Designing NORIA: a Knowledge Graph-based Platform for Anomaly Detection and Incident Management in ICT Systems

Lionel Tailhardat[1,2,*], Yoan Chabot[1] and Raphaël Troncy[2]

[1]Orange, France
[2]EURECOM, Sophia-Antipolis, France

Abstract
To monitor complex systems, such as telecommunication and computer networks, interconnecting heterogeneous data with shared definitions is necessary for efficient interpretation of events and incidents. Semantic Web technologies are essential in this context, as they address the problems of data heterogeneity, knowledge sharing and logical/probabilistic reasoning. Well-established Network Monitoring Systems (NMSs) and Security Information and Event Management systems (SIEMs) do not explicitly use Semantic Web knowledge representation, however. To fill this gap, we propose an end-to-end data processing architecture that combines NMSs/SIEMs design patterns with Semantic Web tools. The platform features batch/stream processing, declarative data mapping with RML, data patching & reconciliation with SPARQL queries and SKOS, provenance auditability with centralized configuration and data management, and semantic data transfer with Kafka. The proposed architecture has been instantiated and tested in an industrial setting, producing an RDF knowledge graph that shows strong potential for addressing cross-domain anomalies from heterogeneous data.

Keywords
Knowledge Graph Construction, Incident Management, ICT systems, Extract Transform Load, RDF Stream Processor, Semantic Service Bus

1. Introduction

Incident management for broad scale Information and Communications Technology (ICT) systems implies scrutinizing massive amount of heterogeneous data for proper definition of remediation strategies. In the best case, crisp reasoning over situations at hand brings fast root cause analysis and high level of confidence for selecting the corrective maintenance actions to carry out. Anomaly detection within decision support systems, such as Network Monitoring Systems (NMSs) and Security Information and Event Management (SIEM) tools, typically rely on expert knowledge translated into logical rules for catching specific situations based on the systems activity traces. However, uncertainty arises whenever the ICT system's activity shows unexpected values or behaviors poorly fitting known activity models. A typical solution would

KGCW'23: 4th International Workshop on Knowledge Graph Construction, May 28, 2023, Crete, GRE

[*]Corresponding author.

lionel.tailhardat@orange.com (L. Tailhardat); yoan.chabot@orange.com (Y. Chabot); raphael.troncy@eurecom.fr (R. Troncy)
0000-0001-5887-899X (L. Tailhardat); 0000-0001-5639-1504 (Y. Chabot); 0000-0003-0457-1436 (R. Troncy)

be to fine-tune the decision support system stack, for example by extending the detection rule set with the new values, or retraining the anomaly detection model. This unfortunately brings computational complexity and overfitting to the diagnosis stage.

A better solution is to keep rules and models consistent by working on semantically equal data. The notions of "ontology" and "data model" solve the challenge of reasoning upon composite alerting signals at the semantic level. Indeed RDF Knowledge Graphs [1] bring an abstraction level for standard interpretation and logical reasoning over heterogeneous data.

Leveraging on Semantic Web tools could bring decision support systems for ICT systems to a next level of diagnosis and recommendation capabilities. However, few feedback exist about the design of such data processing platform from an end-to-end perspective. This paper reports on the design experiments for a knowledge graph-based data platform made at Orange, an international telecommunication infrastructure and service provider. The NORIA platform is part of an ongoing research effort for improved resilience of complex networks. It comprises the following key features: *1)* building a RDF Knowledge Graph from static (IT resources lists, organization) & streamed (trouble tickets, logs) data; *2)* providing data & inferences provenance and confidence indicators; *3)* enabling inline & posterior entity patching and reconciliation; *4)* enabling multi-level & synergical reasoning. Our main contributions with this paper are: setting design methodology and principles for an end-to-end knowledge graph-based data platform, providing Extract Transform & Load (ETL) architecture details and code for handling descriptive datasets and events streams, and sharing lessons learned while building the platform about data mapping strategies and configuration deployment.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 explores design challenges and requirements through a tool chain model proposal. Section 4 details our knowledge graph-based data platform architecture. References to contributed open source code are provided there. Section 5 evaluates the platform features and discusses lessons learned. Finally, Section 6 concludes the paper and discusses some future work.

2. Related Work

Designing a data processing architecture for Incident Management of ICT systems involves various research and technical domains, such as: data transformation and wrangling, computing and service architecture, decision making and business process management. In this section, we review related work from the aspect of current NMSs & SIEMs architectures, and Semantic Web data management solutions.

NMSs [2, 3] and SIEMs [4, 5] are two different product lines due to the nature of the data processed and the expectations regarding the incident management processes in which they are involved (e.g. ISO/IEC 2000[1], NIST SP 800-61[2]). For telecommunication networks, alarms (i.e. a durable or non fugitive fault that happens on an atomic function, as per ITU-T G.7710/Y.1701[3]) are first class citizens that should be reported to a Management and Control System. For cybersecurity, technical logs need to be combined with vulnerabilities and threat intelligence

[1] https://www.iso.org/standard/70636.html
[2] http://dx.doi.org/10.6028/NIST.SP.800-61r2
[3] https://www.itu.int/rec/T-REC-G.7710/

in a Log Collection → Log Normalization → Notifications and Alerts → Security Incident Detection component chain [6] for threat response management.

In both contexts, data processing architectures generally follow the producer/consumer design pattern[4] (a.k.a. observer pattern) and inspirations from distributed computing (i.e. hubs + aggregator). This is done in particularly to enable handling data sources with varied characteristics according to the monitored system uniformity and requirements (e.g. a High-Performance Computing platform with a data transfer and processing service offer *vs* an Internet Service Provider with communication service and PaaS – Platform as a Service – offers). Data persistence/dynamics characteristics and usage objective are driving forces behind the local computing performance through the choice of a best storage technology (e.g. Input/Output performance, storage footprint): *1) daemons and web applications* (e.g. dedicated filesystem for raw logs, binaries and libraries); *2) events and node information* (e.g. PostgreSQL[5] for structured notifications and characteristics); *3) performance data* (e.g. RRD[6] for throughput or CPU usage time series). Other lines of research exist for improving system performance: distributed event management and multi-agent architectures; data sketching; stream processing and anomaly detection subject to resource constraints.

Knowledge Graphs [1] bring an abstraction level for standard interpretation and logical reasoning over heterogeneous data. Graph data structures are sometimes used directly internally in NMSs and SIEMs for cases such as visualizing network topologies and data flows, representing and implementing failure mechanism models for root cause analysis (e.g. with fault tree analysis), or enabling knowledge management and data exploration with a traversal approach. Knowledge representation with Semantic Web technologies is not in use explicitly in well-established tools, however. Despite this, various research projects offer solutions of different complexity and maturity to support the Knowledge Graph Construction (KGC) process and make Incident Management more effective.

In end-to-end frameworks [7, 8, 9, 10], the KGC step is never considered singular, initial or terminal, but rather is the subject of multiple instances of a similar tool/principle within processing flows depending on the application field. In addition, this step is always placed between heterogeneous non-RDF data and a knowledge graph working sometimes as a main data storage, and sometimes as a support for third-party inference processes.

Generic tools devoted to KGC focus on stream processing and reasoning, for example: RML-Streamer [11] applies declarative mapping on the fly to structured data streams (e.g. file, Kafka topic) with RDF Mapping Language (RML) [12] rules; StreamingMASSIF [13] uses basic string substitution for mapping, and allows for real-time reasoning (e.g. SPARQL query processing, Complex Event Time processing); C-SPARQL [14] extends the SPARQL query language for continuous reasoning within a publisher/subscriber platform. For static data: RMLMapper [12] enables data fetching and declarative mapping with RML rules; Ontop [15] creates a virtual graph representation of various data sources via SPARQL queries; and SLOGERT [16] orchestrates log modeling and annotation with Cyber Threat Intelligence tags[7].

[4] The TM Forum's Open Digital Architecture (ODA) aims to improve user experience and Information System (IS) interoperability in the ICT industry beyond general best-practice approaches for Decision Support Systems design.

[5] https://www.postgresql.org/

[6] http://www.rrdtool.org/

[7] As for SLOGERT v0.9.1: with MITRE CEE categories from http://cee.mitre.org/language/1.0-alpha/

This paper's end-to-end data processing approach combines NMSs/SIEMs design patterns with Semantic Web tools. It includes requirements for distributed processing, separation of concerns, data sketching (i.e. enabling both early and posterior reasoning on data), openness to third-party databases/tools, and re-use of well-established frameworks (e.g. declarative mapping, message passing).

3. Design methodology and challenges

This section outlines the methodology followed in constructing a knowledge graph-based data platform with stream processing and reasoning capabilities for Incident Management over ICT systems. It proposes a conceptual tool chain and examines design challenges and requirements to frame the high-level design and implementation work discussed in Section 4.

Conceptual tool chain & design principles. Looking at data integration theory and generic data transformation processes (e.g. "Extract, Transform, Load", CRISP-DM [17]), we remark that none directly take into account the abstraction and reasoning capabilities brought by the Semantic Web technologies and Knowledge Graphs. Furthermore, these design patterns set apart decision making concerns where informed-decisions potentially involve gradual understanding of data (i.e. raw data → information → knowledge) combined with synergical reasoning [18].

Extending on these, we propose a tool chain model (Figure 1) to guide design thinking steps: *unstructured data* (e.g. event logs) enters the tool chain and becomes *structured data* by application of a defined/learnt *structure model*. Semantic mapping is applied for making *annotated data*. These can benefit of additional knowledge from some *enrichment service* (e.g. mapping assets to organization or vulnerability knowledge). *Reasoning service* (e.g. rule-based inference, confidence propagation, link/entity prediction) works from *annotated data* for producing further knowledge (i.e. *interpreted data*). Downstream agents (e.g. operational teams, information system) get informed (e.g. situation awareness) by querying *interpreted data*. This conceptual tool chain is open to complementary processes, such as a direct feed of structured data or recursive loops of the *inference* step.

Based on the above, we posit the following design principles (further discussed below) to streamline integration and improve user adoption: *1) Minimize transformation needs at ingress*: data encoding (serialization & structuration) must be backward compatible early in the processing chain to limit the number of technologies used; *2) Independent downstream usage*: parallel downstream applications may focus on different data facets, so the serialization/structure should allow easy separation of data from meta-data without imposing specific remote procedure calls; *3) Implementation independent*: abstractly describing transformation and processing rules enables system behavior description and transposition independent of implementation; *4) Integrate, customize or build*: prioritize integrating existing frameworks that meet requirements, extend partially meeting frameworks, or develop specific solutions if neither of the previous options apply.

Note that knowledge engineering methodologies (e.g. Competency Questions [19] and Linked-Open Terms [20]) are separate from our proposal. Data models resulting from these methodologies are used in the *annotation* step, but our tool chain is not affected by changes in data models from a functional and technical perspective.

Dataset characteristics organize the processing architecture. Scrutinizing Orange internal datasets and third party datasets based on their TAM Domain/Sub domain[8], we took note of the data structures and technical characteristics (e.g. number and type of features, serialization syntax, schema definition, access protocol, update period) for devising a data integration strategy. The *update period* and *data access method* emerged as key design factors: descriptive datasets (e.g. assets database, network topology, organization) have a low refreshment pace (one day to one week period) and are generally available through file-based platforms (e.g. database API, file dumps), while network operations and events (e.g. interface status change, applications logs, alarms, trouble tickets) are stream feeds with fast-paced time-stamped data (real time to quarter-hour period).

Data wrangling with syntax heterogeneity. When transforming data, we need to simultaneously consider syntax heterogeneity and batch/stream processing. This can be represented using the $ET[P1]L[P2]L$ model embedded in Figure 1, where $P1$ components are for per feed processing and $P2$ components apply at the dataset level. The $P1/L$ interface should comply with standard data transport solutions (e.g. JSON for Kafka messages) and knowledge graph data ingestion methods (e.g. SPARQL Update, periodic/on-demand bulk load), while the $L/P2/L$ interface requires data representation transformation to match $P2$ requirements (e.g. Turtle to JSON-Graph) and integrate results (e.g. time-stamped confidence as a RDF triple) into the knowledge graph-based application data model.

Data wrangling with annotation, patching & reconciliation tasks. We assume data must have meaning (e.g. data is about a hostname or a date, and not just a string of characters) and structure to be useful (in our case, a relational graph structure). Therefore we introduce the concept of data patching & reconciliation to do in-place update of the graph data and link entities from different sources. This includes substituting equivalent literals with controlled vocabulary and normalizing terms and relationships.

Post-processing constraints on the ETL stages. First, it is important to track the data origin for trustworthiness. From a practical standpoint, this allows for: *1)* isolating/correcting contaminated (intentionally or not) data sources; *2)* accessing the original data to restore its original meaning and context; *3)* exposing data characteristics (e.g. freshness, validity period) for refined decision making. Second, it is necessary to make post-processing efficient by considering both the composition of post-processing (e.g. sequential, parallel) and the form of the data for lossless transformation, such as from graph to table. The third goal is to determine how post-processing results are utilized. This involves considering the compatibility of processing blocks and whether the results can be interpreted beyond their original context. It also involves reintroducing the results into the base data space to serve iterative or synergistic reasoning. The nature of the result must also be considered in terms of form and value, as it can add information

[8] i.e. their parent application/research domain within the TM Forum's GB929 Application Framework (TAM), see https://www.tmforum.org/application-framework/

to an existing object (e.g. assigning a cyber security risk level to a network asset) or create a new object (e.g. an alert). Provenance and trust are necessary here, but with different semantics since they affect the product of data interpretation, not the original data.

4. KG-based platform and data processing architecture

Thinking through the design methodology of Section 3, we developed two data integration pipelines and a mechanism for data interpretation (Figure 1) based on well-known open source frameworks (e.g. Apache Kafka[9], Apache Airflow[10], OpenLink Virtuoso[11]), academic projects (e.g. RMLMapper [12], StreamingMASSIF [13], string2vocabulary [21], grlc [22], RDFUnit[12]) and adhoc code (Table 1). The overall system is akin to a Lambda data processing architecture [23].

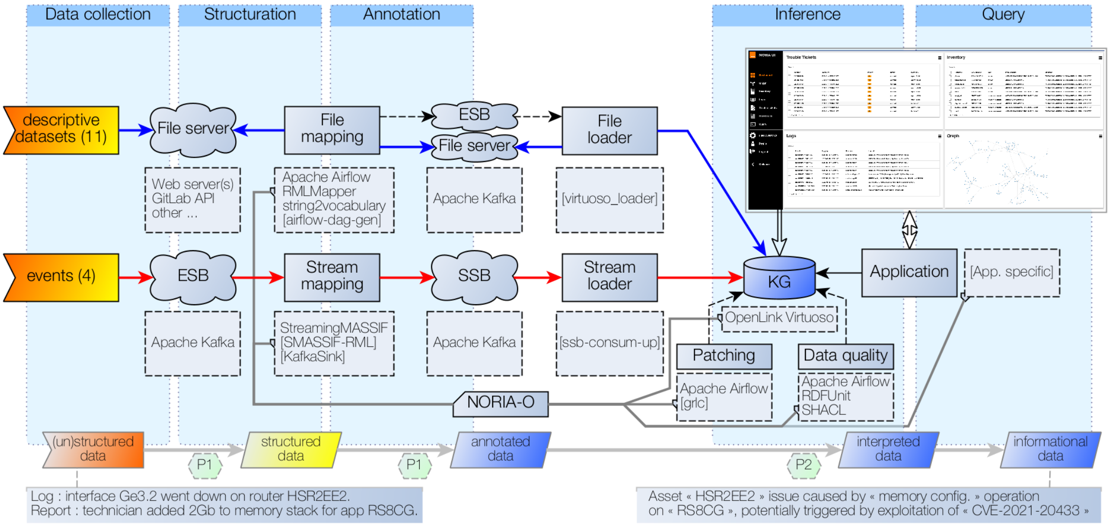

Figure 1: Conceptual tool chain & data platform overview.
Acronyms: ESB = Enterprise Service Bus, SSB = Semantic Service Bus, KG = Knowledge Graph. Plain arrows are for data flows, dotted arrows for control and query flows. Arrows start from the component initiating the flow/transaction. Numbers for the “descriptive datasets” and “events” blocks refer to the number of sources in Table 2. Component names within brackets relate to adhoc code from Table 1. $P1$ and $P2$ stands for Processing components groups in an $ET[P1]L[P2][L]$ reading of the tool chain (i.e. matching the tool chain with the ETL process model).

Knowledge graph management. We manage the knowledge graph using a Virtuoso quad store, with enabled SPARQL endpoint and Faceted Browser services. Named graphs enable fast data access and help track the source of triples in RDF datasets. We use predefined URI patterns that closely match the NORIA-O data model (see below), following the *Graph per Source* and/or *Graph per Aspect* data management patterns [24]. This prior knowledge simplifies the creation

[9] https://kafka.apache.org/
[10] https://airflow.apache.org/
[11] https://virtuoso.openlinksw.com/
[12] https://github.com/AKSW/RDFUnit

Table 1
Adhoc code (complementary developments).

Component name	Role
airflow-dag-gen	Parametric Airflow DAG generator for data integration and patching.
KafkaSink	StreamingMASSIF [11] component for JSON-LD output to Kafka, available at https://github.com/Orange-OpenSource/SMASSIF-RML.
grlc	Enhanced grlc [22] with GitLab connector and SPARQL Update, available at https://github.com/Orange-OpenSource/grlc.
NORIA-O	RDF data model for IT networks, events and operations information, available at https://w3id.org/noria/.
SMASSIF-RML	Modified RMLMapper [12] for StreamingMASSIF [13] component, available at https://github.com/Orange-OpenSource/SMASSIF-RML.
ssb-consum-up	Kafka to SPARQL gateway, available at https://github.com/Orange-OpenSource/ssb-consum-up.
virtuoso_loader	Event-triggered (Kafka) bulk load of remote RDF datasets into Virtuoso.

of linked data using a declarative transformation approach before inserting mapped data into the graph (see below for implementation details).

Batch processing for descriptive datasets [airflow-dag-gen, virtuoso_loader]. A set of Apache Airflow DAGs[13] periodically trigger data downloading, mapping and inserting tasks. DAGs are defined on a per $\langle Source, View \rangle$ basis and are configured using a limited set of parameters: schedule interval, a reference to a `noria:ETL_process_node` entity, and templated ETL tasks to schedule.

`noria:ETL_process_node` entities are configuration nodes stored in the platform's knowledge graph. They include a reference to the data source to download via a `dcat:downloadUrl` property, and a relationship to the RML mapping rules (also stored in the platform's knowledge graph). This allows for centralizing information (configurations and mapped data) with a homogeneous representation, resulting in simplified interrogation and audit of data provenance. Because RML is RDF data, making these rules available from the knowledge graph is as simple as uploading an RML file into the graph store once the mapping implementation done.

Prior starting a mapping thread (i.e. a local `rmlmapper-java` instance), *1)* a `fetchRules` task queries the knowledge graph for the mapping rules and stores them in a temporary file, and *2)* a `fetchData` task downloads the raw data to map. Then mapping is started with complementary output configuration parameters asking for RDF Trig[14] serialization (enables targeting specific named graphs in the downstream graph store with `rr:graph` attributes in the mapping implementation) and provenance metadata generation at the dataset level [25] (relates the mapping activity to the `rml:source` used in the mapping implementation with a `prov:used` attribute). Because `rmlmapper-java` does not include target graph data in the provenance metadata file, we rewrite the file with an `adjustProvenance` step. Once the mapping thread is over, a `loadRequest` signal triggers `fetchMappedData` and `loaderRun`

[13] https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html
[14] https://www.w3.org/TR/trig/

threads on a downstream `virtuoso_loader` component listening to a specialized Kafka topic. This ends the batch processing for descriptive datasets by inserting mapped data and provenance metadata into the knowledge graph.

Speed processing for events [SMASSIF-RML, KafkaSink, ssb-consum-up]. A set of specialized StreamingMASSIF pipelines continuously consume, map and forward data for insertion into the knowledge graph by a downstream `ssb-consum-up` component (a Kafka to SPARQL gateway). The typical form of the pipelines is $KafkaSource \rightarrow RMLMapper \rightarrow (OptionalProcessing) \rightarrow KafkaSink$, where $KafkaSource$ is a StreamingMASSIF native component, $RMLMapper$ is a modified version of the `rmlmapper-java` tool (to handle streamed data as a StreamingMASSIF component), $OptionalProcessing$ can consist of any combination of StreamingMASSIF's reasoning components[15], and $KafkaSink$ is a contributed component that sends the data stream in JSON-LD syntax to a Semantic Service Bus (SSB). Pipelines are defined on a per $\langle Source, View\rangle$ basis and are configured using a limited set of parameters: input topic, reference to an RML rules implementation, and output topic.

We posit that RDF data downstream of the mapping pipelines can serve multiple purposes as distinct streams (e.g. direct update of the knowledge graph, intermediary vocabulary reconciliation, multi-source event logs co-occurrence alerting, notification-triggered dependency calculus). To manage the distribution of streams, we developed the concept of SSB using the Kafka event streaming technology and considering the following features: *1)* forwarding RDF data messages in standard RDF serialization; *2)* providing provenance metadata; *3)* providing named-graph compatibility; *4)* enabling the use of SPARQL Update actions[16].

The Kafka platform uses $\langle key, value\rangle$ pair-based messaging with native JSON compatibility, hence JSON-LD is a natural serialization choice for forwarding RDF data through the SSB. We identified two strategies for mapping JSON-LD with the $\langle key, value\rangle$ message model while handling potentially RDF dataset-shaped streams: *1)* build Kafka messages with `key = NULL`, and `value = <JSON-LD payload>` (with or without named graph); *2)* assuming named graph at the subject level, build Kafka messages with `key = <JSON-LD metadata>` (e.g. {[provenanceMetadata] , [updateAction], [otherMetadata]}) and `value = <JSON-LD payload>` (with named graph). The strategy #1 is the most straightforward and versatile approach, as it delegates all decision making regarding RDF triples to the downstream consumer(s), including which graph update operations to apply (such as INSERT or DELETE). Further, it directly complies with the fact that the Kafka's key is for partitioning and compaction[17], which means that key is akin to a primary key and may not be used for metadata unless made compatible with the partitioning and compaction principles (note that key can be left empty). The strategy #2 enables providing guidance to downstream consumers on how to handle RDF triples through metadata. We remark that `updateAction` can leverage on well-known vocabularies such as `schema:UpdateAction`[18]. Similarly, `provenanceMetadata` can include a `prov:wasGeneratedBy` attribute to keep track of the successive modifications made within a tool chain. A downside of the second strategy is that the scope of the `updateAction` should

[15] As for StreamingMASSIF v0.0.1: Windowing, Filtering, Abstraction, Complex Event Processing (CEP).

[16] https://www.w3.org/TR/sparql11-update/

[17] https://kafka.apache.org/documentation/#compaction

[18] https://schema.org/UpdateAction

match all `<JSON-LD payload>` triples, both in terms of the operations to apply and target named graph. Because the strategy #2 can be later built upon the #1, we chose to implement the #1 as a starter, that is: straightforward payload mapping at the `KafkaSink` level and graph update operations defined at the `ssb-consum-up` level.

Since we are using named graphs at the knowledge graph level and that the Virtuoso graph store requires defining the target graph parameter for data insertion using SPARQL update queries, we explicit the design of the `ssb-consum-up` component with Equations (1) & (2):

$$(s, p, o, g) \xrightarrow[Op]{} Op\{\text{GRAPH}\; g\; \{s\; p\; o\}\} \tag{1}$$

$$(s, p, o) \xrightarrow[Op,g]{} Op\{\text{GRAPH}\; g\; \{s\; p\; o\}\} \tag{2}$$

where Eq. (1) states that incoming messages from the SSB are RDF triples along with a target graph information, thus data insertion into the downstream SPARQL endpoint is akin to translating the messages into SPARQL update queries subject to a user-defined action Op (e.g. `INSERT`); and Eq. (2) reflects the same with an additional user-defined target graph parameter g whenever incoming messages are raw RDF triples. Assuming many StreamingMASSIF pipelines pushing mapped data with target graph information into a same SSB topic, then a single `ssb-consum-up` component instance is sufficient for continuous data insertion (e.g. with $Op = INSERT$, and $g = \langle DefaultGraphURI \rangle$ for filling any gaps).

Data model [NORIA-O]. Based on Orange network & cybersecurity expert panel interviews and Competency Questions analysis [19], four facets structuring the knowledge domain emerge from the entities and properties that we identified for describing incident management over ICT systems: *structural* (network assets such as servers and links), *functional* (network services and flows), *dynamic* (events and states changes) and *procedural* (processes and actions). There are several efforts to propose data models representing computing resources and how they are allocated for hosting services. However, to date, there is no model to describe the multiple interdependencies between the structural, dynamic, and functional aspects of a network infrastructure. In line with the Linked Open Term methodology [20], we have formalized and implemented the NORIA-O conceptual model [26], an OWL-2 ontology that re-uses and extends well-known ontologies such as SEAS[19], FOLIO[20], UCO[21], ORG[22], BOT[23] and BBO[24]. It is used as the main data model for the data integration and exploitation work described in this paper as it can model complex ICT system situations and serve as a basis for anomaly detection and root cause analysis. The NORIA-O data model also provides a set of controlled vocabularies useful for standard interpretation of the knowledge graph entities; for example, with reconciliation (see below) on the network device alarms through the <Notification/EventTypeGroup/SecurityAlarm> concept scheme.

[19] https://ci.mines-stetienne.fr/seas/
[20] https://github.com/IBCNServices/Folio-Ontology
[21] https://ontology.unifiedcyberontology.org/
[22] http://www.w3.org/ns/org
[23] https://w3id.org/bot
[24] https://www.irit.fr/recherches/MELODI/ontologies/BBO

Patching & Reconciliation [airflow-dag-gen, grlc]. The per source and per concept mapping approaches discussed above entails handling data ingest interdependencies with complementary patching & reconciliation tasks. We make use of an Airflow DAGs-based periodic run of ordered patching queries in SPARQL syntax via a enhanced grlc [22] tool instance. For this approach to work, we assume that the NORIA-O data model (see above) is available in the data store, including the controlled vocabularies (a.k.a. NORIA-O KOS).

We observe that patching requests follow a limited number of forms that can be expressed as (arche)types of patching queries, thus leading to a standard approach to patching (Eq. 3, where P stands for Patching Queries and O for Ontology):

$$P_{templates} \times P_{definitions} \xrightarrow{\text{query generator}} P \quad ; \quad O \times P \xrightarrow{\text{patching}} O' \tag{3}$$

We define the three following archetypes, hence making the mapping definition process faster and easier to maintain via patching requirements set in a definition file (e.g. YAML syntax) and query generation (e.g. Python script + templated SPARQL queries in JINJA2 syntax[25]):

1. $literal2KOS := \langle Literal \rangle \rightarrow \langle skos : Concept(Subject) \rangle$. We implement it with SPARQL queries as an exact string match via a `LCASE(STR(x)) = LCASE(STR(y))` statement in order to avoid declaring redundant `skos:altLabel` in the NORIA-O vocabulary files. For example, from Figure 1: “interface went down” → EventRecord.type(< kos/Notification/EventType/StateChange>).

2. $literal2URI := \langle Literal \rangle \rightarrow \langle Subject \rangle$) (but not a KOS URI). Likewise $literal2KOS$, we implement it as an exact string match. For example, from Figure 1: “router HSR2EE2” → Resource.resourceHostName('HSR2EE2').

3. $addShortcut := \{\langle Predicate, Object \rangle\} \rightarrow \langle Subject \rangle$, i.e. a direct property between a subject and an object when these two nodes are related by a given longer path. For example, from Figure 1: “issue potentially triggered by” → EventRecord.conformsTo(Vulnerability(' CVE-2021-20433')).

Complementary iterative processing. Complementary Airflow DAGs trigger data model and data quality audits (e.g. querying the knowledge graph against the NORIA-O competency questions, checking data ingest conformance with the RDFUnit tool), performance evaluations (e.g. queries velocity *vs* NORIA-O expressivity), and application-specific code (e.g. querying the IT network topology from the knowledge graph and then running a graph-based risk assessment method).

5. Lessons learned

Our design currently runs on Orange internal data (10 data sources encompassing 128 used features over 15 tables, see Table 2). Batch processing generates, updates and patches the knowledge graph on a hourly basis. Speed processing works on generated data until further

[25] https://jinja.palletsprojects.com/

integration within the data ecosystem. The size of the resulting RDF dataset at hand is approximately four million triples for 400K entities, including streamed events spanning over 111 days[26].

Table 2

Data sources and mapping overview, with generic names of Orange internal data sources, along with features and mapping statistics. Acronyms: AAA = Authentication, Authorization and Accounting; VM = Virtual Machine. Unused features are due to data redundancy, or data without correspondence with the NORIA-O data model.

Nature	Data source (table)	Features (total)	Features (used)	Features (used ratio)	rr:TriplesMap (count)
Events	Trouble Tickets	28	21	75,00 %	6
	Change Tickets	124	11	8,87 %	2
	Alarm monitoring	152	8	5,26 %	1
	Logs monitoring	164	3	1,83 %	1
Descriptive	AAA groups	6	4	66,67 %	2
	Applications	25	15	60,00 %	2
	Teams	14	8	57,14 %	3
	Users	12	6	50,00 %	2
	Logistic database	51	19	37,25 %	8
	Backbone logical links	14	5	35,71 %	2
	Backbone physical links	14	4	28,57 %	3
	Applications types	63	9	14,29 %	1
	Network topology	16	2	12,50 %	1
	VM management	74	9	12,16 %	3
	VM clusters	57	4	7,02 %	2

By applying the Authoring Test approach [19] with the NORIA-O competency questions[27], the resulting RDF dataset proves to be of great interest from a business perspective as it enables the handling of incident management needs across heterogeneous data through the use of data retrieval techniques and complementary AI-based algorithms. Indeed, out of the 26 competency questions defined with expert panel interviews, a significant number of them (16/26) can be answered using simple SPARQL queries and the NORIA-O ontology, while the remaining questions (9/26) may require additional techniques such as relational learning or anomaly detection algorithms.

From a more technical perspective, the software infrastructure is deployed using an Infrastructure as Code approach. A main project installs and configures the platform using templated scripts based on a host feature inventory. Components can be individually started, stopped, or upgraded thanks to a microarchitecture design. The platform uses nine virtual machines hosted in Orange's private cloud with varying hardware setups (e.g. 1-4 vCPU, 8-16 Gb memory, 20-80 Gb storage). CI/CD is further used for granular version control and performance evaluation, particularly for the NORIA-O data model, where pre-publishing review and expressivity evaluation are enforced as per the LOT methodology [20]. The data model is automatically loaded into the data store when there is a change. The same approach is used for orchestration DAGs, where data integration tasks can change based on data source changes. The latest DAGs releases are downloaded and scheduled via an update signal sent to the Apache Airflow instance.

[26] Due to confidentiality, this dataset is not made public.

[27] https://w3id.org/noria/cqs/

From the Apache Airflow DAGs and Virtuoso logs, we measure that the *map data* and *adjust provenance* tasks are from far the longest tasks of the DAGs (Table 3). As we implemented simple RML rules (i.e. without `rr:joinCondition`), the mapping time can hardly be lowered as it depends on the input file size and `rmlmapper-java` tool implementation. We remark from complementary experiments that `rr:joinCondition` entail a $\times 2$ to $\times 5$ increase of processing time. However, improving provenance data generation for the *adjust provenance file* step (e.g. at the `rmlmapper-java` or file rewriting level) may bring better overall performance with a $\times 4$ increase in throughput.

Table 3
Batch processing performance for three representative (small/medium/big) sources from Table 2. Preprocessing may include: convert to csv, delete first line, convert to UTF-8, crop columns.

	AAA groups		Users		Logistic database		Unit
Input data size	0.16		2.4		45.5		[Mb]
Download data	0.44	6.63 %	0.95	1.54 %	3.32	0.69 %	[s]
Dump rules	0.14	2.11 %	0.19	0.31 %	0.15	0.03 %	[s]
Preprocessing	0.19	2.86 %	9.46	15.37 %	8.66	10.83 %	[s]
Map data	3.27	49.25 %	8.54	13.87 %	79.97	16.70 %	[s]
Adjust provenance	2.27	34.19 %	40.66	66.05 %	374.26	78.16 %	[s]
Notify for loading	0.27	4.07 %	0.29	0.47 %	0.29	0.06 %	[s]
Data bulk load	0.05	0.75 %	1.46	2.37 %	12.17	2.54 %	[s]
Prov. bulk load	0.01	0.15 %	0.01	0.02 %	0.02	0.00 %	[s]
Total time	6.64		61.56		478.84		[s]
Output data	0.52		21		222		[Mb]
	5 110		244 532		2 415 676		[Triples]
Throughput	769.58		3 972.25		5 044.85		[Triples/s]

For speed processing, we confirmed the effectiveness of our `SMASSIF-RML` → `ssb-consum-ub` → `Virtuoso` tool chain based on local experiments with generated data (related to the “events” category from Table 2). However, a thorough load study is yet to be conducted with real data sources. Evaluation is left for future work, which will depend on the evolution schedule of the streaming sources to provide standardized connectors. Besides performance, we observed that although Kafka allows data replay for overcoming subsystems failures, it is a complex system; so materializing mapped data in files (as in our batch processing approach) seems more reliable.

For patching & reconciliation, we make use of 42 SPARQL queries ($literal2KOS = 16$, $literal2URI = 19$, $addShortcut = 7$). From our experience on reconciliation, $literal2KOS$ with exact match is sufficient in a great majority of cases, but will misses advanced text analysis situations, such as for `noria:logText` parsing (e.g. noria:EventRecord.logText("LINK-3-UPDOWN: Interface GigabitEthernet0/0/1, changed state to up") to enrich with dcterms:type <kos/Notification/EventType/stateChange>). Hence we developed two complementary approaches: *1)* we extended the String2Vocabulary tool [21] with named graph processing capabilities for enabling vocabulary reconciliation with a fuzzy match approach as a DAG task consecutive to data mapping; *2)* we experimented with the Slogert framework [16] for complementary `noria:logText` structuring and annotation in a $KG \xrightarrow{Slogert} KG$ fashion through a DAG.

More generally, we remark that combining Airflow with grlc [22] allows quick development of knowledge graph-based applications of the extract-process-report type, and friendly access to data and operations for non-technical/non-expert users. Furthermore, our two step "simple mapping *vs* posterior patching" approach allowed us maximizing direct graph traversal capability with URIs while minimizing duplicates although handling them. This has notably allowed us to keep the knowledge graph's complexity low by avoiding the use of `owl:sameAs` predicates. Finally, we remark that, thanks to a report table such as Table 2, tracking the characteristics of the source files and TripleMaps for comparison is simplified, resulting in time savings for exploring and cross-referencing information. Building this table is possible through scripted process akin to $RMLs \times DataSources \rightarrow \{SourceFile, features_{total}, features_{used}\}$. A nice consequence of this programmatic analysis is the natural emergence of mapping management patterns; this notably led to design a DAG generator tool for convenient management and deployment of the ETL and patching DAGs. Table 2 is also valuable for complementary analysis. First, the "used ratio" indicator reveals sparse/dense data sources for our application domain, raising concerns about information redundancy and database design practice. Second, it suggests potential for additional concepts/relationships, depending on clever understanding of the necessary and sufficient features for a given domain. Third, it enables direct reading of data flow from sources to concepts and graphs, revealing design principles and characteristics behind them.

6. Conclusion and Future Work

In this work, we aimed to design and implement a data processing architecture for knowledge graph-based incident management of broad scale Information and Communications Technology (ICT) systems. We firstly hypothesized that the cross-referencing of semantic representations from multiple sources would enable the evolution of decision support systems for ICT systems to a next level of diagnosis and recommendation capabilities. Next, we developed and deployed a Lambda data processing architecture combining well-known open source frameworks (Apache Kafka, Apache Airflow, OpenLink Virtuoso), academic projects (RMLMapper [12], StreamingMASSIF [13], string2vocabulary [21], grlc [22], RDFUnit) and adhoc code released in open source (grlc[28], SMASSIF-RML[29], ssb-consum-up[30]). The proposed architecture has been instantiated and tested in an industrial setting, producing an RDF knowledge graph that shows strong potential for addressing cross-domain anomalies from heterogeneous data.

The solution notably minimizes the effort for data quality and trust audit thanks to the generalized use of RML, and the centralized storage of both data and mapping configuration within the knowledge graph. Additionally, we open up the possibility of distributed processing or event-triggered processing through the generalization of RDF data transfer by a message-broker software. However, the data provenance tagging at the dataset level leads to a loss of information granularity after the data patching/reconciliation steps and introduces a heavy file adjustment step. Further, a thorough load study is necessary to consider deploying the stream processing pipeline on massive data (e.g. telemetry data from broadband network routers or a fleet of IoT devices).

[28] https://github.com/Orange-OpenSource/grlc
[29] https://github.com/Orange-OpenSource/SMASSIF-RML
[30] https://github.com/Orange-OpenSource/ssb-consum-up

Future work on the NORIA platform will consider both improving the Knowledge Graph Construction (KGC) process and using the resulting knowledge graph for efficient incident management. Focusing on KGC, future work will explore how the full description of the ETL processes could be stored within the knowledge graph with RDF process models [27, 28]. This would allow auditing data platforms through a single language, thanks to a joint representation of data and processing mechanisms. In the same line of thought, automated patching generation can be enabled by browsing RML files for rr:predicateObjectMap [rr:objectMap [rml:reference "< someRef>"]], potentially with an additional toPatchWith(<someGraphPattern>) property for better end-to-end process automation and automated URI template checking. For stream processing, we envision comparing our approach with other frameworks [11, 14]. This should help identify decision boundaries, particularly in terms of energy efficiency and network overhead, in order to move towards a Kappa architecture [23]. This should also provide insights on the signaling mechanisms to be implemented for opportunistic processing (e.g. SKOS reconciliation as a service, in-line graph clustering) and synergical reasoning [18] (cooperative decision making). Finally, scrutinizing knowledge graph pruning and summarization techniques will prevent from ever expanding datasets (e.g. ICT systems situation models *vs* an accumulation of logs), although using generic RDF data models for knowledge representation is already a mitigating factor.

Acknowledgments

The Orange research program[31] supported the research. Mihary Ranaivoson's development work during his internship on the data processing architecture is also acknowledged.

References

[1] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, R. Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, Antoine Zimmermann, Knowledge Graphs, 2020. arXiv:2003.02320.

[2] Pankaj Prasad, Josh Chessman, Market Guide for IT Infrastructure Monitoring Tools, Technical Report G00450400, Gartner, 2019.

[3] Josh Chessman, Magic Quadrant for Network Performance Monitoring and Diagnostics, Technical Report G00463582, Gartner, 2020.

[4] Kelly Kavanagh, Toby Bussa, Gorka Sadowski, Magic Quadrant for Security Information and Event Management, Technical Report G00348811, Gartner, 2018.

[5] Gustavo González-Granadillo, Susana González-Zarzosa, Rodrigo Diaz, Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures, Sensors (2021). doi:10.3390/s21144759.

[6] David Swift, A Practical Application of SIM/SEM/SIEM Automating Threat Identification, White Paper, SANS Institute, 2007.

[31] https://hellofuture.orange.com/

[7] Pierre-Antoine Champin, Alain Mille, Yannick Prié, Vers Des Traces Numériques Comme Objets Informatiques de Premier Niveau, Intellectica - La revue de l'Association pour la Recherche sur les sciences de la Cognition (ARCo) (2013). doi:10.3406/intel.2013.1090.

[8] Xiangnan Ren, Olivier Curé, Li Ke, Jeremy Lhez, Badre Belabbess, Tendry Randriamalala, Yufan Zheng, Gabriel Kepeklian, Strider: An Adaptive, Inference-Enabled Distributed RDF Stream Processing Engine, Proceedings of the VLDB Endowment (2017). doi:10.14778/3137765.3137805.

[9] S. N. Narayanan, A. Ganesan, K. Joshi, T. Oates, A. Joshi, T. Finin, Early Detection of Cybersecurity Threats Using Collaborative Cognition, in: 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), 2018. doi:10.1109/CIC.2018.00054.

[10] Bram Steenwinckel, Dieter De Paepe, Sander Vanden Hautte, Pieter Heyvaert, Mohamed Bentefrit, Pieter Moens, Anastasia Dimou, Bruno Van Den Bossche, Filip De Turck, Sofie Van Hoecke, Femke Ongenae, FLAGS: A Methodology for Adaptive Anomaly Detection and Root Cause Analysis on Sensor Data Streams by Fusing Expert Knowledge with Machine Learning, Future Generation Computer Systems (2021). doi:10.1016/j.future.2020.10.015.

[11] Gerald H, Sitt Min Oo, Gertjan De Mulder, Michiel Derveeuw, Pieter Heyvaert, Wouter Maroy, Vincent Emonet, kmhaeren, Ben De Meester, Dylan Van Assche, Thomas, ajuvercr, RMLio/RMLStreamer, 2022. doi:10.5281/zenodo.7181800.

[12] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, Rik Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Proceedings of the Workshop on Linked Data on the Web, LDOW 2014, co-located with the 23^{rd} International World Wide Web Conference (WWW 2014), CEUR-WS.org, 2014.

[13] Pieter Bonte, Riccardo Tommasini, Emanuele Della Valle, Filip De Turck, Femke Ongenae, Streaming MASSIF: Cascading Reasoning for Efficient Processing of IoT Data Streams, Sensors (2018). doi:10.3390/s18113832.

[14] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus, C-SPARQL: SPARQL for Continuous Querying, in: Proceedings of the 18th International Conference on World Wide Web, Association for Computing Machinery, New York, NY, USA, 2009. doi:10.1145/1526709.1526856.

[15] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalaycı, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, Elena Botoeva, The Virtual Knowledge Graph System Ontop, in: The Semantic Web – ISWC 2020, 2020. doi:10.1007/978-3-030-62466-8_17.

[16] Andreas Ekelhart, Fajar J. Ekaputra, Elmar Kiesling, The SLOGERT Framework for Automated Log Knowledge Graph Construction, in: The Semantic Web, 2021. doi:10.1007/978-3-030-77385-4_38.

[17] C. Shearer, The CRISP-DM Model: The New Blueprint for Data Mining, Journal of data warehousing (2000).

[18] Ben Goertzel, Cassio Pennachin, Nil Geisweiller, Engineering General Intelligence, Part 1: A Path to Advanced AGI via Embodied Learning and Cognitive Synergy, Atlantis Thinking

Machines, Atlantis Press, 2014. doi:10.2991/978-94-6239-027-0.

[19] Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, Robert Stevens, Towards Competency Question-Driven Ontology Authoring, in: 11th European Semantic Web Conference (ESWC), 2014. doi:10.1007/978-3-319-07443-6_50.

[20] María Poveda-Villalón, Alba Fernández-Izquierdo, Mariano Fernández-López, Raúl García-Castro, LOT: An industrial oriented ontology engineering framework., Engineering Applications of Artificial Intelligence (2022). doi:10.1016/j.engappai.2022.104755.

[21] Pasquale Lisena, Konstantin Todorov, Cécile Cecconi, Françoise Leresche, Isabelle Canno, Frédéric Puyrenier, Martine Voisin, Thierry Le Meur, Raphaël Troncy, Controlled vocabularies for music metadata, in: 19th International Society for Music Information Retrieval Conference, 2018. doi:10.5281/zenodo.1492441.

[22] Albert Meroño-Peñuela, Rinke Hoekstra, grlc Makes GitHub Taste Like Linked Data APIs, in: The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Springer, 2016. doi:10.1007/978-3-319-47602-5_48.

[23] Dorota Owczarek, Lambda vs. Kappa Architecture. A Guide to Choosing the Right Data Processing Architecture for Your Needs, https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/, 2022.

[24] Leigh Dodds, Ian Davis, Linked Data Patterns: A Pattern Catalogue for Modelling, Publishing, and Consuming Linked Data, https://patterns.dataincubator.org, 2012.

[25] Anastasia Dimou, Tom De Nies, Ruben Verborgh, Automated Metadata Generation for Linked Data Generation and Publishing Workflows, in: Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with 25th International World Wide Web Conference (WWW 2016), CEUR-WS.org, 2016.

[26] Lionel Tailhardat, Yoan Chabot, Raphaël Troncy, NORIA-O: an Ontology for Anomaly Detection and Incident Management in ICT Systems (2022).

[27] Amina Annane, Nathalie Aussenac-Gilles, Mouna Kamel, BBO: BPMN 2.0 Based Ontology for Business Process Representation, in: 20th European Conference on Knowledge Management (ECKM), Academic Conferences and publishing limited, 2019.

[28] Juan C. Vidal, Manuel Lama, Alberto Bugarın, A High-level Petri Net Ontology Compatible with PNML, Petri Net Newsletter 71 (2006).