the top profitable cells for a given time interval. Listing 4.3 contains the representation of the SytheticFrame, named frFrame:1356995700000, computed by the Synthesize activity (frSynthesize:1356995700000) represented by the SPARQL query presented in Listing 4.2. The SyntheticFrame, wasSynthesizedFrom four different CapturedFrames, and contains two Pixels, associated to two different Cells with different values of profitability (number of DropOffEvent).

4.3 FraPPE 2.0

In order to extend the expressiveness of FraPPE and enable more advanced analysis (see Section 4.4), we propose FraPPE 2.0. We, mainly, extended the original FraPPE by improving the provenance fragment, in order to specialize the Agent concept, and by adding the content related fragment, in order to enable more fine grained analysis.

Figure 4.5 depicts a UML representation of the FraPPE 2.0 model, only the extended parts is presented in the figure. As highlighted by different colors, FraPPE 2.0 is orga-

Listing 4.3: *Fragment of the model that represents a* SYTHETICFRAME

```
frFrame:1356995700000 a fr:SyntheticFrame ;
    fr:contains frPixel:1356995700000_39462,
    frPixel:1356995700000_39463 ;
    fr:samplingTime [ a time:Instant ;
        time:inXSDDateTime "2013-01-01T00:15:00"^^xsd:dateTime ];
    prov:wasGeneratedBy frSynthesize:1356995700000 ;
    fr:wasSynthesizedFrom frFrame:1356994800000,
        frFrame:1356995100000,
            frFrame:1356995400000,
            frFrame:1356995700000.

frSynthesize:1356995700000 a prov:Activity ;
  prov:startedAtTime "2013-01-01T00:15:00"^^xsd:dateTime .

frPixel:1356995700000_39462 a fr:Pixel;
    fr:isIn frFrame:1356995700000 ;
    fr:refers frCell:Cell_1 ;
    fr:hasValue "37"^^xsd:integer.

frPixel:1356995700000_39463 a fr:Pixel ;
    fr:isIn frFrame:1356995700000 ;
    fr:refers frCell : Cell_2 ;
    fr:hasValue "65"^^xsd:integer.
```

nized in different interconnected parts: the white one is related to time, the light gray one to content, and the dark gray one to provenance.

The temporal fragment (the classes Frame, Pixel and Event) and the spatial fragment (the classes Grid, Cell and Place) are inherited from FraPPE 1.0. For a detailed description of these parts we refer the reader to Section 4.2.1

In FraPPE 2.0, the content can be associated to the time-varying classes and carries information about the event it is associated to. At event level the content can be Original or Augmented. The original content represents a simple measure or description of a phenomenon, while any enrichment of an original content produces an augmented content. For instance in a *Tweet* related to the Museum of Modern Art (MOMA), the OriginalContent, in form of a text, contains different entities presented in different surface forms (e.g., moMA, Museum of Modern Art, etc). The Augmentation allows to link those surface forms to a single db-pedia entity[9] (see Listing 4.4). The content related to Pixel or Frame is Synthetic and it is derived by processing event-related contents.

In the extended provenance fragment of FraPPE 2.0, the Agent concept is explicitly defined. Each activity is performed either by a HumanAgent or by a SoftwareAgent. Consequently, the two Agents wasAssociatedTo, respectively, HumanActivity or SoftwareActivity. On the one hand, an example of HumanActivity is the Create activity, exploited to create an OriginalContent. On the other hand, a SoftwareActivity can be exemplified by the Synthesize activity, used to create a Synthetic content associated to a Pixel or a Frame.

We developed FraPPE 2.0 keeping in mind the evaluation based on Tom Gruber's principles presented in Section 4.2.2 and it also complies to them. FraPPE 2.0 keeps satisfying the *clarity* principles because we add common terms in provenance and content analytics domains. FraPPE 2.0 remains as general as possible in order to satisfy the *minimal encoding bias* and the *minimal ontological commitment* principles. FraPPE 2.0

[9] http://dbpedia.org/page/Museum_of_Modern_Art

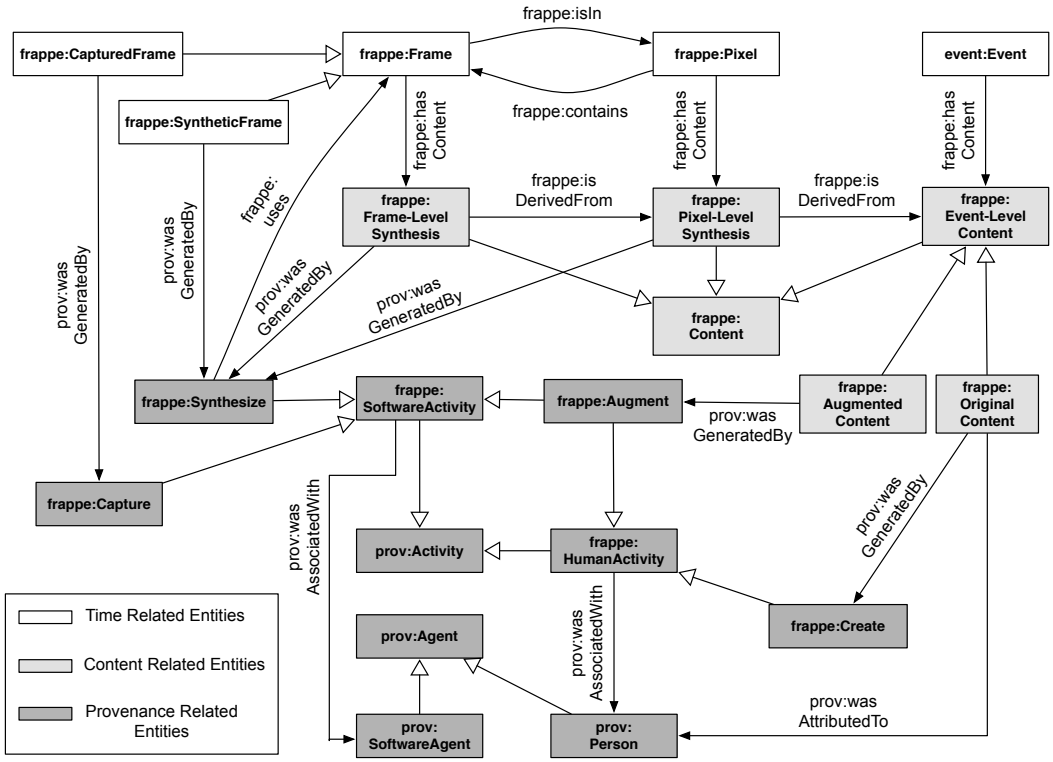


Figure 4.5: *The extended version of the FraPPE model, represented as a colored UML diagram highlighting time-, content-, and provenance- related concepts.*

is still *coherent* and the use cases in the Chapter 7 demonstrates its *extendibility*.

4.4 FraPPE 2.0 and Urban Data Analysis

In this section, we summarize the methods we use in exploiting FraPPE to enable the three classic dimensions of the urban data analysis (see Section 3.2). The use cases, presented in Chapter 7, offer an overview of the FraPPE capabilities in enabling all the classic urban data analysis categories.

Concerning the Content Analysis, as mentioned in the Section 4.3, FraPPE enables the association of the information content to an event, with all the indirect information related to time and space.

Listing 4.4 represents the sending event of a micropost by a social media user. The listing mainly focuses on the FraPPE 2.0 extensions (content and provenance). According to the FraPPE conceptual model, the messageSending Event is described by a set of properties, including the Event-RelatedContent. FraPPE 2.0 distinguishes between two different types of this content, the OriginalContent and the AugmentedContent. In the example, the OriginalContent is the text of the message and is produced by a Create activity, performed by a Person. Contrariwise, the AugmentedContent is created by the DBpediaSpotlight SoftwareAgent that automatically extract the DBpedia entities from the OriginalContent.

FraPPE enables the Spatial Analysis mainly through the concepts of Grid and Cell. Moreover, thanks to their common sense, the Grid and Cell concepts improve the under-

Listing 4.4: *The FraPPE representation of a social message*

```
@prefix frappe:<http://streamreasoning.org/ontologies/frappe#>
@prefix frPlace: <http://streamreasoning.org/frappe/Places/> .
@prefix frEvent: <http://streamreasoning.org/frappe/Events/> .
@prefix frContent: <http://streamreasoning.org/frappe/Content/> .
@prefix frAct: <http://streamreasoning.org/frappe/Activity/> .
@prefix frAgent: <http://streamreasoning.org/frappe/Agent/> .
@prefix frEntity: <http://streamreasoning.org/frappe/Entity/> .
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>
@prefix time:<http://www.w3.org/2006/time#>
@prefix prov:<http://www.w3.org/ns/prov#>
@prefix event:<http://purl.org/NET/c4dm/event.owl#>
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>

frEvent:E a frappe:messageSending ;
  a event:Event ;
  event:time [ a time:Instant ;
    time:inXSDDateTime "2018-07-10T10:00:00"^^xsd:dateTime ] ;
  frappe:location frPlace:P ;
  frappe:hasContent frContent:oc  ;
  frappe:hasContent frContent:ac  .

frContent:oc a frappe:OriginalContent ;
  frappe:litContent "At Museum of Modern Art to see Claude Monet, Water Lilies #
      moMA"^^xsd:string ;
  prov:wasGeneratedBy frAct:cActivity ;
  prov:wasAttributedTo frUser:user .

frAct:cActivity a frappe:Create ;
  a frappe:HumanActivity ;
  prov:wasAssociatedWith frAgent:user .

frAgent:user a frappe:HumanAgent ;
  a prov:Person .

frContent:oc a frappe:AugmentedContent ;
  frappe:dbPediaEntity frEntity:MOMA ;
  frappe:dbPediaEntity frEntity:Claude_Monet ;
  frappe:dbPediaEntity frEntity:Water_Lilies ;
  prov:wasGeneratedBy frAct:aActivity .

frEntity:MOMA a prov:entity ;
  prov:value ".../Museum_of_Modern_Art"^^xsd:anyURI ;

frEntity:Claude_Monet a prov:entity ;
  prov:value ".../Claude_Monet"^^xsd:anyURI ;

frEntity:Water_Lilies a prov:entity ;
  prov:value ".../Water_Lilies_(Monet_series)"^^xsd:anyURI ;

frAct:aActivity a frappe:Augment ;
  a frappe:SoftwareActivity ;
  prov:wasAssociatedWith frAgent:sa .

frAgent:sa a frappe:SoftwareAgent ;
  rdfs:label "DBpediaSpotlight"^^xsd:string;
```

standability and navigability of geographical-based content. They can be instantiated in multiple ways: we may define different types of grids and cells, based on the specific data sets and on the analysis needs. We identify three main categories of grids:

- *Regular squared grid*: a regular Grid dividing the physical space in cells that are uniform for shape, size, and positioning. For instance, in many of our experiences around the city of Milan (see Section 7.1), we defined a Grid of 100 x 100 Cells, each Cell having a size of 250 x 250 meters.
- *Irregular grid with official business-driven meaning*: a Grid of Cells that are different in shape, size and orientation based on some official definition (e.g., the boroughs or zones of a city) or based on some business specification (e.g., the commercial areas of the city). An example of this can be the official city districts defined by the municipality or the areas where a large Event is located. During our experiences in Milan we used the definitions of the official MDW areas to perform aggregated analysis on the data (see Section 7.1).
- *Irregular grid with data-driven definition*: a Grid of Cells defined bottom-up based on the domain data available or on partial analysis and aggregations already performed on them. Some examples include the areas served by different electricity sub-stations, the mobile phone cell coverage, or the areas where mobile phone presence can be clustered with sufficient precision with respect to the location of the antennas. During our work in Como (see Section 7.3), we experienced data-driven definition of different areas related to the mobile network coverage.

Another important feature of the Grid is the coverage of the area of interests. We can define Grids with **total coverage** or **partial coverage**. Typically, regular Grids tend to feature total coverage, while irregular ones, especially when defined starting from business requirements, may offer only a partial coverage of the area.

All the relevant spatial analysis exposed in Section 3.2 (i.e., Dispersion, Distance and relation to places, Correlation and Prediction) can be performed exploiting the concept of Grid and Cell.

The Temporal analysis in FraPPE can be described as the study of the evolution and spreading of signals captured by Pixels, which refers to Cells, over time in different Frames. FraPPE enables all the categories of the temporal analysis described in Section 3.2. The Description in FraPPE describe the signal captured by Pixel-level contents to create a time-series. FraPPE enables the Correlation, Prediction, Anomaly detection and Causality analysis exploiting the concepts of Pixel and Frame.

Finally, FraPPE exploits the links between Cell and Pixel, Grid and Frame to enable the combination of time and space analysis.

4.5 Conclusion

In this chapter, we study the problem of modeling spatio-temporal data to enable analyses that involve time, space and content aspects of the data. The growing availability of geo-located time-varying data, in particular in the urban environment, increased the needs for an holistic conceptual model to describe the data itself, its dynamics and to enable advanced analysis.

To address this problem, we proposed FraPPE conceptual model. FraPPE exploits digital image processing terms to tame three main dimensions of analysis: space, time, and content. It uses image processing common terms to create a bridge between the data engineers and visual interface designers and enables visual analytics on geo-spatial time varying data.

We first developed FraPPE 1.0 using state of the art methodology (METHONTOLOGY). It is formalized using OWL2 and reuses already existing ontologies (see Section 4.2.1 for further details). We then extended FraPPE to version 2.0 by adding concepts related to the provenance and the content (see Section 4.3 for additional details on the extension).

In order to validate Hypothesis Hp.1.1, we checked the adherence of FraPPE 1.0 to the five Tom Gruber's principles (see Section 4.2.2). The *clarity*, *minimal encoding bias* and *coherence* is respected by construction. Infact, the FraPPE 1.0 definitions are documented in natural language, they are formalized using OWL2 standard and all the inferred data is meaningful and consistent. Moreover, FraPPE 1.0 requires a *minimal ontological commitment* because it easily allows specialization, while its *extendibility* is ensured by the number of use cases that are based on it.

Our extended usage of FraPPE 1.0 in real world use cases (see Chapter 7) pushed us to create FraPPE 2.0 that contains the formalization of the concepts that we used more often in our use cases. They are related to the provenance of the information and to the content of the events. Also FraPPE 2.0 results *clear*, *coherent*, *extendable* and with *minimal encoding bias*. Moreover, it still requires a *minimal ontological commitment*, since the new concepts has been formalized because they are shared by two or more use cases.

The overall evaluation, based on Tom Gruber's Principles, validates the Hypothesis Hp.1.1. The presentation of the validation of the Hypothesis Hp.1.2 is postponed to Chapter 7 because it is based on the empirical evaluation of the guessability of the use cases' visualizations.

CHAPTER 5

Computational Model

FINAL - TO BE READ ONE LAST TIME

In this chapter, we propose RI≻EЯ[1] – a variety-proof computational model to deal with streaming data. Section 5.1 introduces the problem of dealing with data characterized by high *Variety* and *Velocity* without forgetting *Volume*. In Section 5.2, we propose a) the background concepts that underpin RI≻EЯ, b) the semantics and textual syntax of the operators of RI≻EЯ and c) the Pipeline Definition Language – a graphic language to ease the modeling of RI≻EЯ plans.

5.1 Introduction and Problem Statement

In our case studies (see Chapter 7), we noticed that data can come from different sources that vary in format (*Variety*) and size (*Volume*), but it always flows (*Velocity*). Even what we normally call "*static data*", e.g., a city street network, is not immutable over time, it slowly evolves.

In 2013, we presented SLD (see Section 2.4), a middleware to ease the deployment of an RSP engine in a real world scenario characterized by heterogeneous streaming data. In five years of SLD usage, we learned that using RDF streams is valuable when (i) data are naturally represented as graphs, (e.g., micro-posts in the larger social graph) and when (ii) the availability of popular vocabularies eases the development of adapters that semantically annotate the external data flows. For instance, we wrote adapters that annotate streams from the major social networks using SIOC [130].

However, we also found out several weaknesses of the RDF-only approach: (i) RDF streams cannot be found in the wild, yet, JSON is largely used in practice (e.g., Twitter

[1] The name RI≻EЯ, and its graphics, is inspired by the data continuously flowing one way into the system.

Streaming APIs[2] and W3C activity stream 2.0 working draft[3]) and (ii) the results of a continuous computation are often relational and forcing them into an RDF stream is suboptimal.

Those reflections inspired the idea to work with the data in its original format as long as possible to reduce the latency caused by the data transformations at ingestion time. We named this approach *Lazy Transformation*.

In order to investigate how *Lazy Transformation* helps in address our research questions – *Is it possible to continuously ingest and reactively analyses a variety of streaming urban data in order to visualize emerging patterns and their dynamics?* – we formulate the hypothesis:

Hp.2.1 The implementation of a streaming computational model that defers as long as possible the data transformation demands less resources and better approximates the correct answer under stress conditions than an implementation of a computational model that cast data into RDF at ingestion time.

5.2 RI≻EЯ

In the next sections, we present RI≻EЯ, a variety proof streaming computational model built around the idea of *Lazy Transformation*.

In Section 5.2.1, we introduce the background concepts that underpin RI≻EЯ. Section 5.2.2 presents in detail the semantics and the textual syntaxes of the RI≻EЯ's operators, the Pipeline Definition Language (PDL) – a graphic syntax to abstract the implementation complexity – and, eventually, examples of physical languages (e.g., EPL, SQL, SPARQL, etc.) to implement the RI≻EЯ's operators. All of those concepts are presented through a running example. Finally, in Section 5.3, we present a reference architecture for systems that implement RI≻EЯ.

5.2.1 Preliminaries

Based on the considerations resulting from the development of our conceptual model (see Chapter 4) and on our past experiences (see Chapter 7), we identify two principles that inspire RI≻EЯ.

(P1) *Everything is a data stream.* According to this principle, a variety-proof stream processing engine must indifferently ingest data with different velocities from any sources and of any size.

For instance, the movements of a car is a *fast* data stream where the information flow records the identity, the positions and the speeds of the cars. In this case, the distance between two subsequent observations can be seconds. On the other side, a city road is a *slowly evolving* data stream, where the information flow contains, for instance, the information about the addition of a bike lane. In this second case, the distance between two subsequent observations can be days or months.

The continuous nature of data streams, and the importance of the information extracted by the most fresh data, require such a category of engine to avoid data loss and, consequently, to implement our second principle: **(P2)** *Continuous Ingestion*. The data

[2] https://dev.twitter.com/streaming/overview
[3] http://www.w3.org/TR/activitystreams-core/

in input is continuously captured by the system and, once arrived, it is marked with an increasing timestamp. Notably, some data sources may natively include their own timestamp too (namely, the application timestamp, presented in Section 2.1.1). It is worth to note that a continuous ingestion mechanism helps to avoid data losses, but continuous analysis is not always needed; an analysis can be reactive even if postponed (see Section 6.4).

In order to challenge the hypothesis Hp.2.1, we propose the *Lazy Transformation* approach. A variety-proof stream processing engine operates on the data in its original format as long as it can, and it transforms data only if it really needs to do so. Indeed, operations like projections, filters or aggregations can operate on generic data without requiring to cast all data in a single format (such as RDF). Therefore, for those operations, we can delay transformations. Contrariwise, a join operation, normally, first requires to cast data in a common format (e.g., the relational one).

So, this kind of system must rely on data of generic type T.

Definition 5.2.1. *(Type to-be-specified-later) A type to-be-specified-later* T *represents the generic type of the atomic object flowing into the system.*

Together with the definition of time (already reported in Section 2.1.1), we can define the information flowing into this kind of systems as a Generic Data Stream.

Definition 5.2.2. *(Time) The time $\mathcal{T}$ is an infinite, discrete, ordered sequence of time instants $(\tau_1, \tau_2, ..., \tau_n)$, where $\tau_i \in \mathbb{N}$. A time unit is the difference between two consecutive time instants $(\tau_{i+1} - \tau_i)$ and it is constant.*

Definition 5.2.3. *(Generic Data Stream) A Generic Data Stream $S\langle T \rangle$ is a potentially unbounded sequence of timestamped data items (d_i, τ_i):*

$$S = (d_1, \tau_1), (d_2, \tau_2), \ldots, (d_n, \tau_n)$$

where d_i is of type T*, $\tau_i \in \mathcal{T}$ is the associated time instant and $\tau_i < \tau_{i+1}$.*

Note that the time instants associated to data items is monotonically increasing. We do not allow contemporary ingestion, because we consider the time as a form of punctuation. So, the data items d_i in a Generic Data Stream $S\langle T \rangle$ is of *types to-be-specified-later* T, and, for instance, it can be, indifferently, a tree representation of a JSON document, a set of tuples in CSV or in parquet format, or a graph in RDF.

Generic Functions and *Generic Types* [131] represents the natural abstraction to model the operations that manipulate information in accordance with the *Lazy Transformation* approach.

Let us now define a Generic Time-Varying Collection as:

Definition 5.2.4. *(Generic Time-Varying Collection) A Generic Time-Varying Collection $C\langle T \rangle$ is a mapping from $\mathcal{T}$ to a finite but unbounded bag of data items d_i, where d_i is of type* T*.*

Differently from a Generic Time-Varying Collection, a Generic Instantaneous Collection defines an unordered bag of data items at a specific time instant.

Definition 5.2.5. *(Generic Instantaneous Collection) A Generic Instantaneous Collection $C\langle T \rangle(\tau)$ is the bag of data items in a collection at τ, a given point in time .*

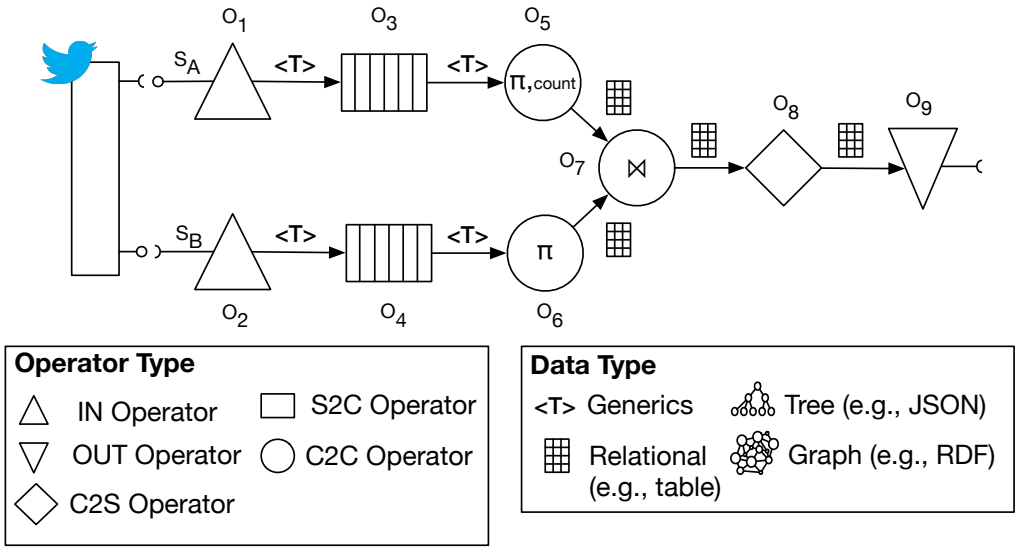


Figure 5.1: *Pipeline that presents an example of the operators and of the typical data type produced during the computation.*

5.2.2 RI≻EЯ's Operators and the Pipeline Definition Language

RI≻EЯ computational model enables users to define computational plans, in the form of pipelines (formally DAGs[4]), composed by different operators that take care of ingesting, processing and emitting Generic Data Streams. In order to ease the definition of the computational plans, we propose the Pipeline Definition Language (PDL). It defines the graphical syntax of the operators.

The reader will be guided into the details via the running example depicted in Figure 5.1. The presented operators and symbols will be detailed discussed in the next paragraphs.

Example 5.2.1. *The example, depicted in Figure 5.1, represents the pipeline to deal with a typical social media analytics use case. The inputs to the pipeline are the post stream S_A and the users' friend network, the stream S_B, in the form of graph. The two streams have to be joined in order to connect users with common friends in the same location.*

More formally, Figure 5.2 depicts the five classes of RI≻EЯ operators and their interactions. The operators, defined as S2C⟨T⟩, C2C⟨T, T′⟩ and C2S⟨T⟩ in RI≻EЯ, are inspired to the CQL processing model (see Section 2.1.1), and allow to move from S⟨T⟩ to C⟨T⟩ and vice-versa. In addition to the CQL-like operators, we introduce the *ingestion* (defined as IN⟨T⟩ in RI≻EЯ) and *emission* (defined as OUT⟨T⟩ in RI≻EЯ) operators. They, respectively, ingest and emit external data flow to/from a system implemented using RI≻EЯ computational model.

In the next paragraphs, we report the details of the operators and present, for each of them, its formal definition, its graphical syntax in PDL, the examples of physical languages for its implementation and its role in the running example (see Figure 5.1).

Ingestion Operator

[4] A finite directed graph with no directed cycles.

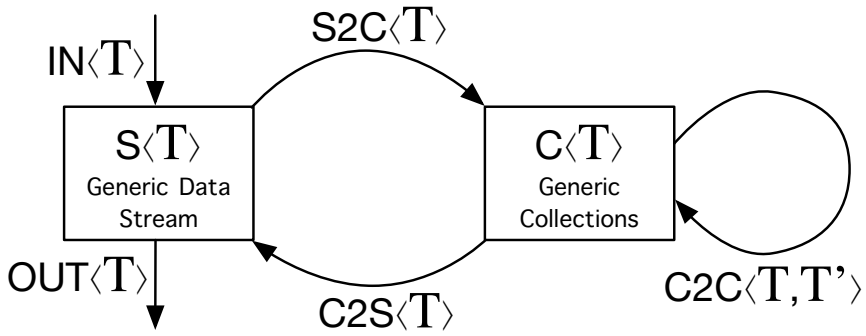


Figure 5.2: *Overview of* RI≻EЯ *operators.*

Definition 5.2.6. *(Ingestion operator) A IN*⟨T⟩ *operator takes an external data flow and inject the items into the system creating a new S*⟨T⟩*. The ingestion operator is type-agnostic, it works independently from the external source data-type. It always transform the items in the external data flow into internal generics (defined as* ⟨T⟩*).*

In PDL we introduce the symbol $\triangle$ to represent the Ingestion operator.

Example 5.2.2. *(cont'd) The external data flows S_A and S_B need to be ingested in order to be analyzed. O_1 and O_2 are implementations of IN*⟨T⟩ *operator for Twitter. They contain the logic for connecting to twitter and retrieve the requested informations. On the one hand, O_1 takes care of the external data flow S_A, by connecting to the Twitter streaming API and ingesting JSON Trees as generics* ⟨T⟩*. Listing 5.1 shows the resulting JSON.*

Listing 5.1: *Example of the data resulting by the ingestion operation performed by O_1.*

```
{
  "data": [{
    "user_id": ":Alice",
    "content": "breathless at #moma",
    "hashtag": [
       {
         "tag_id":"t1"
         "text":"moma"
       }
    ],
    "latitude":40.761620,
    "longitude":-73.977257,
    "time":"2018-09-30T09:00:00"
   }
  ]
 }
```

On the other hand, O_1 manages the external data flow S_B, by polling the Twitter REST API and ingesting RDF Graph as generics ⟨T⟩*. Listing 5.2 shows the resulting RDF Graph.*

Stream-to-collection Operator

Listing 5.2: *Example of the data resulting by the ingestion operation performed by O_2.*

```
:Alice a :User ;
    :userName "Alice"^^xsd:string ;
    :birthDate "1980-06-21"^^xsd:date ;
    :hasFriend :Bob .
```

Definition 5.2.7. *(Stream-to-collection operator) A S2C⟨T⟩ operator transforms a portion of a potentially infinite Generic Data Stream S⟨T⟩ into a Generic Time-Varying Collection C⟨T⟩. S2C⟨T⟩ operator is type-agnostic, the operation is completely independent from* T.

Similarly to CQL, the implementations of the S2C⟨T⟩ operator are based on the concept of *sliding window*. In particular, we can define the concept of Generic Data Window.

Definition 5.2.8. *(Generic Data Window) A window W⟨T⟩(S) is a set of data items (d_1, ..., d_n), of type* T*, extracted from a Generic Data Stream S⟨T⟩.*

Exploiting this concept we can now define two classes of Generic Data Window. The *time-based* Sliding Generic Data Window and the *tuple-based* Generic Data Window.

Time-based Sliding Generic Data Window operator defines its output by sliding an interval of K time units over the stream S⟨T⟩.

Definition 5.2.9. *(Time-based Sliding Generic Data Window) A Time-based Sliding Generic Data Window on a stream S⟨T⟩ takes a time-interval* K *as a parameter and is specified by following S⟨T⟩ in the query with [Range* K*]. The output Generic Instantaneous Collection C⟨T⟩(τ) of S⟨T⟩[Range* K*] is defined as:*

$$C\langle \mathrm{T}\rangle(\tau) = \{s \mid \langle s, \tau'\rangle \in S\langle \mathrm{T}\rangle \wedge (\tau' \leq \tau) \wedge (\tau' \geq \max\{\tau - \mathrm{K}, 0\})\}$$

Tuple-based Sliding Generic Data Window operator defines its output by sliding a window of size N data items over the stream S⟨T⟩.

Definition 5.2.10. *(Tuple-based Sliding Generic Data Window) A Tuple-based Sliding Generic Data Window takes a positive integer* N *as a parameter and is specified by following S⟨T⟩ in the query with [Rows* N*]. The Generic Instantaneous Collection C⟨T⟩(τ) of S⟨T⟩[Rows* N*], consists of data items of type* T *obtained from the* N *elements with the largest timestamps in S⟨T⟩ no greater than τ.*

In PDL, we use the symbol ▭ to represent the S2C⟨T⟩ operator. In particular, the symbol related to the S2C⟨T⟩ operators can give an hint about the operator's implementation. For instance, the S2C⟨T⟩ operators O_3 and O_4, proposed in Figure 5.1, are implemented as two windowers.

Example 5.2.3. *(cont'd) The operators O_3 and O_4 apply Time-based sliding window operations to the streams resulting from the operators O_1 and O_2. Operator O_3 is a S2C⟨Tree⟩, while O_4 implements a S2C⟨Graph⟩*

As anticipated in the introduction, we report the physical implementation of some operators. In this case, both O_3 and O_4 can be implemented exploiting ESPER engine (see Section 2.1.3) and using the EPL clause TIME to create sliding windows with 1

minute duration. Note that the operators are completely agnostic to the data type. O_3 and O_4 are the same query, except for the name of the stream (see Linting 5.3 and Linting 5.4).

Listing 5.3: *EPL query, applied by O_3 operator, to window the stream of JSON trees*

```
SELECT * FROM treeEvent#TIME(1 MIN)
```

Listing 5.4: *EPL query, applied by O_4 operator, to window the stream of RDF graphs*

```
SELECT * FROM graphEvent#TIME(1 MIN)
```

The proposed EPL queries, produce two Generic Instantaneous Collections that contain, respectively, the JSON trees and the RDF graphs ingested in the last minute, without apply any transformation to the data items.

Before producing the output, in order to enable query operation for the downstream operators, O_3 merges the different JSON trees in a single JSON tree which contains an array of elements (see Listing 5.5). The operator O_4 works in a similar way on the input stream of RDF graphs. It windows the stream and creates a single RDF graph in output that contains all the information that entered the system in the last minute (see Listing 5.6).

Collection-to-collection Operator

Definition 5.2.11. *(Collection-to-collection operator) A C2C$\langle$T, T$'\rangle$) operator transforms one or more Generic Instantaneous Collection C$\langle$T$\rangle(\tau)$ in input into a single Generic Instantaneous Collection C$\langle$T$'\rangle(\tau)$.*

The implementations of this class of operators are tailored on the different data format, e.g., JSONiq operators process JSON, SQL operators process relational table, SPARQL operators process RDF graph, etc. Notably, T and T$'$ can be of different types, but they can also be of the same type. For instance, a filter on a table, on a tree or on a graph extract tuples, sub-trees or sub-graphs maintaining the original data type. Contrariwise, as we noticed above, a count aggregation transform the original data type into a table.
In PDL, we use the symbol $\bigcirc$ to represent the C2C$\langle$T, T$'\rangle$ operator.

Example 5.2.4. *(cont'd) The operator O_5 is a C2C$\langle Tree, Relational\rangle$ and performs an aggregation on the output of the O_3 operator. The JSON object from O_3 contains the list of the posts that have entered the system in the last minute. Differently from the generic EPL queries (see Listing 5.3 and Listing 5.4), in order to perform the aggregation query (presented in the Listing 5.7) the operator O_5 needs to know the data format in advance. As specified in the operator's definition, the implementation of the operator change with the input data type and can be choose at design time.*

The query in Listing 5.7 counts the elements in the list, grouped by user_id, and produces a relational table.

Table 5.1 presents the output of O_5: a relational table with the user_id and the associated count.

Listing 5.5: *Example of result of the O_3 operators.*

```
{
  "data": [{
    "user_id": ":Alice",
    "content": "breathless at #moma",
    "hashtag": [
       {
         "tag_id":"t1"
         "text":"moma"
       }
    ],
    "latitude":40.761620,
    "longitude":-73.977257,
    "time":"2018-09-30T09:00:00"
   },{
    "user_id": ":David",
    "content": "Morning at #moma",
    "hashtag": [
       {
         "tag_id":"t1"
         "text":"moma"
       }
    ],
    "latitude":40.761620,
    "longitude":-73.977257,
    "time":"2018-09-30T09:00:30"
   },{
    "user_id": ":Carl",
    "content": "Spending my day at #MoMA",
    "hashtag": [
       {
         "tag_id":"t2"
         "text":"MoMA"
       }
    ],
    "latitude":40.761620,
    "longitude":-73.977257,
    "time":"2018-09-30T09:00:45"
   },{
    "user_id": ":Alice",
    "content": "#picasso at #moma",
    "hashtag": [
       {
         "tag_id":"t1"
         "text":"moma"
       }, {
         "tag_id":"t3"
         "text":"picasso"
       }
    ],
    "latitude":40.761620,
    "longitude":-73.977257,
    "time":"2018-09-30T09:00:55"
   }
  ]
}
```

Listing 5.6: *Example of result of the O_4 operator.*

```
:Alice a :User ;
    :userName "Alice"^^xsd:string ;
    :birthDate "1980-06-21"^^xsd:date ;
    :hasFriend :Bob .
:Bob a :User ;
    :userName "Bob"^^xsd:string ;
    :birthDate "1965-08-10"^^xsd:date ;
    :hasFriend :Alice ;
    :hasFriend :Carl .
:Carl a :User ;
    :userName "Carl"^^xsd:string ;
    :birthDate "1965-05-15"^^xsd:date ;
    :hasFriend :Bob .
```

Listing 5.7: *JSONiq Query for aggregating JSON element, applied by the operators O_5.*

```
FOR $user IN COLLECTION("data")
GROUP BY $user_id := $user.user_id
RETURN { "user_id" : $user_id, "count" : COUNT($user) }
```

In the other branch of the pipeline, which manage the slowly evolving data that enter the system in RDF graph format, the operator O_6, similarly to the operator O_5, extracts information related only to the users who have at least one friend in common.

Listing 5.8: *SPAQL query applied by operator O_6 to the RDF stream to project information about the user.*

```
SELECT ?user_id ?name ?birthDate
WHERE {
    ?user1 a :User;
    :user_id ?user_id ;
    :userName ?name ;
    :birthDate ?birthDate ;
    :hasFriend ?commonFriends .
    ?user2 a :User;
    :hasFriend ?commonFriends .
FILTER (?user1 != ?user2)
}
```

Operator O_6 is a C2C$\langle Graph, Relational \rangle$. It extracts information from the slowly evolving RDF graph stream via the SPARQL query presented in Listing 5.8 and creates a relational table as presented in the Table 5.2.

The operator O_7 joins two collections C$\langle Relational \rangle$ from the two branches using the user_id as key and produces an enriched Collection C$\langle Relational \rangle$. Table 5.3 presents the results of the join operation. It contains the personal information of the users with at least one Friend in common, together with the count of posts tweeted in the last minute.

Collection-to-stream Operator

Definition 5.2.12. *(Collection-to-stream operator) The C2S$\langle$T$\rangle$ operator needs a Generic Collection C$\langle$T$\rangle$ as input, to create a new Generic Data Stream S$\langle$T$\rangle$ as output. This operator is used to emit, as a new Generic Data Stream S$\langle$T$\rangle$, the results over time of C2C$\langle$T, T$'\rangle$ operators.*

Table 5.1: *Example of result produced by the O_5 operator.*

user_id	count
:Alice	2
:Carl	1
:David	1

Table 5.2: *Example of result produced by the O_6 operator.*

user_id	name	birthDate
:Alice	Alice	1980-06-21
:Carl	Carl	1965-05-15

Similarly to CQL, RI≻EЯ introduces three classes of C2S⟨T⟩ operators.

Definition 5.2.13. *(Insert Generic Data Stream) The Istream applied to Generic Collection $C\langle \mathrm{T} \rangle$ contains an element $\langle d, \tau \rangle$, with d of type* T*, iff the data item d is in $C\langle \mathrm{T} \rangle(\tau)$ - $C\langle \mathrm{T} \rangle(\tau - 1)$:*

$$Istream(C\langle \mathrm{T} \rangle) = \bigcup_{\tau \geq 0}((C\langle \mathrm{T} \rangle(\tau) - C\langle \mathrm{T} \rangle(\tau - 1)) \times \{\tau\}).$$

Definition 5.2.14. *(Delete Generic Data Stream) The Dstream applied to Generic Collection $C\langle \mathrm{T} \rangle$ contains an element $\langle d, \tau \rangle$, with d of type* T*, iff the data item d is in $C\langle \mathrm{T} \rangle(\tau - 1)$ - $C\langle \mathrm{T} \rangle(\tau)$:*

$$Dstream(C\langle \mathrm{T} \rangle) = \bigcup_{\tau \geq 0}((C\langle \mathrm{T} \rangle(\tau - 1) - C\langle \mathrm{T} \rangle(\tau)) \times \{\tau\}).$$

Definition 5.2.15. *(Relation Generic Data Stream) The Rstream applied to Generic Collection $C\langle \mathrm{T} \rangle$ contains an element $\langle d, \tau \rangle$, with d of type* T*, iff the data item d is in $C\langle \mathrm{T} \rangle$ at time τ:*

$$Rstream(C\langle \mathrm{T} \rangle) = \bigcup_{\tau \geq 0}(C\langle \mathrm{T} \rangle(\tau) \times \{\tau\}).$$

In PDL, we propose the symbol $\diamondsuit$ to represent the C2S⟨T⟩ operators.

Example 5.2.5. *(cont'd) The operator O_8 is a $C2S\langle Relational \rangle$ and produces a stream $S\langle Relational \rangle$. We exploit ESPER and EPL to extract an Istream from the results of the join performed by the operator O_7 (see Listing 5.9)*

Listing 5.9: *EPL query, applied by O_8 operator, to create a stream after the join operation.*

```
SELECT istream * FROM joinEvent#TIME(1 MIN)
```

Emission Operator

Definition 5.2.16. *(Emission operator) The OUT⟨T⟩ operator lets the results of the computation exit the system. As for the ingestion operator, it is type-agnostic. The emission operator takes a S⟨T⟩ in input, and produces an external data flow following a custom logic.*

Table 5.3: *Example of Results produced by the O_7 operator*

user_id	name	birthDate	count
:Alice	Alice	1980-06-21	2
:Carl	Carl	1965-05-15	1

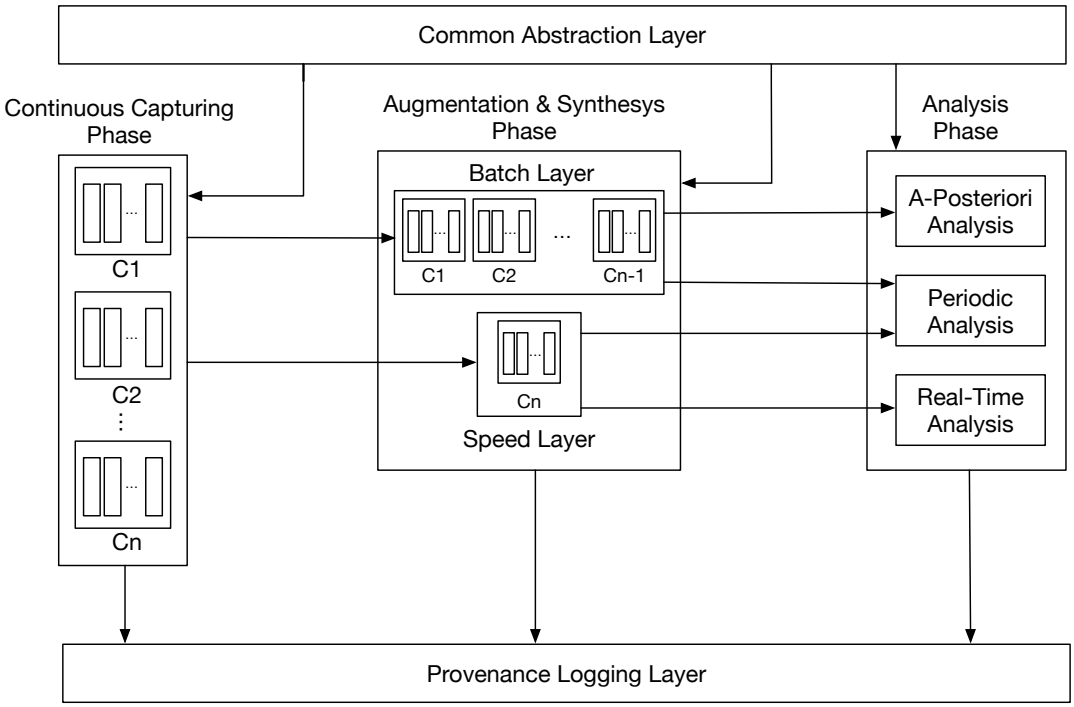


Figure 5.3: *Example of the architecture of a system that implements* RI≻EЯ.

In PDL we propose the symbol $\bigtriangledown$ to represent the OUT$\langle$T$\rangle$ operator.

In the running example, the operator O_9 is an implementation of an OUT$\langle$T$\rangle$ operator for a relational database. It takes a S$\langle$T$\rangle$ in input, where T is relational table with a defined schema, and exploits custom operation to connect and push the results on a target database.

Moreover, in PDL we add packages, represented by $\hexagon$ symbols, as a general purpose mechanism for organizing RI≻EЯ operators into groups. It allows to graphically group different semantically related operators. Typically, it is useful to represent a chain of a S2C$\langle$T$\rangle$, C2C$\langle$T, T$'\rangle$ and C2S$\langle$T$'\rangle$ operators that represents the transition from one or more S$\langle$T$\rangle$ to a single S$\langle$T$'\rangle$. For instance, with reference to Figure 5.1, a package could have in input the Generic Data Streams produced by the ingestion operators O_1 and O_2 and in output the S$\langle$*Relational*$\rangle$ generated by operator O_8.

5.3 Reference Architecture

Figure 5.3 presents an reference architecture for systems that implement RI≻EЯ computational model. Information enters from the left and exits to the right going through three operational phases. Phase 1 (namely, Continuous Capturing Phase) continuously captures data over time. Phase 2 (namely, Augmentation and Synthesis Phases) enriches,

manipulates and transforms captured data. Phase 3 (namely, Analysis Phase) analyses data to compute results. Accordingly to the RI≻EЯ's operators definition, Phase 1 exploits different implementations of the IN⟨T⟩ operator. Phase 2 exploits a combination of S2C⟨T⟩, C2C⟨T, T′⟩ and C2S⟨T⟩ operators implementations. And, finally, Phase 3 exploits OUT⟨T⟩ operator implementations to emit the results.

During the Continuous Capturing Phase the data, which continuously flows in, is just marked with a timestamp, i.e., following the *Lazy Transformation* approach, it is captured in its original form independently from its complexity.

The proposed architecture treats *Volume* as orthogonal to *Variety* and *Velocity*. When *Volume* is present, system must implement the continuous ingestion phase in a partition tolerant way (see Section 6).

The fragment of the architecture, which has in charge the Augmentation and Synthesis Phases, is inspired by a λ architecture (see Section 2.1.2). Let us denote with C_i the information the Speed Layer is able to process while staying reactive, and let us denote with C_n the most recently captured information, and with C_1, ..., C_{n-1} all the data captured. While the Speed Layer processes C_n, the Batch Layer updates C_1, ..., C_{n-2} with the results generated by the Speed Layer while processing C_{n-1}.

The Analysis Phase exploits, based on the information need of the user, indifferently various part of the upstream architecture. The Batch Layer can be used alone for periodic and post-hoc analysis, or in support of the Speed Layer for analysis that needs to compare the most recent data with the historical one. Nevertheless, the speed layer, can be independently used to perform instantaneous analysis.

For instance, a taxi company can exploit the Batch Layer, to synthesize statistics about the cost and the duration of all the rides captured so far in a city. An a-posteriori analysis of those statistics can determine a complete origin-destination matrix for the taxi rides, i.e., a distribution of the durations and the prices of all possible routes from any point to any other point in the city (see Section 4.2.3). At the same time, the taxi company can exploit the Speed Layer to determine the current most profitable routes using the latest incoming data. The comparison between the latest price of the rides (computed in the Speed Layer) with the information in the origin-destination matrix (computed in the Batch Layer) can be useful to foil a fraud.

Two more layers compose the proposed architecture: the Common Abstraction Layer – that contains the abstraction used to model or manipulate data (e.g., FraPPE concepts and RI≻EЯ's operators) and enables OBDA operations – and the Provenance Logging Layer – that contains all the artifact useful to document data lineage and to log the system actions (e.g., in accordance with concepts in the Provenance fragment of FraPPE).

5.4 Conclusion

In this chapter, we investigate the problem of managing data characterized by high variety and velocity without forgetting volume. Taking in account the conceptual model presented in Chapter 4, we concentrate our efforts on the creation of a computational model to deal with data with such characteristics.

We propose RI≻EЯ, a variety-proof streaming computational model based on two main principles (see Section 5.2): **(P1)** everything is a data stream, and **(P2)** Continuous Ingestion. A system based on **(P1)** and **(P2)** can manage flowing data without any data

loss. During this research work, in order to answer the research question, we propose the *Lazy Transformation* and formulate the hypothesis Hp.2.1. A system, which implements the *Lazy Transformation* approach, postpones the data transformation until it can benefit from it.

We present a formal definition of RI⋗EЯ's operators in terms of semantics and textual syntax (see Section 5.2.2), together with the Pipeline Definition Language (PDL) – a graphic language that enables user to create computational plans, in the form of pipelines, to ingest, process and emit data.

RI⋗EЯ represents the formal basis of the implementations proposed in the next chapter (see Chapter 6), that will be exploited to experimentally validate Hp.2.1.

CHAPTER 6

RI≻EЯ Implementations and Evaluations

FINAL - TO BE READ ONE LAST TIME - READ FROM 6.3 ON

In this chapter, we propose different implementations of RI≻EЯ computational model (see Chapter 5) and the evaluations of such implementations from different points of view. Section 6.1 introduces the problem of implementing RI≻EЯ from the scalability perspective. In Section 6.2 we propose Natron – a vertically-scalable implementation of RI≻EЯ –, rvr@Spark and rvr@Hive – two horizontally-scalable implementations of RI≻EЯ. The Section 6.3 and the Section 6.4 presents, respectively, the evaluation of Natron against an already existing system, and an evaluation of Natron against rvr@Spark based on cost-effectiveness.

6.1 Introduction and Problem Statement

In the following sections, we discuss three alternative implementations of RI≻EЯ computational model. Based on our experiences, we identify three situations where the nature of the data, in particular the *Volume*, and the system scalability requirements shape the specific implementation of RI≻EЯ reference architecture (see Section 5.3). In particular, inspired by the benchmarking basic principles (see Section 2.5), we consider the *cost effectiveness* as the most important characteristic of an implementation.

When the amount of data is small and the cost of a complex infrastructure is unaffordable, an ad-hoc implementation results suitable. In this situation, there is no need for scalability and the final artifact can be developed in any language or using any framework, e.g. Python or Java. We work following this direction [25], but we do not report the results because they are out of the scope of this thesis. If the amount of data grows, a scalability requirement arises. An ad-hoc solution results hard to be cost-effective, if compared to a more generic and reusable implementation.

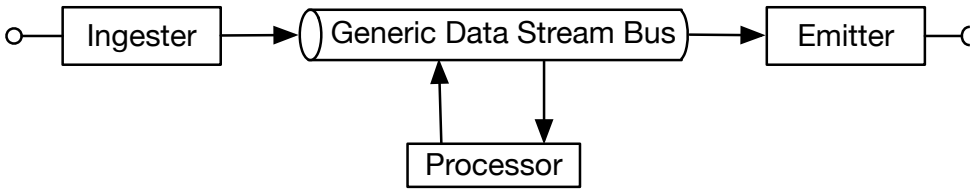


Figure 6.1: *Overview of Natron architecture.*

Conscious that distribution and parallelization does not pay at all scales [89], we developed (i) a vertically scalable single threaded implementation – Natron – and (ii) two horizontally scalable implementations based on Spark – rvr@Spark – and Hive – rvr@Hive. In the next sections, we present Natron, rvr@Spark and rvr@Hive, and the evaluation results that validated the Hypothesis Hp.2.1 (formulated in Chapter 5).

Moreover, in order to cast some light on the research questions taking in account possible alternative implementations, w.r.t. scalability, we formulate the hypothesis:

Hp.2.2 A single-threaded implementation of the streaming computational model from Hp.2.1 is more cost-effective than a distributed implementation of the same model while guaranteeing the reactiveness of the system.

6.2 Implementations

Section 6.2.1 proposes Natron, an implementation based on single-threaded technology able to manage heterogeneous streaming data characterized by medium Volume. When scaling to large volume is required, a single threaded implementation is at risk of loosing cost-effectiveness because, even if the entry cost is much lower than a Big Data implementation, its cost grows exponentially in the size of the data. Therefore, an horizontally scalable solution, using Big Data technology, represents a good choice. The results of the evaluation reported in Section 6.3, convinced us to assume the Lazy Transformation as a principle to be applied in the horizontally scalable implementations of RI≻EЯ computational model. In our work, we employed two different solutions respectively based on Spark (rvr@Spark) and Hive (rvr@Hive). Section 6.2.2 and Section 6.2.3 present an overview of those implementations.

6.2.1 Natron - A Vertically Scalable Implementation

Natron is a single threaded implementation of RI≻EЯ reference architecture (see Section 5.3) able to deal with continuously flowing data characterized by medium Volume, high Variety and very high Velocity. It continuously ingests streaming data represented as a time-stamped data items that are typed, only when needed. The type is declared as an annotation to the captured information.

Figure 6.1 depicts an overview of the Natron internals. The Ingesters allow ingesting external data flows, and push the data on the Generic Stream Bus. As recommended by the *Lazy Transformation* approach, we postpone the transformation as long as possible in the process, only the ingestion time is added. The Processors, e.g. an Information Flow Processor as Esper (see Section 2.1.3), listens to one or more streams $S\langle T\rangle$, computes different operations and produces a new stream $S'\langle T'\rangle$. Emitters allow Natron producing

a new external data flow in multiple formats. In Natron, the window operator can be implemented in two different ways: either using the ingestion timestamp added during the Continuous Capturing Phase, or using an application timestamp, e.g. a time mark added during the Augmentation & Synthesis Phase referring to the notion of Frame in FraPPE.

6.2.2 rvr@Spark - A Horizontally Scalable Implementation Based on Spark

rvr@Spark is an implementation of RI⋗EЯ based on Spark Structured Streaming processing engine (see Section 2.1.3). It enables the users to create pipeline of streaming computation as they are creating a batch computation, and to leave to the Spark SQL engine to manage the incremental update of the results in a transparent way.

rvr@Spark offers different implementations of the IN⟨T⟩ operator that exploits the DataFrames and Datasets API offered by Spark to ingest data from different sources (e.g., filesystem, Kafka, socket, etc.). In the same way, rvr@Spark exploits the sink of Spark Structured Streaming as implementations of the OUT⟨T⟩ operators (e.g., filesystem sink, Kafka sink, Console sink, etc.). The access to the data, during the Ingestion and Augmentation & Synthesis Phase, is guaranteed by OBDA techniques implemented in the various components that exploit FraPPE as data schema. Moreover, the Dataframe APIs enable the implementation of all the RI⋗EЯ's operators and allow the development of the complete stack of layers presented in the RI⋗EЯ reference architecture (see Section 5.3).

Listing 6.1: *Example of Window operator in Spark.*

```
val itemsCounts = inputStream.groupBy(
     window($"ts", "40 seconds", "20 seconds"),
     $"Agg(Count)"
  ).count()
```

In particular, the rvr@Spark implementation of the S2C⟨T⟩ window operator exploits the native Spark Structured Streaming windowing operations on the ingestion time added to the data during the Continuous Capturing phase.

Listing 6.1 presents the code to compute the aggregation *Agg(Count)*, representing the amount of the data items that entered the system in the last 40 seconds, using a window that slides every 20 seconds.

6.2.3 rvr@Hive - A Horizontally Scalable Implementation Based on Hive

rvr@Hive is a distributed implementation of RI⋗EЯ computational model based on Hive (see Section 2.1.3). Hive is a Big Data warehouse solution and is not originally ready for managing streaming data. This limitation can be overcome by chaining, during the Ingestion Phase, the implementations of a IN⟨T⟩ operator and a S2C⟨T⟩ operator (window). This chain enable the system to add the ingestion timestamp, *ts* to each incoming data and to transform the time-varying input into a Hive compatible static format (e.g., Parquet) partitioned by *ts*. The Augmentation & Synthesis Phase exploits OBDA techniques to access data using FraPPE ontology as data schema. The FraPPE Commons Abstractions layer contains the concepts to enrich and transform data according to the FraPPE conceptual model (e.g., adding a reference to FraPPE Frame that groups

the data items by time and space). The final result is then served to the user through implementations of different OUT⟨T⟩ operators that allow the system to save the data in different format (e.g., filesystem, Kafka, websocket).

As for the rvr@Spark, we now focus on the S2C⟨T⟩ window operator in rvr@Hive that exploits the Hive window operator. Differently from Spark, the window operator is not natively supported due to the batch nature of the framework. However, if we augment the data items with a frame ID during the Augmentation & Synthesis Phase, tumbling windows can be implemented grouping by Frame ID.

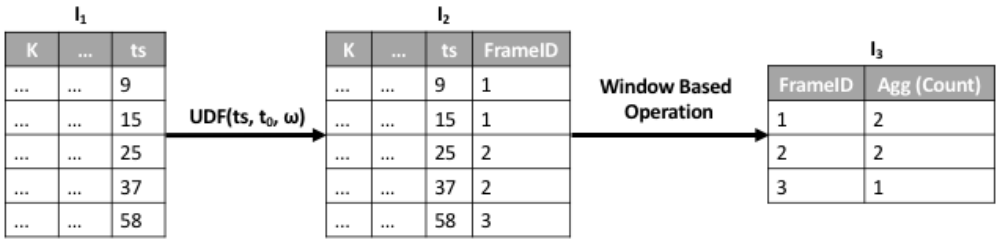


Figure 6.2: *Example of Window operator using Hive.*

Figure 6.2 shows a simple example of a chain of operations that ingest a data stream, augment it with a Frame ID and simulates a tumbling window that counts the number of data items per frame. I_1 represents the data saved on HDFS during the ingestion phase. I_1 is in form of a table containing various attributes and partitioned by the ingestion timestamp *ts*. The data is augmented using a query that uses a User Defined Function (UDF) to attach a Frame ID based on the *ts* and save the result incrementally in a parquet file partitioned by Frame ID. Such an UDF is configured passing the opening time t_0 of the first window, and the length of a Frame ω. In the example $t_0 = 0$ and $\omega = 20$. The Frame groups the data items in windows and enables operation on time-varying data in a batch oriented system such as Hive. I_2 represents the augmented data. The Window Based Operation exploits the Frame ID to perform a simple count aggregation by applying a Group by on the Frame ID. I_3 represents the aggregated data.

6.3 Validation of the Lazy Transformation Approach

As a first evaluation step, we compared Natron against SLD (see Section 2.4). Both of them are single-threaded but, while SLD is based on RDF streams, Natron implements the *Lazy Transformation* approach.

Differently from SLD, Natron i) uses time-stamped generic data items (instead of focusing only time-stamped RDF graphs) and ii) processes them according to their original nature. In particular, we test if a system exploiting i) and ii) results a cheaper (using less memory and CPU), faster (reaching higher maximum input throughput) and more accurate (better approximating the correct answer) version of a streaming computational model.

In the following sections, we expose the problem, the solution design and the experimental settings, and, finally, we bring experimental evidences that validate the hypothesis Hp.2.1.

6.3.1 Problem Settings

As domain, we chose Social Media analysis as done by the Linked Data Benchmark Council (LDBC) in the SNBench[1].

SLD and Natron receive information in the same way, they both connect to a web socket and handle JSON-LD files.

Listing 6.2: *JSON representation of a Twitter micro-post. Due to the lack of space we omitted the context declaration that contains the namespace.*

```
{"@context": { ... },
  "@type": "Collection",
  "totalItems": 1,
  "prov:wasAssociatedWith": "sr:Twitter",
  "items": [{
    "@type": "Post",
    "published": "2016-04-26T15:40:03.054+02:00",
    "actor": {
      "@type": "Account",
      "@id": "user:1",
      "sioc:name": "@streamreasoning"
    },
    "object": {
      "@type": "Content",
      "@id": "post:2",
      "alias": "http://.../2",
      "prov:wasAssociatedWith": "sr:Twitter",
      "sioc:content": "You ARE the #socialmedia!",
      "dct:language": "en",
      "tag": [{
        "@type": "Tag",
        "@id": "tag:3",
        "displayName": "socialmedia"
      }]
    }
  }]
}
```

In Listing 6.2, we propose a JSON-LD serialization of the Activity Stream[2] representation of a tweet as it was injected during the experiments in both systems. The JSON-LD representation of an Activity Stream is a *Collection* (specified by *@type* property) composed by one or more social media items. The *Collection* is described by two properties, i.e., *totalItems* and *prov:wasAssociatedWith*, which tell respectively the number of items and the provenance of the items. The collection in the example contains a *Post* created on *2016-04-26* (*published* property) by an *actor* (Line 6) that produce the *object* (Lines 7-13). The *Actor* has a unique identifier *@id*, a *displayName*, a *sioc:name* and a *alias*. The *Object* has a *sioc:content*, a *dct:language*, zero or more *tag*s, and optionally a *url* and a *to* to represent, respectively, links to web pages and mentions of other actors.

Listing 6.3 shows the RDF produced by SLD in transforming the JSON-LD in Listing 6.2 at ingestion time. The translation exploits well known vocabularies, in particular SIOC[3] to represent the online community information, PROV- O [129] to track the provenance of an item and DCTERMS[4] to represents information about the *object*.

[1] http://www.ldbcouncil.org/benchmarks/snb
[2] https://www.w3.org/TR/activitystreams-core/
[3] http://sioc-project.org/
[4] http://dublincore.org/documents/dcmi-terms/

Listing 6.3: *RDF N3 representation of a Twitter micro-post*

```
<post:2> a sma:Tweet ;
  dcterms:created "2016-04-26T15:40:03.054+02:00"^^xsd:dateTime ;
  dcterms:language "en"^^xsd:string ;
  sioc:content "You ARE the #socialmedia!"^^xsd:string ;
  sioc:has_container "Twitter"^^xsd:string ;
  sioc:has_creator <user:1> ;
  sioc:id "2"^^xsd:string ;
  sioc:link "http://.../status/2"^^xsd:string ;
  sioc:topic <tag:3> .
<tag:3> a sioct:Tag ;
  rdfs:label "socialmedia"^^xsd:string .
<user:1> a sioc:UserAccount ;
  sioc:account_of "StreamReasoning"^^xsd:string ;
  sioc:creator_of <post:2> ;
  sioc:id "1"^^xsd:string ;
  sioc:name "@streamreasoning"^^xsd:string .
```

6.3.2 Solution Design and Experimental Settings

A test consists of sending a constant amount of JSON-LD synthetic data. The data is sent in chunks three times per minute (i.e. at the 10^{th}, the 30^{th} and the 50^{th} seconds of the minute). Each chunk contains the same amount of posts. We tested the configuration for different rates: 1500 posts per minute (i.e., three chunks of 500 posts), 3000 posts per minute, 6000 posts per minute, 9000 posts per minute, 12000 posts per minute and 18000 posts per minute.

The rates and the input methodology test a normal situation for SLD (1500 and 3000 posts per minutes) as well as situations that we know to overload SLD (more than 6000 posts per minute) [22].

the pipelines used for testing the systems (depicted in Figure 6.3a and in Figure 6.3b) are both split into two branch. The first branch produces an *area chart* by computing the number of tweets observed over time, the second one produces a *bar chart* by counting how often hashtags appear in the tweets received in the last 15 minutes.

The two pipelines are coded in SLD and Natron in two different ways. SLD performs the transformations of JSON-LD in RDF by default, on all the input data, independently from the task to perform. Natron keeps the data in its original format as much as possible, i.e., it implements the *Lazy Transformations* approach. In Natron, the results can be continuously computed *i*) using a generic sliding window S2C$\langle Tree \rangle$ operator, which works looking only to the time-stamps of the data items in the generic stream, and *ii*) accessing with C2C$\langle Tree \rangle$ operator implemented as a path expression the *totalItems* property in the JSON-LD file, i.e., the number of items in the collection.

Figure 6.3a presents the two pipelines in SLD with PDL. The input data are translated in RDF as soon as they enter the pipelines by the Ingester O_1, an implementation of an IN$\langle$T$\rangle$ operator. The computations for the area chart and for the bar chart (see the part marked with **A** and **B**) are composed by the same type of components and share the new RDF stream translated by O_1. The pipeline **A** uses two C-SPARQL queries applied to the stream by the operators O_2 and O_3. Both of them represent a chain of 3 different RI≻EЯ operators (a S2C$\langle$T$\rangle$, a C2C$\langle$T, T′$\rangle$ and a C2S$\langle$T′$\rangle$). O_2 (see Listing 6.4) applies a tumbling window of 1 minute, while O_3 aggregates the results using a 15 minutes time window that slides every minute (see Listing 6.5).

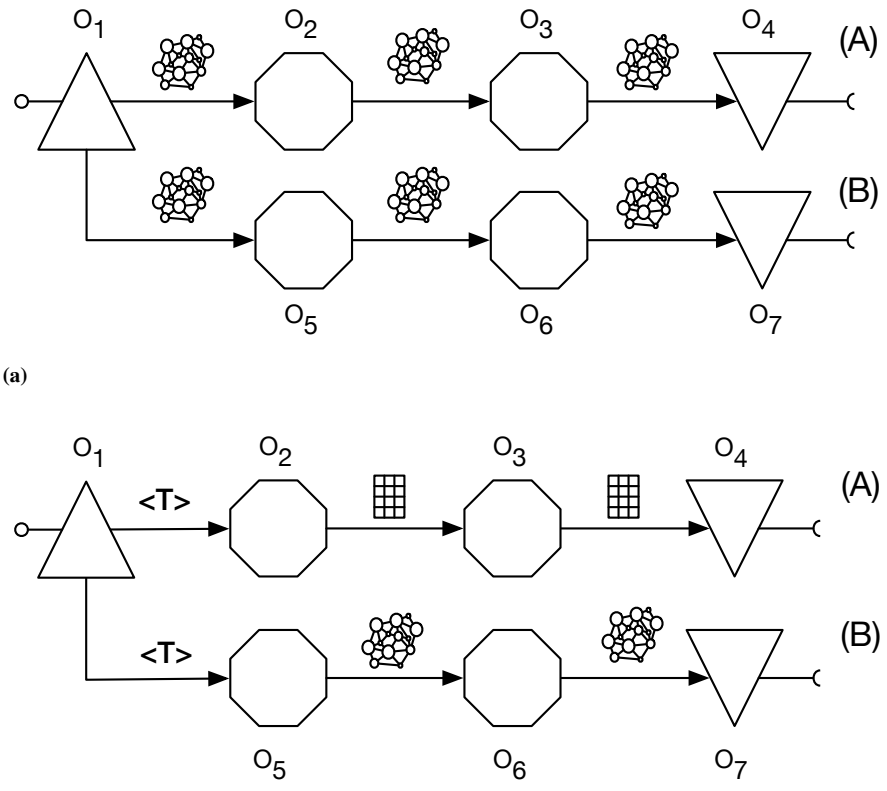


Figure 6.3: *(a) SLD pipeline. Even if both the input and the output data are in JSON-LD format, SLD use RDF graph for the internal computation. (b) Natron pipeline. The input and the output data are both in JSON-LD format, Natron keeps the data in tree format as long as possible during the internal computation. Both the pipelines are presented exploiting the Pipeline Definition Language (see Chapter 5)*

It is worth to note that the first query is an important optimization in terms of memory consumption. It avoids the engine to keep 15 minutes of tweets only to count them. In SLD, we often use this design pattern, we call this first query a *pre-query*. Pipeline **B** also exploits this design; it applies a pre-query through the O_5 operator to reduce the amount of data and, then, a query to produce the final result through the O_6 operator. It is also worth to note that all the C-SPARQL queries use the form REGISTER STREAM ... AS CONSTRUCT ..., because RDF streams are the only means of communication between SLD components. The OUT⟨T⟩ operators of both pipelines, namely O_4 and O_7, make the results available to processes outside SLD. In this case, both O_4 and O_7, write JSON files on disk.

Figure 6.3b presents the pipelines in Natron. As for SLD, the pipeline **A** is for the area chart, while **B** is for the bar chart. Differently from the Ingester in the SLD pipeline, O_1 does not apply any transformation to the input stream, and the data flows in Natron in JSON-LD format. The O_1 operator applies a generic EPL query (presented in Listing 6.6) characterized by a 1 minute long time window. The clauses `FORCE_UPDATE`[5]

[5]The FORCE_UPDATE flow control keyword instructs the view to post an empty result set to listeners if there is no data to post for an interval. Note that FORCE_UPDATE is for use with listeners to the same statement and not for use with named windows. Consider output rate limiting instead.

Listing 6.4: *C-SPARQL query applied by O_2 that count the number of post in the stream from O_1 using a tumbling window of 1 minute.*

```
REGISTER STREAM presocialstr AS
CONSTRUCT { ?id sma:twitterCount ?twitterC }
FROM STREAM <http://.../socialstr> [RANGE 1m STEP 1m]
WHERE {
    SELECT (uuid() AS ?id) ?twitterC
    WHERE {
        SELECT (COUNT (DISTINCT ?mp) AS ?twitterC)
        WHERE { ?mp a sma:Tweet }
    }
}
```

Listing 6.5: *C-SPARQL query applied by O_3 that aggregates the results from O_2 using a 15 minutes time window that slides every minute.*

```
REGISTER STREAM ac AS
CONSTRUCT { ?uid sma:twitterCount ?totTwitter ;
              sma:created_during ?unixTimeFrame
          }
FROM STREAM <http://.../presocialstr> [RANGE 15m STEP 1m]
WHERE {
  SELECT (uuid() AS ?uid)
         ?unixTimeFrame
         (SUM(?twitter) AS ?totTwitter)
  WHERE { ?id sma:twitterCount ?twitter ;
            sma:created_during ?timeFrame .
          ?timeFrame a sma:15mTimeFrame ;
            sma:inUnixTime ?unixTimeFrame
        }
  GROUP BY ?unixTimeFrame
}
```

and START_EAGER[6] tell the stream processing engine, respectively, to emit also empty reports and to start processing the window as soon as the query is registered (i.e., without waiting for the first time-stamped data item to arrive). It is worth to note that this query exploits the event-based nature of the generic stream it is observing. It does not inspect the payload of the events; it only uses their time-stamps.

Listing 6.6: *The generic window query applied by the operator O_1.*

```
SELECT *
FROM event#TIME(1 min,"FORCE_UPDATE, START_EAGER")
```

As explained in Section 6.2.1, processors are the central components of Natron. They can listen to one or more generic stream, compute different operations and push out a generic streams. The type of the input and output streams can be different. The two pipelines use different processors (e.g. RDF translator, windower and SPARQL).

Natron maintains the data format as long as possible in order to reduce the overhead of the translations. It can exploit the tree-based nature of JSON-LD. In pipeline **A**, the

[6]The START_EAGER flow control keyword instructs the view to post empty result sets even before the first event arrives, starting a time interval at statement creation time. As when using FORCE_UPDATE, the view also posts an empty result set to listeners if there is no data to post for an interval, however it starts doing so at time of statement creation rather then at the time of arrival of the first event.

operator O_3 exploits a path expression data to extract *totalItems*, i.e., the number of items in each collection, from the time-stamped JSON-LD items in the generic stream it listens to. It outputs a tuple ⟨timeframe,count⟩ that is aggregated every minute over a window of 15 minutes using an EPL statement.

The pipeline **B** of Natron shares the O_1 operator with the pipeline **A**. The operator O_5 translates the generics in input in RDF graph. This transformation is required to extract information about the hashtags. As for the pipeline **B** of SLD, we use a pre-query design pattern to reduce the amount of data. A SPARQL processor, implementing the operator O_5, applies the SELECT query in Listing 6.7 to every data-item in the stream and pushes out a stream of tuples ⟨hashtagLabel,count⟩. The relational stream is then aggregated with an ESPER processor, which implements O_6, with a 15 minute time window that slides every 1 minute (see Listing 6.8).

Listing 6.7: *SPARQL pre-query applied by the component O_6*

```
SELECT ?htlabel (COUNT(DISTINCT(?mpTweet)) AS ?htTweetCount)
WHERE { ?mpTweet a sma:Tweet ; sioc:topic ?tweetTopic .
        ?tweetTopic a sioctypes:Tag ; rdfs:label ?htlabel }
GROUP BY ?htlabel
ORDER BY DESC(?htTweetCount)
```

Listing 6.8: *EPL query for the bar chart, applied to the stream by the component O_6*

```
SELECT htlabel, SUM(count) as sumHt
FROM HTCountEvent#TIME(15 min)
GROUP BY htlabel
OUTPUT SNAPSHOT EVERY 1 min
```

As for SLD pipelines, the operators O_4 and O_7 of the Natron pipelines are implementations of Emitters and offer the result to the user in the form of JSON files on disk.

6.3.3 Results and Discussion

As key performance indicators (KPIs), we measure the resources consumption of the two systems and the correctness of the results. For the resource consumption, we measure every 10 seconds: *i*) the CPU load of the system thread in percentage, *ii*) the memory consumption of the thread in MB and *iii*) the memory consumption of the Java Virtual Machine (JVM). For the correctness, we compared the computed results with the expected results. Being the input a constant flows of tweets that only differ for the ID, the area chart is expected to be flat and the bar chart is expected to count exactly the same number of hashtags every minute.

Figure 6.4 offers an overview of the results of the experiments. The full results are reported at the end of this section in Figure 6.5. On the X, axis we plot the median of the CPU load in percent, while on the Y axis, we plot the memory allocated by the engine thread. The size of the bubble maps the median of the error of the area chart. Bubbles in the lower left corner correspond to the experiment where we sent 1500 tweets per minute.

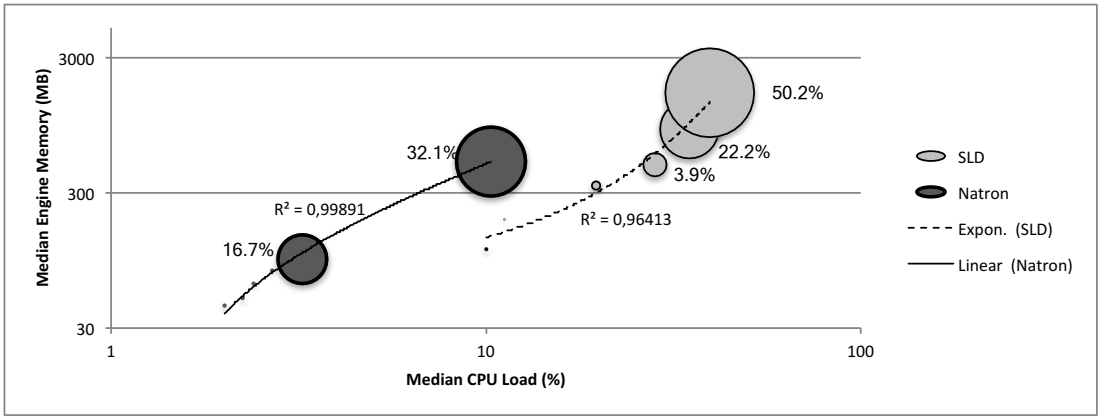


Figure 6.4: *An overview of the experimental results; larger bubbles means greater % errors.*

Increasing the throughput results in more memory consumption and CPU load for both systems. However, Natron consumes less memory than SLD and occupies less CPU. Moreover, Natron presents a linear increment for both these KPIs, while the resource usage for SLD grows exponentially with the throughput (note the R^2 in Figure 6.4). Also the error in the results increases with the throughput: SLD already shows an median error greater than 3% in the bar chart at 3000 tweets per minutes and in the area chart at 9000 tweets per minute; Natron is faster - i.e. it reaches higher maximum input throughput - and more accurate – i.e. it reaches 3% error level only for 18000 tweets per minutes, providing more precise results than SLD.

Figure 6.6 presents the recorded time-series for CPU load and memory usage in both systems. The memory usage graphs contain two different time series. The blue one represents the memory usage of the system thread, while the orange one shows the total memory usage for the JVM.

The memory usage of the system thread accounts for all the components and data in the pipeline. Notably, when the system under testing is not overloaded, the memory usage is constant over time, while when the system is overloaded it grows until the system crashes. The total memory usage of the JVM shows, instead, the typical pattern of the garbage collector that lets the JVM memory grow before freeing it. Also in this case, when the system it is overloaded, the garbage collector fails to free the memory.

During the experiments the median of the memory used by SLD spans from 115 MB, when loaded with at 1500 posts/min, to 1.6 GB, when loaded with 18000 posts/min. For Natron, instead, it spans from 44 MB to 511.5 MB in the same load conditions. The experimental results clearly shows that Natron consumes (in average) three times less memory than SLD.

The same considerations can be proposed for the CPU load. The median of the CPU load spans from 2% to 10% for Natron, while it spans from 10% to 39.5% for SLD. Natron consumes in average 4 time less CPU time than SLD. Moreover, it offers higher level of stability for both the parameters in all the experiments.

The correctness results are summarized in Figure 6.7. The X axis of each plot shows the percentage of error; it ranges from 0% to 100%. The Y axis is the percentage of results with that error; it also ranges from 0% to 100%. A bar as tall as the Y axis in the left side of the graph means that all results where correct. The smaller that bar is and

		SLD					Natron				
post/min	KPI	Min.	1st Qu.	Median	3rd Qu.	Max.	Min.	1st Qu.	Median	3rd Qu.	Max.
1500	memory (MB)	112	114	115	115	116	42	43	44	45	48
3000		188	190	191	192	193	45	48	50	52	58
6000		330	337	340	342	353	52	61	64	67	78
9000		478	481	485	488	504	59	73	80	84	102
12000		684	783	888	956	1015	66	88	97	107	144
18000		871	919	1652	2565	4451	212	392	511,5	597,2	774
1500	CPU load (%)	7,3%	9,3%	10,0%	10,8%	14,1%	1,3%	1,8%	2,0%	2,2%	3,4%
3000		8,3%	10,5%	11,2%	12,1%	16,2%	1,3%	2,0%	2,2%	2,5%	3,8%
6000		16,1%	18,6%	19,7%	21,0%	29,5%	1,4%	2,1%	2,4%	2,8%	4,5%
9000		24.3%	26,3%	28,3%	30,4%	37,0%	1,5%	2,3%	2,7%	3,2%	5,1%
12000		31.4%	32,6%	34,9%	36,9%	43,3%	1,3%	2,6%	3,2%	4,2%	8,9%
18000		33.8%	35,2%	39,6%	46,1%	72,5%	2,8%	7,2%	10,3%	13,8%	26,2%
1500	area chart error	0,0%	0,0%	0,0%	0,1%	66,7%	0,0%	0,0%	0,0%	0,0%	33,3%
3000		0,0%	0,0%	0,0%	0,1%	33,4%	0,0%	0,0%	0,0%	0,0%	33,3%
6000		0,0%	0,1%	0,6%	8,1%	61,8%	0,0%	0,0%	0,0%	0,0%	33,3%
9000		0,1%	1,2%	3,9%	9,5%	30,2%	0,0%	0,0%	0,0%	0,0%	33,3%
12000		0,1%	14,1%	22,2%	26,0%	82,4%	0,0%	0,0%	16,7%	33,3%	33,3%
18000		38,2%	44,8%	50,2%	59,5%	93,1%	0,0%	16,9%	32,1%	53,7%	100,0%
1500	bar chart error	0,0%	0,0%	2,2%	2,3%	6,7%	0,0%	0,0%	0,0%	0,0%	93,3%
3000		0,0%	0,1%	4,5%	4,5%	4,5%	0,0%	0,0%	0,0%	0,1%	2,2%
6000		0,1%	2,3%	2,3%	4,1%	5,7%	0,1%	0,1%	2,3%	2,3%	2,3%
9000		1,8%	3,4%	4,2%	5,4%	7,5%	1,6%	2,4%	2,4%	2,4%	3,2%
12000		13,6%	19,4%	21,0%	25,0%	28,0%	0,2%	1,5%	2,5%	4,7%	6,1%
18000		49,5%	51,1%	52,6%	54,7%	57,9%	2,9%	9,4%	12,6%	16,3%	22,4%

Figure 6.5: *The experimental results.*

the greater the number of bars to the right is, the more errors were observed.

In general, the results show that Natron is more accurate (the result error is smaller) than SLD and, consequently, validate the hypothesis Hp.2.1. For the area chart the distribution shows that Natron percentage of error is very low when the input throughput is between 1500 posts/min and 9000 posts/min. When it is higher (i.e., 12000 and 18000 posts/min) also Natron starts suffering and the percentage of errors starts growing. For SLD, errors are present even at lower input rate, the graph shows that the error distribution starts moving to the right at 6000 posts/min. Similar consideration can be proposed for the bar chart error distribution. The degradation of performance of SLD starts a very low rate, a substantial presence of errors around 7% can be seen with 6000 posts/min in input.

Figure 6.6 and Figure 6.7 show the deep correlation between resources usage and errors. Clearly, a growing input throughput drives the systems to be less reliable. For both Natron and SLD the correctness of the results decreases as soon as the machine is overloaded and the resources usage starts rising out of control.

Differently from SLD, that uses only the ingestion time to optimize the reactiveness, Natron, also accept the application time. It guarantees the accuracy of the computation at the expense of reactiveness. It is worth to note that it is important to find a good trade-off between the two KPIs, this trade-off is domain dependent.

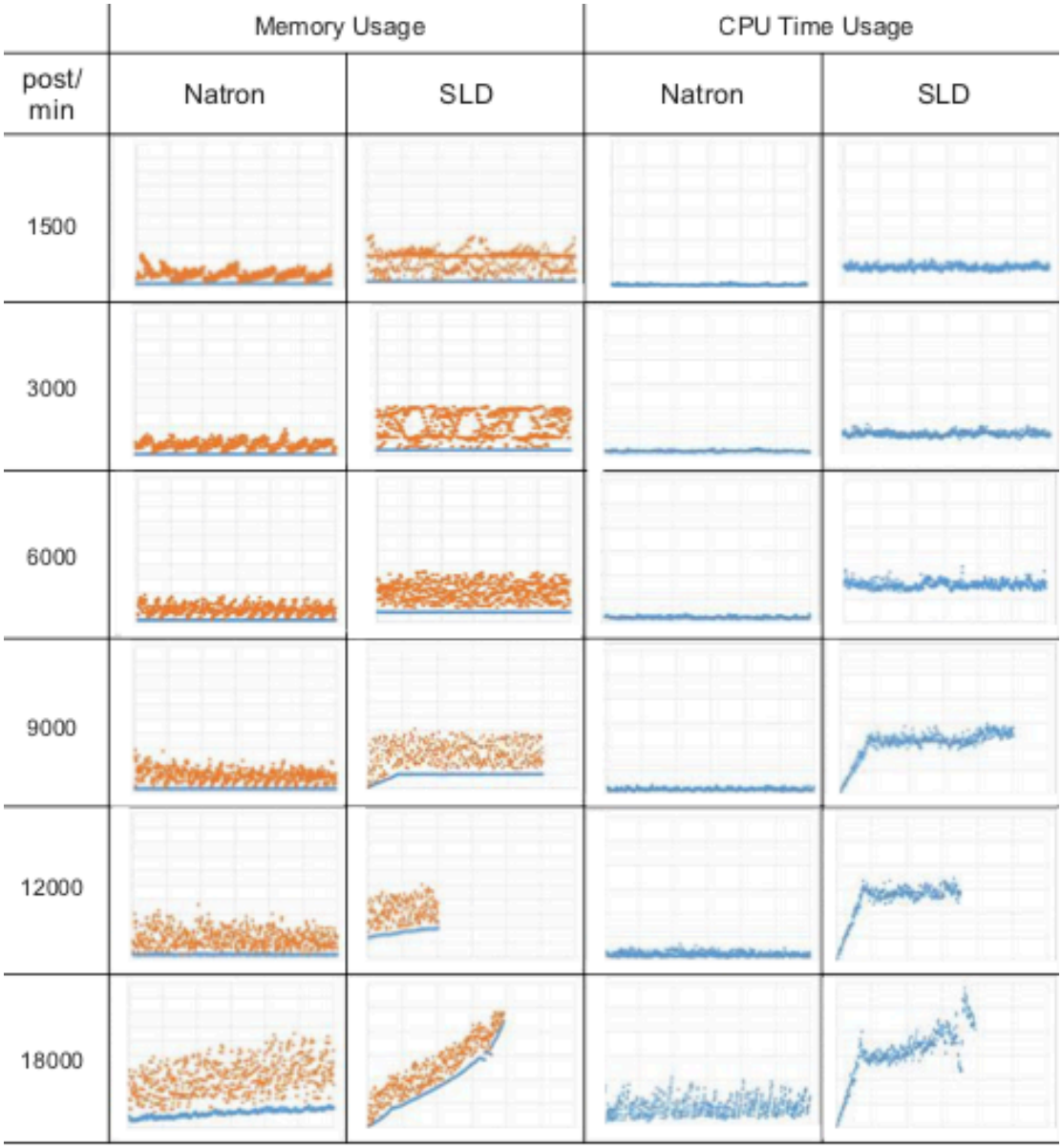


Figure 6.6: *Memory and CPU usage over time. In the Memory Usage columns, the blue dots represents the memory usage of the system thread, while the orange dots shows the total memory usage for the JVM. In the CPU Time Usage columns, the blue dots represents the CPU time usage of the system thread.*

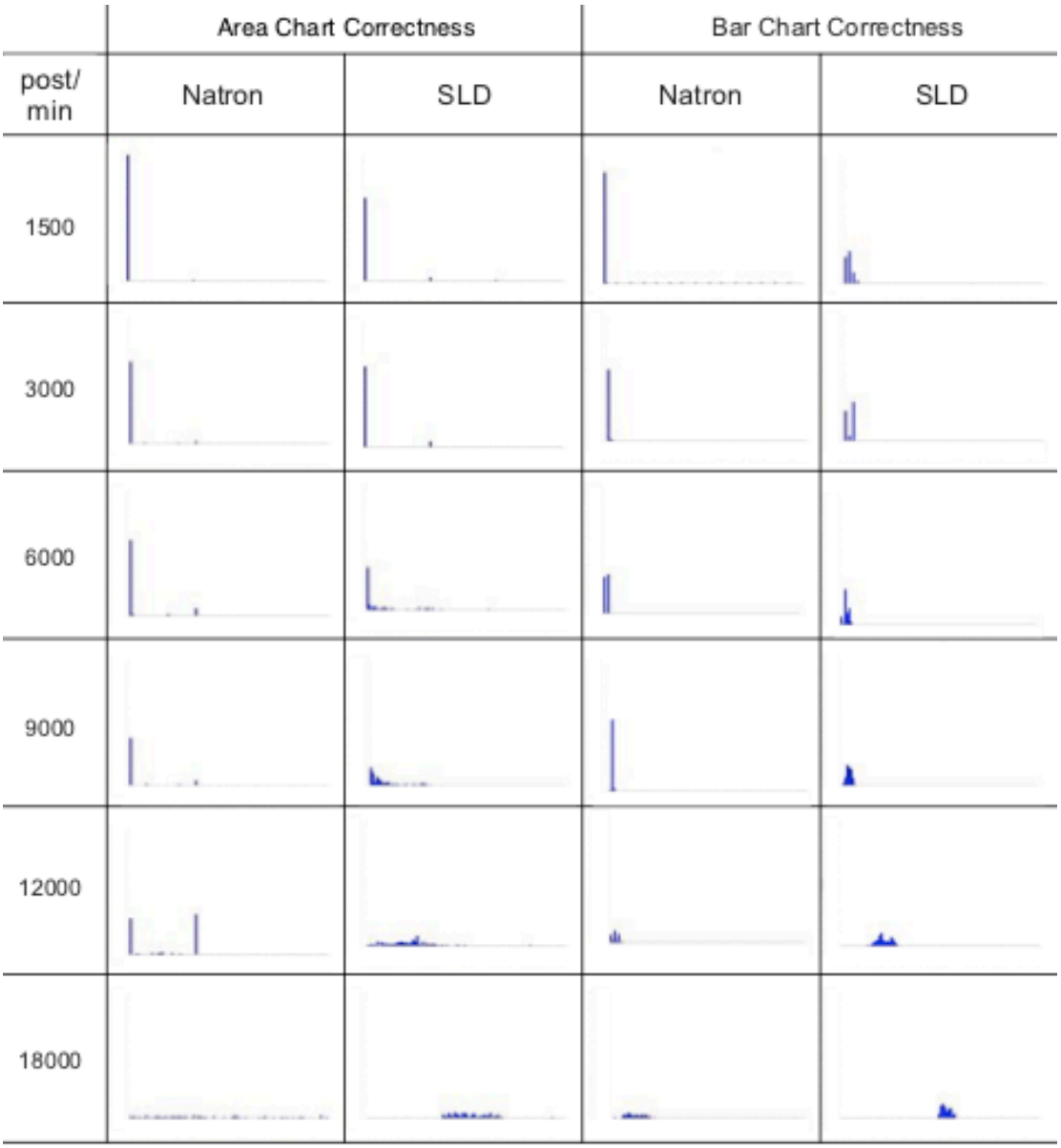


Figure 6.7: *Area chart and bar chart errors distributions*

6.4 COST-Aware Evaluation: Distributed vs. Single-Threaded

In [24], in order to validate the hypothesis Hp.2.2 and inspired by COST (see Section 2.5.3), we propose an empirical comparison between a rvr@Spark and Natron for a streaming data analysis task. The focus of our analysis is less on performance, and more on the total cost of solving the task. This shift is motivated by the industrial setting in which this work is conceived. In industry, solutions must be evaluated both in terms of cost-effectiveness and efficacy. The research question we want to answer is – *What is the most cost-effective solution for streaming data analysis when comparing distributed and single-threaded deployments?*

It is well established that performance metrics are frail when they ignore cost-related indexes (see Section 2.5.1). For this reason, differently from previous works [78,79] that focuses on latency and throughput, we base our analysis on the total solution cost. This cost is obtained by multiplying the price-per-second of the machines storing the data and running the solution by the execution time needed for the analysis task. With this choice, we want to highlight the cost-effectiveness of a solution.

Our use case is an on-line anomaly detection task. Our goal is to detect unusually crowded areas in a city. Our dataset consists of the mobile phone connection data collected in Milan during 2016 (see Section 7.1.3). The possibility to perform this task is well documented in [8, 104, 132]. Both of our solutions use the same anomaly detection strategy. This consists in a statistical model-based anomaly detector trained on historical data [23] (for more information, see Chapter 7).

We compare the performance of rvr@Spark and Natron. In both cases, RDF is not used at all and data are kept in their format as long as possible. In the follows, we describe the tuning of both solutions to our particular use case. Then, we compare them on the total cost required to solve the anomaly detection task. In order to assess the solutions' scalability, the analysis is replicated multiple times and for different data volumes.

The design of an industrial solution also requires operational considerations. With the term operational, we refer to the choices regarding when and how data is ingested, stored, and processed. Depending on the use case, there might be different operational requirements. In our use case, data is generated continuously from the mobile phone network. To avoid data losses, our only operational requirement is that data must be ingested continuously (see Section 5.1). For our analysis, we consider the following two consumption policies: (i) *continuous* – data is consumed in real-time as soon as it is ingested – and (ii) *periodic* – data is consumed at regular time intervals (e.g., once a day, or once a week). Those choices influence the total solution cost. For example, if we want to analyze data continuously, we need dedicated hardware running 24/7.

6.4.1 Problem Settings

In this work, we use mobile phone data, in particular the CDRs (see Section 3.1), collected in the city of Milan, Italy, during the months of February, March, April and June 2016. Data was made available thanks to the collaboration with TIM – Telecom Italia.

We model data using FraPPE (see Chapter 4) . The city was overlaid by a Grid, each grid Cell represents a 250x250 meters square. In order to preserve user privacy, data is aggregated at Pixel level using 15-minutes-long Frames. Consequently, we created a

film of Frames, which shows the evolution of the city, by counting the number of distinct mobile phone users in each Pixel. For privacy preserving reasons, if the counting goes below a given threshold, it is set to zero.

The data collected in the month of April is the most significant; in this period the city of Milan hosts a design festival[7] that attracts half a million of visitors, and an anomalous density of people can be detected in the 11 districts of Milan that host the 1.151 events [23] of the festival. This dataset comprises CDRs of calls and SMSs collected between April 13th and April 17th 2016. CDRs of Internet connections are filtered out since this data is missing in the majority of the months considered. This one-week dataset occupies 1.7GB, and contains around 24 millions calls and 17 millions SMS records. We name this dataset Mobile 1, and we shorten it as MOB1.

We use the rest of the data (March, February, June) for training the models described in Section 6.4.1. The cost of this activity is not considered in the paper.

In order to include the scalability dimension in our analysis, we generated several datasets by scaling our original MOB1 dataset. The scaling procedure takes as input an integer scaling factor k, and it replicates each CDR in the dataset k times.

Through scaling, we generated several additional datasets for our experiments. The most representative ones are:

- MOB1 (1.7GB), original dataset, representative of weekly mobile traffic (excluding Internet connections) in a large metropolitan area (Milan).
- MOB10 (17GB), $k = 10$, representative of weekly mobile traffic (including Internet connections) in a large metropolitan area (Milan).
- MOB30 (50GB), $k = 30$, representative of weekly mobile traffic in a country (Italy).
- MOB50 (83GB), MOB100 (170GB), extreme situations.

Representative sizes are based on internal TIM metrics. Unfortunately, all datasets used in this study are not available for public disclosure under TIM policies. Aggregated data similar to the one we produced internally when processing the raw CDRs is available as part of the TIM Big Data challenge 2015 dataset[8].

In our use case, we are interested in finding out which areas of a metropolitan city are unusually crowded. The people present in a certain area can be approximated by the number of active mobile phones in the area.

We can cast this use case into an on-line time series anomaly detection problem. Anomaly, or outlier, detection is a data analysis task such as classification or clustering [133]. Anomaly detection consists in identifying the most anomalous data patterns in a data set. An anomalous pattern could be composed of a single or several data elements. Anomaly detection relies on the ability of building a model of normality for a system or phenomenon. The model is then used to detect anomalies by computing the "distance" between the model and the anomalous element.

In our case, an anomaly represents an infrequent event in the city, which attracts a large number of people. A model of normality can be built by analyzing mobile phone data in periods where no event occurs. This is usually known as *training* in the machine

[7] http://archivio.fuorisalone.it/2016/en
[8] http://www.telecomitalia.com/tit/en/bigdatachallenge.html

learning community. Then, the trained model is compared with the collected data to detect anomalies.

We perform an online anomaly detection analysis (see Section 3.2). For training, we consider a the evolution of each Pixel. Following [23], we assume each Pixel follows a Gaussian distribution, and we approximate its parameters by computing the sample mean and variance in periods where no sizable event happens (i.e., in February, March, and June). We repeat this process for weekdays and weekends, since they present different mobile activity patterns. This accounts for $2 \times 24 \times 4 = 192$ models for each pixel, i.e., 1.92 million models considering the 10.000 pixels the city is divided in. Anomalies are detected at runtime by joining each pixel measurement with the corresponding model distribution. Measurement x is reported as an anomaly if its z-score is larger than 3, that is

$$\frac{|\bar{\mu} - x|}{\bar{\sigma}} > 3 \tag{6.1}$$

where $\bar{\mu}$ and $\bar{\sigma}$ are the estimated mean and standard deviation for x's pixel in the corresponding fifteen minutes slot.

Note that there exists a plethora of more advanced anomaly detection techniques (for an extensive reference see [133]). Finding the most accurate detector is outside the scope of this work. We use the Gaussian model since it has been shown [23] to be well-fit for the problem at hand. In particular, our method can be executed in parallel on a cluster of computers, since every pixel can be analyzed independently from the others. As already mentioned. the only operational requirement for our use case is that data is collected in real-time to avoid data losses.

6.4.2 Solution Design

The cost of an analytics solution depends on infrastructural, architectural, and operational choices. The proposed solution is a specialization of the reference architecture presented in Section 5.3.

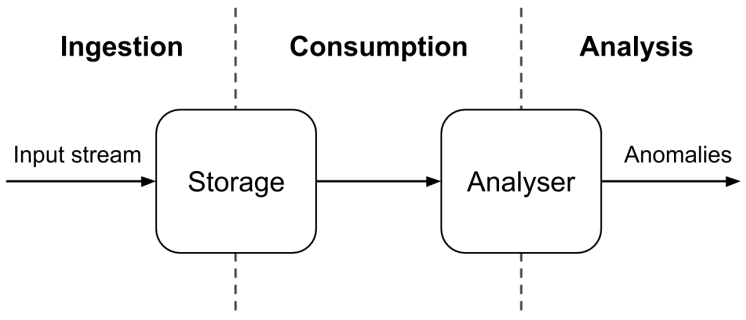


Figure 6.8: *General architecture of our solution*

being a specialization of RI≻EЯ reference architecture, our data analysis task can be decomposed into three main phases (see Figure 6.8):

- data ingestion – data is collected from the mobile network and transferred to a storage layer.
- data consumption – data is transferred from the storage layer to the analysis layer.

- data analysis – data is processed and results are generated by joining streaming data with the static models.

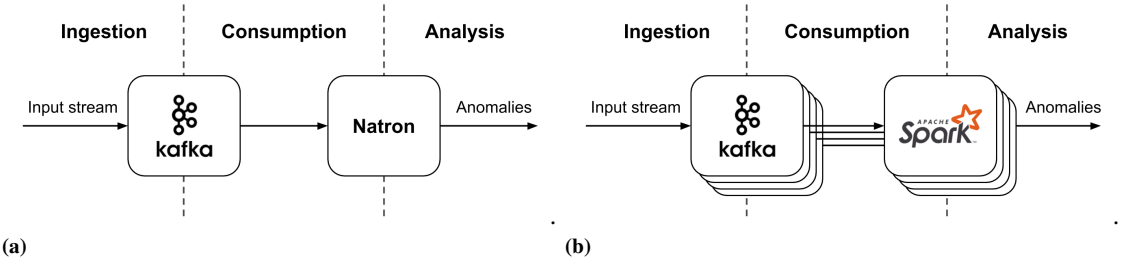
Note that we add a storage layer between ingestion and consumption to decouple the two phases. This means that we can ingest data in real-time, and analyze it at a later stage. This also enables various operational scenarios. An infrastructural choice specifies where a solution is deployed. The hardware used to run an application can be bought, or rented from a cloud service provider. We restrict our analysis to cloud services, since they usually reduce the operational cost of the solution.

When instantiating virtual machines (VMs), cloud service providers usually offer two types of billing policies: pay-per-use instances and reserved instances. Reserved instances (RIs) can be held for a fixed amount of time at a reduced price with respect to pay-per-use instances. RIs are well-fit to reduce the cost of continuous data analysis solutions, while pay-per-user instances are better fit for bursty workloads, such as periodic analysis tasks. In the following, we refer to pay-per-use instances as shared.

Table 6.1: *Azure VM sizes (January 2018)*

VM Type	Cores	RAM (GB)	S/R (€/month)
VM1	2	4	64.62/60.33
VM2	4	8	127.99/121.58
VM3	8	16	256.61/242.42
VM4	16	32	513.23/484.92

Table 6.1 presents the characteristics of the virtual machines used in this study. The last column contains the approximated cost of running a shared instance versus a reserved instance. The reported costs and characteristics refer to Fsv2-series VMs of Microsoft Azure[9]. We chose the Fsv2-series because it is equipped with computation optimized hardware that fitted our needs at affordable cost. Nevertheless, reported costs do not differ significantly from those of other cloud service providers.



(a) (b)

Figure 6.9: *(a) Architecture for the single-threaded solution. (b) Architecture for the distributed solution*

In this work, we use Kafka (see Section 2.1.3) as our streaming data storage. We use two different configurations, one for each solution. The single-threaded solution, based on Natron, reads data from a single VM1 machine. The distributed solution, based on rvr@Spark, reads data from a Kafka cluster composed of four VM2. In the distributed setting, we set the number of partitions for each topic to eight. We choose this value by

[9]https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-compute

considering the number of executors used in the experiments, since executors can read in parallel from different partitions.

In the single-threaded solution, the consumption phase is implemented using a Kafka Ingester that polls the data from the server in comma separated value format. The architecture for this solution is depicted in Figure 6.9(a). The Ingester connects to an Apache Kafka server that provides the data. The data enters the system as a stream of generic objects. Each object contains its event timestamp. Downstream to the Ingester, a Processor takes the data from the Bus and transforms each element into a domain-specific Java object (i.e. a Java representation of a CDR, named PixelCDR).

Then, an Processor based on Esper (see Section 2.1.3) performs the analysis. The internal stream of PixelCDRs flows into Esper, which performs the query presented in Listing 6.9. The query counts, every 15 minutes, the number of calls/SMSs grouped by pixelId, i.e. the pixel identifier. The window operation is performed on the event timestamp. We use Kafka exactly-once message delivery to analyze the whole data stream. The query produces the list of anomalous pixels. The anomalies are identified using the *isAnomalous* user defined function, that access the models file, stored in memory, and implements Equation (6.1). The query results are then saved to the file system by an Emitter.

```
SELECT pixelId, MAX(timestamp)
FROM PixelCDR.WIN:EXT_TIMED_BATCH(timestamp, 15 min)
GROUP BY pixelId
HAVING isAnomalous(pixelId, COUNT(*), MAX(timestamp))
```

Listing 6.9: *EPL query performed by Esper Processor.*

We implemented our distributed streaming pipeline using rvr@Spark and we register both the static models table, and the CDR data stream as temporary views that can be queried through the Structured Streaming API. The CDR view is actually a dynamic table that gets updated as data is ingested. The anomaly detection method is implemented as a SQL query that performs a join on the aforementioned tables, and filters the results based on the anomaly condition defined in Equation (6.1). Listing 6.10 contains the pseudocode for the query.

```
SELECT pixelId, timestamp
FROM (
  SELECT cdrs.pixelId, cdrs.timestamp, COUNT(1)
  FROM cdr_stream AS cdrs
  WINDOW ON cdrs.timestamp EVERY 15 minutes
  GROUP BY cdrs.pixelId
) AS windowed_cdrs LEFT JOIN models
ON models.timestamp = windowed_cdrs.window.start
WHERE isAnomalous(windowed_cdrs.value, model.mean, model.sd)
```

Listing 6.10: *Spark SQL anomaly detection query.*

The distributed application is deployed on a multi-node Spark cluster, while data is ingested from a multi-node Kafka cluster. This deployment is represented in Figure 6.9(b). Spark is integrated with Kafka to provide parallel reads from multiple Kafka partitions.

We choose Apache Spark for our distributed solution due to its wide spread use in

industry, and the availability of previously developed source code and expertise. Operational requirements are related to business choices (see Section 3.2). They deal, for example, with how often a result report should be produced. We consider the following two operational scenarios:

- *Continuous ingestion – continuous consumption and analysis*. This scenario includes real-time use cases, such as crowd monitoring for security purposes. Data is consumed as soon as it is produced, and the delay with which results are produced corresponds to the latency of the system. In this regime, results are produced continuously with whatever latency the system might have. This scenario requires the continuous utilization of reserved resources, since the solution must run without interruptions.
- *Continuous ingestion – periodic consumption and analysis*. Periodic analysis represents a common scenario. In many use cases, the results of the analysis can be summarized in a periodic report, and the real-time analysis is not necessary. The ingestion layer must still run continuously to avoid data losses. On the other hand, the analysis layer can be allocated only for the amount of time needed to perform the analysis and generate the results.

Another important considerations when designing an industrial analytics system are fault-tolerance and redundancy. Apache Kafka and Apache Spark respectively provide out-of-the-box redundancy and fault-tolerance. Nonetheless, we do not include these aspects in our analysis, since the total solution cost of a fault-tolerant system can be approximated as the total solution cost multiplied by the redundancy factor. If we apply this consideration to both solutions, it does not affect our final results.

6.4.3 Experimental Settings

The goal of our experimental methodology is to find the most cost-effective solution for the given problem. To assess this, we run our solutions on both real and simulated problem instances. The real data MOB1 is collected from the mobile phone network of TIM. Starting from this real data we generated several other datasets (MOB10, MOB30, etc.). Those datasets were generated to analyze the scalability of our solutions.

We compare our solutions based on their total cost when they both provide correct results. This is not always the case since the most economic single-threaded configurations struggle to deal with the most demanding problem instances. The solution cost is computed by multiplying the cost of the solution (i.e., price-per-second of the used VMs) with the execution time of the experiment (if completed correctly). The cost of the solution also depends on the operational requirements, e.g., a continuous solution can run on reserved instances, thus reducing the price-per-second.

We executed all experiments on Microsoft Azure Linux VMs. For each experiment, we performed five experimental runs. All reported results are average over four runs by discarding the worst outcome. We do not include error bars in the plots since their bounds are so tight that they simply overlap with the point shapes and clutter the images.

We do not consider latency in our analysis due to the following reasons:

1. In the continuous analysis scenario, at regime the latency of the system does not influence the stream of results. Moreover, the latency to analyze one minute of data is below 1.5 seconds for both solutions, which is appropriate for our use case.

2. In the periodic analysis scenario, the latency of both systems is negligible with respect to the periods considered (i.e. every day or every week).

Thus, in the following we omit latency from our discussion.

Our goal is to find the most cost-effective configuration which solves the problem. We restrict our analysis to Fsv2-series VMs under the assumption that in cost-aware scenarios more general-purpose VMs are preferable to workload-optimized VMs, since they can be shared and used by different workloads. Figures 6.10(a) and 6.10(b) show the solution cost as a function of the scale factor for different configurations of Natron and rvr@Spark.

Natron was deployed using a docker container to create a sandbox environment and to ease the monitoring operations for CPU and memory consumption. The whole infrastructure needed a single VM for each experiment in addition to the VM needed for the data provider, i.e. a single partition Kafka server on a VM1. We run multiple experiments for each dataset and remove the outliers, e.g. the first run of each experiments was considered as a system setup, collect data result for correctness check, i.e. anomalies, and CPU/memory consumption log to monitor the health of the infrastructure. During each run the container exploits all available virtual machine resources for the computation. We vary the dimension of the VMs in azure to stress the environment and get the upper limit of the resource needed to handle a given amount of data.

We experimented with three configurations, having different number of cores, RAM, disk I/O, and network I/O available: (i) Natron1 with a single VM1, (ii) Natron2 with a single VM2, and (iii) Natron3 with one VM4.

The single-threaded implementation suffers from the volume of the data, a single VM cannot scale horizontally to deal with a continuously increasing amount of data. Figure 6.10(a) clearly shows that the different configurations can bear different loads of data. Natron1 can handle dataset MOB1, which represents the original data size, and can perform the anomaly detection in about 120 seconds. This configuration can handle up to dataset MOB10, but bigger dataset results are problematic. Configurations Natron2 and Natron3 can bear at most dataset MOB50 and MOB100 (respectively), but are more expensive than configuration Natron1 . The three chosen configurations widely explore the hardware offerings in order to find the best solution related to the data loads. Due to the variability of configurations' behaviors, we tested the system against more dataset than the ones listed in Section 6.4.1, i.e. we tested dataset with scale factor k=2, k=3, k=5, and k=20.

We compare all the three Natron configurations with the best configuration chosen for the distributed system in order to have a complete overview for the different input volumes. During the experiments, regardless of the Natron configuration, the normality models are loaded in memory, while streaming data is read from the Kafka cluster described in Section 2.1.3.

We deployed rvr@Spark application on a Spark cluster tuned using the total solution cost as a metric, and experimenting with three parameters which commonly affect Spark's performance. Our intention here is to present our findings on the best Spark configuration for our specific use case, datasets, and problem setting. We implemented our Apache Spark cluster using Azure Linux VMs (see Table 6.1).

We experimented with the following cluster parameters: i)the virtual machine size, ii) the number of executors per worker (or number of cores per executor), and iii) the

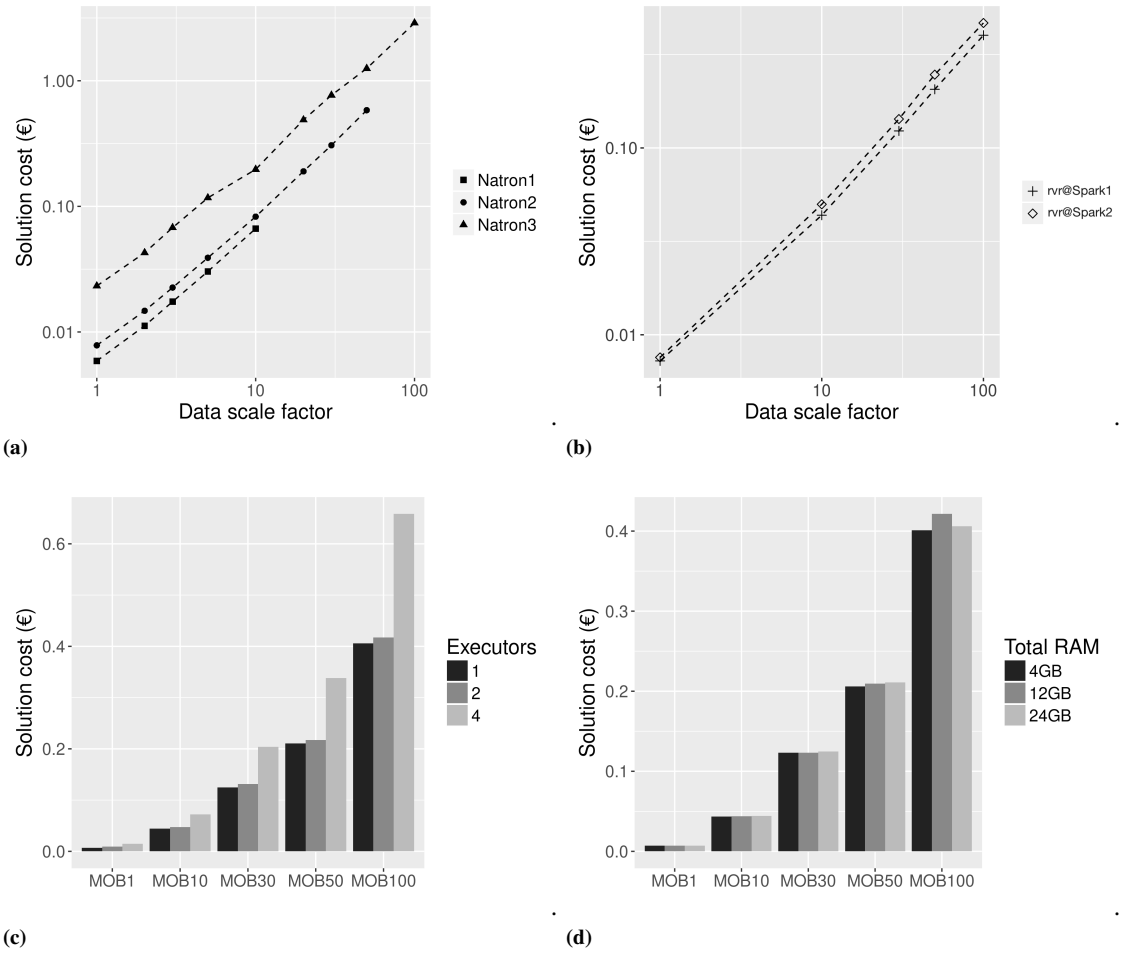


Figure 6.10: *(a) Solution cost over data scale for different Natron configurations. (b) Solution cost over data scale for different rvr@Spark configurations. (c) Solution cost over number of executors per worker on different datasets (RAM at 24GB). (d) Solution cost over total number of RAM in GB for different datasets (1 executor per worker).*

memory allocated per executor. All other parameters were set to their default values. Note that, since in Microsoft Azure each virtual core (vCPU) corresponds to a single thread, in the following we use the terms core and thread interchangeably.

Cloud service providers offer several VM types. Those types vary depending on the number of cores, RAM, disk I/O, and network I/O available to user applications running on the VM. Thus, an important consideration when deploying a cloud solution is the choice of VM type.

We evaluated two different cost-equivalent configurations for our Spark cluster (refer to Table 6.1 for VM characteristics): (i) rvr@Spark1 with one VM2 as a master and four VM2 workers, and (ii) rvr@Spark2 with a singly VM2 as a master and two VM3 workers.

Note that we also experimented with smaller cluster configurations (e.g., a single VM2 worker). However, we found that these were not as cost-effective as the configurations described above. This might seem counterintuitive. However, consider that a

smaller configuration usually takes more time to perform the analysis. Since our metric is the total solution cost, to be cost-effective a solution's cost reduction should compensate for its performance penalty.

As an example, we found out that a cluster with a single VM2 worker takes from 2.75 to 3 more time (depending on the dataset) to perform the task with respect to our rvr@Spark1 configuration, while only costing 2.5 less.

Figure 6.10(b) shows the cost of the solution for both configurations. All Spark settings were set to their default values (all available cores, 1GB of RAM per executor). The figure highlights that the rvr@Spark1 configuration tend to be more cost-efficient, even though the total number of used cores is the same in both configurations.

Even after tuning both clusters, i.e. by changing the default parameters, we could not find a configuration for rvr@Spark2 outperforming rvr@Spark1. We used rvr@Spark1 for all other experiments. The two following sections provide more details on the experiments we performed measuring the sensitivity of the selected configuration to changes in the number of cores per executor and in the amount of RAM per executor.

An important parameter in Spark configuration is the number of cores allocated to each executor. The default configuration allocates all available cores. Incidentally, the number of cores per executor also determines the number of executor processes that a worker can spawn. Thus, we perform our sensitivity analysis in term of executors per worker. We fixed the total RAM to 24GB and varied the number of executors per worker machine. Figure 6.10(c) shows our results. We can see that having a single executor on each worker outperforms other configurations. This is supposedly due to the fact that when multiple executors reside on the same machine, the JVM must handle a large volume of I/O network traffic in order for them to communicate. This could possibly influence application performance.

Another important parameter is the amount of RAM designated to each executor. In this case, we picked the best configuration from the previous analysis, i.e. one executor per worker, and varied the RAM allocated to each executor. Figure 6.10(d) shows our results to this sensitivity analysis. We can notice that the amount of memory allocated to each executor does not seem to affect execution time. This is surprising, considering the common knowledge that Spark performance is proportional to the amount of main memory available. However, our particular use case, i.e. windowed and watermarked relational query, is executed considering one window of data at a time. Even at maximum scale (x100), our windows do not exceed 1GB of RAM, and therefore in this particular scenario the system is not memory-bounded.

All the following experiments were executed using configuration rvr@Spark1 with 4 cores and 3GB of RAM per executor. The normality models are stored in a static file over the Spark cluster, while streaming data is read from the Kafka cluster described in Section 2.1.3.

6.4.4 Results and Discussion

In this section, we present our experimental results. We organize our discussion based on the operational requirements considered in Section 6.4.2. The analyzed scenarios are summarized in Table 6.2.

The resulting monthly solution costs per scenario are represented in Table 6.3. All costs refer to the MOB100 dataset. Periodic scenarios (S2 and S3) refer to analysis

Table 6.2: *Operational scenarios. Each layer of the system can run continuously (C) or periodically (P), and on shared (S) or reserved (R) hardware. Data ingestion and consumption are both handled by Apache Kafka, therefore they are always executed on the same hardware.*

Scenario	Ingestion	Consumption	Analysis
S1	C/R	C/R	C/R
S2	C/R	P/R	P/S

Table 6.3: *Monthly solution costs. The monthly cost of our solution depending on the operational scenario. Notice that if we perform continuous ingestion, the consumption costs are included (Incl.). The third scenario represents the case in which ingestion costs are fixed, i.e. they do not depend on the number of machines, but only on data throughput. The most cost-effective solution is highlighted.*

Scenario		Ingestion	Consum.	Analysis	Total
S1	rvr@Spark1	€486.32	Incl.	€607.9	€1094.22
	Natron3	€60.33	Incl.	€484.92	**€545.25**
S2	rvr@Spark1	€486.32	Incl.	€12,41	€498,73
	Natron3	€60.33	Incl.	€76.85	**€137.18**
S3	rvr@Spark1	Fixed	€9.93	€12.41	**€22.34**
	Natron3	Fixed	€9.68	€76.85	€86.53

carried out daily, i.e., 30 times per month.

S1 – Continuous ingestion – continuous consumption and analysis In this scenario, we consider the case in which we require a continuous analytics solution. The whole infrastructure must be continuously up and running to support the ingestion, consumption and analysis phases. We can compute a monthly solution cost by considering the reservation price of all VMs used in the solution.

From Table 6.3, we can see the estimated monthly solution cost for scenario S1. Ingestion cost is calculated using reserved instance price, since these machines must run continuously. This is the same for analysis cost. Consumption cost is included in the ingestion, since the Kafka VMs perform both phases continuously. The single-thread cost is calculated considering configuration Natron3.

In this case, we can clearly see that the single-threaded implementation is the most cost-effective solution for the problem.

S2 – Continuous ingestion – periodic consumption and analysis This scenario represents a use case where the continuous analysis is not necessary, but periodic reports are needed. Table 6.3 contains the cost analysis for this scenario. The costs of ingestion and consumption are equivalent to S1. The analysis cost is computed on the more demanding dataset MOB100, using rvr@Spark1 and Natron3 configurations. We report the monthly cost for an analysis performed daily. The ingestion phase must be continuous and, consequently, the infrastructure that support the ingestion and consumption phases can be deployed on reserved hardware. The analysis is periodic (once a day), and can be executed on pay-per-use VMs which can be turned on only for the duration of the analysis.

In this scenario, we can see that the rvr@Spark system is more cost-effective with respect to the analysis phase, but not to the ingestion phase. The cost of continuously

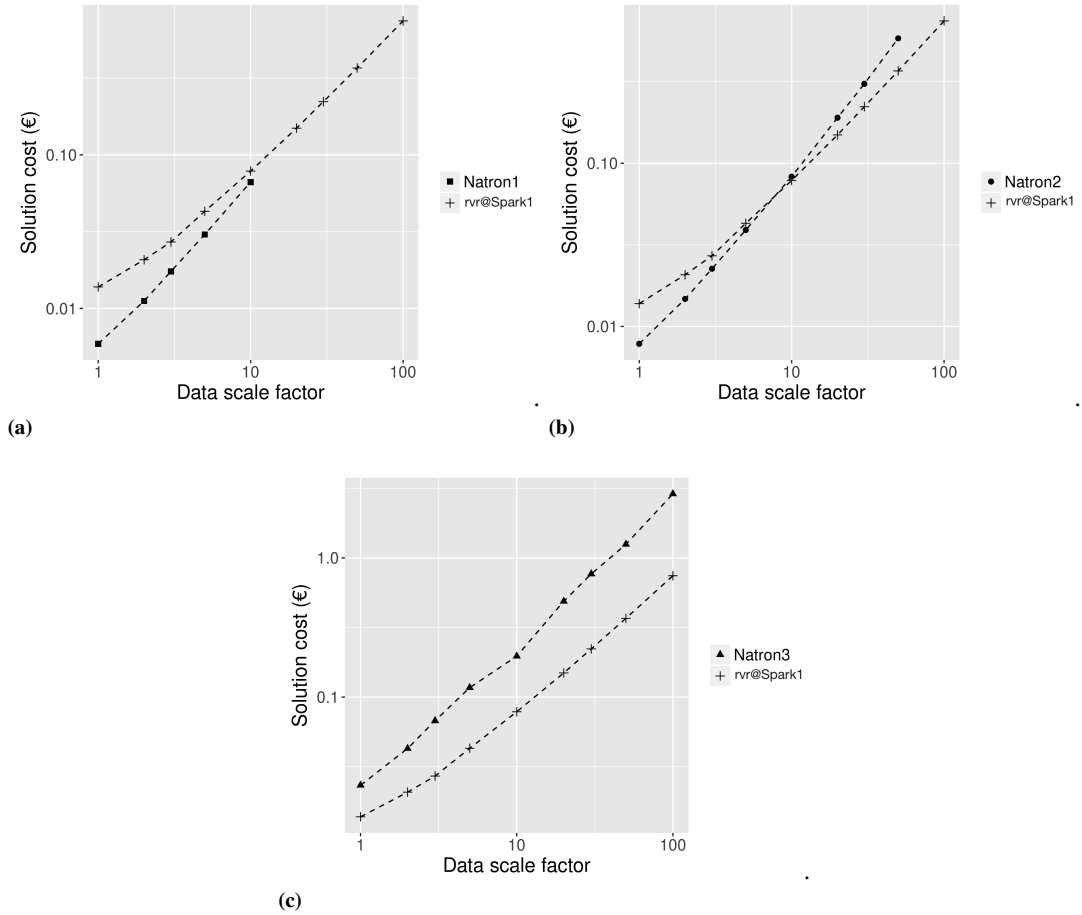


Figure 6.11: *(a) Total solution cost for S3: Natron1 vs rvr@Spark1. Natron1 is the lowest cost solution, but it can handle only datasets of modest size. (b) Total solution cost for S3: Natron2 vs rvr@Spark1. The two solutions are cost-equivalent at a scale factor around 10. After that, rvr@Spark1 becomes the most cost-effective solution. (c) Total solution cost for S3: Natron3 vs rvr@Spark1. Natron3 can handle all datasets considered, however it is less cost-effective than the distributed system at all scales.*

ingesting data using a distributed cluster outvalues the benefits of processing such data in parallel. This is still true at lower data scales, where the convenience of the single-threaded solution is even more evident.

After realizing this fact we included a final scenario (*S3*) in our analysis. This scenario is a situation where data ingestion is provided at a fixed and small price, i.e. it does not depend on VMs cost but only on data throughput and retention. This is the case with some particular offers from cloud providers such as Confluent[10]. Since the throughput and the retention are fixed, in this scenario the ingestion cost is the same for both solutions.

S3 – Continuous ingestion at fixed/small price – periodic consumption and analysis In this scenario the total cost of the solution depends on the number of machines active during the analysis phase, and on the duration of this phase. Thus, if the additional costs of using more machines in the distributed setting implies reducing the execution time by the same factor, then the distributed solution is the most cost-effective.

Table 6.3 presents the results for this scenario. The results compare configuration rvr@Spark1 versus configuration Natron3 when processing the dataset MOB100. We assume the analysis is carried out periodically each day. We can see that the reduced execution time for the analysis makes up for the increased number of VMs. This makes the distributed solution around 3.8 times more cost-efficient than the single-threaded system.

We provide more insight on this scenario by considering different data scales. We compare configuration rvr@Spark1 with the less expensive Natron configuration that can handle a given data scale: configuration Natron1 for a scale factor up to 10, configuration Natron2 for a scale factor up to 50, and configuration Natron3 for the dataset MOB100.

We can see that, in this setting, the most cost-effective solution depends on the data size. At small data scales, configuration Natron1 is the most cost-effective solution. The configuration Natron1 can only deal with data volumes up to scale factor 10 (city scale), but, until this point, it is more cost-effective than configuration rvr@Spark1 (Figure 6.11(a)). When the data size increases, the solutions first become equivalent in term of cost around city scale (Figure 6.11(b)), and, then, configuration rvr@Spark1 becomes the most cost-effective solution (Figure 6.11(c)) when dealing with national and extreme scales.

The results presented in this section show that in case of continuous analysis, the single-threaded solution is the most cost-effective option.

When periodic analysis is considered, the distributed solution is the most cost-effective in analyzing the data. However, this benefit is outvalued by the costs of distributed data ingestion. Thus, the single-threaded application remains the best choice also in this case.

Finally, if we assume that data ingestion costs only depends on data throughput and retention, i.e. they are fixed and small, we show that the most cost-effective choice depends on the data size. The single-threaded application is cost-effective when managing small datasets, which is our setting are the CDRs generated by Milan when including Internet or those of the entire Italy if limiting the analysis to calls and SMSs. However,

[10] `https://www.confluent.io`

as the data size grows to the size of Italy including Internet, the distributed solution becomes the most cost-effective option.

6.5 Conclusion

In this chapter, we investigate how to implement RI≻EЯ computational model from different points of view. We concentrate our effort on the implementation of the *Lazy Transformation* approach.

We present Natron (see Section 6.2.1) – a single-threaded, vertically scalable implementation of the RI≻EЯ computational model – and we evaluate it against our Streaming Linked Data Framework (SLD) that applies data transformation at ingestion time (see Section 6.3). The result of this evaluation validates Hypothesis Hp.2.1: Natron results better than SLD under both resource consumption and correctness points of view.

Comforted by the results of this performance evaluation, we assume the Lazy Transformation as a third principle **(P3)** of our computational model and we apply it in the horizontally scalable implementations based on distributed technologies (see Section 6.2.2 and Section 6.2.3): rvr@Spark and rvr@Hive.

In order to reaffirm the importance of the cost-effectiveness metric in the evaluation of streaming based system, we perform an empirical comparison between Natron and rvr@Spark for a streaming data analysis task (see Section 6.4). The overall results partially validates the hypothesis Hp.2.2. Natron results cost-effective when managing medium size datasets (up to the scale of a large city like Milan), but, as the data size grows, rvr@Spark becomes the most cost-effective option.

CHAPTER 7

Case Studies

FINAL - TO BE READ ONE LAST TIME.

In this chapter, we investigate our research question (see Section 1.2) in real world cases through the hypothesis:

Hp.3 A solution using FraPPE conceptual model and an implementation of RI≻EЯ computational model, can create a bridge between data analytics and data visualization that enhances the comprehension of a variety of spatio-temporal data and, at the same time, allows reactive decisions.

We provide evidence that Hp.3 is valid presenting five case studies where we exploited, together, FraPPE (see Chapter 4) and one of the implementations of the RI≻EЯ computational model (see Chapter 6) to represent, ingest, augment, synthesize and analyze urban spatio-temporal streaming data. Section 7.1 presents our three-years-long experience related to Milano Design Week. Section 7.2 presents the case study of the Milano Fashion Week and Section 7.3 presents the work we carried on for the municipality of Como.

7.1 Milano Design Week

In the next sections, we report our experience in monitoring Milan Design Week (MDW). The monitoring project spreads across three editions (2013, 2014 and 2016), and represents a first attempt to put at work a system based on FraPPE, Natron, rvr@Hive and rvr@Spark to monitor a city-scale event (CSE) that lasts days and is spread across a city. Indeed, MDW takes place in hundreds of places that host thousands of small-scale events, attracting half a million people. The aim of the project is to feel the pulse of Milan during the MDW using data from different sources.

7.1.1 MDW2013 - Understanding the Data

During MDW2013, relying on our past experiences [5, 13], we concentrate our effort on the social data sources [134]. We name Social Signal the collection of the digital foot-prints left by MDW attendees on social media. During the preliminary study, we recognized that such a signal is strong enough to build a graph linking people, places, and events, but it is not sufficiently reliable (or, more precisely, statistically sound) to identify the most crowded MDW places. To tell a reliable story of MDW and identify the most crowded places, we need to show that the social signal is correlated to another, statistically sound, signal. Our hypothesis is that mobile phones can generate such a signal. We work in partnership with Telecom Italia in order to access privacy preserving aggregates of the mobile phone activity of the people present in Milan area. The availability of such a data allows us to compute an index of anomaly that shows off the discrepancies between the expected and the actual activities on the mobile telecommunication network in a given area (for more details, see Section 6.4.1).

We study the nature of the social and mobile anomaly signals to understand which visualization better fits the purpose to tell the MDW story to its attendees and we use FraPPE concepts (marked by special font in the paragraphs) to represent data.

Milan was in overlaid by a square Grid of 10,000 Cells (100 per side), each of which has a size of 250 x 250 meters. We considered three sources of Events at Places: mobile phone calls/sms/internet-accesses, geo-referenced micro-posts related to the Milan Design Week, and the 1,200 long-lasting events that are organized in 600 places spread around Milan during the Design Week.

The spatial analysis (see Section 4.4) of the social and telco data, based on FraPPE, requires to assign each OriginalContent to a specific Cell.

The analysis of the social data requires a mapping action. A social message can be explicitly geo-located and can be easily assigned to a specific Cell, but, more often, a message has no assigned latitude and longitude. In most of the social media posts, related to the MDW, the location can be extracted from the content and linked to the MDW locations exploiting Named Entity recognition and linking techniques during the augmentation phase.

The analysis of mobile phone data is based on CDR (Call Data Record) analysis. CDRs are generated by telecommunication networks to log the activity of the users, associated to a mobile phone cell for billing purposes. Every mobile phone cell has a unique identifier, the Cell Global Identity (CGI). The CGI is characterized by the country, the Mobile Network Operator (MNO), the Location Area of the cell, the latitude and longitude of the barycenter of the cell and of the antenna, the distance between barycenter and antenna, and other properties. The CDR analysis requires to map each CGI to the Cells of the Grid. There are many techniques to do so, we opted for assigning a coverage percentage to each Cell based on the orientation of the antenna, and the land usage of the covered area.

In order to perform a temporal analysis (see Section 4.4), we captured a Frame every 15 minutes. We did so, because we empirically observed that there is always at least a micropost per Frame – the social signal is less dense than the telco one.

We analyzed 2 months of CRDs from Milan to synthesize two Gaussian models for each cell (for further details see Section 6.4.1): one grouping the Frames by working days, and one grouping them by week-end days. We were able to build 1.92 millions

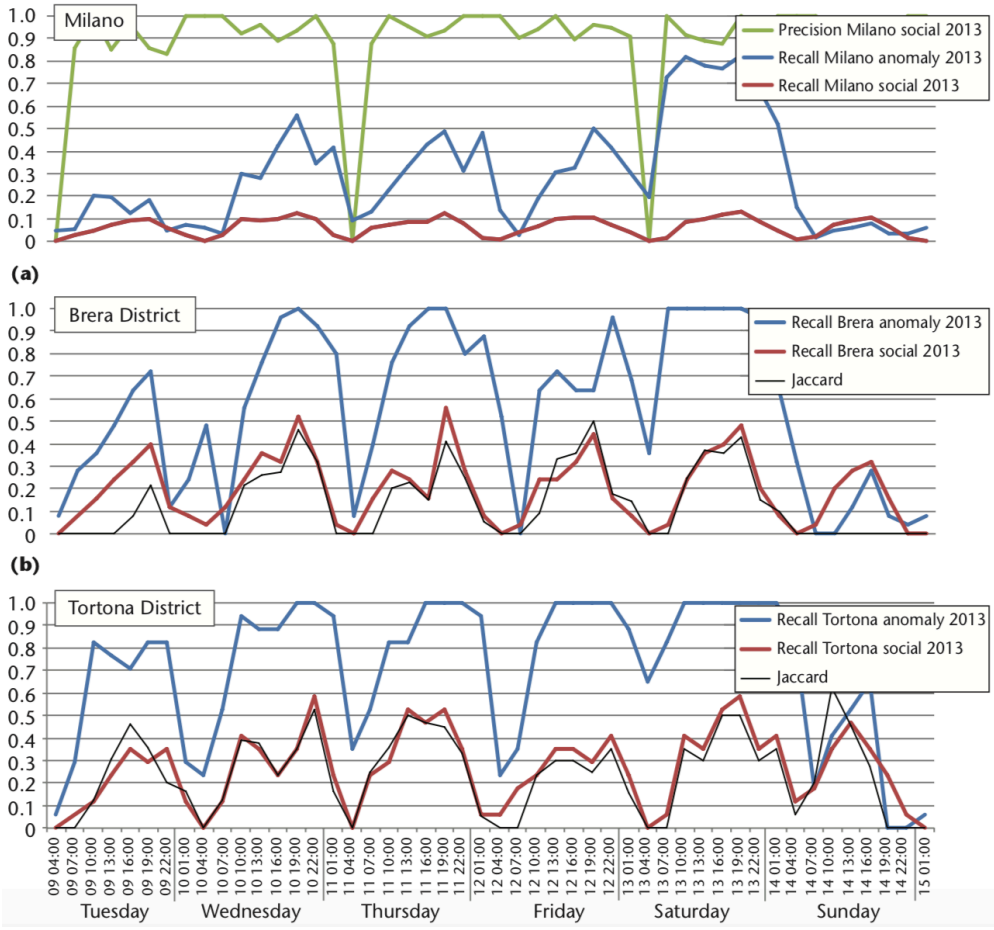


Figure 7.1: *Precision and recall of the social and mobile anomaly signals in identifying pixels where Milan Design Week (MDW) events happen in (a) Milano and the (b) Brera and (c) Tortona districts.*

Gaussian models[1]. The anomaly index is obtained by computing how far the number of calls/sms/internet-accesses (which we refer as n) is from the average behavior (which we refer as avg), keeping into account the computed standard deviation (which we refer as std). The formula to obtain the anomaly index can be compactly written as:

$$2\Phi_{avg,std^2}(n) - 1$$

where $2\Phi_{avg,std^2}$ is the cumulative distribution function of a Gaussian random variable with mean avg and variance std. Anomalies are identified by filtering all the records with an anomaly index greater than a given threshold.

We, then, created a gold standard (MilanP) consisting of all the pixels that contain at least one MDW2013 place. We also created two subsets of this gold standard (BreraP and TortonaP) that focus on the pixels that belong to the districts of Brera and Tortona, where the most important events were located. For each frame, we measured the ability of the social and mobile anomaly signals in identifying pixels in MilanP, BreraP, and TortonaP (see Figure 7.1) measuring precision and recall.

[1] 4 frames per hour X 24 hours X 2 day types (working and weed-end days) X 10.000 pixels.

The results of the analysis show that the social signal has high precision (between 0.8 and 1 for 95 percent of the frames) in identifying pixels in MilanP. Its recall is low in identifying pixels in MilanP, but it is good (above 0.4 for around 60 percent of the frames) in identifying pixels in BreraP and TortonaP. In all cases, the signal volume is low: the average number of microposts linked to MDW per hour is 15 for MilanP, three for BreraP, and five for TortonaP. However, the social signal is correlated to the anomaly index signal (see Figure 7.1) during the working days. The Pearson correlation for MilanP is 0.61 with a p-value of $2x10^{-4}$ for BreraP, it is 0.76 with a p-value of $4.6x10^{-7}$; and for TortonaP, it is 0.56 with a p-value of $3.1x10^{-5}$.

Moreover, not only the two signals are correlated, but they also identify the same pixels. To measure this, we computed the Jaccard similarity coefficient[2] between the pixels identified by the social signal and those identified by the mobile anomaly signal. As we can observe in Figure 7.1, the Jaccard similarity coefficients in Brera and Tortona almost constantly overlaps with the recall of the social signal – that is, if a pixel is identified by the social signal, then it's among those identified by the mobile anomaly signal.

The data analysis phase (see phase 3 of the reference architecture presented in Section 5.3) bring evidences that: (i) Telco Big Data can provide a very relevant and dynamic overview of the presence of people in the context of a specific city or territory, aggregated at the level of the single cell tower, and (ii) Social data represent a precious source to build the interaction graph between people, events and places. However, since building the morphology of the territory can impact on the effectiveness of the cells, the CDR information can describe the city's dynamics at the macro level, but it cannot be used to precisely represent such dynamics at the micro level, e.g. input/output flows into one specific street or square. The social data, thanks to its public content and the precise location of each message, can help in filling this gap.

7.1.2 MDW2014 - CitySensing Public Installation

During the Milan Design Week 2014 (MDW2014), exploiting the knowledge acquired analyzing the previous edition, we use Natron and rvr@Hive to realize the proposed CitySensing platform to collect and analyze social and telecommunication data. The visualization, enabled by CitySensing, is presented in a public installation in Mediateca Santa Teresa[3].

During the MDW2014, we consider 21,782 micro-posts from Twitter and Instagram, data describing MDW2014 places and events from three heterogeneous sources (fuorisalone.it, breradesigndistrict.it, and tortonaroundesign.com), and Telecom Italia's CDRs (19,719,629 calls, 20,240,485 SMSs, and 197,767,245 Internet data accesses counted in the Milan area from 8 to 14 April 2014). rvr@Hive analyze calls/sms/internet-accesses in real-time by aggregating them for each pixel and for each frame and by computing how anomalous they are comparing each of them against the predictions of the Gaussian models built at set-up time.

As detailed in [23], the anomalous pixels correspond with high precision to pixels in which events of the Milan Design Week are happening. This allows us to provide experimental evidence that the extra 400,000 people that come to Milano for the De-

[2]The Jaccard coefficient measures similarity between finite sample sets

[3]`http://www.mediabrera.it/index/index.php`

(a) (b)

Figure 7.2: *Social media used to explain the reason of anomalous peaks of presence in some pixels: (a) shows the most popular hashtags posted in the anomalous pixels during Milan Design Week, whereas (b) highlights the emergent hashtags, i.e., the non predicted ones. While the generic most popular tags contains also hashtag about a popular TV show (i.e., Amici or Emma), the emergent hashtags are those of Milan Design Week.*

sign Week generate extra calls/sms/internet-accesses from the cells that contain the 600 locations of the 1,200 Milan Design Week events.

To process social streams, we use Natron (see Section 6.2.1). The original data stream are injected in Natron in Activity Stream 2.0 format[4]. Natron semantically augments them using our custom Named Entity recognition and linking solution tailored on Milan Design Week [135]. A continuous query captures a frame every 15 minutes counting the number of distinct hashtags and semantic entities present in the georeferenced microposts for each pixel. The results of this continuous query is a stream modeled in FraPPE.

As illustrated in Figure 7.2.(a) a (partial) *semantic* explanation of the mobile anomalies, can be attempted aggregating the top-10 hashtags used in those pixels. For instance, in Brera district the Italian hashtag of Milan Design Week (i.e., Fuorisalone) emerges. However, this technique is not dependable. For instance, in Tortona district also the hashtags of a popular TV show (i.e., Amici) and its protagonists (e.g., Emma) appear. Once again, the solution is in the ability to compare the current top hashtags against the those predicted by a statistical model. This allows highlighting only the emergent hashtags of this frame for the selected pixels (see Figure 7.2.(b)).

As one can expect, the simple Gaussian model used for the mobile activity is not appropriate to predict hashtag usage. We found, instead, that an Holt-Winter method can be used [136] to predict the usage over time of a specific hashtag (e.g., #milan). In order to use Holt-Winter, we use FraPPE: we build SyntheticFrames that aggregate the CapturedFrames in five parts of a day (i.e., 2am-7am, 7am-11am, 11am-2pm, 2pm-7pm and 7pm-2am). Moreover, as for the CDRs, we distinguish between working days and week-ends. This approach allows to build effective predictive models for hashtags about the points of interest of Milan and about popular TV shows. Figure 7.3 illustrates how this method detects the anomalous usage of #milan during the Milan Design Week, which is highly correlated to the usage of #mdw – the official hashtag of Milan Design

[4] http://www.w3.org/TR/activitystreams-core/

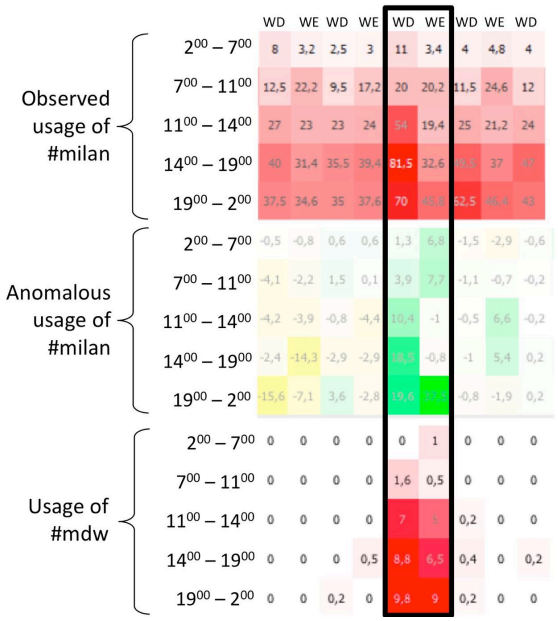


Figure 7.3: *Highlighting of anomalies in hashtag usage: The hashtag #milan is used more often during the Milan Design Week. Forecasting the #milan time-series using Holt-Winter method, we were able to identify the anomalous usage, which is highly correlated to the usage of #mdw – the official hashtag of Milan Design Week.*

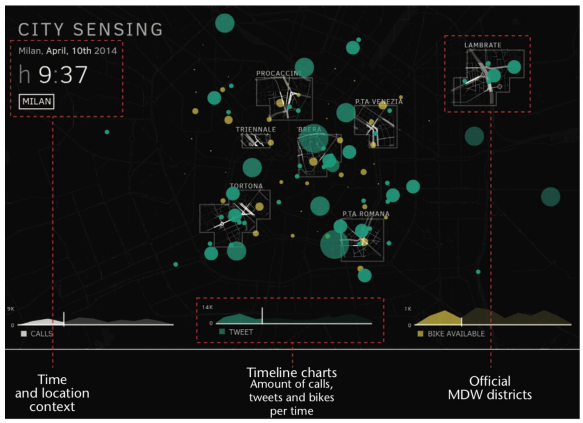
Week.

During the Milan Design Week, using Natron to compare those models with the observed usage of an hashtag, we detect in real-time emerging hashtags. Figure 7.3 illustrates how the extra usage of #milan is correlated to the appearance of the official hashtag of Milan Design Week (i.e., #mdw).

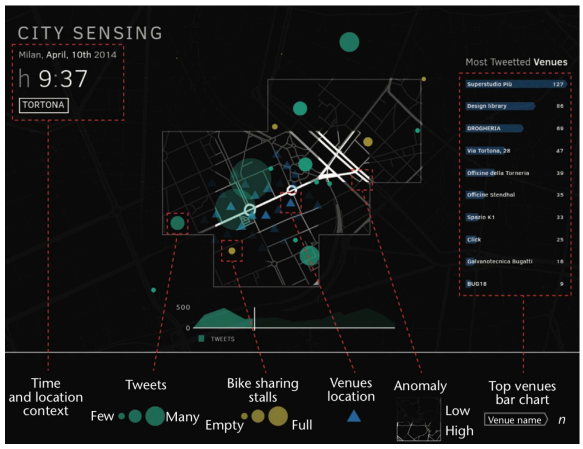
Using the analyses described above, CitySensing identifies pixels where people is talking about Milan Design Week. As detailed in Section 7.1.1, those pixels are not as numerous as those identified as anomalous using the CDRs. However, they match with almost absolute precision the pixels in which Milan Design Week events happen. The most interesting finding is that almost all those pixels are contained in mobile anomalous ones. This provides further experimental evidence that the anomalies observed in the CDRs are caused by the people coming to Milan for the Design Week.

The particular scheduling and geographical organization of the events of Milano Design Week, with most of the events concentrated in some specific areas of the city, enable also to perform analysis with irregular grid, based on the official areas of Fuorisalone.

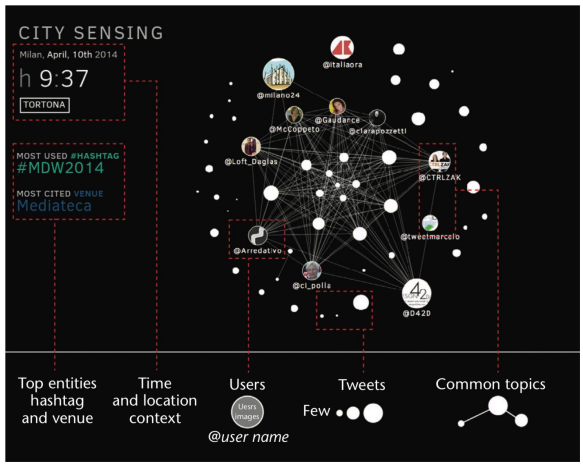
The CitySensing visualizations exploit the FraPPE concepts to relate time and space. They comprise two main views: a geographical view (see Figures 7.4(a) and 7.4(b)) that displays signals on a static map of Milan, and a graph view (see Figure 7.4(c)) that displays the evolution of the graph of people visiting MDW2014 places and events. Both can be zoomed in at the city or district level. The system underpinning the views enables the story to be told in nearly real time, but the visualized phenomenon is better viewed quickly, with the system playing a day in few minutes. To this end, a new frame,



(a)



(b)



(c)

Figure 7.4: *MDW2014 CitySensing installation geographical view (a) at city level, where the visualization highlights the pixels of official MDW districts, and (b) at district level, the map is zoomed and centered on the district. (c) The graph view displays the evolution of the graph of people visiting MDW2014 places and events. (source [23]).*

which aggregates 15 minutes of data, is displayed every 2 seconds[5].

Finally, we perform an empirical evaluation of FraPPE in order to understand how the CitySensing visualizations ease the attendees' understanding of data dynamics and, consequently, validate the Hp.3.

We start from the author-driven perspective, illustrating the visualizations as the people who watched the installation in Mediateca Santa Teresa experienced them. Then, we take the reader-driven perspective and report on the results of a questionnaire meant to assess whether the audience could guess the visualizations' intended message.

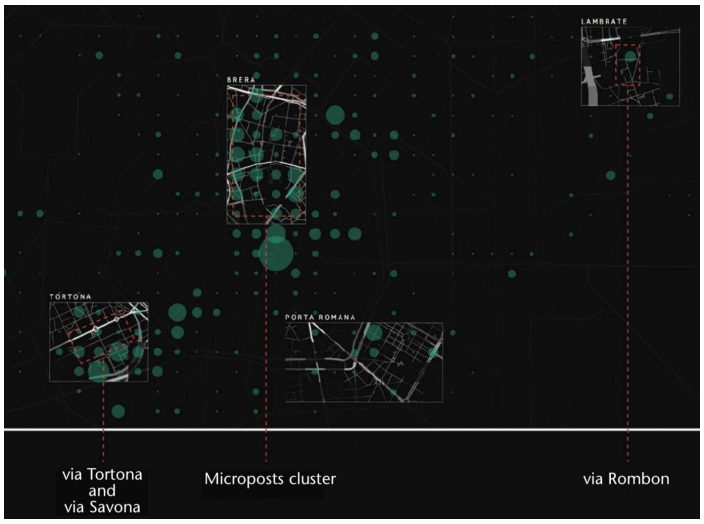
Figure 7.5(a) illustrates how the correlation between the social and mobile anomaly signals is readable in the story told by the geographical view at the city level. The figure represents a cumulative view of the frames between 6:00 and 24:00 on 10 April (the most active day in 2014). The majority of the streets of MDW districts "spot out" that is, the pixel highlighted by the mobile anomaly signal are MDW places. Furthermore, a clear pattern emerges: the MDW social signal (the green circles) originates from MDW districts.

Figure 7.5(b) presents the geographical view, but it focuses on the Brera district. This view illustrates the evolution of the social and mobile anomaly signals over three frame groups (06:00–12:00, 12:00–18:00, and 18:00–24:00) on 10 April 2014. The places where MDW events are held normally open in the late morning, but the majority of the events start in the afternoon. This is clearly visible both in the value of the mobile anomaly signal (mapped to the opacity of the pixels) and the volume of the social signal (mapped in the size of the green circles): both signals increase in most of the pixels and especially in those containing MDW places (blue triangles).

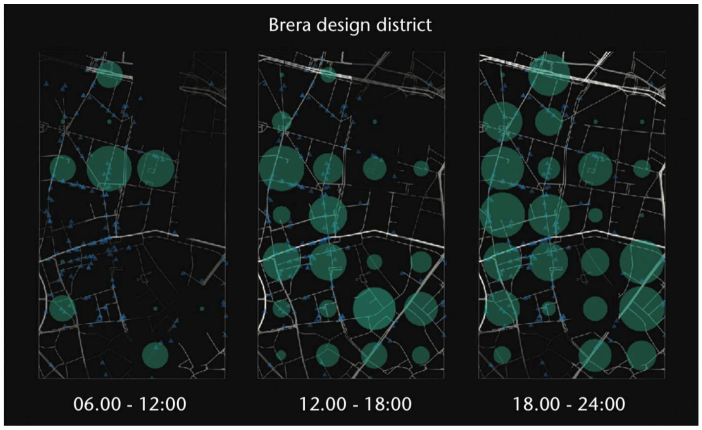
Figure 7.5(c) illustrates the graph view. It shows the evolution of the same three frame groups on 10 April 2014. During the morning, few users are linked by topics related to MDW; in the afternoon, a cluster of people talking about places and events of MDW appears; and in the evening, just few users remain unlinked. This is a direct consequence of the long-tail distribution of the discussion topics: 80 percent of the users talk about 20 percent of the places/events, while the remaining 20 percent of the users talk about the other 80 percent of places/events.

This pattern repeats over the days at city scale and in the Brera and Tortona districts. It disappears when MDW ends. To verify the Hp.3, we asked people without specific skills in data visualization and analytics to guess the message of the views shown in Figure 7.5(a) and 7.5 (c). We asked them the six questions reported in Figure 7.6. In four cases, we asked true-or-false questions, and in two cases, we asked questions that had no correct answer (see "uncertain" in the figure). The correct answers are underlined in Figure 7.6. As the distribution of the answers of the 23 responders shows, the messages we intended to transmit were correctly guessed. The responders correctly correlated the social and the mobile anomaly signals when the correlation was not evident and could not guess the correct answer when the correlation was not present. The same happens when they guess the meaning of the graph that links people based on the places and events that they jointly discussed. Those results validate Hypothesis Hp.3.

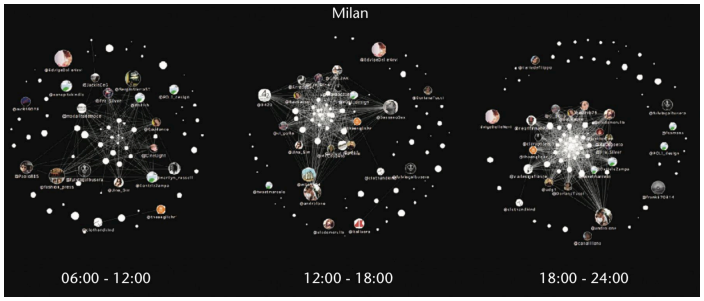
[5] http://citysensing.fuorisalone.it, http://youtu.be/MOBie09NHxM



(a)



(b)



(c)

Figure 7.5: *(a) The correlation between the social and mobile anomaly signals is readable in the story told by the geographical view at city level. On 10 April 2014, the social and mobile anomaly signals tell the success of MDW districts: Brera and Tortona beat all others. (b) A geographical view in which the social and mobile anomaly signals tell the daily pattern of activity in Brera. (c) This graph view highlights the scale-free nature of the graph of people when connected by the places, events, and hashtags they discuss on social media. (source [23]).*

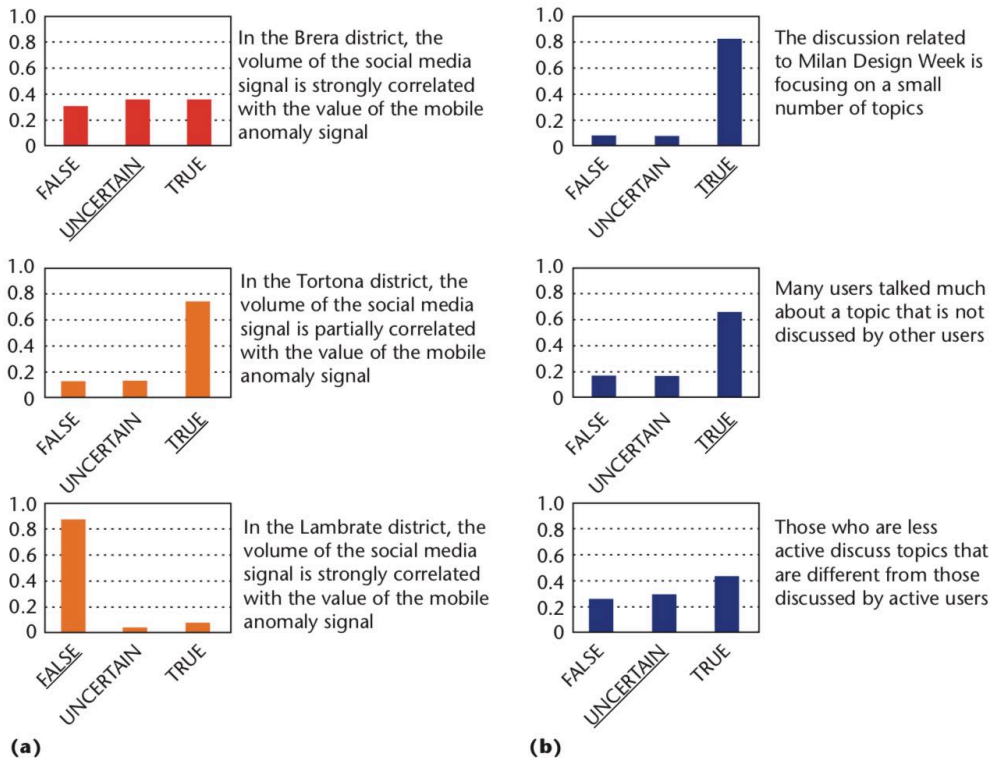


Figure 7.6: *The results of the questionnaire about the visualizations presented in (a) Figure 5 and (b) Figure 7. As we can see from the distribution of the answers of the 23 responders, the messages we intended to transmit (underlined) were correctly guessed. (source [23]).*

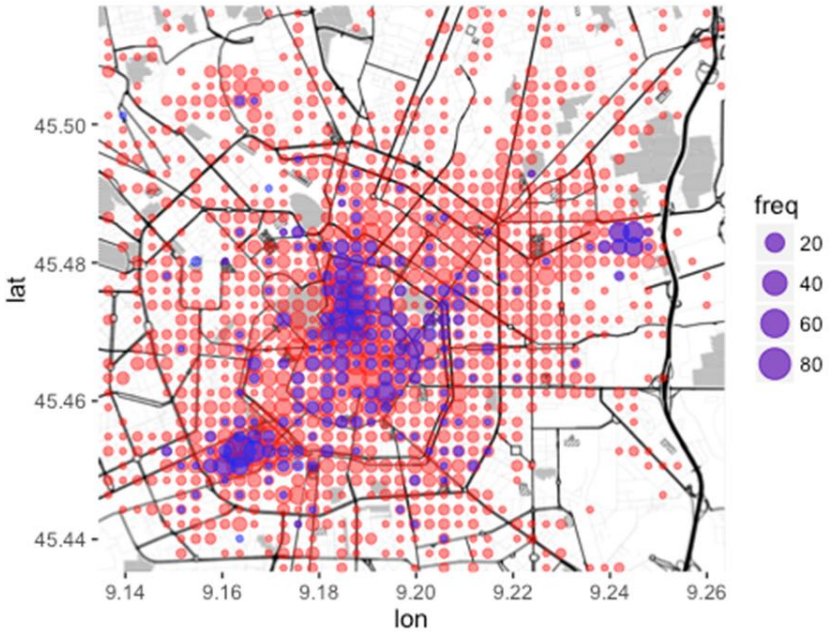
7.1.3 MDW2016 - Advanced Visualizations

Comforted by the guessability level of the CitySensing visualization, we try to reach an higher level of complexity in the data presentation. During the 2016 edition of the event we had the opportunity to collect data coming from an additional relevant source: the official mobile application of Fuorisalone. In particular, we had access to the GPS positions of places where the users open the App and the events inserted in the agenda on the App. Also, in this context, we apply the method of squared grid tessellation of the city, in order to analyze the correlation between pairs of different signals.

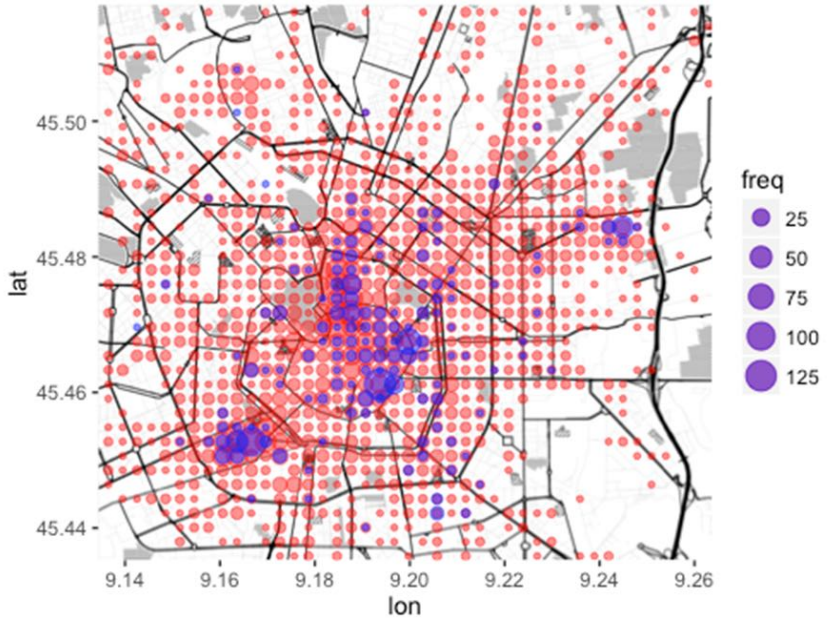
The Figures 7.7 and Figures 7.8 depicts the results, obtained exploiting FraPPE, Natron and rvr@Spark, of different use cases that involves data from heterogeneous sources (i.e., GPS record of the usage of the official Fuorisalone App, public social network, official schedule of Fuorisalone events).

Figure 7.7(a) shows the correlation between the use of the App and the number of Fuorisalone events. To visualize such a phenomenon, we consider as events the use of the App in a place, that generates a GPS record, and the scheduled event of MDW with their place that users put in their agenda. Data is aggregated grouping by Pixels and capturing daily Frames.

Another available source of Events is the geo-located activity on the public social networks: we collect the Twitter and Instagram posts geo-located in the Places contained in each Cell and we aggregate them gropuping by Pixels containing the GPS observa-



(a)



(b)

Figure 7.7: *The Figure (a) shows the number of Mobile App GPS observations collected in one day inside each square of the grid (red dots) correlated with the number of events of Milano Design Week scheduled for the same day in the same square. The Figure (b shows the correlation between GPS observations (red dots) and geo-located posts on social networks (blue dots) inside each square.*

tions, as shown in Figure 7.7(b). Both Frames show the increasing of the Events in the areas of the Fuorisalone.

Another interesting use case is represented by the analysis of the provenance of the visitors during the Milano Design Week. In order to estimate the provenance of the visitors we use the GPS information collected by the App, extracting the GPS position of the first observation logged for each user before the days of the MDW (assuming that the users download the App at home). Using appropriate shape files we map each GPS location to a Country in the world, obtaining the provenance of the user.

Mapping the GPS observation events in the grid of pixels, it is possible to visualize, for each daily frame and for each people group, which are the most popular areas of the city. Figure 7.8(a) shows the localization of the five largest groups of European foreign visitors and United States visitors. Figure 7.8(b) shows the region of provenance of Italian visitors and their distribution in the events.

As for the public installation during MDW2014, we test the guessability of the advanced visualizations. The new advanced visualizations are created for a more professional audience (i.e. Studiolabo[6], the main organizer and stakeholder of Fuorisalone). A user-centric study we conducted observing how Studiolabo used the visualizations during a workshop, allowing us to affirm that our stakeholder was able to understand the visualizations and exploit the analysis results. Therefore, these empirical evaluation of the MDW2016 visualizations enforces the validity of Hypotheses Hp.3.

7.2 Milano Fashion Week

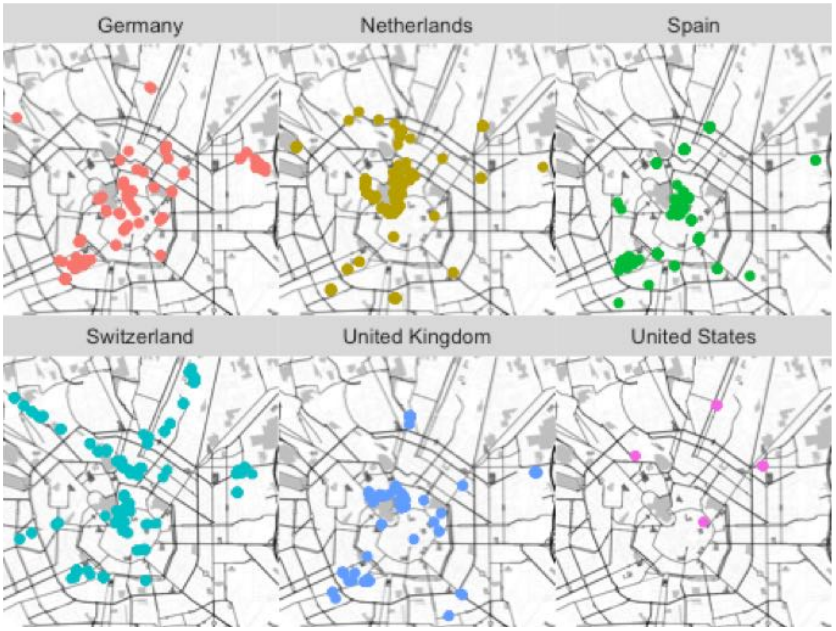
The Milano Fashion Week (MFW) represents another example of CSE in Milan. This experience deals with the problem of understanding the social media response of the MFW occurred from the 24^{th} to the 29^{th} February of 2016. We use Natron to analyze the behavior of users who re-acted (or pro-acted) in relationship with each specific fashion show during the week. MFW represents the most important meeting between market operators in the Italian fashion industry. Out of the 170 shows, we are interested only in the catwalk shows, which are the core of the fashion week. The whole set of catwalks includes a total of 73 brands; among them, 68 brands organize one single event, 4 brands organize 2 events, and 1 brand organizes 3 events.

We initially extract posts by invoking the social network APIs of Twitter and Instagram; for identifying the social reactions to MFW, we use a set of 21 hashtags and keywords provided by domain experts in the fashion sector, i.e., researchers of the *Fashion in Process* group (FIP) of Politecnico di Milano[7]. We focused on 3 weeks: the one before, the one after and the one of the event. In this way, Natron and rvr@Hive collected 106K tweets (out of which only 6.5% geo-located) and 556K Instagram posts (out of which 28% geolocated). Eventually, we opt for considering only Instagram posts, as they represent a much richer source for the particular domain of Fashion with respect to Twitter [137, 138].

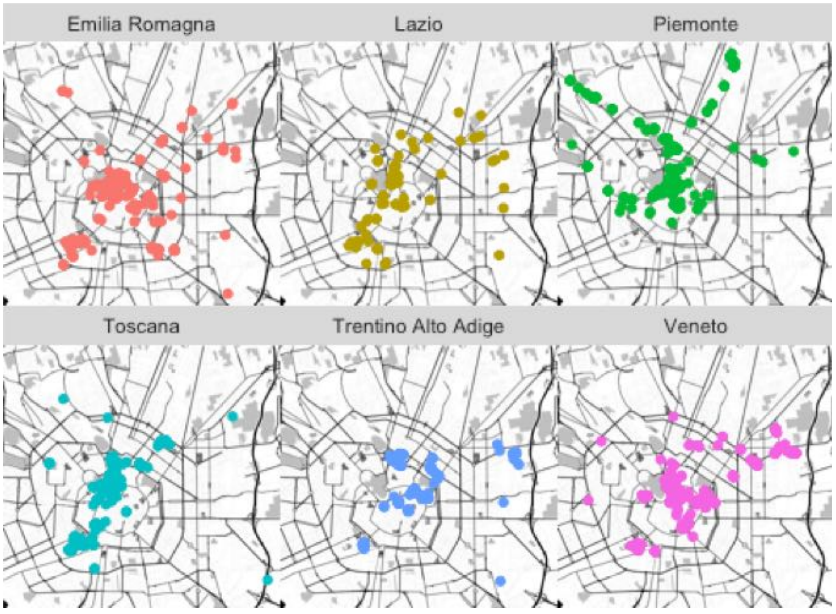
We model the data using FraPPE and exploit Natron and rvr@Hive to enable temporal, spatial and content analyses on the modeled information. Following the FraPPE approach, we build a regular Grid of cells above the area of Milan, and assigned each

[6] http://studiolabo.it/

[7] http://www.fashioninprocess.com/

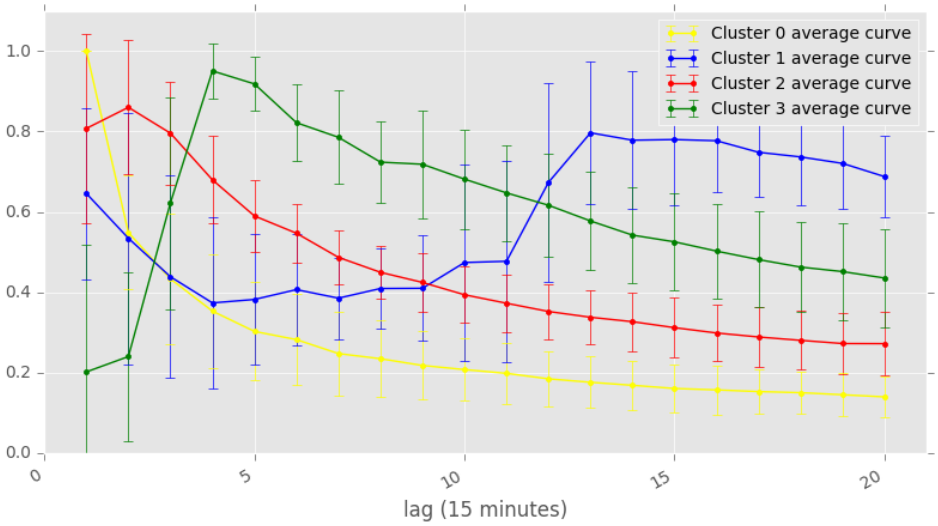


(a)

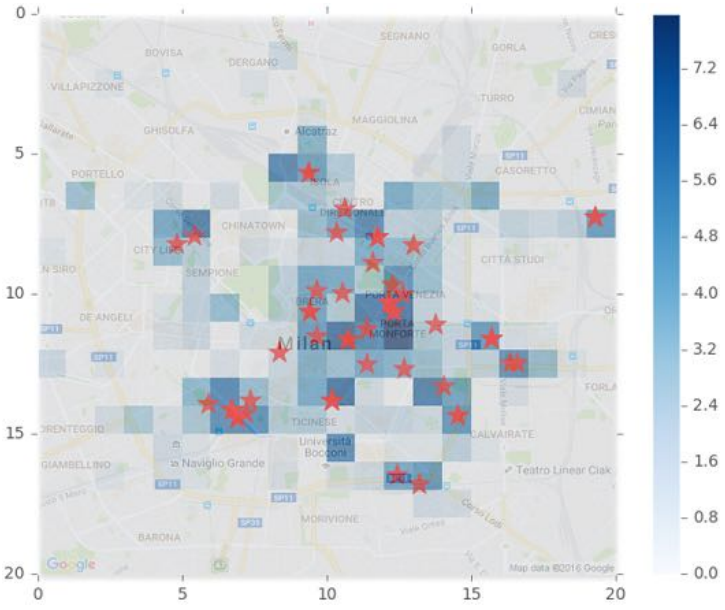


(b)

Figure 7.8: *The Figure presents (a) the localization of the five largest groups of European and US visitors and (b) Italian visitors.*



(a) *Granger Causality tests result*



(b) *Density of posts*

Figure 7.9: *(a) Granger causality test curves between physical events and social media response of each brand during the MFW, clustered by similarity of behavior; (b) Geographical dispersion over the cells of physical events (red stars) and density of social media activity (blue).*

post to the appropriate Cell. The Grid has a square shape, with sides of $10km$, divided into 20 rows and 20 columns, for a total of 400 Cells of $500m \times 500m$.

According to the FraPPE model, an Event is organized by a brand at time τ_n, hosted in the Place, located the Cell of a Grid. Each Cell is related to a Pixel and the Grid is captured by a CapturedFrame. An Agent (a user) may contribute with some OriginalContent (e.g., Instagram post), related to an Event, which in turn is going to be augmented by an automated enriching and analysis process that may add entities, as well as extract visual properties (color, pattern, ...) and concepts (objects, people, ...) from posted images.

We extend FraPPE with FrameLevelSynthesis and PixelLevelSynthesis activities in order to represent the augmentation activities on the OriginalContent. We also add further concepts to FraPPE. We name *Alive pixels* those where the percentage of posts shared in the considered pixel is more than 1% of the total number of posts in the frame. We name *Active pixels* those where the percentage of posts shared in the considered pixel is more than 10% of the total number of posts in the frame. We name *Strongly Active pixels* those where the percentage of posts shared in the considered pixel more than 20% of the total number of posts in the frame. We compute the number of alive, active and strongly active cells for all brands; we also compute the differences between subsequent durations (e.g. 3h - 6h) by counting how many cells changed their state.

Our first goal is to perform a **temporal analysis** aiming at characterizing the time at which social media respond to the events which appear in the official calendar and are linked to specific brands. Exploiting frame level synthesis we observe either peaks of reactions which then quickly disappear, or instead slower reactions that tend to remain observable for a longer time. Estimating the time latency of social responses to events is important for the brands, which could reactively plan more accurate reinforcement actions, essentially by adding well-planned social actions so as to sustain their social presence over time. We run Granger causality for each brand to compare the physical events and the social media reaction, and then we exploite k-means algorithm to cluster the brand by similarity of the Granger curves. Figure 7.9(a) shows the clusters of Granger causality curves of the brands.

Our second goal is to **analyze the geographical dispersion** of social media response. We have two different spatial signals: (i) the calendar events; and (ii) the volume of social media posts on the Web with geographical information attached, i.e., latitude and longitude. Given these two signals, several features can be computed in order to describe the spatial dispersion of posts following an event. We compute different measures that reflect the dispersion of the social media signal over time, using: *Gini coefficient*, *Average distance* of the social media signals from the event location, and the number of *alive*, *active* and *strongly active* cells. Figure 7.9 (b) shows the map representing the geographical distribution of events (represented by red stars) and post density using a synthetic frame where the pixel are darker where the density is higher.

The final result of the analysis is a set of advanced visual interfaces to be exploited by professional users. The stakeholder (FIP), exploiting those visual interfaces, was able to observe that: (i) as the duration of the frame increases, the number of *alive pixels* also increases. Moreover, the number of *active* and *strongly active pixels* is floating in the range from 1 to 3, with very few brands reaching 4 *active pixels*. (ii) At the start of the event, posts are shared near the event location, but, as looking at the bigger picture, including 24 hours or even the entire period of 24 days, the *average distance* is

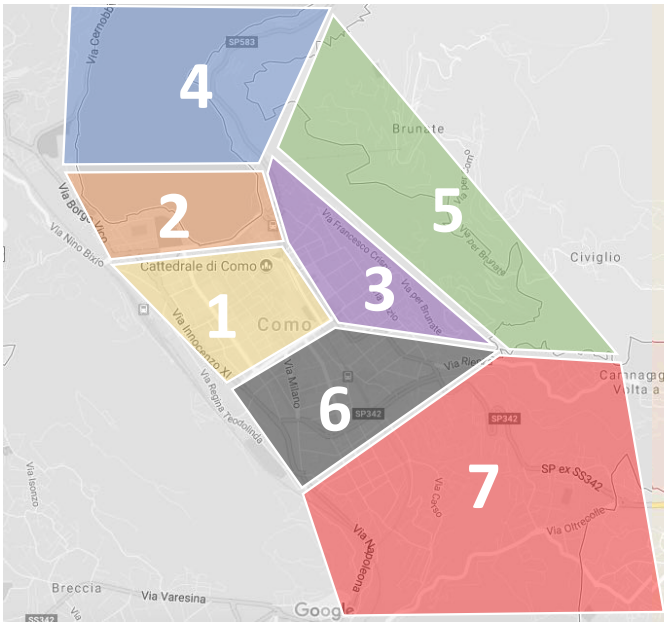


Figure 7.10: *The seven irregular data-driven cells based on mobile phone data that cover the Como territory.*

increasing, showing the growing dispersion of the social signal. (iii) The *Gini coefficient* proves how the concentration of the social signal remains always high, due also to the fact that the low percentage of users that allows Instagram to geo-tag their own photo is reducing the number of authors implied in this study, and so the few authors with high volumes of posts generated are biasing the results. However, looking at the *Gini alive coefficient*, which refers to the Gini coefficient computed only over the pixels that result alive for at least one brand in the specific frame, they can see a weak smoothing of the concentration strength with the increasing of the time-scope.

This proposed visual interfaces, based on FraPPE concept (in particular, Frame and Pixel), result guessable and enable visual correlation of the presence of an event (stars) to the density of the social activity (darker pixels) over time. The guessability of this visualizations validates, once more, Hypothesis Hp.3.

7.3 Como Smart City for Smart Citizens

Como Smart City for Smart Citizens (ComoSC^2) is a big-data integration project started in 2016 where we involve the Municipality of Como and TIM-Telecom Italia. The purpose of the project is to create a system for the integration, analysis and interpretation of the large amount and heterogeneous data coming from different sources, in order to understand Como's urban dynamics and support the decision making process of Como's local government.

As in MDW, we analyze the dynamics of **mobile phone traffic** (preventively anonymized and aggregated, according to privacy-preserving policies) in different areas of the city. Differently from the experiences in Section 7.1 and 7.2, in ComoSC^2 we use FraPPE with and irregular Grid composed by seven data-driven cells built according to the distribution of the phone antennas and the characteristics of the area. We

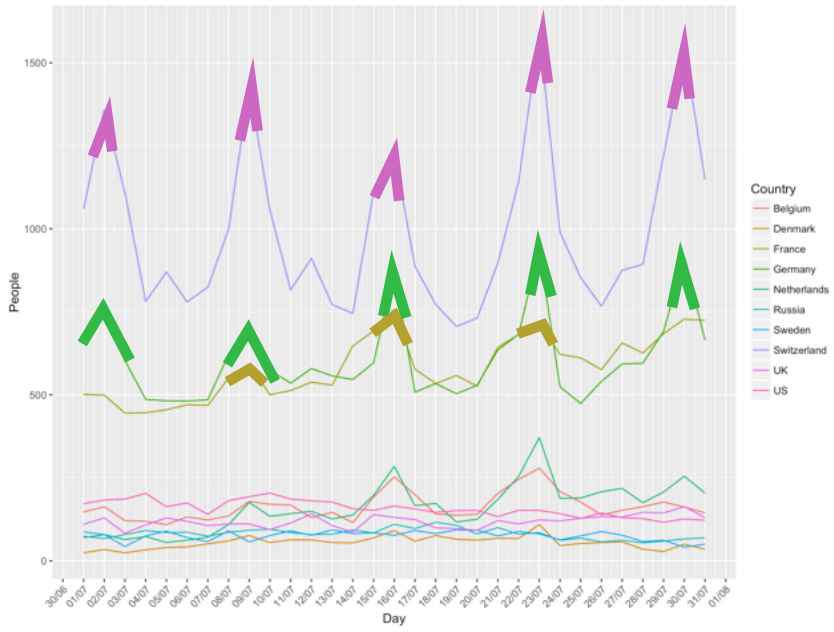
name those cells: historical city center, lakeside promenade, touristic areas outside from the historical center, lake area, mountain area around the city, business and universities area, industrial outskirts. The map in Figure 7.10 represents the distribution of the seven cells.

We collect the mobile phone traffic data during several months, in particular we focuse on the summer period (from May to October 2016). Inside each cell, exploiting rvr@Spark, we analyze the trend of mobile phone traffic capturing frames with different duration (one hour, one day) and different coverage (the complete grid or a group of cells). Anonymized mobile phone data contains also information about the SIM (like international dial-code) and demographics information about the owner of the SIM (like gender or age-range). As a result, we can perform analysis not only about the events of people presence but also about the characteristics of people (event content) and to produce visualizations to ease the understanding of complex data. For instance, Figure 7.11 shows the comparison between the number of visitors from neighboring countries of Italy. In particular Figure 7.11(a) shows the amount of visitors during the summer, while Figure 7.11(b) refers to July.

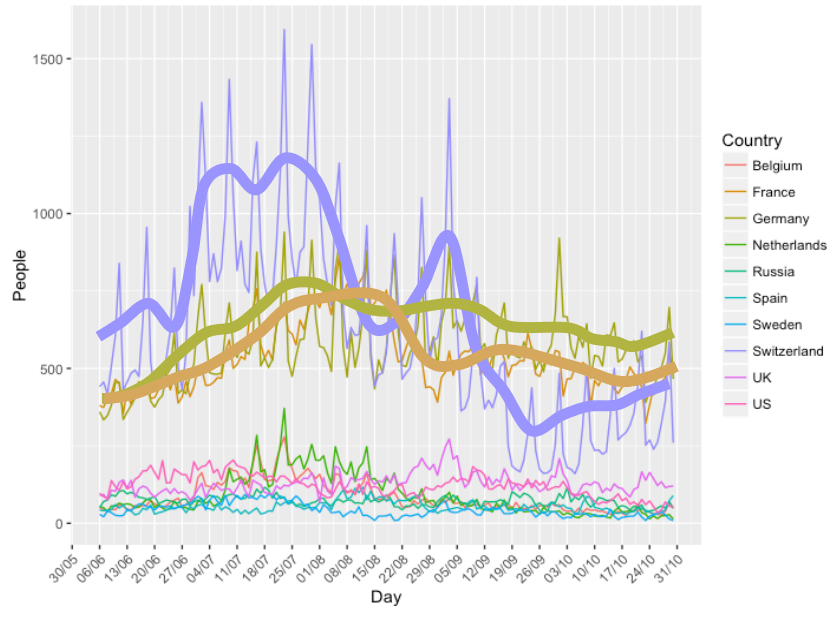
Besides mobile phone data analysis, we instrumented the Cathedral Square of Como (*Piazza Duomo*) with a set of **IoT (Internet of Things) sensors for counting people** passing in the square. The installation of the IoT sensors covers all the access to the square, and each sensor count how many people pass from the access every minute and push the results in real-time to rvr@Spark. We collect the data from IoT sensors and model it with FraPPE. Exploiting CapturedFrames with different time duration (hour, day and week), we construct the trend of passages *from* and *to* the Duomo Square according to the day of the week and the hour of the day (see Figure 7.12). This trend represents the starting point to analyze trending pattern and anomalies.

The result of the analysis is a report containing an overview of the crowd movement around the city. In order to validate Hypothesis Hp.3, we organized a workshop for the data analysis team of the Municipality of Como. Exploiting the report on the analysis of mobile traffic data, the analysts were able to identify particular trends. For instance, the analysis of Figure 7.11 shows that Swiss people usually comes to Como for shopping on Saturdays in July, but this trend dramatically decreases in September. Moreover, exploiting the passage trend analysis, the team, identifies two different patterns of people presence in Duomo Square: one for working days and one for weekends, with a significant increase of people during Saturdays and Sundays, with respect to working days. More in details, they found some differences inside the two patterns. For example in week-ends clearly emerge a difference during the evenings: Saturday evening shows a sort of persistence of people presence, while Sunday evenings appear more similar to working-day evenings. Another significant difference is the increase of people presence on Tuesday and Thursday mornings, with respect to the other working days. They are significant because the local market activities in the square during such mornings induces higher flows than usual.

The report eases the access to the results and is exploited by the Municipality of Como to enable decision making process to change the urban aspect of the city center (i.e, the creation of a new pedestrian zone). Once again, we collected experimental evidence that the FraPPE approach, together with the rvr@Spark implementation of RI⊱EЯ, enables a series of different analytics on data streams characterized by high



(a)



(b)

Figure 7.11: *(a)Number of foreign visitors per country per day in July in Como: Swiss, German, and French are the most present in the weekends. (b) Number of foreign visitors per country per day in Como from June to October. One can notice that Swiss visitors decrease sensibly in September.*

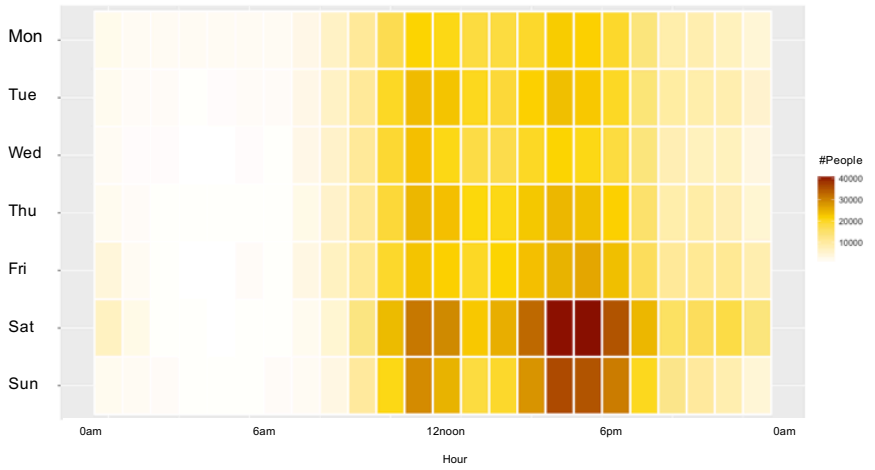


Figure 7.12: *Patterns of people presence in Duomo Square on working days and weekends: Tuesday, Thursday and Saturday are more crowded due to open market in the streets; Week ends are extremely crowded (including Saturday night).*

variety. The understandability of complex data and the enabled decision making process validate Hypothesis Hp.3.

7.4 Conclusion

In this section, we conclude the chapter by showing the correlation of the case studies with: (i) the adoption and the evolution of the FraPPE concepts, and (ii) the implementation of RI⋗EЯ that better fits the analytics needs. Table 7.1 proposes an analysis of how important each of the FraPPE concept is in the various experiments and shows the extendibility of FraPPE, while Table 7.2 shows which implementation of RI⋗EЯ is used in the various use case.

The basic FraPPE concepts were introduced during the Milan Design Week experiences. The accent is on *Place* and *Event*. Also the *Cell* and the *Pixel* – its time-variant counterpart – are important to bridge the gap between the analyzed data and the visual analytics we intend to enable [23]. The *Grid* and *Pixel* – its time-variant counterpart – were useful as abstractions, but they did not play a key role. The provenance of all steps of the analysis were documented using the generic *Action* concept; captured and synthesized were only possible value of an attribute of the action. The data of the MDW2013 and MDW2014 experiences, is analyzed using SLD (see Section 2.4.1), while the MDW2016 is the first real world validation for the *lazy transformation* approach implemented in Natron, and for the distributed implementations of RI⋗EЯ in Spark and HIVE. Indeed, during the 2016 edition, the data from telco network (CDR) is characterized by an high volume, so we first aggregate it using rvr@Spark and rvr@Hive implementations of RI⋗EЯ. Natron, instead, is used to analyze social network data, enrich it and merge the different data aggregations.

The Milano Fashion Week experience is key to extend the original FraPPE with the concepts that allow describing the content as well as to extend FraPPE with some provenance concepts. Specifically for this experience, in FraPPE 2.0.we introduce the distinction between *original* and *augmented* content at *event-level*. We also perceive

Table 7.1: *A comparison of how the FraPPE concepts are used in the two large-scale events of Milan Design Week (MDW) and Milan Fashion Week (MFW) as well as in a longitudinal analysis we performed Como. The 'x' symbols have the following meaning: xxx – key concept; xx – important concept; and x – useful concept. The lack of stars means that the concept was not used.*

		MDW	MFW	Como
Spatial	Place	xxx	xxx	x
	Cell	xx	x	xx
	Grid	x	x	xx
Temporal	Event	xxx	xxx	x
	Pixel	xx	x	xx
	Frame	x	x	xx
	- CapturedFrame	xx		xxx
	- SyntheticFrame	xx		xxx
Content	Content	xxx	x	
	- Event-level content		xxx	x
	- Original content		xxx	x
	- Augmented content		xx	x
	- Pixel-level synthesis		xxx	xxx
	- Frame-level synthesis		xx	xxx
Provenance	Action	x		
	- Capture		x	xx
	- Synthesize		xx	xx
	- Augment		xx	xx

Table 7.2: *An overview of analysis technologies used in the two large-scale events of Milan Design Week (MDW) and Milan Fashion Week (MFW) as well as in a longitudinal analysis we performed Como.*

	MDW2013	MDW2014	MDW2016	MFW	ComoSC^2
SLD	x	x			
Natron			x	x	
rvr@Hive			x	x	
rvr@Spark			x		x

the need to model the content that we connected to each pixels, namely the *pixel-level synthesis*. We reflecte this extension also in the provenance part of FraPPE introducing the *capture*, the *augment* and the *synthesize* actions. The data of the MFW was analyzed exploiting Natron and rvr@Hive.

The longitudinal analysis, which we performed on Como in the context of ComoSC^2, serve as validation for FraPPE 2.0. All the concepts are used, although their use and benefit depends on the different types of analysis performed. In particular, more emphasis is posed on the *Grid*, the *Frames* and the *frame-level synthesis*, which is introduced in FraPPE during this experience. The data involved in the Como experience, in particular the demographically enriched CDR, must be pre-aggregated in order to manage its volume. We exploit rvr@Spark to perform this task and to aggregate the results.

The experiences in Milan (MDW and MFW) and in Como demonstrate the validity of the FraPPE approach in the data representation, and the robustness of the implementations of RI≻EЯ. The guessability of the data visualizations improved the understanding of the urban environment from different points of view and enabled decision making

processes by the stakeholders. The results of these experiences validate Hypothesis Hp.3.

CHAPTER 8

Conclusion

During the PhD research work reported in this thesis we develop the research question exploiting the Macro, Mezzo and Micro methodology. At Macro level, we focus on relevancy and formulate the question: *Is it possible to support reactive decisions by managing data characterized by velocity and variety without forgetting volume?*

In order to specify a problem for which we can find a viable solution, at Mezzo Level, we concentrate our effort on the task of visually making sense of spatio-temporal streaming data. The result of those reflections is the question: *Is it possible to visually making sense of a variety of spatio-temporal streaming data by enabling continuous ingestion and reactive analysis?*

While trying to formalize a question for which we can find a solution that can be evaluated, we start looking for spatio-temporal data sources to enable the processes of formal and empirical evaluations. We observed that the modern cities offer a growing volume of heterogeneous flowing data from sensors, telecommunication infrastructures, time tables of public services, and, last but not least, from the people who leave the city every day (e.g., citizens, commuters, tourists, etc.). Thanks to the nature and the availability of those urban data, the interest around them are growing fast. So, in this PhD thesis we investigate the question: *Is it possible to continuously ingest and reactively analyses a variety of streaming urban data in order to visualize emerging patterns and their dynamics?*

8.1 Review of the Contributions

In this section, we offer an overview of the thesis' contributions in terms of the problems solved and how they offer a valid solution for the research question. The development of each contribution is related to one or more research problems and its validation is

guided by the formulation of one or more hypotheses.

Reflecting on the research question, we split the research work in two different sublines. On the one hand, we face the problem of data modeling, on the other hand, we tackle the data computation problem.

While focusing on the problem of modeling spatio-temporal data to enable time, space and content analyses a first problem emerges:

Rp.1 Defining a conceptual model to represent a variety of streaming data.

To address Rp.1 we formulate the hypotheses Hp.1.1 and Hp.1.2:

Hp.1.1 A conceptual model containing terms from the image processing domain can represent spatio-temporal data in an extendable and coherent way with a minimal encoding bias and a minimal ontological commitment.

Hp.1.2 Visual analytics interfaces built directly on data represented with the conceptual model of Hp.1.1 are guessable.

In order to validate Hp.1.1 and Hp.1.2, we propose FraPPE ontology (see Chapter 4) that exploits digital image processing terms to tame three main dimensions of analysis (i.e., space, time, and content) and enables OBDA operations on heterogeneous spatio-temporal data.

FraPPE bridges the gap between the data engineer perspective and the visual analytics perspective. We formally evaluated FraPPE by checking its adherence to the Tom Gruber's principle and, in doing so, we validate the Hypothesis Hp.1.1. From the visual analytics perspective, we validate the Hypothesis Hp.1.2 by empirically checking the guessability of the visualizations created exploiting the data modeled using FraPPE.

While focusing on the data computation research line two problems emerge:

Rp.2 Defining a streaming computational model to enable analysis on a variety of data.

Rp.3 Defining appropriate technical instantiations of the computational model in Rp.2.

In order to investigate the research questions and address Rp.2 and Rp.3, we formulate the hypotheses:

Hp.2.1 The implementation of a streaming computational model that defers as long as possible the data transformation demands less resources and better approximates the correct answer under stress conditions than an implementation of a computational model that cast data into RDF at ingestion time.

Hp.2.2 A single-threaded implementation of the streaming computational model from Hp.2.1 is more cost-effective than a distributed implementation of the same model while guaranteeing the reactiveness of the system.

Aiming at validating the hypotheses, we propose RI⋗EЯ computational model and its implementations (see Chapter 5 and Chapter 6).

RI⋗EЯ is a streaming computational model inspired by two principles: **(P1)** *everything is a data stream*, and **(P2)** *continuous ingestion*. It is built around the idea of *Lazy Transformation*. A system that implements RI⋗EЯ postpones data transformations until it can really benefits from them.

In order to test the validity of the *Lazy Transformation* approach, we propose Natron a single-threaded vertically scalable implementation of RI⊱EЯ and evaluated it against our Streaming Linked Data (SLD) engine that apply data transformation (to RDF) at ingestion time. We evaluate the systems in terms of correctness and resource consumption. The result of such evaluation validates the Hypothesis Hp.2.1 and convince us in assuming the Lazy Transformation as a third principle **(P3)**.

In order to prove the adequacy of RI⊱EЯ in different work conditions, we propose two horizontally scalable implementations based on distributed technologies (rvr@Spark and rvr@Hive). Aiming at reaffirming the importance of cost-effectiveness in Stream Processing field we evaluate Natron against rvr@Spark. The overall results of the this evaluation validate Hypothesis Hp.2.2.

Focusing on assessing the validity of the proposed solutions, a last problem emerges:

Rp.4 Assess, in real world scenarios, the feasibility and the effectiveness of the instantiations developed addressing Rp.3 using the models developed in solving Rp.1 and Rp.2.

In order to solve Rp.4, we formulate the hypothesis:

Hp.3 An implementation of RI⊱EЯ computational model presented in Chapter 5, can create a bridge between data analytics and data visualization that enhances the comprehension of a variety of spatio-temporal data and, at the same time, allows reactive decisions.

Aiming at validating Hypothesis Hp.3, we put at work FraPPE and RI⊱EЯ's implementations in four week-long case studies in Milan (during the Milano Design Week in 2014, 2015 and 2016, and the Milano Fashion Week 2016) and a 6 months-long use case in Como. We evaluate the guessability of the visualizations by collecting the answers to a questionnaire proposed to the public audiences and by organizing workshops for the use cases' stakeholders – Studiolabo[1], one of the biggest organizer of the Milano Design Week events and the Municipalities of Como. Both of the audiences were asked to visually correlate event and to find data patterns through the visualizations. The collected results demonstrate the validity of the FraPPE approach in the data representation, and the robustness of the implementations of RI⊱EЯ.

8.2 Limitations and Future Directions

In this section, we discuss the limitations we identified in this research work and the future directions of the research in order to overcome those limitations.

The FraPPE evaluation shows its effectiveness in making spatio-temporal data ready for visual analytics. We prove the validity of the model at Micro and Mezzo level, but we have no evidence of it at Macro level. In the future, it is worth to keep working on the validation of the model by involving different data for different reactive tasks.

Moreover, in order to reduce the ontology complexity and enable OBDA operations, FraPPE exploits only the terms of external ontologies, without any axiomatization. The relation between OBDA and data model complexity represents a challenge in multiple fields and it is an hot research topic. For instance, in spatial reasoning, the transitive

[1] http://www.studiolabo.it

relations play a crucial role, but they are not compatible with an OWL-QL data model and, consequently, with OBDA. In this field, Eiter et al. in [139] exploit a DL-Lite data model to present a query rewriting approach for the traffic, while in [140] they present two automatic routing use cases. In parallel, Kontchakov et al. in [141] propose a new query language, based on datalog, for performing spatial analysis. Also the data analytics world suffers the problem. Mehdi et al. in [142] propose an innovative approach to create ontologies for the data analysis, while Kharlamov et al. in [143] present a typical use cases of reasoning for data analysis. Artale et al. in the survey [144] on time series analysis methods demonstrate how this problem is hot and wide, while Brandt et al. in [145] proposes different methodology for dealing with it. In the future it is worth to keep working in the field by improving FraPPE expressiveness, by broadening the range of usage fields and by pushing for the FraPPE adoption.

From the data computation point of view, RI⋗EЯ and the Pipeline Definition Language (PDL), at this development stage, help users in designing computational plans. Moreover, they do not offer any concrete help in abstracting from the physical implementation of each operator. In particular, PDL, is only a graphical syntax, and it is still missing an editor and a compiler. Next step is to use RI⋗EЯ and PDL for automating optimizations of the streaming computation. Future developments of RI⋗EЯ and PDL should concern static optimization – regarding the automatic selection of the physical operators – and dynamic optimization – regarding the automatic decision about the best moment to perform a data transformation. Investigating on the formal definition of operators' cost model represents a viable solution to ease the proposed optimizations. The operators' cost model allows to estimate the overall cost of the pipelines and to automatically detect the best computational plan from the cost-effectiveness perspective. From the external evaluation point of view, a formal cost model allows a formal comparison of RI⋗EЯ against already existing computational model. Last but not least, we also aim at automating the code generation. To this end, we need to work on the alignment of the algebraic representations of RI⋗EЯ to the existing stream processing engines that we want to target with our code generation.

As for FraPPE, it is worth to keep working on the validation of RI⋗EЯ through its implementation by broadening the type of involved tasks and data. In particular, the development of the distributed implementation of RI⋗EЯ is still at an early stage. Distributed framework is an hot topic and, in the future, it is worth to keep working on rvr@Spark and rvr@Hive in order to exploit their potential. Moreover, it is worth to keep working on comparison by broadening the range of the involved system (e.g., CQELS Cloud [70], Strider [72], etc.) and exploiting well known benchmark (e.g. Yahoo! benchmark).

8.3 Reflections

In this thesis, we proposed a complete set of instruments to enable reactive decision making through the visual analysis of a variety of spatio-temporal data. The results of the formal and empirical evaluations show that FraPPE conceptual model, RI⋗EЯ streaming computational model and its implementations (Natron, rvr@Spark and rvr@Hive) represent valid, effective and feasible solutions to the research questions and improve the state of the art.

From the data modeling point of view the presented conceptual model propose an holistic perspective on the spatio-temporal data in order to enable data visualization and different analysis. From the data computation perspective, the proposed computational model with its implementation represent a valid solution to ingest, analyze and emit a variety of streaming data. Moreover, we validated the proposed solutions in five different urban use cases.

However, our approach presents different limitations from both the perspective, such as the missing axiomatizations of the terms from the imported ontologies in the conceptual model and the missing definition of a formal cost-model for the computational model. Those limitations open the door to further investigations and optimizations.

Bibliography

[1] Rob Kitchin. The real-time city? big data and smart urbanism. *GeoJournal*, 79(1):1–14, 2014.

[2] Marco Balduini, Irene Celino, Daniele Dell'Aglio, Emanuele Della Valle, Yi Huang, Tony Kyung-il 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. *J. Web Sem.*, 16:33–41, 2012.

[3] 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 Sem.*, 27:26–33, 2014.

[4] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1082–1090, New York, NY, USA, 2011. ACM.

[5] 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 *International Semantic Web Conference (2)*, volume 8219 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013.

[6] Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[7] Richard A. Becker, Ramón Cáceres, Karrie Hanson, Ji Meng Loh, Simon Urbanek, Alexander Varshavsky, and Chris Volinsky. A tale of one city: Using cellular network data for urban planning. *IEEE Pervasive Computing*, 10(4):18–26, 2011.

[8] Francesco Calabrese, Massimo Colonna, Piero Lovisolo, Dario Parata, and Carlo Ratti. Real-time urban monitoring using cell phones: A case study in rome. *IEEE Trans. Intelligent Transportation Systems*, 12(1):141–151, 2011.

[9] Desislava Hristova, David Liben-Nowell, Anastasios Noulas, and Cecilia Mascolo. If you've got the money, i've got the time: Spatio-temporal footprints of spending at sports events on foursquare. In *CitiLab@ ICWSM*, 2016.

[10] Marco Balduini, Irene Celino, Daniele Dell'Aglio, Emanuele Della Valle, Yi Huang, Tony Kyung-il Lee, Seon-Ho Kim, and Volker Tresp. Reality mining on micropost streams - deductive and inductive reasoning for personalized and location-based recommendations. *Semantic Web*, 5(5):341–356, 2014.

[11] Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[12] 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.

[13] Marco Balduini, Alessandro Bozzon, Emanuele Della Valle, Yi Huang, and Geert-Jan Houben. Recommending venues using continuous predictive social media analytics. *IEEE Internet Computing*, 18(5):28–35, 2014.

[14] Jeffrey R. Lacasse and Eileen Gambrill. Making assessment decisions: Macro, mezzo, and micro perspectives. In *Critical Thinking in Clinical Assessment and Diagnosis*, pages 69–84. Springer, 2015.

[15] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246. ACM, 2002.

[16] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. 1997.

[17] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[18] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? In *HotOS*. USENIX Association, 2015.

[19] Fabrizio Antonelli, Matteo Azzi, Marco Balduini, Paolo Ciuccarelli, Emanuele Della Valle, and Roberto Larcher. City sensing: visualising mobile and social data about a city scale event. In *AVI*, pages 337–338. ACM, 2014.

[20] Marco Balduini and Emanuele Della Valle. Frappe: A vocabulary to represent heterogeneous spatio-temporal data to support visual analytics. In *ISWC (2)*, volume 9367 of *LNCS*, pages 321–328. Springer, 2015.

[21] Emanuele Della Valle and Marco Balduini. Listening to and visualising the pulse of our cities using social media and call data records. In *BIS (Workshops)*, volume 228 of *LNBIP*, pages 3–14. Springer, 2015.

[22] Marco Balduini, Emanuele Della Valle, and Riccardo Tommasini. SLD revolution: A cheaper, faster yet more accurate streaming linked data framework. In *ESWC (Satellite Events)*, volume 10577 of *LNCS*, pages 263–279. Springer, 2017.

[23] Marco Balduini, Emanuele Della Valle, Matteo Azzi, Roberto Larcher, Fabrizio Antonelli, and Paolo Ciuccarelli. Citysensing: Fusing city data for visual storytelling. *IEEE MultiMedia*, 22(3):44–53, 2015.

[24] Marco Balduini, Sivam Pasupathipillai, and Emanuele Della Valle. Cost-aware streaming data analysis: Distributed vs single-thread. In *DEBS*, pages 160–170. ACM, 2018.

[25] Marco Balduini, Marco Brambilla, Emanuele Della Valle, Christian Marazzi, Tahereh Arabghalizi, Behnam Rahdari, and Michele Vescovi. Models and practices in urban data science at scale. *Big Data Research*, 2018.

[26] Marco Balduini. On the continuous and reactive analysis of a variety of spatio-temporal data. Submitted to ISWC 2018 DC.

[27] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 2016.

[28] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

[29] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In *ESWC*, 2010.

[30] 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 (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 370–388. Springer, 2011.

[31] Jean-Paul Calbimonte, Óscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2010.

[32] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW2011*, pages 635–644, 2011.

[33] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

[34] Irina Botan, Roozbeh Derakhshan, Nihal Dindar, Laura M. Haas, Renée J. Miller, and Nesime Tatbul. SECRET: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.

[35] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

[36] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[37] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[38] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.

[39] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[40] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: fault-tolerant streaming computation at scale. In Michael Kaminsky and Mike Dahlin, editors, *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pages 423–438. ACM, 2013.

[41] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured streaming: A declarative API for real-time applications in apache spark. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 601–613. ACM, 2018.

[42] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - A warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[43] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[44] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[45] Michael R. Genesereth and Nils J. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1988.

[46] Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J Carroll, and Brian McBride. Rdf 1.1 concepts and abstract syntax. *W3C recommendation*, 25(02), 2014.

[47] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10), 2013.

[48] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.

[49] Mari Carmen Suárez-Figueroa. *NeOn methodology for building ontology networks: specification, scheduling and reuse*. PhD thesis, Technical University of Madrid, 2012.

[50] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[51] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

[52] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient OWL reasoner. In *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[53] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfox: A highly-scalable RDF store. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.

[54] Boris Motik, Yavor Nenov, Robert Edgar Felix Piro, and Ian Horrocks. Incremental update of datalog materialisation: the backward/forward algorithm. In *AAAI*, pages 1560–1568. AAAI Press, 2015.

[55] Raphael Volz, Steffen Staab, and Boris Motik. Incrementally maintaining materializations of ontologies stored in logic databases. *J. Data Semantics*, 2:1–34, 2005.

[56] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.

[57] Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. A first step towards stream reasoning. In *FIS*, volume 5468 of *Lecture Notes in Computer Science*, pages 72–81. Springer, 2008.

[58] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010.

[59] Srdjan Komazec, Davide Cerri, and Dieter Fensel. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In *DEBS*, pages 58–68. ACM, 2012.

[60] Yuan Ren and Jeff Z. Pan. Optimising ontology stream reasoning with truth maintenance system. In *CIKM*, pages 831–836. ACM, 2011.

[61] Fredrik Heintz and Patrick Doherty. Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 15(1):3–13, 2004.

[62] Fredrik Heintz. *DyKnow : A Stream-Based Knowledge Processing Middleware Framework*. PhD thesis, Linköping University, Sweden, 2009.

[63] Daniel de Leng and Fredrik Heintz. Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks. In *AAAI*, pages 957–963. AAAI Press, 2016.

[64] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012.

[65] Emanuele Della Valle, Daniele Dell'Aglio, and Alessandro Margara. Taming velocity and variety simultaneously in big data with stream reasoning: tutorial. In *DEBS*, pages 394–401. ACM, 2016.

[66] Davide Francesco Barbieri and Emanuele Della Valle. A proposal for publishing data streams as linked data - A position paper. In *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*, 2010.

[67] 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):17–44, 2014.

[68] Jean-Paul Calbimonte, Hoyoung Jeung, Óscar Corcho, and Karl Aberer. Enabling query technologies for the semantic sensor web. *Int. J. Semantic Web Inf. Syst.*, 8(1):43–63, 2012.

[69] Mikko Rinne, Esko Nuutila, and Seppo Törmä. INSTANS: high-performance event processing with standard RDF and SPARQL. In *International Semantic Web Conference (Posters & Demos)*, volume 914 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[70] 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 *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 2013.

[71] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*. USENIX Association, 2010.

[72] Xiangnan Ren and Olivier Curé. Strider: A hybrid adaptive distributed RDF stream processing engine. In *International Semantic Web Conference (1)*, volume 10587 of *Lecture Notes in Computer Science*, pages 559–576. Springer, 2017.

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

[74] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[75] 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.

[76] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[77] Jim Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[78] Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In *VLDB*, pages 480–491. Morgan Kaufmann, 2004.

[79] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Peng, and Paul Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *IPDPS Workshops*, pages 1789–1792. IEEE Computer Society, 2016.

[80] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. Benchmarking distributed stream processing engines. *CoRR*, abs/1802.08496, 2018.

[81] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthikeyan Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy V. Ryaboy. Storm@twitter. In *SIGMOD Conference*, pages 147–156. ACM, 2014.

[82] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[83] Thomas Scharrenbach, Jacopo Urbani, Alessandro Margara, Emanuele Della Valle, and Abraham Bernstein. Seven commandments for benchmarking semantic flow processing systems. In *ESWC*, volume 7882 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 2013.

[84] Daniele Dell'Aglio, Jean-Paul Calbimonte, Marco Balduini, Óscar Corcho, and Emanuele Della Valle. On correctness in RDF stream processor benchmarking. In *International Semantic Web Conference (2)*, volume 8219 of *Lecture Notes in Computer Science*, pages 326–342. Springer, 2013.

[85] Ying Zhang, Minh-Duc Pham, Óscar Corcho, and Jean-Paul Calbimonte. Srbench: A streaming RDF/S-PARQL benchmark. In *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 641–657. Springer, 2012.

[86] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. Citybench: A configurable benchmark to evaluate RSP engines using smart city datasets. In *International Semantic Web Conference (2)*, volume 9367 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2015.

[87] Maxim Kolchin, Peter Wetz, Elmar Kiesling, and A Min Tjoa. Yabench: A comprehensive framework for RDF stream processor correctness and performance assessment. In *ICWE*, volume 9671 of *Lecture Notes in Computer Science*, pages 280–298. Springer, 2016.

[88] Riccardo Tommasini, Emanuele Della Valle, Marco Balduini, and Daniele Dell'Aglio. Heaven: A framework for systematic comparative research approach for RSP engines. In *ESWC*, volume 9678 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2016.

[89] Christoph Boden, Tilmann Rabl, and Volker Markl. Distributed machine learning-but at what cost? Private Communication, 2018.

[90] Francesco Calabrese, Laura Ferrari, and Vincent D. Blondel. Urban sensing using mobile phone network data: A survey of research. *ACM Comput. Surv.*, 47(2):25:1–25:20, 2014.

[91] Marta C Gonzalez, Cesar A Hidalgo, and A-L Barabasi. Understanding individual human mobility patterns. *arXiv preprint arXiv:0806.1256*, 2008.

[92] Julián Candia, Marta C González, Pu Wang, Timothy Schoenharl, Greg Madey, and Albert-László Barabási. Uncovering individual and collective human dynamics from mobile phone records. *Journal of physics A: mathematical and theoretical*, 41(22):224015, 2008.

[93] Gavin McArdle, Giusy Di Lorenzo, Fabio Pinelli, Francesco Calabrese, and Erik Van Lierde. Analyzing social events in real-time using big mobile data. *IEEE COMSOC MMTC E-Letter*, 2015.

[94] Ramón Cáceres, James Rowl, Christopher Small, and Simon Urbanek. Exploring the use of urban greenspace through cellular network activity, 2012.

[95] Carlo Ratti, Dennis Frenchman, Riccardo Maria Pulselli, and Sarah Williams. Mobile landscapes: using location data from cell phones for urban analysis. *Environment and Planning B: Planning and Design*, 33(5):727–748, 2006.

[96] Marco De Nadai, Jacopo Staiano, Roberto Larcher, Nicu Sebe, Daniele Quercia, and Bruno Lepri. The death and life of great italian cities: a mobile phone data perspective. In *Proceedings of the 25th International Conference on World Wide Web*, pages 413–423. International World Wide Web Conferences Steering Committee, 2016.

[97] Jane Jacobs. *The death and life of American cities*. 1961.

[98] Amy Wesolowski, Nathan Eagle, Abdisalan M Noor, Robert W Snow, and Caroline O Buckee. The impact of biases in mobile phone ownership on estimates of human mobility. *Journal of the Royal Society Interface*, 10(81):20120986, 2013.

[99] Myeong Lee, Rosta Farzan, and Brian S Butler. This is not just a café: Toward capturing the dynamics of urban places. In *CitiLab@ ICWSM*, 2016.

[100] Achilleas Psyllidis, Alessandro Bozzon, Stefano Bocconi, and Christiaan Titos Bolivar. A platform for urban analytics and semantic data integration in city planning. In *CAAD Futures*, pages 21–36. Springer, 2015.

[101] Vivek K. Singh, Mingyan Gao, and Ramesh Jain. Social pixels: genesis and evaluation. In *ACM Multimedia*, pages 481–490. ACM, 2010.

[102] Federico Botta, Helen Susannah Moat, and Tobias Preis. Quantifying crowd size with mobile phone and twitter data. *Royal Society open science*, 2(5):150162, 2015.

[103] Daniele Quercia, Giusy Di Lorenzo, Francesco Calabrese, and Carlo Ratti. Mobile phones and outdoor advertising: measurable advertising. Institute of Electrical and Electronics Engineers, 2011.

[104] Francesco Calabrese, Francisco C Pereira, Giusy Di Lorenzo, Liu Liang, and Carlo Ratti. The geography of taste: Analyzing cell-phone mobility and social events. In *Pervasive*, volume 10, pages 22–37. Springer, 2010.

[105] Daniele Quercia, Neal Lathia, Francesco Calabrese, Giusy Di Lorenzo, and Jon Crowcroft. Recommending social events from mobile phone location data. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 971–976. IEEE, 2010.

[106] Kaoutar Ben Ahmed, Mohammed Bouhorma, and Mohamed Ben Ahmed. Smart citizen sensing: a proposed computational system with visual sentiment analysis and big data architecture. *IJCA*, 152(6), 2016.

[107] Bartosz Hawelka, Izabela Sitko, Euro Beinat, Stanislav Sobolevsky, Pavlos Kazakopoulos, and Carlo Ratti. Geo-located twitter as proxy for global mobility patterns. *Cartography and Geographic Information Science*, 41(3):260–271, 2014.

[108] Francesco Calabrese, Giusy Di Lorenzo, and Carlo Ratti. Human mobility prediction based on individual and collective geographical preferences. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 312–317. IEEE, 2010.

[109] UN Habitat. Urbanization and development: emerging futures; world cities report 2016. *Nairobi, UN Habitat*, 2016.

[110] Matthew R Sanderson, Ben Derudder, Michael Timberlake, and Frank Witlox. Are world cities also world immigrant cities? an international, cross-city analysis of global centrality and immigration. *International Journal of Comparative Sociology*, 56(3-4):173–197, 2015.

[111] Pierre Deville, Catherine Linard, Samuel Martin, Marius Gilbert, Forrest R Stevens, Andrea E Gaughan, Vincent D Blondel, and Andrew J Tatem. Dynamic population mapping using mobile phone data. *Proceedings of the National Academy of Sciences*, 111(45):15888–15893, 2014.

[112] Jakob RE Leimgruber. The management of multilingualism in a city-state. *Multilingualism and language diversity in urban areas: Acquisition, identities, space, education*, 1:227, 2013.

[113] Ofelia García and Joshua A Fishman. *The multilingual apple: languages in New York City*. Walter de Gruyter, 2001.

[114] Guus Extra and Kutlay YaÇ§mur. *Urban multilingualism in Europe: Immigrant minority languages at home and school*, volume 130. Multilingual matters, 2004.

[115] Dan Tasse, Jennifer T Chou, and Jason I Hong. Generating neighborhood guides from social media. In *CitiLab@ ICWSM*, 2016.

[116] Michela Arnaboldi, Marco Brambilla, Beatrice Cassottana, Paolo Ciuccarelli, Davide Ripamonti, Simone Vantini, and Riccardo Volonterio. Studying multicultural diversity of cities and neighborhoods through social media language detection. In *CitiLab@ ICWSM*, 2016.

[117] Eszter Bokányi, Dániel Kondor, László Dobos, Tamás Sebők, József Stéger, István Csabai, and Gábor Vattay. Race, religion and the city: twitter word frequency patterns reveal dominant demographic dimensions in the united states. 2015.

[118] Marco Quaggiotto, Donato Ricci, Gaia Scagnetti, Giorgio Caviglia, Daniele Guido, Michele Graffieti, and Samuel Granados Lopez. New maps from the media-city. citymurmur as a tool for the visualization of urban space. In *Nouvelles cartographies, nouvelles villes. HyperUrbain.2*. Europia Production, 2010.

[119] Shlomo Yitzhaki. On an extension of the gini inequality index. *International economic review*, pages 617–628, 1983.

[120] Emanuele Della Valle, Irene Celino, Daniele Dell'Aglio, Ralph Grothmann, Florian Steinke, and Volker Tresp. Semantic traffic-aware routing using the larkc platform. *IEEE Internet Computing*, 15(6):15–23, 2011.

[121] Dieter Fensel, Frank van Harmelen, Bo Andersson, Paul Brennan, Hamish Cunningham, Emanuele Della Valle, Florian Fischer, Zhisheng Huang, Atanas Kiryakov, Tony Kyung-il Lee, Lael Schooler, Volker Tresp, Stefan Wesner, Michael J. Witbrock, and Ning Zhong. Towards larkc: A platform for web-scale reasoning. In *ICSC*, pages 524–529. IEEE Computer Society, 2008.

[122] Volker Tresp, Yi Huang, Markus Bundschus, and Achim Rettinger. Materializing and querying learned knowledge. *Proc. of IRMLeS*, 2009, 2009.

[123] Yi Huang, Volker Tresp, Markus Bundschus, Achim Rettinger, and Hans-Peter Kriegel. Multivariate prediction for learning on the semantic web. In *International Conference on Inductive Logic Programming*, pages 92–104. Springer, 2010.

[124] Kristin A Cook and James J Thomas. Illuminating the path: The research and development agenda for visual analytics. 2005.

[125] Jackie Moyes and Patrick W Jordan. Icon design and its effect on guessability, learnability, and experienced user performance. *People and computers*, (8):49–60, 1993.

[126] Robert Battle and Dave Kolas. Geosparql: enabling a geospatial semantic web. *Semantic Web Journal*, 3(4):355–370, 2011.

[127] Jerry R. Hobbs and Feng Pan. Time Ontology in OWL, September 2006.

[128] Yves Raimond and Samer Abdallah. The event ontology, 2007. http://motools.sf.net/event.

[129] Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. PROV-O: The PROV Ontology. Technical report, W3C, 2012.

[130] John G. Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. SIOC: an approach to connect web-based communities. *IJWBC*, 2(2):133–142, 2006.

[131] Mehdi Jazayeri, Rüdiger Loos, and David R. Musser, editors. *Generic Programming, International Seminar on Generic Programming, Dagstuhl Castle, Germany, April 27 - May 1, 1998, Selected Papers*, volume 1766 of *Lecture Notes in Computer Science*. Springer, 2000.

[132] Gautier Krings, Francesco Calabrese, Carlo Ratti, and Vincent D Blondel. Urban gravity: a model for intercity telecommunication flows. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(07):L07003, 2009.

[133] Charu C Aggarwal. Outlier analysis. In *Data mining*, pages 237–263. Springer, 2015.

[134] Alex Pentland. Social signal processing [exploratory dsp]. *IEEE Signal Processing Magazine*, 24(4):108–111, 2007.

[135] 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.

[136] Prajakta S Kalekar. Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, 4329008:1–13, 2004.

[137] Marco Brambilla, Stefano Ceri, Florian Daniel, and Gianmarco Donetti. Temporal analysis of social media response to live events: The milano fashion week. In Jordi Cabot, Roberto De Virgilio, and Riccardo Torlone, editors, *Web Engineering: 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*, pages 134–150, Cham, 2017. Springer International Publishing.

[138] Marco Brambilla, Stefano Ceri, Florian Daniel, and Gianmarco Donetti. Spatial analysis of social media response to live events: The case of the milano fashion week. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 1457–1462, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[139] Thomas Eiter, Josiane Xavier Parreira, and Patrik Schneider. Spatial ontology-mediated query answering over mobility streams. In *ESWC (1)*, volume 10249 of *Lecture Notes in Computer Science*, pages 219–237, 2017.

[140] Thomas Eiter, Thomas Krennwallner, Matthias Prandtstetter, Christian Rudloff, Patrik Schneider, and Markus Straub. Semantically enriched multi-modal routing. *Int. J. Intelligent Transportation Systems Research*, 14(1):20–35, 2016.

[141] Roman Kontchakov, Laura Pandolfo, Luca Pulina, Vladislav Ryzhikov, and Michael Zakharyaschev. Temporal and spatial OBDA with many-dimensional halpern-shoham logic. In *IJCAI*, pages 1160–1166. IJCAI/AAAI Press, 2016.

[142] Gulnar Mehdi, Sebastian Brandt, Mikhail Roshchin, and Thomas A. Runkler. Semantic framework for industrial analytics and diagnostics. In *IJCAI*, pages 4016–4017. IJCAI/AAAI Press, 2016.

[143] 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 Sem.*, 44:54–74, 2017.

[144] Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Theoretically optimal datalog rewritings for OWL 2 QL ontology-mediated queries. In *Description Logics*, volume 1577 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[145] Sebastian Brandt, Elem Güzel Kalayci, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Ontology-based data access with a horn fragment of metric temporal logic. In *AAAI*, pages 1070–1076. AAAI Press, 2017.