Reactive Knowledge Management

Stefano Ceri
Politecnico di Milano
Milano, Italy
stefano.ceri@polimi.it

Anna Bernasconi
Politecnico di Milano
Milano, Italy
anna.bernasconi@polimi.it

Alessia Gagliardi
Politecnico di Milano
Milano, Italy
alessia1.gagliardi@mail.polimi.it

***Abstract*—Today's large knowledge graphs are conceived mainly for supporting search and e-commerce within large companies such as Google or Amazon, with well-crafted knowledge creation rules. Our recent experience of the COVID-19 pandemic, when knowledge has grown at unprecedented rates and has been often contradictory, inspired us to capture a huge gap in existing concepts and technology: today's knowledge management does not adequately support such a disruptive process. In this article, we propose the design and prototyping of the next generation of knowledge management concepts and systems, which will support domain diversity and scientific evolution as foundational ingredients.**

Change management is based on a reactive approach, well-established in database systems, but so far lacking in knowledge systems. We propose the reactive interaction of several knowledge hubs, each developed within a scientific domain and "owner" of a portion of a common knowledge representation. Knowledge is represented as graphs, with nodes and edges; edges may interconnect nodes from different hubs. Most importantly, reactive rules cross the hub's borders and create the premises for a disciplined knowledge evolution, even under the pressure of crises. Similar challenges are not restricted to the recent pandemic and can address other crisis scenarios, including the catastrophic consequences of climate change or the recent (r)-evolution in artificial intelligence, studied by several scientific communities, whose management requires complex and controversial choices.

***Index Terms*—Knowledge Management, Knowledge Hubs, Reactive Processing**

I. Introduction

COVID-19 created an unprecedented challenge for the production of scientific knowledge. Several scientific communities have been called into action to understand and model the impact of COVID-19, resulting in a huge and swift production of amazing scientific results. While biotech experts, virologists, and clinicians were on the front line in fighting the pandemic, many other communities have been mobilized – to name a few, epidemiologists, economists, social scientists, and psychologists. Unexpected events – such as the rise of new viral variants, the governance of vaccination programs, and the use of non-pharmacological measures to circumvent the contagion – have reshaped anti-pandemic strategies throughout the last two years. Among the sources of scientific controversies, a striking example is the debate on "how, how long, how strictly" that put into action containment measures for limiting viral spreading (such as lockdowns or travel restrictions), considering an array of negative effects on the economy and society at large [1], [2].

Similar challenges, which were exacerbated by the pandemics, are indeed characterizing many aspects of societal evolution, where sudden events radically change a scenario of incremental evolution. Other highly impacting challenges are posed by climate change, similarly studied within different scientific communities, featuring an increase in frequency and effects of catastrophic events that need to be contrasted not only by short-term interventions but also by highly controversial long-term policies [3], [4]. Similarly, last year's evolution of AI technologies has the potential of introducing a new radical change in many aspects of our society [5], [6].

The rapid acquisition of new knowledge was not assisted by adequate technology for knowledge management. The current practice is not appropriate to react to changes; large knowledge graphs that empower the search engines of major companies – such as Google [7] and Amazon [8] – change incrementally, according to strict graph creation rules, enforced to guarantee a uniform and robust search. The community of the semantic web has produced technologies for linking semantic resources [9], but most efforts are focused on data integration, harmonization, and linking rather than dealing with controversial decision-making scenarios. Our vision has the objective of enhancing knowledge management for dealing with radical knowledge changes, by formalizing the concepts and designing the models, methods, and technology for a new class of reactive knowledge management systems.

Along this view, knowledge management is organized around the concept of *knowledge hubs*; each hub reflects the requirements of a given scientific community (which may internally host interdisciplinary contributions) and uses the most suitable models and methods for knowledge creation and management. However, knowledge hubs do not act in isolation; they communicate by means of declarative data abstractions that interconnect them and support reasoning at a global level. In our vision, selected portions of the available knowledge at each hub will contribute to the creation of knowledge to be integrated and shared with other hubs. More specifically, each knowledge hub will have control over a specific portion of a *partitioned knowledge graph*, by having full responsibility for the management (i.e., creation, update, or deletion) of the nodes assigned to that partition. While most edges will also be managed within the graph, a few well-crafted edges, designed according to agreed data linking protocols, define knowledge bridges by connecting nodes managed by different hubs. In this way, available knowledge

at each hub will be integrated, thereby generating a single (but partitioned) knowledge graph.

Our driving principle for managing knowledge change is to augment knowledge management with reactive components, which perceive local changes and produce – as a reaction – new global knowledge; they augment classic methods already implemented and used in database management systems, but still completely lacking in knowledge management practice. Today's knowledge managers are passive: they are not equipped to radically change their content based on occurring events. Surprisingly, the work on active databases [10], [11] has not yet influenced the development of knowledge managers. However, the addition of reactive components is perhaps even more relevant for knowledge managers, which are affected by major changes at a coarse granularity, rather than databases, where the so-called "database triggers" are enacted as a result of elementary changes.

In this vision, *reactive knowledge managers* support a highly expressive reactive rule language, enabling high-level reasoning; rules follow a classic event-condition-action paradigm, where events are raised at knowledge creation (i.e., creation or deletions of nodes and edges), conditions (renamed *guards*) check if potential hazards occur, and – if so – actions (renamed *alerts*) look for critical conditions and, if occurring, produce side effects on the knowledge (i.e., addition or deletion of nodes and edges). Ideally, a query language over such graphs should add to classic relational calculus also recursion, data aggregation, and statistical extensions (e.g., various expressions for defining statistical significance). In order to effectively reflect knowledge partitioning, guards should be evaluated within a single hub, whereas alerts could in the most general case involve several hubs, both in terms of required knowledge and side effects.

An important question is to evaluate which underlying technology is most suited in order to support this vision. Technological solutions provided by the semantic web community provide many ingredients towards our objectives, including expressive knowledge languages (e.g., RDFS/OWL) and powerful data linking methods. However, we are also witnessing important progress in graph databases, in terms of scalable solutions for big data management as well as semantic enrichment in the direction of query language standardization [12], [13] and richer support of classical relational concepts, such as schemas and keys, in the context of property graphs. In particular, we subscribe to this effort with our proposal of defining standard triggers for property graphs. Thus, along with our proposal, we also briefly examine how it could be supported by means of suitable extensions of graph databases.

This article is organized as follows: after the related work (Section II), we introduce reactive knowledge graphs (Section III), by first discussing its partitioning into knowledge hubs, then illustrating the reactive rules with their components (events, guards, alerts), presenting orthogonal rule classes, and finally discussing how logging can be supported in our framework, using the concept of alert nodes. We also present, in section (Section IV), a prototype implementation of these concepts on Neo4j, one of the most widely used graph database systems. In the discussion (Section V), we indicate future research topics and explain how reactive knowledge management can shape our responses to crisis scenarios.

Our proposal is general and suited to manage arbitrary crisis scenarios; to give substance to our vision, we consider, as a running example, a simple scenario inspired by the COVID-19 pandemic, and use it to exemplify each aspect of the proposal. In our COVID-19 related research, we developed the CoV2K model [14], a knowledge graph interconnecting large datasets of SARS-CoV-2 sequences (EpiCoV [15] and GenBank [16]) and epitopes (IEDB [17]) with knowledge entities representing variants, their effects (in terms of disease severity, transmissibility, vaccine escape, etc.), their composition (in terms of sets of mutations, which have specific positions along the structure of the viral genome/proteins), the peculiarities of mutations (due to their original and alternative nucleotide or amino acid residues), and the definition of particular regions of the viral genome, with given functions. A public API (http://gmql.eu/cov2k/api/) allows users to perform targeted searches that interconnect data and knowledge. In CoV2K, nodes are clustered within areas, representing topics of interest that are studied by different communities; thus, CoV2K is a partitioned knowledge graph, in agreement with the definition that will be given in Section III-A.

II. Related Work

Knowledge graphs are knowledge bases that use a graph-structured data model to integrate data. The term became popular around 2010 when knowledge graphs gained popularity both in academia and industry. In general, they are *directed labeled graphs* composed of nodes and edges, used for the organization and integration of information about known "entities". One of the most important initiatives is Wikidata's Knowledge Graph [18], a collaboratively edited open knowledge graph providing data, which includes information extracted from Wikipedia. As of 2023, Wikidata contains more than 100 million distinct entities (or objects) [19] and more than one billion relationships among them, extracted from different catalogs in various languages and published by independent data providers. Knowledge graphs are part of the search engines hosted by major companies, such as Google [7], Bing [20], and Amazon [8].

Graph databases are increasingly employed for representing the connections that exist in the real world [21]. Thanks to their expressive query languages and strong performance, they are steadily more used to store large knowledge graphs. Standards for graph databases are emerging, most importantly the Graph Query Language (GQL) [22]; GQL is strongly inspired by Cypher [23], the query language of Neo4j [24], which in turn is the most successful graph database engine to date [25].

The research community has proposed various approaches for enriching the semantics of graph databases, first by shaping them in the form of *Property Graphs* [26], and then by defining the notions of *PG-Keys* and *PG-Schema*. PG-Keys [27] are unique identifiers assigned to arbitrary subsets

of nodes and edges within a property graph database. A *PG-Schema* [28] adds to property graph databases a formalized schema, addressing the need for a standardized approach to schema management, enabling users to define and enforce data constraints, specify relationships, and establish a clear structure for their graph data.

In a recent manuscript, we proposed *PG-Triggers* [29]. Triggers exist since the birth of relational databases [10]; they have been studied in [11] and formalized in the ISO-ANSI SQL3 Standard [30]. So far, they have not been formalized by the graph database research community, although they can be informally supported by various graph database systems. Hence, we introduced PG-Triggers as a proposal for influencing future standard development and suggesting new directions to the evolution of graph databases.

III. Reactive Knowledge Graphs

A. Partitioned Knowledge Graphs

A Knowledge Graph is modeled as a **property graph**, whose elements are nodes and edges, which are directed links between nodes; we assume both nodes and edges to be labeled, and they can include valued properties. Although this is a minimum requirement, knowledge graphs could be further enriched with other semantic aspects, e.g., those discussed in PG-Schema [28] and PG-Keys [27].

In our approach, knowledge graphs are partitioned. More precisely, each node is assigned to a **knowledge hub**, which is responsible for creating and administering these nodes and the internal relationships (i.e., edges) connecting them. Selected relationships may connect nodes assigned to different graphs, thereby linking knowledge hubs and allowing exchange among different communities. Data linking has a long tradition, following Tim Berners-Lee's terminology [31] that has been later adopted by the Semantic Web Community [32], [33].

Running Example: COVID-19 and SARS-CoV-2. In this article, we consider a reactive knowledge graph for monitoring the spreading of a dangerous viral mutation in a geographical region. We identified four knowledge hubs, represented in Fig. 1:

- *Experimental hub* (E) that studies mutations' effects.
- *Analysis hub* (A) that performs viral genome sequencing within a given region and associates each human-collected sample with known viral variants.
- *Clinical hub* (C), located at a given hospital within a region.
- *Regional hub* (R), responsible for deciding the policies for a given region; it collects information from sequencing centers and hospitals in that region.

For a precise definition of the data graph, we use the abstractions from the PG-Schema proposal [28], taking advantage of its rich semantics. The adoption of PG-Schema makes graph databases more similar to relational databases, especially with a `STRICT` graph type definition, where nodes and relationships are uniquely identified by labels, in the same way as table names identify relational tables. The PG-Schema of our running example is shown in Fig. 2.

B. Reactive Rules

Reactive rules for knowledge graphs follow an event-condition-action paradigm, where events monitor **graph changes**, i.e., creation and deletion of nodes and edges or the setting and removal of labels and properties; the condition, also called **guard**, monitors the situation that is created after the knowledge change; if the situation requires further analysis, the action, called **alert**, checks for a critical condition; if it occurs, the alert produces side effects. A simple and minimally impacting side effect is to create a special node, called **alert node**, whose content summarizes the critical condition observed in the graph. Thus, a reactive rule is defined by the quadruple <*Event, Guard, Alert, Alert node*>. Next, we detail the role of each component, exploiting the Cypher[1] query language [23].

- The *Event* is any of the following pairs: creation/deletion of nodes/relationships, or setting/removal of labels/properties. Along the tradition of relational triggers, each specific node or relationship that is changed in the graph is denoted by the transition variable `NEW`. Events referring to a relationship should normally refer to an internal relationship, i.e., relationships interconnecting nodes managed by a given hub.
- The *Guard* is an existential predicate, written in Cypher, normally focused on information managed within a hub; when true, it reveals situations that deserve further investigation.
- The *Alert* is a Cypher query of arbitrary complexity that further analyzes the situation, to determine whether it is critical: when this happens, the Alert produces a new node that retrieves from the entire knowledge graph the information necessary to manage the critical situation.
- The *Alert node* is labeled `Alert`, it has three mandatory properties, the associated `rule`, the `hub` producing the rule, and the `creation time` of the alert; then, it has additional properties that depend upon the specific critical situation.

Each rule is created within a hub, responsible for specifying the guard and alert and for designing the information carried by the Alert node; the hub is assigned based on the specific domain knowledge mastered at that hub. The produced Alert nodes are shared by all hubs; the history of Alert nodes is made accessible by means of a suitable data structure (as described in Section III-D).

Note that graph database engines (e.g., Neo4j) provide a coarse granularity for events, as they capture just the insertion or deletion of nodes and edges, with their labels and properties. We instead force the guards to be relevant only to nodes and relationships with a given label – this is similar to the

[1]Cypher syntax has rounded brackets for (nodes) and arrows for -[:relationships]->. Queries represent graph patterns through the database data, e.g., (node)-[:is_connected_to]->(otherNodes). The keyword `MATCH` (like SELECT in SQL) searches for an existing pattern in the database; a match can be optional or conditioned to a `WHERE` predicate. Parts of a pattern/query can be referenced by means of variables, multiple query parts can be chained together using `WITH`.

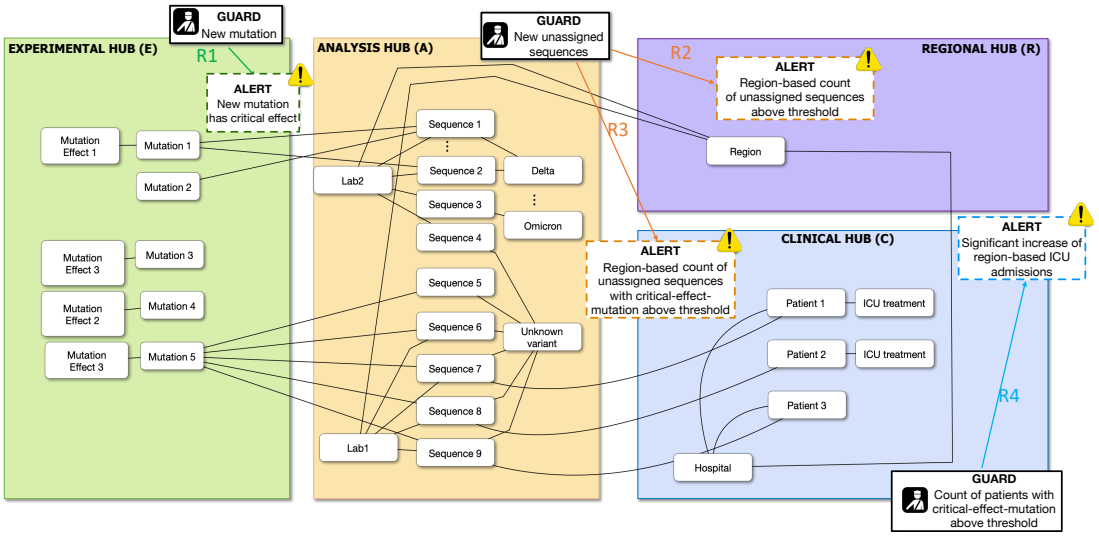

Fig. 1. Partitioned knowledge graphs for the running example; it includes the experimental (E), analysis (A), clinical (C), and regional (R) hubs. Nodes of the graph describe (from left to right) mutation effects, mutations, labs, sequences, variants, hospitals, regions, patients, and their treatments. Four rules R1–R4 are specified by pairing guards and alerts.

```
CREATE GRAPH TYPE CovidGraphType STRICT {
  (mutationType: Mutation {name STRING, protein STRING}),
  (criticalEffectType: CriticalEffect {description STRING}),
  (sequenceType: Sequence {accession-id STRING}),
  (variantType: Variant {name STRING}),
  (labType: Laboratory {name STRING}),
  (patientType: Patient {patient-id STRING, admissionDate DATE,
       name STRING, sex CHAR}),
  (icuPatientType: IcuPatient {admittedToIcu BOOL}),
  (hospitalType: Hospital {name STRING}),
  (regionType: Region {name STRING}),
  (:mutationType)-[riskType: risk]->(:criticalEffectType),
  (:mutationType)-[foundInType: foundIn]->(:sequenceType),
  (:sequenceType)-[belongsToType: belongsTo]->(:variantType),
  (:sequenceType)-[sequencedAtType: sequencedAt]->(:labType),
  (:patientType)-[hasSampleType: hasSample]->(:sequenceType),
  (:labType)-[locatedInType: locatedIn]->(:regionType),
  (:patientType)-[treatedAtType: treatedAt]->(:hospitalType),
  (:patientType)-[treatedWithType: treatedWith]->(: icuPatientType),
  (:hospitalType)-[locatedInType: locatedIn]->(:regionType)
  // Constraints
  FOR (x:sequenceType)
  EXCLUSIVE MANDATORY SINGLETON x.accession-id,
  FOR (x:patientType)
  EXCLUSIVE MANDATORY SINGLETON x.patient-id }
```

Fig. 2. PG-Schema specification for the running example.

triggers targeting tables, a mechanism that occurs in relational databases. Note also that, by limiting the alert action to creating a new Alert node, in this paper, we do not further discuss the real-world reaction that will deal with the critical situation. Finally, note that alerts with arbitrary side effects can cause the cascade of node and relationship changes, thereby introducing problems such as the termination and confluence of reactive computations [11], [34].

Running Example: Reactive Rules. Let us consider rule R2 of Fig. 1, restated as follows: *when a new unassigned sequence is created, if the number of unassigned sequences for a geographical region goes above a critical threshold, then define an alert with suitable information*. Fig. 3 shows the quadruple <Event, Guard, Alert, Alert node> both in visual form (A) and as Cypher code (B). Specifically: (1) the triggering event is the creation of a new node of type `Sequence`; (2) hence the guard includes a check on the corresponding label and a condition that the `NEW` sequence is unassigned; (3) the alert is a complex query counting the unassigned sequences in any laboratory of the region and returning a true value if such count is above a given threshold; (4) the new Alert node includes the rule acronym, the hub, the creation time, and the value of the counter.

C. Rule Classification

We distinguish two orthogonal classes of reactive rules; based on their **scope**, rules are:

- *intra-hub*, when their scope (nodes and relationships affected by guards and alerts) is within a single hub;
- *inter-hub*, when their scope is arbitrary.

Considering instead the possibility of consulting **state information**, rules are:

- *single-state*, when their Alert part refers just to the current state of the knowledge graph;
- *multi-state*, when their Alert part requires comparing several states of the knowledge graph.

Running Example: hub-specific reactive rules. Fig. 1 illustrates four reactive rules; all of them monitor events of type "*new node created*". We next present them informally; the full account of the four rules, expressed in Neo4j APOC-triggers (see Section IV), is in the GitHub repository [35].

The **Experimental Hub** presents a single-state and intra-hub rule R1. The guard checks for a mutation creation, and

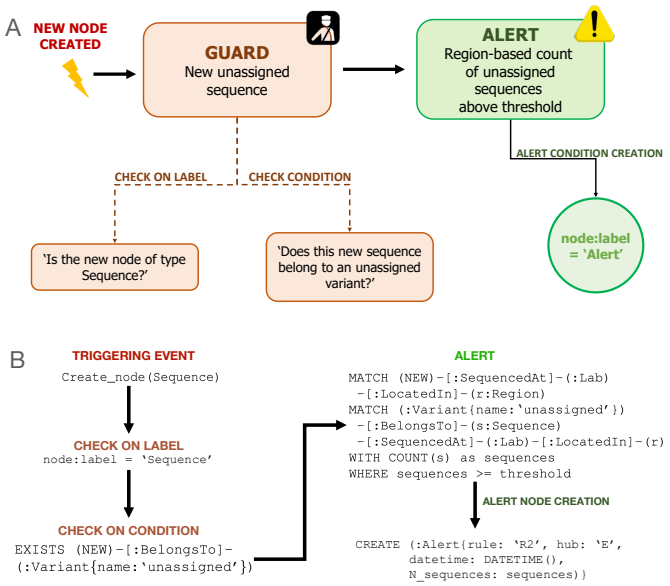

Fig. 3. Schematic representation of rule R2 (A) and Cypher code (B) for each element of the scheme.

the Alert then checks if the new node has a connection with a node of type Effect and critical property.

The **Analysis Hub** presents two reactive rules R2 and R3; they share the Guard, which looks for the existence of new sequence nodes whose variant has not been defined yet (i.e., unassigned sequences). Both rules refer to the labs within a given region; they are inter-hub (as they need to access the region node in the region hub; the second one needs also to access the experimental hub) and single-state, as their predicates use fixed thresholds.

Finally, the **Clinical Hub** has a single rule R4 whose Guard counts the number of patients associated with a mutation with a critical effect within a region and returns a true value when the counter exceeds a threshold. The Alert compares the number of patients in the ICU in the current and past states and produces an Alert node if such comparison indicates a significant increase; thus, the rule is inter-hub, multi-state.

D. Alert nodes logging and its exploitation

As mentioned in Section III-C, rules may compare multiple states of the knowledge in the graph. While this problem is well-managed by active databases for the relational model by means of the `NEW` and `OLD` transition variables, respectively denoting the state transition associated with the transaction that causes the trigger activation, in a graph database the concept of OLD state of the graph is missing, and more in general we do not want to exploit the concept of transaction to define knowledge rules - as knowledge graphs may be contributed by processes which are not under centralized transactional control. Thus, we need to think of an out-of-the-box solution to tackle this problem.

Given that reactive knowledge bases can apply to a variety of scenarios, we opted for considering application-specific time intervals, whose length may vary (e.g., from minutes to hours, days, weeks). Accordingly, a **period of observation** refers to a specific time interval during which alert information is recorded and stored; the comparison of alert information over various periods allows for identifying trends and patterns over time.

```
CREATE GRAPH TYPE EssentialSummary STRICT {
 (summaryType: Summary {date DATE}),
 (alertType: Alert {rule STRING, hub STRING, dateTime DATETIME, OPEN }),
 (currentType: summaryType & Current),
 (:summaryType)-[nextType: next]->(:summaryType),
 (:summaryType)-[hasType: has]->(:alertType)
 // Constraints
 FOR (x:summaryType)
 EXCLUSIVE MANDATORY SINGLETON x.date,
 FOR (x:alertType)
 EXCLUSIVE MANDATORY SINGLETON x.dateTime }
```

Fig. 4. PG-Schema specification for the Essential Summary

To manage this information, we designed an auxiliary graph data structure, called **Essential Summary**, whose function is to cluster Alert nodes relative to the same period. In this model (shown in Fig. 4), each time interval is associated with a `Summary node`, progressively labeled and carrying a specific time point as a property. The most recently created summary node is denoted as *Current node*; when new Alert nodes are created (with the usual properties and an `OPEN` schema to allow for additional ones), they are linked to the current node. At the end of the current interval, a new summary node is created, labeled with its specific time, representing the new current node. As shown in Fig. 5, the Summary nodes (including the current one) are chained by a `next` relationship, which allows traversing the Essential Summary data structure from the oldest summary up to the current node.

As a whole, the knowledge base consists of the entire knowledge graph, which is subject to arbitrary changes in its content, and the Essential Summary structure, containing all the nodes of type alert created along the knowledge base history. Rules may refer to arbitrary parts of the knowledge base and Essential Summary. We argue that this model, once coupled with careful modeling of the information contained in the Alert nodes, elegantly supports concepts that recall the old and new states of active databases, although with a periodic nature set by the need of the knowledge management scenario. Indeed, this model can be much more powerful, as properties associated with Alert nodes are arbitrarily complex.

Running Example: Alert node logging. Recall that rule R4 of the Clinical Hub requires comparing the number of ICU patients between two subsequent states (periods); we here discuss an alternative implementation of R4, that we denote as R4′; Assume that an auxiliary rule R5, activated by each new patient admitted or discharged from ICU treatment (not shown in Fig. 1), computes their daily count over a given region; hence, at each period (of 24 hrs), summary nodes are linked to Alert nodes, written by rule R5, with properties `Region` and `IcuPatients`. Let us focus on the `'Lombardy'` region.

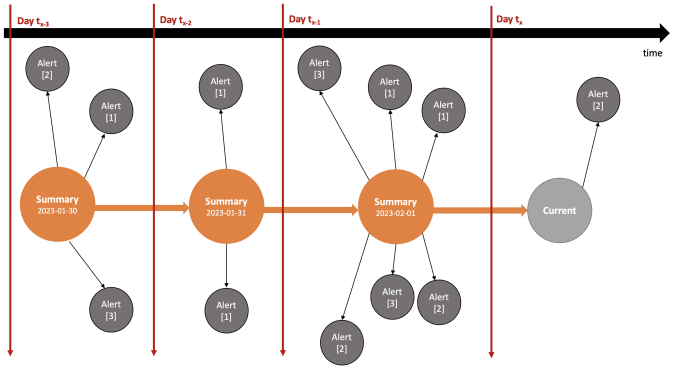

Fig. 5. Schematic representation of the Essential Summary data structure, showing four periods, each one day long.

Then, the daily difference (either positive or negative) of ICU patients can be computed by summing the current value of ICU patients in Lombardy and comparing it with the counter of ICU patients in Lombardy of the previous day, available in the Essential Summary. Then, in the Alert of rule R4′, a first query counts today's number of ICU patients in Lombardy and a second query reads the counter of patients in Lombardy from yesterday's summary. The Alert code for R4′ is then:

```
MATCH (:Region{name:'Lombardy'})
        -[:LocatedIn]-(:Hospital)
        -[:TreatedAt]-(:Patient)
        -[:TreatedWith]-(i:IcuPatient)
WITH count(i) as TodayIcuLomb
MATCH (a:Alert{rule:'R5',Region:'Lombardy'})
        -[:has]-(:Summary)-[:Next]-(:Current)
WITH a.IcuPatients as YesterdayIcuLomb,
        TodayIcuLomb
WHERE toFloat(TodayIcuLomb-YesterdayIcuLomb)/
        toFloat(TodayIcuLomb) > 0.1
```

Finally, rule R4′ creates the Alert node:

```
CREATE (:Alert{rule: 'R4', hub: 'C',
    datetime: DATETIME(), Region: 'Lombardy',
    description: 'Significant Increase
    of ICU patients'})
```

The availability of daily counts of IcuPatients along the Essential Summary structures makes it possible to compute more interesting statistics, such as mobile averages (e.g., over the last seven days) so as to monitor critical situations more carefully, by smoothing peaks.

IV. Mapping Reactive Knowledge Rules to Neo4j

In this section, we consider how reactive rules can be deployed on the Neo4j prototype. We concentrate upon the concepts introduced in Section III, by dealing with knowledge partitioning, rule translation, and alert logging.

A. Partitioned Knowledge Graphs

As knowledge partitioning into distinct hubs reflects the presence of several diverse knowledge domains, we expect that knowledge within each hub will be produced autonomously, as a result of the models and methods of a well-identified community, that is typically of a scientific nature but may also include regulatory committees, e.g., from geographic regions. Each community manages data in arbitrary formats, but we expect that a data subset will be eventually produced in graph format, for supporting knowledge exchange and sharing, yielding to the partitioned knowledge graph presented in Fig. 1.

A partitioned graph database can be supported by many underlying architectural choices. Each partition could be managed by a different database engine, in the context of a federated system; in such cases, links among partitions can be supported along classic methods discussed in the semantic web and linked data community [9]. Alternatively, the entire graph could be supported within a single, scalable database engine, possibly deployed on the cloud. For what concerns data modeling, we simply expect that each node of the graph will include, as a mandatory property, the identity of the hub where the corresponding information is owned. Then, any Cypher query over such a graph can be interpreted as a federated database query, and executed in a way that depends upon the underlying architecture.

B. Rule Translation

Reactive rules in databases are typically implemented using event-driven triggers; they are not directly supported in Cypher, thus we consider APOC (Awesome Procedures on Cypher), a community-contributed library for augmenting the Cypher query language supported by Neo4j[2]. The library includes over 450 procedures, providing functionalities for utilities, conversions, graph updates, data import, data transformation, and manipulation. We here provide a general scheme for the syntax-directed translation from knowledge rules into Neo4j APOC triggers, by considering a rule activated on node creation; along [29], the translation can be easily generalized to all rule events.

We use the `apoc.trigger` procedures, which handle events and their processing. By means of these procedures, triggers can be created, deleted, paused, and resumed; the creation of a trigger uses the following syntax (excluding parameter `config`, not relevant for the trigger translation):

```
apoc.trigger.install(dbName, triggerName, statement,
                                        selector)
```

We next describe the syntax-directed translation of reactive rules into APOC triggers, described in Fig. 6. The APOC `install` procedure has four parameters: the `dbName`, the `triggerName`, the `statement` (surrounded by a red rectangle) and the `selector` (i.e., action time). The richest parameter is the `statement`, which is constructed as a sequence of three sub-statements.

- The first one is a call to the `UNWIND` clause, returning each node created by the rule into the `cNode` variable.
- The second one is used to translate the Condition part of the rule, using the `cNode` variable.

[2]We refer to Version 5.10, available as of August 2023.

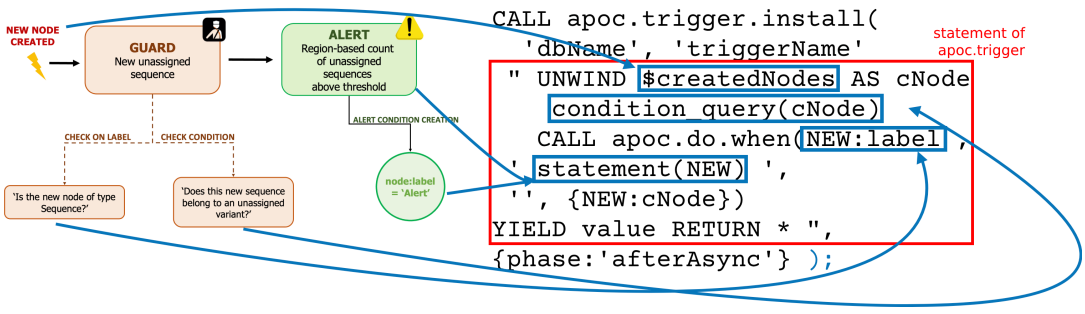

Fig. 6. Syntax-directed translation from reactive rules to Neo4j APOC triggers.

```
CALL apoc.trigger.install('dbName','triggerName',
"UNWIND $createdNodes AS cNode
 MATCH (cNode)-[:BelongsTo]
   -(:Variant{name:'unassigned'})
 CALL apoc.do.when(
   NEW:Sequence,
   'MATCH (NEW)-[:SequencedAt]-(:Lab)
        -[:LocatedIn]-(r:Region)
   MATCH (:Variant{name:'unassigned'})
        -[:BelongsTo]-(s:Sequence)
        -[:SequencedAt]-(:Lab)-[:LocatedIn]-(r)
   WITH COUNT(s) as Nsequences
   WHERE Nsequences >= threshold
   CREATE (:Alert{rule: 'R2', hub: 'E',
    datetime:DATETIME(), N_sequence: Nsequences})',
   '', {cNode: NEW})
 YIELD value RETURN *",
{phase: 'afterAsync'});
```

Fig. 7. Translation of the rule presented in Fig. 3 into a Neo4j APOC trigger, generated along the syntax-directed translation of Fig. 6.

- The third one uses the APOC `do.when` procedure for translating the Action part of the rule, and possibly creating the Alert node.

The `do.when` procedure, in turn, has four parameters: the condition, the action if the condition *is* met, the action if the condition *is not* met, and the operands that can be used in the condition and action. In particular, the fourth parameter assigns the `cNode` value to the `NEW` variable, which can be used in the `do.when` condition and action.

The `do.when` condition checks that the `NEW` variable is upon a node having the correct label. The first `do.when` action is executed when the condition is true; it uses the Alert condition (taken from the left side) and is completed by the Alert node creation (also taken from the left side), which is performed when the Alert condition reveals a critical situation. Finally, the trigger is completed by fixed code (`YIELD value RETURN *`), which has no side effects.

Let us consider the rule presented in Fig. 3; its translation into a Neo4j APOC trigger, generated along the syntax-directed translation scheme of Fig. 6, is shown in Fig. 7.

C. Management of Essential Summary

Essential summaries must be periodically managed, by tracking each change of period. In Neo4j, this processing

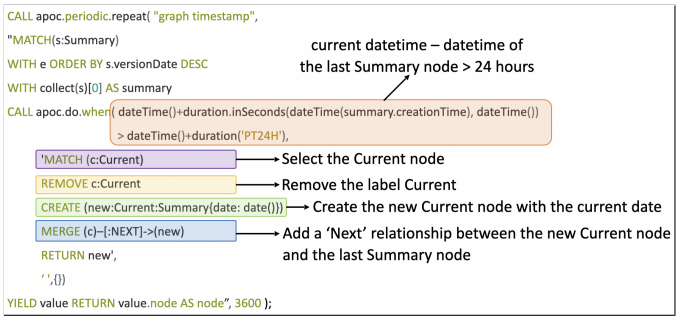

Fig. 8. Creation of the Essential Summary. The APOC-based periodic repeat query activates at every hour. The first query checks that more than 24 hours have elapsed since the last creation of a new summary node. The second query is activated only when a new Current node must be created and appended to the chain of summary nodes.

can be performed by using the `apoc.periodic` procedure, capable of periodically executing a statement, with the syntax:

```
apoc.periodic.repeat('name',statement,
                     repeat-rate-in-seconds)
```

Parameters refer to the statement name, the Cypher statement to be executed, and the period of execution; another parameter `config` is not relevant for the periodic execution. The method is illustrated in Fig. 8. The latest summary (LS) is initially retrieved, and then a test is performed about the time elapsed between the LS daytime and the current time; when it exceeds 24 hours, the Essential Summary structure is extended by adding a new summary node, labeling it as the current node and linking it to LS. Note that this solution requires a centralized management process, whereas all other aspects of reactive knowledge management can be distributed and parallelized.

D. Experiments over large graphs

We considered the development of an alerting system based upon reactive rules, signaling those regions where severe cases are significantly increasing; in particular, a critical situation occurs when admissions grow by 10% during two consecutive days. With a naive design, this condition is recognized by a rule whose guard is simply the creation of a new patient; the alert is a comparison of counters of patients, grouped by region, which extracts critical regions and creates Alert nodes reporting the Region's name and the number of patients in the

two consecutive days. The execution time of the trigger for an increasing number of patients is shown in Fig. 9, where both execution time and patient number are in logarithmic scale. The system scales linearly; the execution of 1M trigger instances takes about one thousand seconds (i.e., 16.6 minutes). We executed experiments on a one-node server powered by an Intel Xeon CPU E5-2660 with 56 cores and 378GB of RAM; here, Neo4j runs on a dedicated docker image with a maximum allowed RAM of 200 GB and a cache of 80 GB.

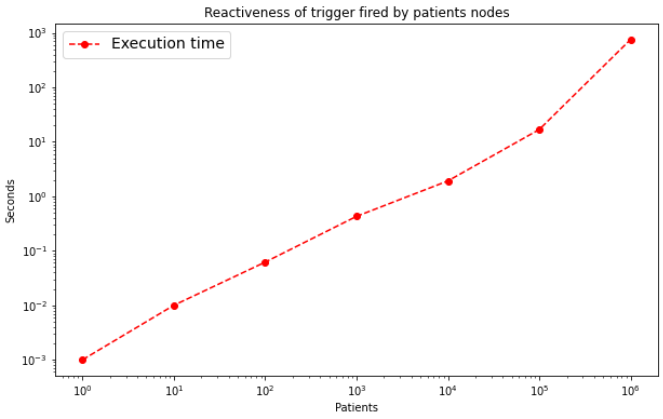

Fig. 9. Execution time for triggers enacted at each new patient

We then considered how to improve rule execution times while keeping the same semantics, and observed that the computation of summary information by regions is repeated for each patient, while in principle it should be performed just once for each region; we then designed a second version of the rules, which uses essential summaries. We added to the patient creation script a new operation, linking each daily summary node to regional statistics, storing for each region the counter of patients. Then, we designed a new rule triggered by the creation of regional daily statistics, that monitors the creation of summary nodes and compares, for each region, the counters in the current day's summaries with the counters in the previous day's summary, found by accessing the `current` node and traversing the Essential Summary data structure. The second rule design performs globally much better, as shown in Fig. 10; we show the summary computation time and the execution time of triggers, both in the order of seconds, while patients increase are still in logarithmic scale. We note that the former execution time grows with the number of patients, and the latter is essentially stable, as it mostly depends on the number of regions, which is fixed.

V. Discussion

We explored the design of knowledge management systems capable of coping with crisis scenarios, when knowledge is rapidly produced within different scientific communities; we demonstrated these concepts at work in a prototype, inspired by our recent research on COVID-19, and we also showed that graph databases can adequately support these concepts, once deployed on big data graphs.

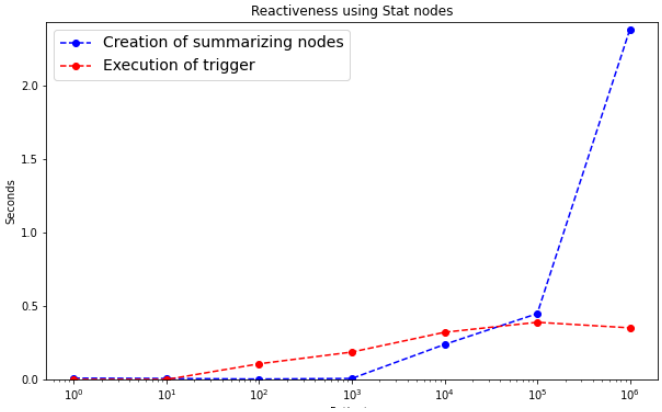

Fig. 10. Execution time for summary computation and triggers enacted at each new summary creation

In our future work, we intend to push these concepts further, by turning them into the design of a solid architecture for reactive knowledge management, with well-identified architectural components, also considering that each knowledge hub will be deployed by distinct scientific organizations or regulatory bodies using its own computing infrastructure; we will therefore deal with supporting knowledge management in a distributed/federated setting, both from a computational and organizational point of view.

Rule design is a complex and intriguing field of research, briefly touched on in our experimental section, where we showed that data summarization in rule design may lead to significant global savings; indeed, several problems open up, from a careful design of Alert nodes describing critical situations to adding full reactive capacity to rules by extending rules beyond the creation of Alert nodes. Supporting reactive processing directly within rules can be very effective, as reactions to critical conditions may be programmed directly within knowledge systems, but may cause rule cascading and therefore introduce other rule design problems, including guaranteeing their termination.

We would also like to experiment with how reactive processing can support what-if scenarios, by designing different cases of rule reactions and then showing their effects, by letting the knowledge management system evolve in different ways. This of course requires deeper changes to the infrastructure and a tight integration of reactive processing with hypothetical reasoning.

In summary, this paper opens up a number of interesting applications and research scenarios, that we intend to follow up on in the future.

Resources

A prototype Neo4j graph database, with scripts for data creation and population and APOC triggers for the four reactive rules presented in Section III.C, is provided on a GitHub repository [35].

ACKNOWLEDGMENT

The authors would like to thank Andrea Colombo and Francesco Invernici for their support in conducting the experiments over large graphs. This paper is supported by PNRR-PE-AI FAIR project funded by the NextGeneration EU program.

REFERENCES

[1] S. Chang, E. Pierson, P. W. Koh, J. Gerardin, B. Redbird, D. Grusky, and J. Leskovec, "Mobility network models of covid-19 explain inequities and inform reopening," *Nature*, vol. 589, no. 7840, pp. 82–87, 2021.

[2] G. Bonaccorsi, F. Pierri, F. Scotti, A. Flori, F. Manaresi, S. Ceri, and F. Pammolli, "Socioeconomic differences and persistent segregation of italian territories during covid-19 pandemic," *Scientific reports*, vol. 11, no. 1, p. 21174, 2021.

[3] P. Gazzotti, J. Emmerling, G. Marangoni, A. Castelletti, K.-I. v. d. Wijst, A. Hof, and M. Tavoni, "Persistent inequality in economically optimal climate policies," *Nature Communications*, vol. 12, p. 3421, 2021.

[4] A. Carlino, M. Tavoni, and A. Castelletti, "Self-adaptive multi-objective climate policies align mitigation and adaptation strategies," *Earth's Future*, vol. 10, no. 10, p. e2022EF002767, 2022.

[5] Think Tank European Parliament, "Artificial intelligence act," 2023, last accessed: Nov. 24th, 2023. [Online]. Available: https://www.europarl.europa.eu/thinktank/en/document/EPRS_BRI(2021)698792

[6] The White House, "FACT SHEET: President Biden Issues Executive Order on Safe, Secure, and Trustworthy Artificial Intelligence," 2023, last accessed: Nov. 24th, 2023. [Online]. Available: https://www.whitehouse.gov/briefing-room/statements-releases/2023/10/30/fact-sheet-president-biden-issues-executive-order-on-safe-secure-and-trustworthy-artificial-intelligence/

[7] A. Uyar and F. M. Aliyu, "Evaluating search features of Google Knowledge Graph and Bing Satori: entity types, list searches and query interfaces," *Online Information Review*, vol. 39, no. 2, pp. 197–213, 2015.

[8] X. L. Dong, "Challenges and innovations in building a product knowledge graph," in *Proceedings of the 24th ACM SIGKDD International conference on knowledge discovery & data mining*, 2018, pp. 2869–2869.

[9] T. Heath and C. Bizer, *Linked data: Evolving the web into a global data space*. Springer Nature, 2011.

[10] K. P. Eswaran, "Aspects of a trigger subsystem in an integrated database system," in *Proceedings of the 2nd International Conference on Software Engineering*, ser. ICSE '76. Washington, DC, USA: IEEE Computer Society Press, 1976, p. 243–250.

[11] J. Widom and S. Ceri, Eds., *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.

[12] "ISO/IEC 39075. Information technology - Database languages - GQL. Standard. International Organization for Standardization," 2023, geneva, CH.

[13] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels *et al.*, "Graph pattern matching in gql and sql/pgq," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 2246–2258.

[14] T. Alfonsi, R. Al Khalaf, S. Ceri, and A. Bernasconi, "CoV2K model, a comprehensive representation of SARS-CoV-2 knowledge and data interplay," *Scientific Data*, vol. 9, p. 260, 2022.

[15] Y. Shu and J. McCauley, "GISAID: Global initiative on sharing all influenza data–from vision to reality," *Eurosurveillance*, vol. 22, no. 13, 2017.

[16] E. W. Sayers, M. Cavanaugh, K. Clark, K. D. Pruitt, S. T. Sherry, L. Yankie, and I. Karsch-Mizrachi, "GenBank 2023 update," *Nucleic acids research*, vol. 51, no. D1, pp. D141–D144, 2023.

[17] R. Vita, S. Mahajan, J. A. Overton, S. K. Dhanda, S. Martini, J. R. Cantrell, D. K. Wheeler, A. Sette, and B. Peters, "The immune epitope database (iedb): 2018 update," *Nucleic acids research*, vol. 47, no. D1, pp. D339–D343, 2019.

[18] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[19] Wikidata Team, "Wikidata:Statistics," 2023, last accessed: Nov. 24th, 2023. [Online]. Available: https://www.wikidata.org/wiki/Wikidata:Statistics

[20] D. Wilkinson and M. Thelwall, "Search markets and search results: The case of Bing," *Library & Information Science Research*, vol. 35, no. 4, pp. 318–325, 2013.

[21] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz *et al.*, "The future is big graphs: a community view on graph processing systems," *Communications of the ACM*, vol. 64, no. 9, pp. 62–71, 2021.

[22] A. Green, P. Furniss, T. Lindaaker, P. Selmer, H. Voigt, and S. Plantikow, "Iso, tech. rep: Gql scope and features," https://s3.amazonaws.com/artifacts.opencypher.org/website/materials/sql-pg-2018-0046r3-GQL-Scope-and-Features.pdf, 2019, last accessed online: Nov. 24th, 2023.

[23] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 international conference on management of data*, 2018, pp. 1433–1445.

[24] Neo4j, "Neo4j," https://neo4j.com/, 2023, last accessed online: Nov. 24th, 2023.

[25] solid IT consulting, "Db-engines ranking of graph dbms," https://db-engines.com/en/ranking/graph+dbms, 2023.

[26] A. Bonifati, G. Fletcher, H. Voigt, N. Yakovets, and H. V. Jagadish, *Querying Graphs*. Morgan & Claypool Publishers, 2018.

[27] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K. W. Hare, J. Hidders, V. E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savković, M. Schmidt, J. Sequeda, S. Staworko, and D. Tomaszuk, "PG-Keys: Keys for Property Graphs," in *Proceedings of the 2021 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2423–2436.

[28] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, A. Green, J. Hidders, B. Li, L. Libkin, V. Marsault, W. Martens, F. Murlak, S. Plantikow, O. Savković *et al.*, "PG-Schema: Schemas for Property Graphs," in *Proceedings of the 2023 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2023.

[29] A. Gagliardi, A. Bernasconi, D. Martinenghi, and S. Ceri, "PG-Triggers: Triggers for Property Graphs," *arXiv preprint arXiv:2307.07354*, 2023.

[30] J. Melton and A. R. Simon, *SQL: 1999: understanding relational language components*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[31] T. Berners-Lee, "Linked Data," 2006, last accessed: Nov. 24th, 2023. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[32] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, "Linked data on the web (ldow2008)," in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 1265–1266.

[33] J. Šimko, M. Tvarožek, and M. Bieliková, "Semantics discovery via human computation games," in *Semantic Web: Ontology and Knowledge Base Enabled Tools, Services, and Applications*. IGI Global, 2013, pp. 286–308.

[34] E. Baralis, S. Ceri, and J. Widom, "Better termination analysis for active databases," in *Rules in Database Systems: Proceedings of the 1st International Workshop on Rules in Database Systems, Edinburgh, Scotland, 30 August–1 September 1993*. Springer, 1994, pp. 163–179.

[35] A. Gagliardi, "Reactive Knowledge Management," https://github.com/Alessia-G/Reactive_Knowledge_Management, 2023, last accessed online: Nov. 24th, 2023.