Internet of Things 31 (2025) 101557

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

CONCERN: A model-based monitoring infrastructure

Antonello Calabrò [a,*], Eda Marchetti [a], Paweł Skrzypek [b], Jan Marchel [b]

[a] CNR Istituto di Scienza e Tecnologie dell'Informazione "A.Faedo", Via Giuseppe Moruzzi, 1 - 56124, Pisa, Italy
[b] AI Investments, 7Bulls.com Al. Armii Ludowej 26, Warsaw, 00-609, Poland

ARTICLE INFO

Keywords:
Monitoring
System engineering
Anomaly detection
Predictive maintenance

ABSTRACT

Characteristics such as quality, trustworthiness, and cybersecurity are becoming essential in the ecosystems of Internet of Things (IoT) systems. Although significant efforts in industry and academia have been made to create solutions that ensure, monitor, and evaluate these attributes throughout the development process, time-to-market urgency, demands for higher productivity, and competitive pressures often result in faster release schedules. This situation calls for methods and means to quickly and accurately identify and avert anomalies and hazardous circumstances. Among them, a monitoring system is an effective solution for evaluating functional and non-functional attributes during the online execution. This paper proposes an integrated Runtime Monitoring solution called CONCERN (COmplex eveNt proCEssing monitoR iNfrastructure) for guaranteeing a consistent quality of services, managing the possible execution exceptions and interruptions, and providing a high system availability. The paper describes the CONCERN supporting architecture and proof-of-concept implementation of its components, validation of the CONCERN usability, and its performance in a real scenario taken from the financial domain. Finally, the paper reports the discussion, lessons learned, and the future work suggested by the showcase.

1. Introduction

Attributes like quality, trustworthiness, and cybersecurity are becoming crucial characteristics of Internet of Things (IoT) systems and their ecosystems. Despite significant industrial and academic research efforts to develop solutions that ensure, monitor, and evaluate these attributes throughout the development process, time-to-market pressures, productivity demands, and competitive urgency often lead to accelerated release schedules. These fast-paced timelines can compromise overall product quality and heighten the risk of vulnerabilities and weaknesses. Establishing practical and integrated methodologies and mechanisms to address these challenges is essential. These should focus on enhancing the understanding and management of the quality attributes and properties, continuously assessing development risks, and mitigating potential vulnerabilities. By doing so, organizations can create a more robust framework that supports safe and reliable IoT systems and can contribute to safer technology deployment and user trust. This holistic approach can ensure that in IoT development, security and quality are not sacrificed in the race for innovation.

Among the methods for quickly and accurately identifying and averting abnormalities and hazardous circumstances in an existing Internet of Things ecosystem or system, the monitoring system is strongly suggested by the literature as an effective and practical solution [1].

Indeed, monitoring infrastructure is specifically designed to assess explicit requirements during online execution to promptly raise alarms or implement countermeasures in case of unattended behavior detection [2]. Monitoring activities usually focus both

* Corresponding author.
E-mail addresses: antonello.calabro@isti.cnr.it (A. Calabrò), eda.marchetti@isti.cnr.it (E. Marchetti), pskrzypek@7bulls.com (P. Skrzypek), jmarchel@7bulls.com (J. Marchel).

https://doi.org/10.1016/j.iot.2025.101557
Received 31 January 2025; Received in revised form 18 February 2025; Accepted 23 February 2025
Available online 3 March 2025
2542-6605/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

on: (i) *Functional Properties*, i.e., properties of the essential actions and capabilities of the IoT system. They describe what the system does, including its ability to sense environmental inputs, process data, and communicate effectively, focusing, for instance, on device connectivity, data transmission, and the automation of actions based on specific inputs. (ii) *Non-Functional Properties:*, i.e., properties that define the performance and quality of the IoT system's operations. They target system characteristics such as performance, scalability, reliability, usability, and security.

A monitoring engine observes, detects, and responds to specific occurrences or changes (events) from several systems and system components (including sensors) within a defined environment in real-time or near real-time. These events include user actions, sensor data changes, system performance thresholds, or external triggers. The system's behavioral complex patterns are inferred upon detecting an event, each linked to distinct functional and non-functional characteristics (represented through specific monitoring rules). For the complex event analysis, a processor is typically included in the monitor architecture [3–5]. This processor is responsible for assessing the monitoring rules during monitoring execution to initiate predefined actions, such as logging data, sending alerts, or activating automated responses. Event-based monitoring systems are commonly used in IoT to ensure timely decision-making and efficient operations. Therefore, monitoring activities can be summarized as gathering and analyzing data from various sources (such as sensors, software, and hardware components or devices), evaluating the IOT's functional and non-functional properties, identifying property violations, raising specific alarms, and, if required, actuating countermeasures.

In literature, comparable approaches for identifying or forecasting run-time violations or vulnerabilities can be classified into three groups [1] according to the type of data used in their activities: *Using (previous) knowledge*: In this case, previous data collections or past examples of failures are used to predict the output's quality. The proposals are usually based on either deep learning approaches as in [6] or rely on a separate system to monitor and predict a target model's failure (as in [7], or on perceptions system [8] or methods for prediction learning (as in [9]).

Using input data: This group includes monitoring methods based on the analysis of the stream of input data coming from different sources, like, for instance, sensors, components, devices, systems, or models (as in [10–12]).

Using confidence estimations: In this case, confidence learning and uncertainty estimation are used for the output evaluation (as in [5,13,14]).

According to the Reactive manifesto principles,[1] the proposal of this paper belongs to the second group by providing an integrated Runtime Monitoring solution called *CONCERN: COmplex eveNt proCEssing monitoR iNfrastructure* focusing on *Responsiveness* for guaranteeing a consistent quality of services; *Resilience* for managing the possible execution exceptions and interruptions and providing a high system availability; *Elasticity* for allowing the use of several Complex Event Processors (CEPs) and channels of communication and avoiding central bottlenecks; *Message driven* for guaranteeing asynchronous and loosely coupled messages between components involved in the evaluation.

Consequently, CONCERN implementation provides features for:

- managing heterogeneity of events from different contexts and environments to improve portability and adaptability;
- managing different Complex Event processors (CEP) and multiple data storage to guarantee high performance and availability;
- communicating through different interfaces based on sockets and messages to improve interoperability and integration;
- optimizing message processing to improve the quality of services;
- integrating the use of smart agents for functional and non-functional properties detection and violation reactions to improve sustainability and responsiveness.

The paper aims to present the technological implementation of the features mentioned above that are helpful in monitoring and assessing the behavior of systems, components, or devices from functional and non-functional properties. We therefore describe the CONCERN supporting architecture and proof-of-concept implementation of its components.

The use of CONCERN in test-bed, simulation and real scenarios taken from the financial domain provides validation of the CONCERN usability and its performance. The paper is structured as follows: Section 3 presents its novelty and motivations. Section 5 describes CONCERN architecture and its components. Section 6.2 shows CONCERN test-bed simulation, while Section 7 presents the case study used for CONCERN performance evaluations. Section 8 describes results obtained and lessons learned. Section 2 provides an overview of related work, while Section 9 concludes the paper and discusses future works.

2. Related work

Researchers have investigated the adoption of monitoring solutions across multiple domains, including smart cities [15,16], industrial automation [17], and healthcare monitoring [18]. However, the analysis of real-time events, coming from various sources, such as sensors, log files, or transactional systems, and with important data volume makes monitoring an essential solution for mission-critical IoT systems [19–21]. For instance, to analyze patterns useful to identify abnormal air conditioning behavior [22], for reliable data analysis in healthcare systems [23] or for detecting specific faults or vulnerabilities and ensuring IOT reliability [24]. Since the first conceptualization of the monitoring system, several proposals have been presented in the literature for targeting the different monitoring realization challenges [20,21]. Commercial monitoring solutions are currently popular due to their enterprise-

[1] https://www.reactivemanifesto.org/.

grade features, user-friendly interfaces, dedicated support, and optimized performance. However, their cost and complexity may limit their adoption in small and medium-sized industrial contexts and research fields. Focusing primarily on this latter group, Table 1 provides in its columns a list of the most important challenges (first column), a brief motivation of their importance in IoT domain (second column), and some of the current solutions for addressing the challenges.

Additionally, as cyber attacks continue to rise in frequency and sophistication in IoT, using effective monitoring solutions to reduce associated risks has become an important activity. Therefore, in parallel with the traditional CEP-based monitoring solutions (such as Apache Flink,[2] Siddhi[3] and the proposal of this paper), which are more focused on real-time threat detection and response, several, innovative technologies have emerged as promising options for targeting specific security issues, including Security Information and Event Management (SIEM) such as Watzu,[4] Security Orchestration, Automation, and Response (SOAR), eXtended Detection and Response (XDR), and Endpoint Detection and Response (EDR) [54].

SIEM systems play a pivotal role in cybersecurity by collecting, aggregating, and normalizing a vast array of data from various events that may indicate potential security breaches within an organization's infrastructure [55]. This comprehensive data collection gives organizations a centralized view of their security posture, facilitating quicker detection and response to threats.

In contrast, SOAR represents an evolution of conventional SIEM systems by incorporating orchestration, automation, and proactive response capabilities [56]. SOAR tools enable organizations to streamline their incident response processes, allowing for more dynamic interactions across all stages of incident management. This is particularly beneficial for addressing known and emerging threats in an increasingly complex environment. Meanwhile, XDR and EDR solutions can significantly enhance the analytical processes of SIEM systems. By providing additional context and advanced tools, XDR and EDR help refine the analysis performed within SIEM platforms, ultimately leading to more accurate threat detection and response strategies [57]. To support this intricate data processing, organizations often utilize a Data Lake to store the collected information [57]. This repository is invaluable for advanced forensic analysis, as it allows security teams to sift through historical data, uncover patterns, and conduct thorough investigations into incidents that have occurred. By leveraging these modern technologies, organizations can bolster their security frameworks and better protect against the ever-evolving landscape of cyber threats.

As discussed more in detail in Section 3.3 CEP-based monitoring solutions and SIEM. SOAR, XDR, and EDR should not be seen as alternatives but as integrated solutions for joining security mechanisms with correlation and real-time analysis to help organizations optimize their level of quality and trustworthiness.

3. Motivations and SoTA advancements

The following sections provide an overview of the monitoring peculiarities in the IoT domain and the CONCERN advancement of the State-of-the-Art.

3.1. CONCERN monitoring in IoT

Monitoring is critical in ensuring IoT systems' effectiveness, reliability, and security during development and operational phases. It provides the foundation for managing quality attributes, identifying risks, and mitigating vulnerabilities in complex IoT ecosystems [58]. In this context, CONCERN proposes itself as a practical, integrated approach that can leverage IoT development and quality and management in several aspects. Specifically, CONCERN can have a role in:

- *Quality Assurance and Assessment*: CONCERN plays a crucial role in continuously evaluating IoT systems' functional and non-functional features. This can ensure optimal performance, reliability, and efficiency during development and in real-time use. It also allows dynamic adaptations, like boosting energy efficiency in resource-limited IoT devices and improving efficiency and efficacy [42,43].
- *Risk Management*: By continuously monitoring IoT systems, CONCERN can identify and mitigate development risks before deployment. The ability to detect or predict hardware failures, software bugs, or misconfigurations enhances IoT efficiency and helps prevent breakdowns [4,59–61].
- *Security Threat Detection*: By interacting with a smart prediction mechanism or the Digital Twins solution, CONCERN can proactively identify vulnerability threats, such as unauthorized access, abnormal traffic patterns, or potential cyberattacks [12,62,63]. It can also collect live operational data that is useful for enabling simulation and predictive analysis or improving decision-making. Finally, utilizing secure communication, encryption integrity, and access logs can enhance the trustworthiness of CONCERN's analyses and interactions.
- *Performance Optimization*: CONCERN can monitor IoT operations and track resource usages, such as CPU, memory, and bandwidth while computing performance metrics like latency, throughput, response time, and load balancing.
- *Ensuring Reliability During Online Execution*: CONCERN can detect and address anomalies during live IoT system executions, implementing redundancy or recovery strategies to enhance fault tolerance. It can also predict and mitigate security breaches in IoT systems before they occur.

[2] https://flink.apache.org/.
[3] https://siddhi.io/.
[4] https://wazuh.com/.

Table 1
Challenges, Descriptions, and Solutions for IoT with Complex Event Processing (CEP).

Challenge	Description	Solution
Data Volume, Velocity, and Variety	Large volumes of heterogeneous IoT data require managing diverse formats, protocols, and metadata, with real-time processing speeds.	Use of distributed stream processing frameworks like Apache Spark [25], Apache Flink [26], Apache Storm [27], Apache ActiveMQ Artemis[a] and Esper [28].
Latency and Real-Time Processing	IoT applications demand low-latency responses to identify complex patterns in real-time, even with high-speed data streams.	Traffic management, congestion control, load balancing [29], and frameworks like AWS IoT Greengrass[b] or Azure IoT Edge[c].
Scalability	Handling thousands or millions of devices requires scaling both horizontally and vertically without performance loss.	Platforms such as AWS Lambda[d] and Azure Stream Analytics[e] can be adopted.
Usage of innovative technologies	Integration of modern AI or ML pattern recognition of anomaly detection, prediction capabilities, Digital Twins, and adoption of ontologies and structured language for F&-NF properties.	Available solutions include [30–36].
Integration with IoT Protocols	Compatibility with IoT protocols (e.g., MQTT[f], CoAP[g], Zigbee [37], BLE [38]) for seamless data ingestion.	Adopt protocol translators, IoT gateways, or middleware tools like Eclipse Mosquitto or Apache ActiveMQ Artemis for protocol interoperability. Adopt protocol translators, IoT gateways, and messages Brokers (like Apache ActiveMQ Artemis) or provide RESTful API and services.
Data Collection and Instrumentation	Effectiveness and Efficacy of IOT monitoring requires accurate, efficient, and reliable data collection and processing.	Possible mechanisms for data collection are: event-based mechanism, agents, publish–subscribe models, aggregated device data, network traffic, edge analytics, and digital twins. Available solutions include [39–43].
Behavioral specification	Ensures a robust, adaptable, and aligned with business needs monitoring activity, particularly dynamic and large-scale IoT environments.	Model-driven specification like BPML [44], Petri Nets [45] or use of Smart Agent and DT [35,46] can be used for behavioral specification and enabling automation and fostering collaboration.
Event management, correlation and context awareness	Managing and Correlating events from multiple devices with context (e.g., location, time) to detect meaningful patterns in noisy data.	Use of temporal logic, sliding windows, and sequence analysis to detect intricate event patterns [20] and query languages like SiddhiQL[h] or Esper [28], DROOLS[i]. [3–5,12,47–51].
Security and Privacy	Protecting data and ensuring privacy, despite increased computational load from encryption and secure protocols.	Apply lightweight encryption algorithms and secure communication standards like TLS/DTLS while ensuring compliance with regulations (GDPR[j]).
Distributed Processing	Distributing processing across edge, gateway, and cloud while maintaining consistency and minimizing data transfers.	Implement hybrid architectures using frameworks like Apache NiFi[k] to balance load and maintain consistency [52].
Complexity of Event Rules	Defining and managing complex rules for diverse IoT applications, complicating rapid implementation.	Provide user-friendly interfaces for rule definition (e.g., Node-RED) and support for rule creation and management [30,53].
Energy Efficiency	When using battery-powered devices or energy harvesting systems, CEP engines must minimize power consumption while maintaining performance.	Reduce high data volume and real-time processing demands, especially in limited power-devices, by improving latency, scalability, and dynamic network topologies.

[a] https://activemq.apache.org/components/artemis/.
[b] https://docs.aws.amazon.com/greengrass/.
[c] https://azure.microsoft.com/it-it/products/iot-edge.
[d] https://aws.amazon.com/it/lambda/.
[e] https://azure.microsoft.com/en-us/products/stream-analytics.
[f] Message Queuing Telemetry Transport.
[g] Constrained Application Protocol - and UDP-based application protocol.
[h] https://siddhi.io/.
[i] https://www.drools.org/.
[j] https://eur-lex.europa.eu/eli/reg/2016/679.
[k] https://nifi.apache.org/.

3.2. CONCERN addressed challenges

Considering the challenges in Table 1, the CONCERN proposal integrates or leverages several suggested solutions. In particular, considering:

- *Data Volume, Velocity, and Variety*: CONCERN manages the heterogeneous IoT data using the Apache Artemis processing framework because it can guarantee low-latency, high-throughput messaging in event-driven systems like IoT, financial transactions, and real-time alerts. Additionally, it relies on mechanisms for dynamic data loading and reconfiguration and Apache ActiveMQ Artemis as a baseline format for managing the events.
- *Latency and Real-Time Processing*: Even if CONCERN adopts state-of-the-art mechanisms for traffic and transmission management, specific attention is devoted to improving the response times (see Section 6.1). Additionally, the use of smart agents mitigates packet loss or corruption (see Section 4.1).
- *Usage of innovative technologies*: CONCERN can use Ontology models to manage IoT knowledge. In particular, ontology models help select functional and non-functional properties to be monitored and provide guidelines for their parameter specifications [36]. Additionally, CONCERN has been conceived to be integrated with the prediction and simulation mechanism based on Digital Twin or Machine learning. This feature is handy for critical IoT systems having strong constrain in trustworthiness [64,65].
- *Integration with IoT Protocols*: As detailed more in Section 5, CONCERN incorporates an internal mechanism for guaranteeing interoperability between m JMS and JSON messages and seamless communication transmitted via Apache ActiveMQ Artemis or through its exposed REST interface (see infrastructure shown in Fig. 4). Additionally, the use of Apache ActiveMQ Artemis offers built-in support for *MQTT*, *AMQP* (Advanced Message Queuing Protocol), Zigbee, and *STOMP* (Simple Text Oriented Messaging Protocol) and ensures compatibility with a wide range of IoT protocols.
- *Data Collection and Instrumentation*: CONCERN relies on an event-based mechanism for collecting data from different IoT components and sensors and better-managing streams and clouds of messages. It also uses *Smart Agents* data collection for implementing local data filtering and sensor fusion [41–43]. Finally, reliable channels, flow control, and data acknowledgment mitigate the risk of data loss (see Section 4.1.)
- *Behavioral specification*: CONCERN can use different kinds of behavioral specifications (like BPML and Ontology models) [36]. Additionally, it can also integrate prediction derived from Smart Agent and DT execution. [35].
- *Event management, correlation, and context awareness*: CONCERN relies on the event-based mechanism. It uses the DROOLS and ESPER languages to manage the complex event processing, sliding windows, and sequences [3–5,12]. Finally, to reduce noise in data threshold-based filtering, pre-process data event management can be applied (see Section 4.1).
  The choice has been due to a compromise between having stateful event correlation, rule management, complex decision-making, and better flexibility and adaptability.
- *Security and Privacy*: CONCERN communications are secured using SSL connections. Integrating GDPR standards and encryption algorithms is part of future work.
- *Complexity of Event Rules*: CONCERN integrates an ontology-based approach [36]. For defining and managing complex rules. These automatically translate into the internal DROOLS language to support rapid monitor instantiation. A dedicated user-friendly interface will be included in future work.
- *Energy Efficiency*: The CONCERN activity and resource consumption can be dynamically adjusted through the online configuration of smart agents (see Section 4.1). It allows efficient resource management, including energy consumption. Additionally, CONCERN improves energy efficiency by putting in place strategies such as (i) event filtering and aggregation for reducing redundant data; (ii) use of efficient communication protocols such as MQTT and CoAP for minimizing power-hungry transmissions [66]; iii) Implementing an adaptive CEP Rules management focused on context-aware properties are useful for reducing unnecessary computations. Finally, the CONCERN core components can be deployed on ARM-based devices (e.g., Raspberry Pi) or hardware and software systems with limited computational resources (e.g., dual-core processors with 4 GB of RAM) [12,66].

The *Scalability* and *Distributed Processing* challenges of Table 1 have been addressed by implementing in CONCERN state-of-the-art mechanisms. The choice was a compromise between improving the CONCERN's performance and guaranteeing its portability, independence from specific and proprietary platforms, and adaptability to different contexts. In particular, CONCERN data streaming has been implemented using Smart Agent, supported by an internal mechanism for message queuing. More details are provided in Section 5.

3.3. Leveraging existing solutions

As anticipated in Section 2, CEP-based monitoring often works with SIEM, SOAR, XDR, and EDR by leveraging them to provide real-time event correlation, reducing false positives and enabling faster response times. Integrating CEP with these security solutions enhances detection accuracy, incident response efficiency, and proactive threat mitigation, positively impacting the organization's quality and trustworthiness. This section discusses existing monitoring solutions, such as XDR, EDR, and SIEM, and the main CONCERN improvements.

Leveraging the XDR. Considering the XDR, the main CONCERN advancement is in leveraging the mechanism of the data lake to make it more suitable for prompt analysis and reactions. XDR uses a comprehensive approach based on a data-lake paradigm for detection and response. It collects and secures data on activities on multiple levels and provides automated analysis of this data to detect threats. As a result, security analysts are equipped to conduct deeper investigations and adopt more rapid responses. CONCERN leverages the multiple levels of automated data analysis of the XDR by exploiting a federated set of Complex Event Processors (CEPs). Indeed, each CEP is dedicated to either a specific threat of functional or non-functional property verification or to analyzing a particular data type. The CEPs are strongly interconnected and able to exchange complex events. This allows evaluation to be executed faster than XDR to speed up the detection of emerging threats and possible malicious behavior.

Leveraging the EDR. Considering the EDRs, the main CONCERN advancement is leveraging countermeasures management by providing a more flexible and customizable mitigation approach. Indeed, CONCERN lets the users define their countermeasures to avoid stopping the runtime activities as a unique alternative. These can be associated with being launched by CONCERN during the monitoring activity in case of violation detection. In the current implementation of CONCERN, the following countermeasures have been provided, including (ordered by increasing complexity).

1. Notify a specific message in which logs and data of the detected violation are reported;
2. Stop the execution of the component or device responsible for the violation;
3. Execute a mitigation action to return the system to a safe condition using Smart Agents and specific artifacts.

Leveraging the SIEM. Considering the SIEM solutions, the main CONCERN advancement is leveraging the mechanisms for gathering the events generated by the monitored system or component for storage and post-analysis activity and online validation of possible functional and non-functional property violations. Indeed, CONCERN exploits this knowledge to evaluate ongoing behavior and detect and notify threats promptly.

Additionally, by incorporating various data sources, CONCERN enhances existing monitoring solutions analysis and event correlation. CONCERN allows the integration of data streams from endpoints, networks, cloud environments, IoT devices, and applications. Furthermore, the ability to scale the CONCERN framework across different infrastructures and environments supports its adoption in various operational scenarios. Finally, by integrating Digital Twin for smart prediction [67] and the possibility of enabling AI/ML capabilities, CONCERN's ability can be improved to identify misbehavior and anomalies and detect both known and unknown threats.

4. Monitoring basic concepts

As anticipated in the Introduction (Fig. 1), during the monitoring activity, the systems, the components, and the environment produce different kinds of events that need to be collected, filtered, and aggregated to identify unusual behaviors and guarantee the system's reliability or performance [68–70].

An event in a monitoring system represents a significant atomic action, state, or change detected within the system or a monitored component. It can indicate normal activity (e.g., completion of an operation) or an anomaly (e.g., an error or a threshold violation). Usually, these actions or state-change detections are encapsulated in an object/message to be notified to the monitoring platform. Indeed, the events are the most commonly used data during the monitoring activity; consequently, the publish/subscribe communications paradigm is the most adopted in the robust, adaptable, and generic monitoring architecture [71].

CONCERN models the functional and non-functional properties to be monitored through rules. These are the sequences of expected events, expressed in a specific language, representing the correct behavior of the monitored entities. Fig. 1, depicts the conceptual model of the CONCERN execution that includes:

1. Event Producer: it collects events from the different IoT entities (system, components, sensors, or devices);
2. Complex Event Processor: It uses the events to evaluate the rules (i.e., the functional and non-functional properties) and detect violations or misbehavior;
3. Event Consumer: promptly raises alarms or executes specific recovery action.

The following sections provide more details about the CONCERN basic concepts.

4.1. Probes and smart agents

Event data collection is usually implemented through a mechanism called *Probes* in a monitoring system. A *Probe* is a low-intrusive software or hardware artifact that collects information from the system, network, or application on which it is deployed. Probes are deployed to gather data useful for assessing performance, availability, latency, or resource usage. They can actively simulate user interactions (active monitoring) or observe ongoing processes (passive monitoring) to ensure systems function as expected. Probes can be implemented through the publish/subscribe communications paradigm over TCP when available for reliable transmission. They can be realized using several techniques, including black-box or white-box solutions, depending on the required access to the internal structure of the monitored entity. Data are encapsulated in a structured format described in Section 4.2 that provides checksum facilities for error detection and data integrity.

The evolution of the *Probe* is called *Smart agent*. The *Smart agent* is a software component injected within the IoT system and represents an active and adaptable element in monitoring. It can collect data useful to the monitoring platform for assessing

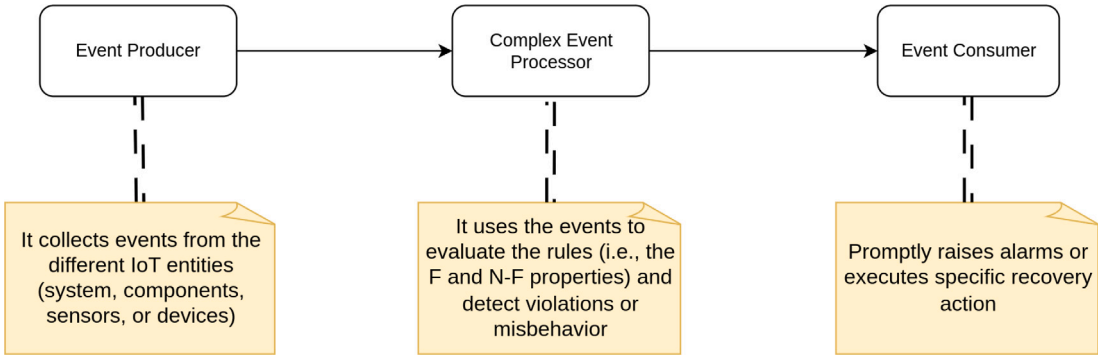

Fig. 1. CONCERN overview.

properties (like basic *Probe*), autonomously execute actions useful for improving the IOT quality, mitigating possible risks, or changing its behavior according to the notification received by the monitoring platform. Smart Agent can also help mitigate data noise risks by locally filtering the data stream or discarding duplicate or incorrect information. For instance, if the network traffic is higher than normal, the monitoring system can order the *Smart agent* to reduce the sampling interval of the event it uses to minimize the network congestion and improve resource consumption. Another example is in case of a property violation. Monitor platform can order *Smart agent* to promptly execute a mitigation countermeasure directly to the IoT system to reduce the reaction time and safeguard the overall system activities.

Fig. 2 conceptually schematizes, using the BPM (Business Process Management) [72], a formal representation of the possible processes implemented in CONCERN for data collection through Probes artifacts. As in the figure, when an entity needs to be instrumented, four different instrumentation types can be selected:

- *Input/Output capture*: This less invasive methodology follows a black-box approach: the entity captures its input and output on a socket or channel and sends it as raw data to the CONCERN. This last packets the messages and translates them in the proper event format;
- *Message-Based notification*: the entity providers instrument it with the required mechanisms to send the messages containing the events. In this case, the messages must be structured according to the CONCERN predefined format and protocol as detailed in Section 4.2;
- *CONCERN probe/smart agent*: the entity is instrumented using either CONCERN pre-build probes or the CONCERN Smart Agent. In the first case, the probe is a Java implementation composed of a jar library that allows the event-message notifications according to the CONCERN format and protocol. Alternatively, the CONCERN Smart Agents are the evolution of the CONCERN Probes. They allow local filtering criteria on the events sent to the monitoring component, enable traffic reduction, and optimize the event transmission by aggregating them. Additionally, CONCERN Smart Agents expose a port/socket enabling monitoring countermeasures. These can be predefined or, on the fly, established by the monitoring component according to the forecasted events. For example, in case of overload, Smart Agents can trigger the monitored entity to reduce the frequency of some actions or to disable some non-critical processes. To reduce the risks of a possible backdoor, the communication to and from CONCERN probes and Smart Agents are implemented according to cybersecurity best practices;
- *Automatic instrumentation*: An external service is used to instrument the monitored entity. This solution wraps the entity and triggers actions when specific events occur. In such a situation, overload risks should be evaluated to avoid bias in performances or communication delays.

Additionally, the different approaches for entity instrumentation contribute to reducing CONCERN intrusiveness and increasing its compatibility, especially when adopted in constrained environments. Finally, specific *Instrumentation guidelines* are available to better guide the entity's instrumentation to be monitored. In Fig. 2, the guidelines are represented by the top left document connected to the activities through dashed lines.

4.2. Event message and session

For general purposes, service-based architectures or micro-services rely on a communication pattern based on a request–response protocol. This approach fits with services that behave like websites or interfaces where we trigger an action and wait for something to happen or a result to come back. The approach must change in the heterogeneous world, where multiple, growing, and independent services are developed with different behaviors and subsystems deployed and interacting. A possible solution is to ensure that the services and systems they are running rely on strict SLA; another is to break the synchronous request–response paradigm and move to an asynchronous message paradigm. The proposed monitoring framework is based on event-message communication, in which events represent facts in the system under test or notifications generated by the monitoring itself. Events are essential for loose coupling and managing facts following a stream or replicating and using it for concurrent analysis [73].

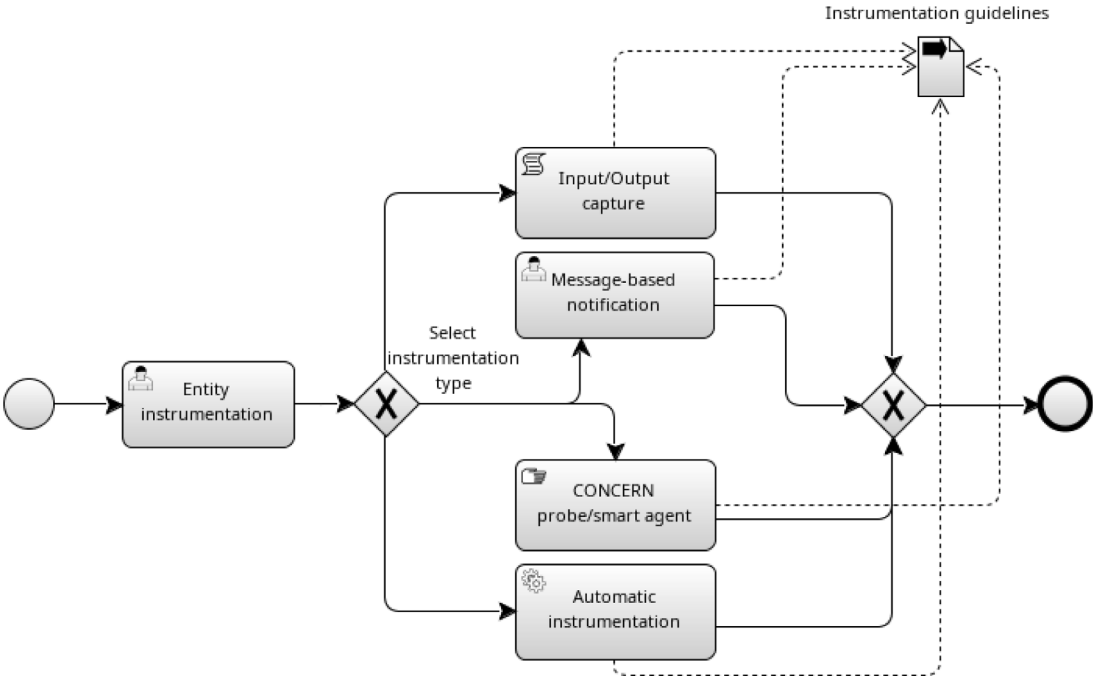

Fig. 2. CONCERN probing techniques.

Fig. 3 shows the CONCERN event structure used for managing facts and notifications in a class diagram representation. In particular, the Figure depicts on the left side the generic *Event* object(in green), in the middle the interface, and on the right side the event implementation ready for instantiation (*ConcernAbstractEvent⟨T⟩* in yellow). This last object has generic data parameters (*⟨T⟩*) expressed as Java features. This class defines the minimum subset of information needed for the CONCERN Complex Event Processor execution that is:

- *Timestamp:* for storing the instant in which the event is captured;
- *sender:* for storing the name or identifier of the event sender;
- *destinationID:* for identifying the destination of the message (i.e., the specific monitoring instance);
- *sessionID:* for identifying a specific monitoring session in case of multiple parallel executions;
- *checksum:* for checking the integrity of the data field;
- *name:* for storing the name of the event;
- *data:* for storing the concrete payload of the event;
- *cepType:* for addressing the event to a specific CEP of a monitoring system.

4.3. Rules

Complex Event processing is based on searching and detecting patterns of events to infer more complex events, emerging trends, or potential threats [74]. For managing event patterns and aggregating low-level events (simple events) into complex (high-level) events, it is essential to specify their *Rule*, i.e., the sequence of expected events and the timing at which they will occur [33]. Rules can be specified according to a specific language or, more generally, using ontologies or generic language-independent formats like the proposals presented in [36,75].

As reported in listing 1, usually, at the abstract level, the rule is composed of four main sections:

1. *Variable declaration:* where variables are declared;
2. *Attributes:* where attributes are declared and variable instantiated;
3. *Conditions:* where the conditions to be satisfied from detected events are declared;
4. *Actions:* where the actions to be executed when the condition is satisfied are described.

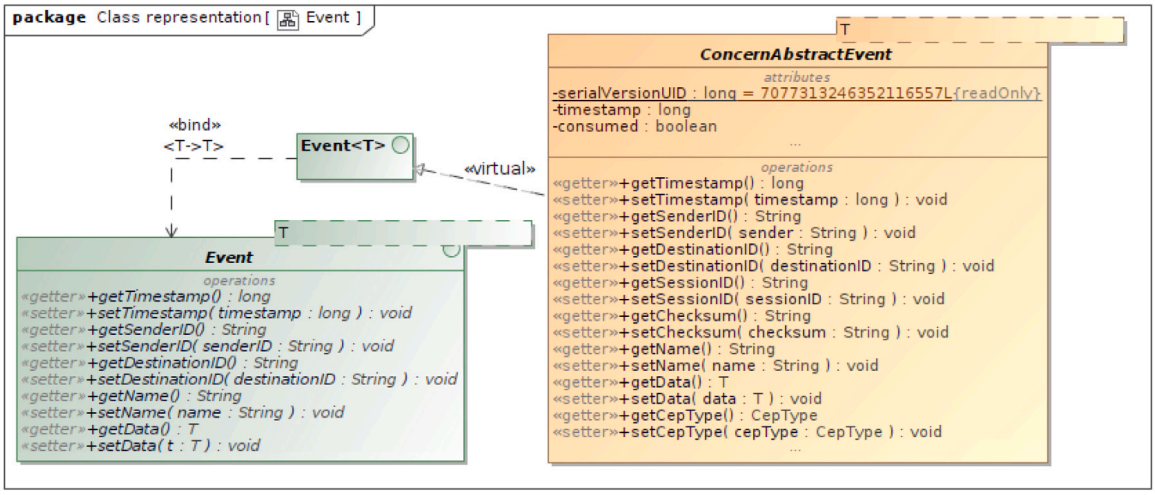

Fig. 3. CONCERN event representation.

```
declare   //Variable declaration
rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
```

Listing 1: Abstract rule rule template

As detailed more in Section 5, to better satisfy the challenges listed in Table 1 and increase interoperability CONCERN implementation implements two Complex Event Processor (CEP) engines: the former based on ESPER[5] specification language, and the later on DROOLS.[6] The following listings report two examples of rules developed in DROOLS. Specifically, listing 2 shows a *Simple Rules* able to assess if the event named *EVENTNAME_A* contains a value that is greater than a certain threshold. This rule is defined using only basic logic operators. In listing 3, the rule assesses a sequence of events. i.e., an event named *EVENTNAME_A* that contains a Data value greater than a certain threshold is followed by an event named *EVENTNAME_B* with a value in the field Data equal to another threshold. As in the listing, the Rule is a composition of *Simple Rules*.

```
rule "check value threshold"
    no-loop
    salience 20
    dialect "java"
when
    $aEvent : ConcernBaseEvent(
                        this.isConsumed == false,
                        this.getEventName == "EVENTNAME_A",
                        this.getData > #THRESHOLD# );
then
    System.out.println("Failure detected");
end
```

Listing 2: Simple Rule example

```
rule "check values"
    no-loop
    salience 20
    dialect "java"
when
    $aEvent : ConcernBaseEvent(
```

[5] https://www.espertech.com/esper/.
[6] https://www.drools.org.

```
7                   this.isConsumed == false,
8                           this.getEventName == "EVENTNAME_A",
9                           this.getData > #THRESHOLD# );
10
11      $bEvent : ConcernBaseEvent(
12                          this.isConsumed == false,
13                          this.getEventName == "EVENTNAME_B",
14                          this.getData = #EQUALVALUE# );
15  then
16      System.out.println("Failure detected");
17  end
```

Listing 3: Complex Rule example

More examples of rules, instantiated for actual monitoring patterns, are provided in Section 7.1. This has been used for monitor validation in the reported industrial system.

5. CONCERN architecture

As reported in Fig. 4, the CONCERN behavior is realized by independent modules that can be enacted according to the desired configuration or operational profile. All the CONCERN components can be instantiated as stand-alone nodes or containers on top of a Docker[7] architecture. This choice provides a more substantial decoupling and lets the components deploy on different machines to improve resource management.

In detail, the CONCERN architecture includes:

Main. that is the launcher of the architecture. It manages the execution of the components required for a specific profile in the correct order. The project and the dependencies between libraries are managed through Maven,[8] i.e., the message broker embedded into the system is *Apache ActiveMQ Artemis*[9] (an open-source multi-protocol Java-based message broker); for the interface that exposes the interaction with REST interfaces, through Mosquitto.[10]

Rest. This component exposes a REST interface that allows CONCERN to receive commands through POST messages and a webpage on which the status of the runtime monitoring is shown 7. The POST messages can be used for *Start, Stop* the monitoring engine, or for executing more specific actions during the execution time like *Load Rules, Delete rules* for interaction with the Complex Event Processor. There are also GET calls available that allow browsing the database containing rules (*Browse RuleDB*) or reading the notifications generated by the CEP: *Notification* method.

These calls are exposed through a Glassfish Grizzly Server,[11] and access to the exposed functionalities is provided through a basic token authentication mechanism. However, for the CONCERN execution, running the *Rest* component is not mandatory.

Logger. It provides common logging facilities across the CONCERN components. It is implemented using Log4j.[12]

Broker. The execution of this component is optional. It executes an internal instance of a message broker and provides facilities for sending messages (events and requests) between the artifacts involved in a monitoring session. Depending on the technology used for the communication, two solutions can be supported (also at the same time):

- using a configuration that runs an Apache ActiveMQ Artemis instance. In this case, the JMS Message Broker is exposed on port 61 616. The instance execution is managed following the pattern Singleton, and the artifact is included using maven dependencies in the Project Object Model (POM) of the project;
- using *Mosquitto*, i.e., an open-source message broker that implements *MQTT* protocol, allows light transmission between low-power devices using publish/subscribe messaging techniques.

Register. This component traces and stores all the information related to the connections between the internal CONCERN components and external artifacts involved in a monitoring session. For instance, the *Register* collects the list of the Complex Event Processor instance executed, the channel used for listening for events, and the parameters they use. This information is collected using the object *TopicAndProperties* as detailed in Fig. 5 As in the Figure, the class provides the information needed for connecting to a specific channel, routing a message or a request to the channel on which a dedicated Complex Event Processor instance is running.

[7] https://hub.docker.com/r/acalabro/concern.
[8] https://maven.apache.org/.
[9] https://activemq.apache.org/.
[10] https://mosquitto.org/.
[11] https://javaee.github.io/grizzly/.
[12] https://logging.apache.org/log4j/2.x/.

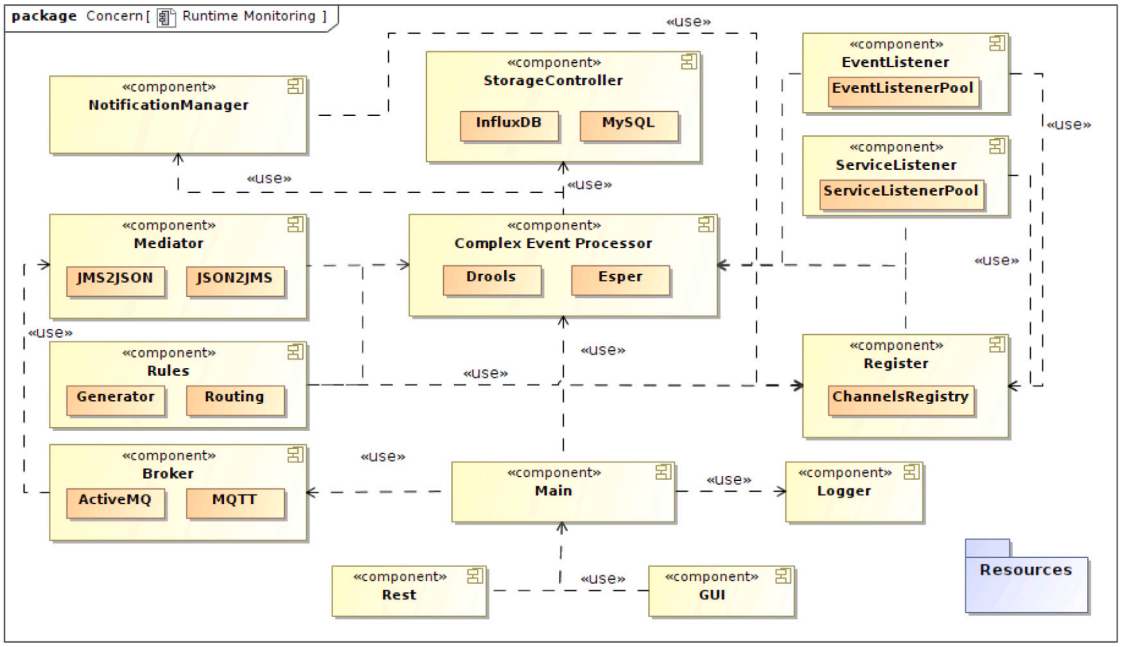

Fig. 4. CONCERN components.

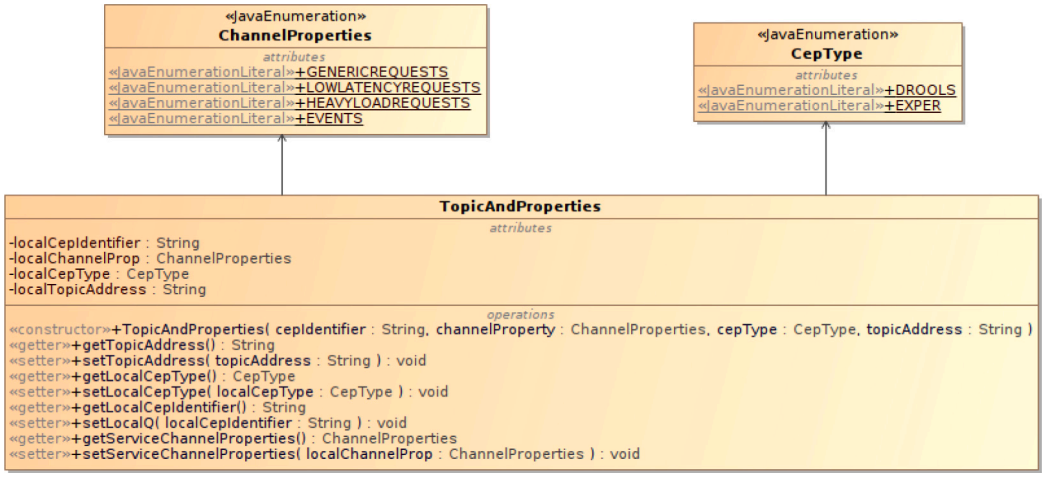

Fig. 5. Topic and Properties class diagram.

EventListener and ServiceListener:. These two components manage and guarantee the scalability of the communication layer of the overall monitoring architecture proposed. They rely on a basic structure of thread poll, capable of deploying more resources in channels and threads when the system faces an increasing number of requests or different topics. The strategies for handling the increasing number of requests rely on a simple auto-scaling algorithm that looks at the queue size.

Each thread listens on a specific channel:

- the event messages generated by probes (*EventListener Task*);
- the requests for the evaluation of one or more rules on (generic or specific) Complex Event Processor (*ServiceListener Task*).

Complex event processor. The Complex Event Processor manages the automatic rule execution and assessment. It can be executed considering two different engines in parallel: *Drools* or *Esper*. In particular, Drools is an engine based on the RETE algorithm [76], based on analysis executed on forward and backward chains. It can be instructed through a specific Domain-Rule-Language. Drools guarantee high speed and scalability thanks to the rule engine engendered by the RETE algorithm (RETE-OO). Esper uses the Event Processing Language rule language that implements and extends the SQL-standard language. Usually, the Esper engine has more

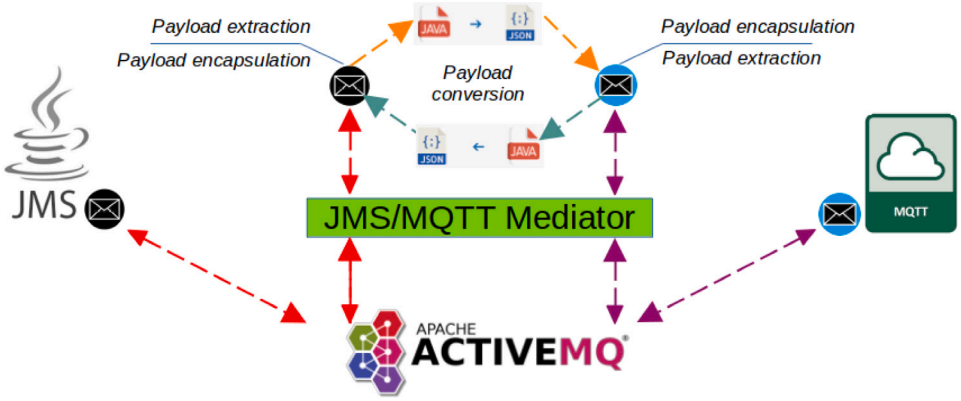

Fig. 6. JSON2MQTT Mediator behavior.

performant memory management than Drools, which relies on a garbage collector that is not performant in the case of millions of events. In the proposed complex event processing solution, the development relies on Java technologies (OpenJDK 19[13]) for the core functionalities and Drools engine for the main Complex Event Processor.

Rules. This component enacts rules generation or the self-injection process when meta-rules are invoked during the Runtime Monitoring execution. The *Rules* component is also in charge of routing the messages that contain rules to the target Complex Event Processor. It uses *CepType*, the parameter stored on the event, and the analysis of the *ActiveCep* list stored in the *Register*.

Mediator. The *Mediator* component is the CONCERN solution to guarantee seamless interoperability between the *MQTT*[14] (Message Queuing Telemetry Transport) and *JMS* (Java Message Service) messaging protocols. The *Mediator* component has two roles: (i) it converts *MQTT JSON* (JavaScript Object Notation) message format into the *JMS* one used when employing *Apache ActiveMQ Artemis* broker. In this case, the *Mediator* encapsulates the payload into a structured JMS event and precisely injects it into a designated channel. (ii) it converts JMS messages containing critical events into JSON format, enabling flawless interaction with entities that do not support the JMS messaging paradigm. With the support of *Apache Artemis* as the JMS broker and *Mosquitto* as MQTT manager, the *Mediator* component acts as an intermediary, ensuring smooth message translation and enhancing communication efficiency. Its adaptability and precision make it a cornerstone for enabling cohesive operations between diverse components and external systems in the runtime monitoring environment.

In Fig. 6 the two main functionalities of the *Mediator* are depicted:

1. converting a JSON message into a JMS Message encapsulating payload into the event described in Section 4.2 and injecting it into one of the dedicated channels;
2. converting a JMS message containing the event described in Section 4.2 into JSON when a notification needs to be sent to an entity that does not support the JMS message paradigm.

StorageController. This component stores all the events and data generated by CONCERN. The *StorageController* allows (simultaneously) two implementations:

1. using a generic MySQL[15] server deployed outside Runtime Monitor to store simple events;
2. using a database structured for time series, such as InfluxDB,[16] allows for managing vast amounts of events faster.

An interface template to explore data has been developed using Grafana.[17]

Notification manager. This component manages the notification of failure sent by the Complex Event Processor. It forwards the failure notification to the specific channel, gathering the correct information (channel details) from the *ChannelRegistry* component.

GUI. The *GUI* component, shown in Fig. 7, has been developed as a basic web page and shows CONCERN logs, the status of each of its components, and the tools for managing the Complex Event Processor (CEP) Knowledge Base (KB) on which rules will be loaded.

[13] https://openjdk.org/.
[14] https://mqtt.org/.
[15] https://www.mysql.com/.
[16] https://www.influxdata.com/.
[17] https://grafana.com/.

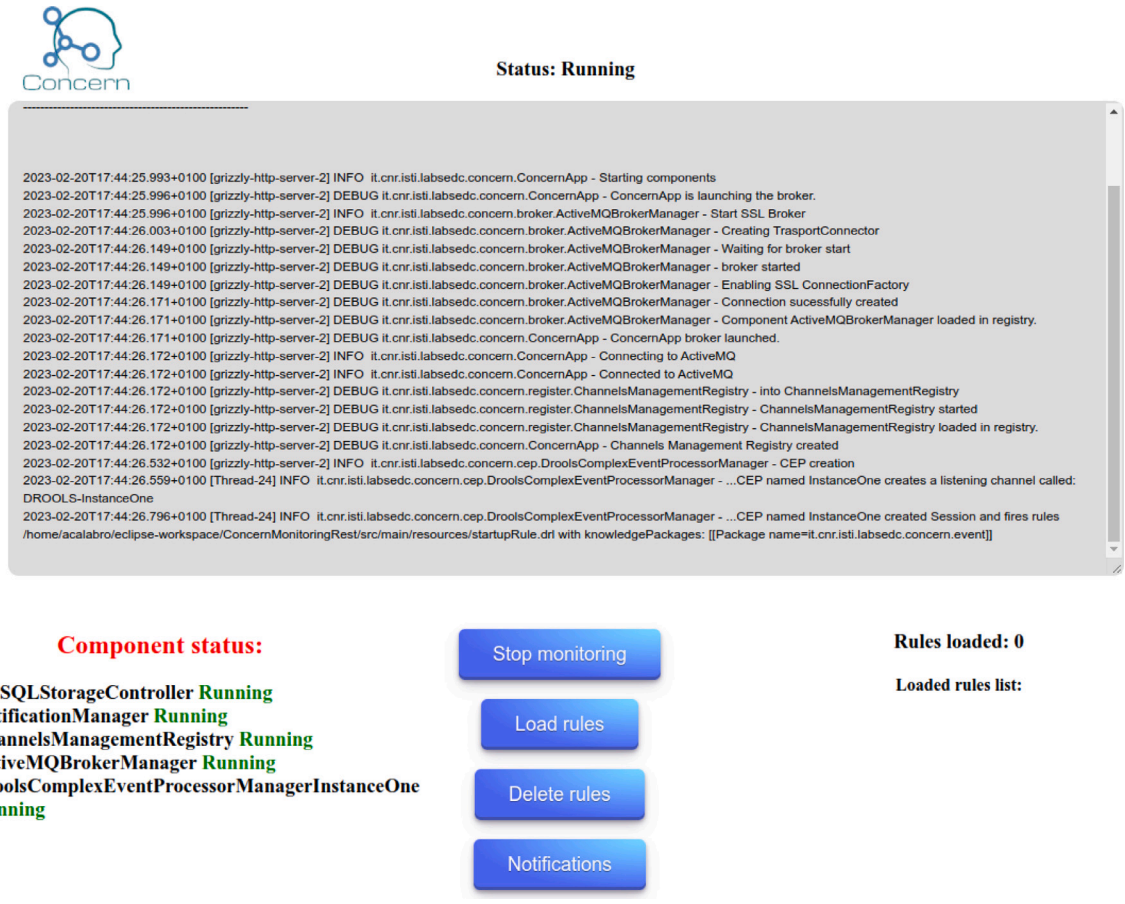

Fig. 7. CONCERN GUI.

In particular, the central text area provides different information about the execution. The left bottom corner shows the status of the main CONCERN components (like *MySQL Storage Controller* or *Notification Manager*). The central buttons are dedicated to managing the monitoring activities, and the bottom right corner shows data about the loaded rules. The monitor architecture presented in Fig. 4 has been developed referring reference architecture for even based-monitor (see [20,68–70] Indeed, CONCERN realizes and leverages all the components of the standard architecture (CEP, Storage manager, Rule manager), and adds specific ones for targeting specific monitoring challenges (such as Rule Generator for Smart prediction [64] or ontology integration) and improving the Monitoring performance, (such as the *Mediator* and *Service Listener* and *Event Listener* for manage and scale multiple events flows).

6. CONCERN in-vitro assessment

Three types of in-vitro assessment have been executed to evaluate the performance of the CONCERN implementation before its application in a real-world environment: inside a test bed for quality attributes assessment (Section 6.1) and inside an IoT simulation environment (Section 6.2). In the next sections, details of the two experiments are provided.

6.1. CONCERN quality attributes assessment

CONCERN performances have been assessed using the testbed presented in [65]. In particular, the evaluation targets the following properties: (1) the impact of loading simple vs. complex monitoring rules and (2) the capability to fire a rule.

It is out of the scope of this paper to provide details about the test bed development; we refer to [65] for more details. The purpose of this section is to summarize the final collected results.

As in [65] the metrics are defined as:

1. **Rules Loading Time**: the time between receiving and loading a rule. It is calculated with the formula shown in (1).

$$\Delta t_{loading} = t_1 - t_0 \tag{1}$$

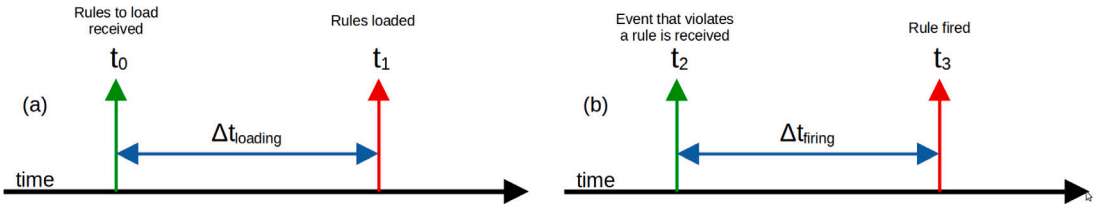

Fig. 8. (a) Rules loading metric and (b) Rules firing metric.

Table 2
Experimental execution results.

Experiment n°	Simple		Complex	
CONCERN State	Cold	Hot	Cold	Hot
$\Delta t_{loading}$	Lower: 397 ms Average: **418 ms** Higher: 439 ms St.Deviation: 14.8 ms	Lower: 52 ms Average: **69 ms** Higher: 86 ms St. Deviation: 12.5 ms	Lower: 499 ms Average: **685 ms** Higher: 944 ms St.Deviation: 120.9 ms	Lower: 208 ms Average: **239 ms** Higher: 268 ms St.Deviation: 22.15 ms
Δt_{firing}	Lower: 9 ms Average: **11 ms** Higher:14 ms St.Deviation: 1.67 ms		Lower: 9 ms Average: **12 ms** Higher: 16 ms St.Deviation: 2.44 ms	

where (t_0) is the time in which the rule is received by the *ServiceListenerPool* and (t_1) is the time in which the rule is loaded on a *Complex Event Processor*. Fig. 8(a), provides a graphical representation of the metric.

2. **Rules Firing Time**: the time between a violation detection and the notification sent. It is calculated with the formula in (2).

$$\Delta t_{firing} = t_3 - t_2 \tag{2}$$

where (t_2) is the time when a received event triggers a rule on the *Complex Event Processor* and (t_3) is the time in which notification of the violation is sent by the *NotificationManager*.
Fig. 8(b) provides a graphical metric representation.

Table 2 reports the average data collected during the CONCERN testing execution. In particular, results have been collected considering

- Complex Event Processor loading different sizes either of *Simple Rules* or of *Complex Rules* sets;
- Complex Event Processor in *Cold* state, i.e., CEP has not yet been initialized, or *Hot* state, i.e., CEP has already been set up and run.

The term *Simple Rules* identifies a set of rules to load on the *Complex Event Processor* that are defined using basic logic operators. The term *Complex Rule* identifies a set of rules to load on the *Complex Event Processor* that are derived by the composition of *Simple Rules*.

Cold startup involves initializing components from scratch, leading to longer times due to the need for full configuration, warm-up, and dependency setup. In contrast, *Hot* startup benefits from pre-initialized states, reduced dependency overhead, and residual readiness, resulting in faster component activation. In this case, the differences are related to the initialization of the working memory of the *Complex Event Processor* and all the related connected components for evaluating events and sending back evaluation notifications. The experimental results are reported in Table 2. The system runs on a machine executing Kali Linux OS 2023.03, which is equipped with an Intel i7-10610U CPU @ 1.80 GHz with SSD and 16 GB RAM. More details of the proposed experiment are described in [65]. As reported in Table 2, the time for loading and firing, the rules can sensibly change in different situations. However, the purpose of the experiment was to define the boundary conditions under which CONCERN can provide suitable performances without compromising the overall environment execution. Indeed, the data collected have been analyzed by the personnel in charge of the showcase execution reported in Section 7 and positively evaluated for the use of online use of CONCERN within their production line to avoid the creation of bottlenecks in their systems.

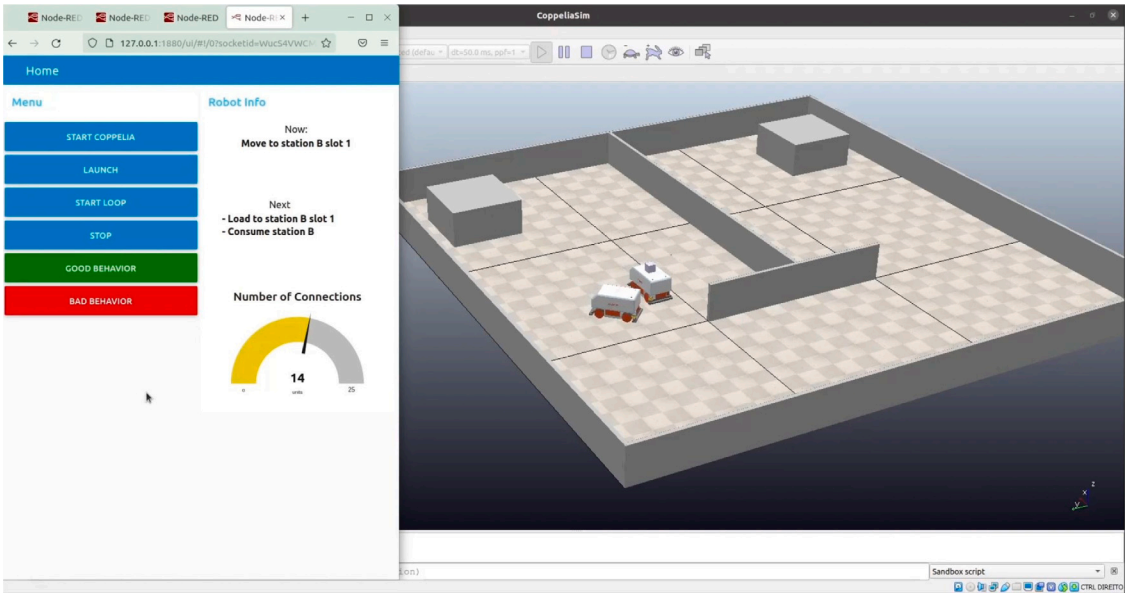

Fig. 9. CoppeliaSim simulation [36].

6.2. CONCERN in simulation environment

A specific CONCERN instance, which integrates an ontology manager for defining functional and non-functional properties, has been integrated into the simulation environment of a multi-robot autonomous navigation scenario [36] for the security assessment of its intralogistics.

In the experiment, the CoppeliaSim[18] has been used to settle up the simulation of two autonomous ground vehicles responsible for transporting material between two different workstations. The CONCERN system enables real-time monitoring and validation of the robot's operations and movements, ensuring its ability to safely and efficiently explore the area. For this proposal, a set of specific rules has been selected. For instance, considering the mutual interaction between the component managing the local logistics (called *Local Planner*) and the component in charge of autonomous navigation (called *Global Planner*), the following rules have been defined and implemented in the DROOLS language: *Maximum number of established simultaneous connections is 1*.

Referring to [36] for more details, the experiment included the simulation of malicious code injection through the UI of one of the robots. As shown in Fig. 9, this caused an increase in the connections between the Local planner and the Global planner. Consequently, the CONCERN instance observed the violation of the above-mentioned rule (*A maximum number of established simultaneous connections is 1*), immediately sent a notification message, and triggered the associate countermeasure. Once the system had been reconfigured, another round of CONCERN execution was performed, verifying the system returned to a safe and trusted condition.

Having the possibility of this dynamic monitoring activity was particularly crucial in this complex environment, where vehicles may encounter various obstacles and hazards, especially in the presence of people. CONCERN was a valid solution for early problem detection and swift corrective measures. This ensured that the vehicle maintained safe and trustworthy operation at all times.

6.3. CONCERN vs. existing solutions

This section summarizes the main results of the comparison between CONCERN, two existing commonly adopted open-source CEP monitors (Apache Flink and Siddhi), and an SIEM/XDR solution (Wazuh). Here, the main peculiarities of each of them are reported. We refer to the specific website for more documentation and details.

Apache flink. is an open-source monitoring framework designed for real-time data analytics. It enables high-throughput, low-latency processing of large-scale data streams, making it ideal for fraud detection, anomaly detection, and cybersecurity monitoring. It offers features for managing state efficiently across distributed nodes and ensuring reliable data processing. Flink is widely used in finance, cybersecurity, IoT, and AI applications, powering real-time decision-making and predictive analytics at scale.

[18] https://www.coppeliarobotics.com/.

Siddhi. is open-source monitoring for capturing, processing, and analyzing high-velocity data streams with low latency. Siddhi supports event correlation, filtering, aggregation, and pattern matching. It is a suitable solution for real-time analytics and IoT data processing applications. It provides an SQL-like query language for defining stream processing logic and seamlessly integrates with various data sources, messaging systems, and cloud platforms. With its lightweight and high-performance architecture, Siddhi is widely used in industries requiring real-time decision-making.

Wazuh. is an open-source security platform with threat detection, visibility, compliance, and incident response capabilities. It is widely used for SIEM (Security Information and Event Management) and XDR (Extended Detection and Response) functionalities, offering robust security monitoring across endpoints, cloud environments, and containers and providing a powerful interface for configuration, execution, and post-analysis. Wazuh offers features for logging suspicious behavior, detecting unauthorized changes to critical system files, and security weaknesses. It is a popular alternative to proprietary SIEM solutions due to its cost-effectiveness, scalability, and open-source nature.

For comparing the applications mentioned above, we consider the properties definition provided in ISO/IEC 25002:2024,[19] and specifically:

- **Functionality**, i.e., the set of provided functions and their specified properties. In particular, we considered the *Interoperability* and *Security*;
- **Reliability**, i.e., software's capability to maintain performance level under stated conditions for a stated time. In particular, we considered the *Fault tolerance* and *Recoverability*;
- **Usability**, i.e., the extent to which specified users can use a product to achieve specified goals.
- **Efficiency**, i.e., the attributes about the relationship between the software's performance level and the amount of resources used under stated conditions. In particular, we considered the *Time behavior* and *Resource Utilization*;
- **Maintainability**, i.e., the attributes about the effort needed to make specified modifications. In particular, we considered the *Testability*;
- **Portability**, i.e., the attributes of the software's ability to be transferred from one environment to another. In particular, we considered the *Installability*.

To compare the solutions mentioned above, we analyze the available documentation, examples, and specifications downloadable from each tool website. Wherever available, we also execute the provided test cases on different monitors. In this last case, to allow their execution, a suitable translation into the proper tool language was performed.

Thanks to the analysis and the results collected, the evidence reported in Table 3 has been derived. As a general consideration, the comparison highlighted that CONCERN is oriented to the research field or small and medium enterprise environments, while Siddhi and Flink also target the industrial context. Wazuh, an effective alternative when monitoring security, is the primary challenge. However, Flink and Wazuh pay for their performances and the cost of being suitable for expert users in environments with many required resources. Siddhi and CONCERN are suitable for basic-knowledgeable users because they provide either a user-friendly interface or ready-to-use code snipped. These considerations are also confirmed by the results obtained when evaluating the time behavior. Due to its simplicity, CONCERN can reduce around 30% the execution time of the Flink test suite while it provides similar performance when executed with the Siddhi test suite.

It is important to specify that for collecting the time behavior results, we executed the test cases available on the Flink and Siddhi websites on each of them, respectively. Then, to collect the CONCERN time behavior results, we translated the set of test cases into Drools language and executed the CONCERN implementation. No comparison has been performed with Watzu because, by construction, its response time has a different scale order. Indeed, Watzu was conceived to collect logs in batches and analyze them retrospectively; therefore, the response times are often not real-time. Conversely, a CEP monitor is designed for continuous, real-time data stream analysis and instant decision-making.

Due to their peculiarities, CONCERN, Flink, Wazuh, and Siddhi can be valid alternatives depending on the monitoring purposes and the required performance across key attributes. Indeed, Flink can be more suitable in real-time data processing, requiring high throughput and robust fault tolerance, such as big data and streaming analytics. Wazuh could be a key solution when no real-time security monitoring (e.g., fraud detection) is necessary and delays under heavy workloads are tolerated. Siddhi can provide the best performance in environments that require real-time, high-throughput, and low-latency event processing, even if it could require sufficient CPU, memory, and a well-optimized streaming data infrastructure. Finally, CONCERN could be a smart solution in case of limited resources and infrastructure support, such as edge devices or drones.

[19] https://www.iso.org/standard/78175.html.

Table 3

.

Attribute	CONCERN	Flink	Wazuh	Siddhi
Interoperability	Supports interoperability with many IoT protocols. Can integrate security tools.	Integrates with big data ecosystems but primarily focused on stream processing.	Interoperability with SIEM solutions, and endpoint security tools.	Supports integration with Kafka, MQTT, DB, and cloud platforms.
Security	It provides authentication mechanism for streams [43,77].	Provides encryption, authentication, but is not a security-focused tool.	Strong security focus: log analysis, intrusion detection, compliance monitoring.	It includes authentication, role-based access control, and security policies.
Fault Tolerance	Limited fault tolerance. Depends on the underlying system.	Good fault tolerance through state snapshots, exactly-once processing, distributed recovery.	Some resilience features, but mainly for security log analysis.	Supports distributed execution with failover mechanisms.
Recoverability	Can recover logs and alerts; no built-in recovery mechanisms.	Strong recoverability with distributed checkpointing and savepoints.	Can recover logs and alerts, but system state recovery depends on configuration.	Supports event replay and state persistence.
Usability	Good mechanism to generate rules through ontology [78,79].	Developer-friendly API but requires expertise in stream processing.	Complex configuration of rules using defined metrics.	Provides a SQL-like EPL for ease of use.
Time Behavior	Optimized for low latency in processing high-volume of events.	Optimized for low-latency, high-throughput stream processing.	Efficient for log collection and analysis. Experience latency in high-volume env.	Built for ultra-low-latency event processing.
Resources Usage	Low resources. Can run on low-powered devices [12,66].	High resource consumption; scales well for big data workloads.	High resource consumption; requires tuning for large-scale security monitoring.	Moderate lightweight but scales well in distributed environments.
Testability	Validated in [65].	Available test suite and benchmarking tools.	Available test suite but require extensive validation.	Provides built-in testing tools.
Installability	Require manual setup, single package deployable on AMD64 and ARM.	Requires setup of distributed components but has automation tools.	Relatively easy to install. Pre-built packages for various platforms.	Lightweight and easy to deploy in various environments.

7. AI investments showcase

Most business-related ICT solutions involve integration or cooperation with other (third-party-developed) components. This supply chain fragmentation represents a severe risk to the overall quality and security. It must be continuously monitored to properly recognize the problems, react to misbehavior, and avoid vulnerability propagation. Undetected criticalities can be highly critical in the case of financial applications like the investment portfolio optimization platform. In this case, consumers consistently rank trust as a critical factor in deciding where to deposit or invest their money. Online investment platforms are complex ICT systems that manage sensitive data and need to be trustworthy, so it is necessary to secure and monitor them accordingly.

Thus, in collaboration with 7Bulls[20], a private Polish company specializing in software development and integration of IT systems, CONCERN has been used inside their *Complete AI-based Investments platform* (AII). This is an innovative platform for optimizing financial portfolio management and risk control using the most advanced AI and deep learning solutions available. It was created to improve the investment results of hedge funds and other trading and investing companies. The AII application fetches data from various sources, combines and normalizes them, and then uses the latest achievements in time series forecasting and optimization to create the most suitable investment portfolio. Then, transactions are automatically executed using the selected stock broker.

In detail, the AII relies on a microservices architecture based on Kubernetes.[21] It runs in a multi-cloud or hybrid cloud environment, including auto-scaling mechanisms from on-premises infrastructure to the cloud. It is integrated with financial services brokers and data providers. Integration is used in real-time to retrieve up-to-date market data and place trade orders. Fig. 10 shows the overall AII architecture that includes:

- *Master AIT-Processor:* This is responsible for business logic, task preparation, results gathering, and system UI. Also integrated with financial brokers;
- *AIT-DB:* that manages data storage. It contains MariaDB[22] relational database and MongoDB[23] NoSQL database;

[20] www.7bulls.com.
[21] https://kubernetes.io/.
[22] https://mariadb.org/.

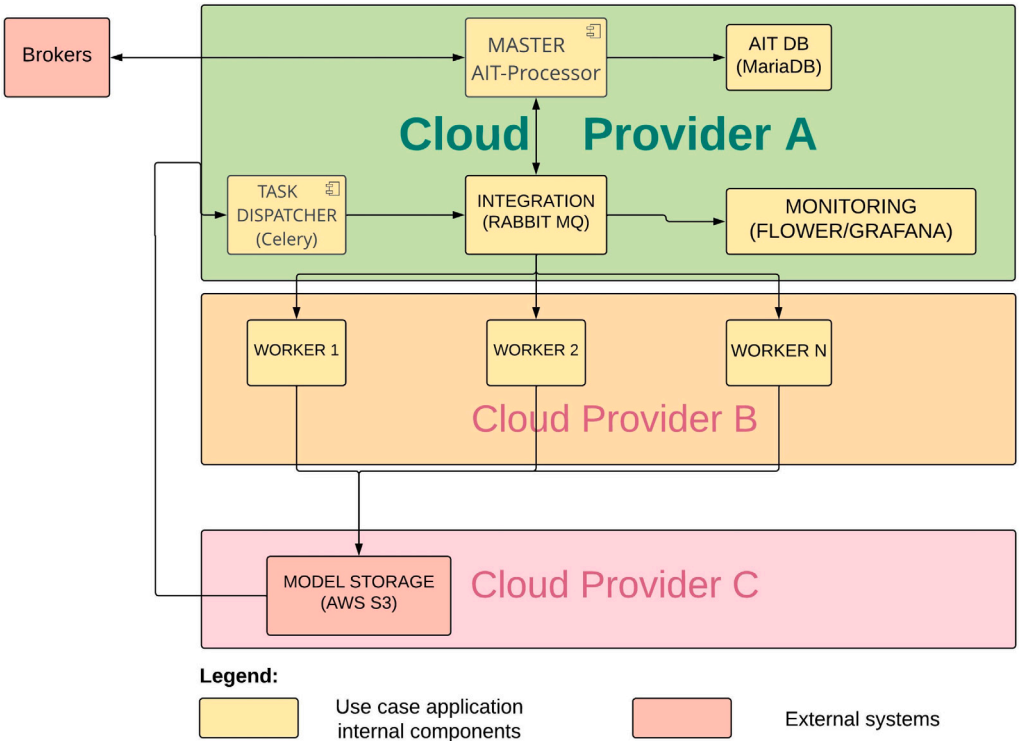

Fig. 10. AI Investments architecture overview.

- *Integration:* is the integration layer of all system components. The implementation leverages RabbitMQ17[24];
- *Task Dispatcher:* This is the training and re-training tasks dispatcher component. It is responsible for dispatching training tasks to different worker components. As task dispatcher is used, Celery18[25];
- *Monitoring:* This is the component responsible for monitoring all the core activities in the system. The technologies used for monitoring are Prometheus[26] and Grafana;
- *Worker:* This component is responsible for training and retraining machine learning models;
- *Model Storage:* This essential component manages the AI Investments system's data storage, effectively leveraging the robust capabilities of AWS S3.[27] Renowned for its scalability and reliability, AWS S3 is ideal for handling large volumes of data, offering high availability and security, and is perfect for supporting machine learning models.

Due to the handling of financial data and management of capital of large size (10 mUSD+), security issues are significant for AII; therefore, in collaboration with 7Bulls, the CONCERN solution has been included to guarantee continuous monitoring of security properties not assessed by the already existing AII Monitoring component.

Fig. 11 describes integrating the CONCERN monitoring infrastructure into the AII platform. In this case, a Docker instance of CONCERN[28] is executed on a virtual machine running on a cloud server. The established security business rules can be loaded through the CONCERN interface, and the CONCERN Probes can be included in the *Complete AI-based Investments platform*.

7.1. Concern set up

As anticipated in the previous section, the use of CONCERN has been focused on the properties not assessed by the AII embedded monitoring system, such as detecting if a specific event does not occur within a given period. In this case, the CONCERN rules can express the configuration of time frames and events that should occur to detect abnormal behavior patterns. This analysis exploits

[23] https://www.mongodb.com/.
[24] https://www.rabbitmq.com/.
[25] https://docs.celeryq.dev/.
[26] https://prometheus.io/.
[27] https://aws.amazon.com/it/s3/.
[28] https://hub.docker.com/r/acalabro/concern.

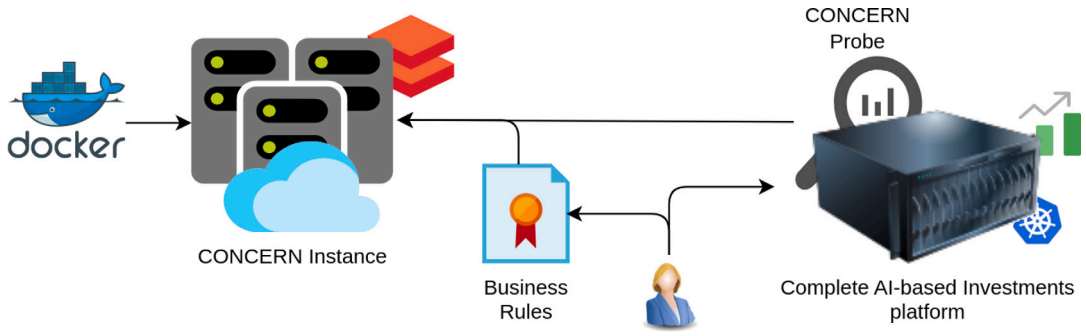

Fig. 11. Showcase setup deployment.

the peculiarity of CONCERN of monitoring individual events rather than their aggregates (such as the average or total number of events) and patterns of occurrence of events rather than just their number or numerical statistics.

The CONCERN solution was integrated into the AI Investments application using the provided *CONCERN probe/smart agent* released in Java (see Section 4.1). The integration process involved embedding specific libraries into the application framework, with an approach that perceives the AI system as an opaque entity, or 'black box'. This methodology facilitates monitoring without the need to understand or interfere with the internal workings of the AI algorithms. Each integrated component plays a specific role in this context, focusing on detecting and alerting security issues without delving into the AI's proprietary processes.

In particular, the following artifacts have been included in the AII Platform for enacting monitoring through CONCERN:

- *EventData:* This Java class encapsulates event-related data within the AI Investments application. It includes specifics such as the event type, timestamp, and other relevant data that could signify a security breach or notable events;
- *EventRedirector:* This component manages the distribution of event data throughout the system. It redirects event information for processing, analysis, or immediate response, especially in the case of potential security breaches;
- *MessageReceiver:* This class handles the reception of messages or notifications concerning security events. It forms a part of the messaging system that generates actively administrators or related systems about particular event types;
- *ProbeSender:* This module is responsible for sending events to the event monitoring system within the AI Investments application. Its primary function is to generate actively and dispatch events that are then analyzed by the monitoring system.

The integration was performed to enable ongoing monitoring of the AI Investments application and to provide data to CONCERN to detect events indicative of a possible security breach. The security rules described above were configured in CONCERN. Monitoring of security violations was carried out in the CONCERN GUI, where all events and their statistics are available, and most importantly, alerts are generated in case of security rule violations.

In the following, the complete list of the security rules included in the CONCERN implementation for the AII showcase execution is provided:

1. *Number of transactions per minute*: the rule focuses on the number of transaction orders sent to the broker in a window time (1 min). This number should not exceed a threshold due to the possible downgrade of the quality of the service. In case of violation, an alert is raised;
2. *Missing quotation*: the rule focuses on the quotation sent by the broker. A broker quotation is a security value, such as a quote from an active market related to a stock provided by a third party. Broker quotes are typically used with other valuation processes to determine the fair value of a stock. In this case, at least ten quotation messages per minute should be received. As a general application condition, the rule must be used only during working days (Monday to Friday). In case of rule violation, an alarm should be raised;
3. *Number of training requests*: the rule focuses on the number of training requests in the queue to be executed. Training requests represent training the neural networks related to reinforcement learning on the AI Investment platform. In particular, an alarm is raised if the number of training requests waiting in the queue exceeds the threshold of 100;
4. *Optimization of training requests*: the rule focuses on the number of training requests in the queue for being optimized. In this case, the alarm is raised when the number of optimization requests waiting in the queue exceeds the threshold of 50.

As examples, the formal representations of the first two rules, written using DROOLS language, are provided in Listings 4 and 5, respectively.

The listing 4 refers to the first rule: *Number of transactions per minute*. More in detail, in rows 2–6, the event is defined, and in row 5, the event time-to-live is declared: 1 min. Rows 13–15 define the mechanism for counting occurrences of specific events. Events may occur in a specific time frame, and they are selected according to the following parameters: type (*ConcernBaseEvent*), sender: (*SevenBullsSimProbe*) and name: (*TransactionStart*). In the end, row 17 contains the mechanism for notifying the violation if the threshold is overcome.

Listing 4: Number of transactions per minute

```
1  dialect "java"
2  declare ConcernBaseEvent
3      @role( event )
4      @timestamp( timestamp )
5      @expires(1m)
6  end
7
8  rule "counting transactions in a timeframe"
9  no-loop
10 salience 10
11 dialect "java"
12
13 when
14     Number( longValue > 100 ) from accumulate(
15     ConcernBaseEvent(name == "TransactionStart", senderID == "SevenBullsSimProbe" ) over window:time( 60s ), count(1))
16 then
17     NotificationManager.NotifyViolation("SevenBullsSimProbe", "Warning - more than 100 transactions per minute have been
           fired - Rule fired:" + drools.getRule().getName(), drools.getRule().getName(), drools.getRule().getMetaData());
18 end
```

The listing 5 refers to the second rule: *Missing quotation*. As before, the event declaration and time to live are declared in rows 2–6. In row 9, the *calendars* command allows the CONCERN Complex Event Processor to check an internal calendar and evaluate whether the rule can be processed. The time frame working days from (Monday to Friday) has been defined with *business-hours* variable.

Row 15 establishes that the rule is fired when no event of the type *ConcernBaseEvent*, with parameters name, equals to *QuotationFromBroker*, sender equals to *SevenBullsSimProbe* is received in a specific time frame: *60* m. At the end, row 19 contains the mechanism for notifying the violation.

Listing 5: Missing quotation from brokers

```
1  dialect "java"
2  declare ConcernBaseEvent
3      @role( event )
4      @timestamp( timestamp )
5      @expires(60s)
6  end
7
8  rule "Missing quotation from brokers"
9  calendars "business-hours"
10 no-loop
11 salience 10
12 dialect "java"
13
14 when
15     not($evt : ConcernBaseEvent( name == "QuotationFromBroker",
16         senderID == "SevenBullsSimProbe" )
17     over window:time(60m))
18 then
19     NotificationManager.NotifyViolation("SevenBullsSimProbe",
20         "Warning - the system has not received any quotation in the last 60 minutes - Rule fired:" +
               drools.getRule().getName(),
21         drools.getRule().getName(),
22         drools.getRule().getMetaData());
23 end
```

8. Evaluation results

The showcase execution of the AI Investments integrated with the CONCERN solution was developed in three stages following the guidelines of the software engineering standards.[29] Thus, the first unit, integration, and system tests were performed in the AI testing environment; successively, the online execution was observed for a certain period. More details for each stage are reported in the following:

Execution validation. This stage included the unit and integration test and focused on correctly executing all components. Thus, test messages, events, and notifications have been sent among components in isolation (unit test) and between progressively larger integrated sets (integration test). The detected problems, mainly due to the communication among components, have been solved, and possible improvements have been applied.

[29] https://www.computer.org/education/bodies-of-knowledge/software-engineering.

Execution in a controlled environment. This stage includes the system test focused on security and privacy properties assessment. In this case, the AI Investments integrated with the CONCERN solution phase have been run in a controlled environment, i.e., an in vitro realistic environment that can be exploited for validation and testing. During this stage, the penetration testing approach[30] and specific test scenarios have been executed to assess the AII platform security risks. Test scenarios simulating the firing of the rules mentioned in Section 6.1 have also been executed to assess their correct implementation and establish their reaction time. This stage has been executed hourly, and problems have been detected, mainly due to tuning parameters. Rules (like events time to live) have been refined for better monitoring performance.

On-line execution. This stage includes the online execution of the AI Investments integrated with the CONCERN solution within the actual operational environment. The AII has been observed for about three months of production operation, collecting monitoring data and violations. Indeed, during the online observation, CONCERN sent a violation message of one of the rules listed in Section 7.1 twice. A post-analysis of the collected monitoring data revealed that a never-detected AII internal software error caused the violations, not a malicious security breach.

As a final result, the use of CONCERN in the AI Investments platform has been positively evaluated by 7Bulls, which was able to detect, correct, and improve the overall quality of the AII. In particular, 7Bulls recognized the possibility of customizing specific rules as an essential point of strength. They desired a security level according to the AII peculiarity, the availability of collected and analyzed data on isolated events, and the possibility of evaluating their temporal correlation to detect violations or suspected security breaches.

The experiment suggested 7Bull's future integration of a dedicated monitor solution focused on security and privacy property assessment on the AII platform. The experiment also evidences to the CONCERN developers the necessity to improve the methodology for the rule specifications to let the not expert in monitoring activity better focus on the proper functional and non-functional requirements and security aspects.

9. Conclusions

This paper describes the architecture and the implementation of the CONCERN monitoring solution, which is useful for runtime assessment of functional and non-functional properties. Results of its assessment in a test-bed, simulation, and real-world financial environment have also been reported. The results collected show the effectiveness of the CONCERN as a means of creating value and guaranteeing business continuity. Ongoing and future works involve validating CONCERN in different environments and real use cases as well as continuing the comparison, considering another CEP-based monitor, such as (Apache Kafka Streams,[31] WSO2 CEP[32]), and possibly some non-open source solutions. The collaboration with the industrial partner evidences the necessity of refining the CONCERN GUI interface to improve its usability by not expert users. Integrating features to improve rule development and management based on artificial intelligence or ontology management is also part of our future work. Finally, to improve CONCERN execution in IoT environments having stringent energy consumption constraints, we will improve the current implementation by: (i) dynamically adjusting computational resources based on workload demand, reducing power consumption during periods of low activity. For example, exploiting facilities provided by the DROOLS engine in scheduling rules execution according to an ad-hoc energy-aware algorithm; (ii) evaluating the energy impact of the CONCERN architectural elements and selecting the best energy-efficient CEP engine and components to minimize overall consumption; (iii) using AI to enable the generation of rules that aim to reduce the potential overload of the complex event processor engine by distributing the calculation on more instances of CEP that can be deployed on several federated nodes.

CRediT authorship contribution statement

Antonello Calabrò: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Data curation, Conceptualization. **Eda Marchetti:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Data curation, Conceptualization. **Paweł Skrzypek:** Validation. **Jan Marchel:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[30] https://www.computer.org/education/bodies-of-knowledge/software-engineering.
[31] http://kafka.apache.org/documentation/streams/.
[32] https://wso2.com/.

Acknowledgments

The author sincerely thanks 7Bulls.com in the person of PawełSkrzypek and Jan Marchel for their collaboration and support in conducting the CONCERN industrial validation. Their expertise, resources, and technical assistance were fundamental for instrumenting their system, collecting real data events, and achieving the objectives of this study. The author would also like to thank UNINOVA in the person of José Barata and Ricardo Peres for collaborating in using CONCERN inside the CoppeliaSim environment.

This work was partially supported by the project RESTART (PE00000001) and project SERICS (PE00000014) under the NRRP MUR program funded by the NextGenerationEU.

Data availability

No data was used for the research described in the article.

References

[1] Q.M. Rahman, P. Corke, F. Dayoub, Run-time monitoring of machine learning for robotic perception: A survey of emerging trends, IEEE Access 9 (2021) 20067–20075, http://dx.doi.org/10.1109/access.2021.3055015.
[2] W. Hironori (Ed.), Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0, IEEE Computer Society, 2024.
[3] A. Bertolino, A. Calabrò, F. Lonetti, A. Sabetta, GLIMPSE: a generic and flexible monitoring infrastructure, in: F.D. Giandomenico (Ed.), Proceedings of the 13th European Workshop on Dependable Computing, EWDC '11, Pisa, Italy, May 11-12, 2011, ACM, 2011, pp. 73–78, http://dx.doi.org/10.1145/1978582.1978598.
[4] A. Calabrò, F. Lonetti, E. Marchetti, Monitoring of business process execution based on performance indicators, in: 41st Euromicro Conference on Software Engineering and Advanced Applications, EUROMICRO-SEAA 2015, Madeira, Portugal, August 26-28, 2015, IEEE Computer Society, 2015, pp. 255–258, http://dx.doi.org/10.1109/SEAA.2015.73.
[5] A. Calabrò, F. Lonetti, E. Marchetti, G.O. Spagnolo, Enhancing business process performance analysis through coverage-based monitoring, in: M.C. Paulk, R.J. Machado, M.A. Brito, M.G. ao, V. Amaral (Eds.), 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016, Lisbon, Portugal, September 6-9, 2016, IEEE Computer Society, 2016, pp. 35–43, http://dx.doi.org/10.1109/QUATIC.2016.015, URL: https://doi.ieeecomputersociety.org/10.1109/QUATIC.2016.015.
[6] D.M. Saxena, V. Kurtz, M. Hebert, Learning robust failure response for autonomous vision based flight, in: 2017 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, pp. 5824–5829.
[7] S. Mohseni, A. Jagadeesh, Z. Wang, Predicting model failure using saliency maps in autonomous driving systems, 2019, arXiv:1905.07679.
[8] S. Rabiee, J. Biswas, IVOA: Introspective vision for obstacle avoidance, 2019, arXiv:1903.01028.
[9] S. Hecker, D. Dai, L.V. Gool, Failure prediction for autonomous driving, 2018, arXiv:1805.01811.
[10] P. Antonante, D.I. Spivak, L. Carlone, Monitoring and diagnosability of perception systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021, IEEE, 2021, pp. 168–175, http://dx.doi.org/10.1109/IROS51168.2021.9636497.
[11] P. Barsocchi, A. Calabrò, A. Crivello, S. Daoudagh, F. Furfari, M. Girolami, E. Marchetti, A privacy-by-design architecture for indoor localization systems, in: M.J. Shepperd, F.B. e Abreu, A.R. da Silva, R. Pérez-Castillo (Eds.), Quality of Information and Communications Technology - 13th International Conference, QUATIC 2020, Faro, Portugal, September 9-11, 2020, Proceedings, in: Communications in Computer and Information Science, vol. 1266, Springer, 2020, pp. 358–366, URL: https://doi.org/10.1007/978-3-030-58793-2_29.
[12] P. Barsocchi, A. Calabrò, E. Ferro, C. Gennaro, E. Marchetti, C. Vairo, Boosting a low-cost smart home environment with usage and access control rules, Sensors 18 (6) (2018) 1886, http://dx.doi.org/10.3390/s18061886.
[13] A. Bertolino, A. Calabrò, F. Lonetti, E. Marchetti, Towards business process execution adequacy criteria, in: D. Winkler, S. Biffl, J. Bergsmann (Eds.), Software Quality. the Future of Systems- and Software Development, 8th International Conference, SWQD 2016, Vienna, Austria, January 18-21, 2016, Proceedings, in: Lecture Notes in Business Information Processing, vol. 238, Springer, 2016, pp. 37–48, http://dx.doi.org/10.1007/978-3-319-27033-3_3.
[14] C. Corbière, N. Thome, A. Bar-Hen, M. Cord, P. Pérez, Addressing failure prediction by learning model confidence, in: H.M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E.B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 2898–2909, URL: https://proceedings.neurips.cc/paper/2019/hash/757f843a169cc678064d9530d12a1881-Abstract.html.
[15] M. Won, Intelligent traffic monitoring systems for vehicle classification: A survey, IEEE Access 8 (2020) 73340–73358, http://dx.doi.org/10.1109/ACCESS.2020.2987634.
[16] R.-P. Fotescu, R. Constantinescu, B. Alexandrescu, L.-M. Burciu, System for monitoring the parameters of vehicle, in: Proceedings Volume 11718, Advanced Topics in Optoelectronics, Microelectronics and Nanotechnologies, 2020, p. 10, http://dx.doi.org/10.1117/12.2569928.
[17] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, N. Meskin, Cybersecurity for industrial control systems: A survey, Comput. Secur. 89 (2020) 101677.
[18] M.A. Santos, R. Munoz, R. Olivares, P.P. Rebouças Filho, J. Del Ser, V.H.C. de Albuquerque, Online heart monitoring systems on the internet of health things environments: A survey, a reference model and an outlook, Inf. Fusion 53 (2020) 222–239.
[19] A. Ziehn, Complex event processing for the internet of things, in: Z. Abedjan, K. Hose (Eds.), Proceedings of the VLDB 2020 PhD Workshop Co-Located with the 46th International Conference on Very Large Databases (VLDB 2020), ONLINE, August 31 - September 4, 2020, in: CEUR Workshop Proceedings, vol. 2652, CEUR-WS.org, 2020, pp. 1–4, URL: https://ceur-ws.org/Vol-2652/paper01.pdf.
[20] S.S. Kumar, S. Agarwal, Rule based complex event processing for IoT applications: Review, classification and challenges, Expert Syst. (2024) e13597.
[21] A. Akbar, S.S. Chaudhry, A. Khan, A. Ali, W. Rafiq, On complex event processing for internet of things, in: 2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences, ICETAS, IEEE, 2019, pp. 1–7.
[22] C.Y. Chen, J.H. Fu, T. Sung, P.-F. Wang, E. Jou, M.-W. Feng, Complex event processing for the internet of things and its applications, in: 2014 IEEE International Conference on Automation Science and Engineering, CASE, IEEE, 2014, pp. 1144–1149.
[23] A.M. Rahmani, Z. Babaei, A. Souri, Event-driven IoT architecture for data analysis of reliable healthcare application using complex event processing, Clust. Comput. 24 (2) (2021) 1347–1360.
[24] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz, J. Boubeta-Puig, A stream processing architecture for heterogeneous data sources in the Internet of Things, Comput. Stand. Interfaces 70 (2020) 103426.
[25] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, et al., Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.

[26] C. Calavaro, G.R. Russo, V. Cardellini, Real-time analysis of market data leveraging Apache Flink, in: Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems, 2022, pp. 162–165.

[27] M. Yang, R.T. Ma, Smooth task migration in apache storm, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 2067–2068.

[28] M. Adam, C. Cordeiro, L. Field, D. Giordano, L. Magnoni, Real-time complex event processing for cloud resources, in: Journal of Physics: Conference Series, Vol. 898, IOP Publishing, 2017, 042020.

[29] B. Yan, Q. Liu, J. Shen, D. Liang, B. Zhao, L. Ouyang, A survey of low-latency transmission strategies in software defined networking, Comput. Sci. Rev. 40 (2021) 100386.

[30] L. Aceto, A. Achilleos, D.P. Attard, L. Exibard, A. Francalanza, A. Ingólfsdóttir, A monitoring tool for linear-time μ HML, in: Coordination Models and Languages: 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held As Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings, Springer, 2022, pp. 200–219.

[31] L. Aceto, A. Achilleos, E. Anastasiadi, A. Ingolfsdottir, Axiomatizing recursion-free, regular monitors, J. Log. Algebraic Methods Program. 127 (2022) http://dx.doi.org/10.1016/j.jlamp.2022.100778.

[32] L. Aceto, I. Cassar, A. Francalanza, A. Ingólfsdóttir, Bidirectional runtime enforcement of first-order branching-time properties, Log. Methods Comput. Sci. 19 (1) (2023) 14:1–14:44, http://dx.doi.org/10.46298/LMCS-19(1:14)2023.

[33] E. Mulo, U. Zdun, S. Dustdar, Domain-specific language for event-based compliance monitoring in process-driven SOAs, Serv. Oriented Comput. Appl. 7 (1) (2013) 59–73, http://dx.doi.org/10.1007/S11761-012-0121-3.

[34] M.U. Şimsek, İ. Kök, S. Özdemir, Cepair: an AI-powered and fog-based predictive CEP system for air quality monitoring, Clust. Comput. (2024) 1–15.

[35] A. Calabrò, E. Cioroaica, S. Daoudagh, E. Marchetti, BIECO runtime auditing framework, in: J.J.G. Prego, J.G. de la Puerta, P.G. Bringas, H. Quintián, E. Corchado (Eds.), 14th International Conference on Computational Intelligence in Security for Information Systems and 12th International Conference on European Transnational Educational (CISIS 2021 and ICEUTE 2021), Bilbao, Spain, 22-24 September, 2021, in: Advances in Intelligent Systems and Computing, vol. 1400, Springer, 2021, pp. 181–191, http://dx.doi.org/10.1007/978-3-030-87872-6_18.

[36] S. Daoudagh, E. Marchetti, A. Calabrò, F. Ferrada, A. Oliveira, J. Barata, R.S. Peres, F. Marques, Daemon: A domain-based monitoring ontology for IoT systems, SN Comput. Sci. 4 (5) (2023) 632, http://dx.doi.org/10.1007/S42979-023-01975-Y.

[37] D. Geer, Users make a beeline for ZigBee sensor technology, Computer 38 (12) (2005) 16–19, http://dx.doi.org/10.1109/MC.2005.422.

[38] E. Mackensen, M. Lai, T.M. Wendt, Bluetooth Low Energy (BLE) based wireless sensors, in: SENSORS, 2012 IEEE, 2012, pp. 1–4, http://dx.doi.org/10.1109/ICSENS.2012.6411303.

[39] D. Fahland, C. Gierds, Analyzing and completing middleware designs for enterprise integration using coloured Petri nets, in: C. Salinesi, M.C. Norrie, O. Pastor (Eds.), Advanced Information Systems Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 400–416.

[40] L. Aceto, D.P. Attard, A. Francalanza, A. Ingólfsdóttir, A choreographed outline instrumentation algorithm for asynchronous components, 2021, CoRR abs/2104.09433, arXiv:2104.09433, URL: https://arxiv.org/abs/2104.09433.

[41] A.B. Hamida, A. Bertolino, A. Calabrò, G.D. Angelis, N. Lago, J. Lesbegueries, Monitoring service choreographies from multiple sources, in: P. Avgeriou (Ed.), Software Engineering for Resilient Systems - 4th International Workshop, SERENE 2012, Pisa, Italy, September 27-28, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7527, Springer, 2012, pp. 134–149, http://dx.doi.org/10.1007/978-3-642-33176-3_10.

[42] A. Bertolino, A. Calabrò, G.D. Angelis, Adaptive SLA monitoring of service choreographies enacted on the cloud, in: A.D. Ionita, G.A. Lewis, M. Litoiu (Eds.), 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2013, Eindhoven, The Netherlands, September 23, 2013, IEEE Computer Society, 2013, pp. 92–101, http://dx.doi.org/10.1109/MESOCA.2013.6632741.

[43] A. Bertolino, A. Calabrò, G.D. Angelis, A generative approach for the adaptive monitoring of SLA in service choreographies, in: F. Daniel, P. Dolog, Q. Li (Eds.), Web Engineering - 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 7977, Springer, 2013, pp. 408–415, http://dx.doi.org/10.1007/978-3-642-39200-9_34.

[44] Y. Kirikkayis, F. Gallik, M. Winter, M. Reichert, BPMNE4IoT: A framework for modeling, executing and monitoring IoT-driven processes, Futur. Internet 15 (3) (2023) 90, http://dx.doi.org/10.3390/FI15030090.

[45] N. Majma, S.M. Babamir, Model-based monitoring and adaptation of pacemaker behavior using hierarchical fuzzy colored Petri-nets, IEEE Trans. Syst. Man Cybern. Syst. 50 (9) (2020) 3344–3357, http://dx.doi.org/10.1109/TSMC.2018.2861718.

[46] Z. Wen, J. Wei, L. Wenchao, H. Yubin, Y. Tongdong, W. Qiujie, Research on CEP dispatching decision model based on digital twin grid, in: 2023 IEEE 6th International Conference on Information Systems and Computer Aided Education, ICISCAE, 2023, pp. 458–464, http://dx.doi.org/10.1109/ICISCAE59047.2023.10392599.

[47] D. de Freitas Bezerra, V.W.C. de Medeiros, G.E. Gonçalves, Towards a control-as-a-service architecture for smart environments, Simul. Model. Pr. Theory 107 (2021) 102194, http://dx.doi.org/10.1016/j.simpat.2020.102194, URL: https://www.sciencedirect.com/science/article/pii/S1569190X20301337.

[48] T. Zoppi, A. Ceccarelli, A. Bondavalli, Madness: A multi-layer anomaly detection framework for complex dynamic systems, IEEE Trans. Dependable Secur. Comput. 18 (2) (2011) 796–809.

[49] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, B. Becker, Model-driven architectural monitoring and adaptation for autonomic systems, in: Proceedings of the 6th International Conference on Autonomic Computing, 2009, pp. 67–68.

[50] U. Breitenbücher, Eine Musterbasierte Methode Zur Automatisierung Des Anwendungsmanagements (Ph.D. thesis), University of Stuttgart, Germany, 2016, URL: https://nbn-resolving.org/urn:nbn:de:bsz:93-opus-ds-87810.

[51] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann, From pattern languages to solution implementations, in: Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications, PATTERNS 2014, 2014, pp. 12–21.

[52] A. Pérez-Vereda, C. Canal, R. Hervás, Distributed mobile CEP for collaborative social computing, Computing 106 (3) (2024) 889–913.

[53] A. Mdhaffar, G. Baklouti, Y. Rebai, M. Jmaiel, B. Freisleben, RL4CEP: reinforcement learning for updating CEP rules, Complex Intell. Syst. 11 (2) (2025) 1–15.

[54] J.N. Pedro Ramos Brandao, Extended detection and response importance of events context, 2021, pp. 1–6, http://dx.doi.org/10.31112/Kriativ-Tech-2021-10-58.

[55] S. Bhatt, P.K. Manadhata, L. Zomlot, The operational role of security information and event management systems, IEEE Secur. Priv. 12 (5) (2014) 35–41.

[56] G. González-Granadillo, S. González-Zarzosa, R. Diaz, Security information and event management (SIEM): analysis, trends, and usage in critical infrastructures, Sensors 21 (14) (2021) 4759.

[57] I. Holubová, M. Vavrek, S. Scherzinger, Evolution management in multi-model databases, Data Knowl. Eng. 136 (2021) 101932.

[58] C. Krieger, U. Breitenbücher, M. Falkenthal, F. Leymann, V. Yussupov, U. Zdun, Monitoring behavioral compliance with architectural patterns based on complex event processing, in: A. Brogi, W. Zimmermann, K. Kritikos (Eds.), Service-Oriented and Cloud Computing - 8th IFIP WG 2.14 European Conference, ESOCC 2020, Heraklion, Crete, Greece, September 28-30, 2020, Proceedings, in: Lecture Notes in Computer Science, vol. 12054, Springer, 2020, pp. 125–140, http://dx.doi.org/10.1007/978-3-030-44769-4_10.

[59] A. Bertolino, A. Calabrò, F.D. Giandomenico, M. Martinucci, P. Masci, Automated refinement of dependability analysis through monitoring in dynamically connected systems, in: 10th International Symposium on Autonomous Decentralized Systems, ISADS 2011, Tokyo & Hiroshima, Japan, March 23-17, 2011, IEEE Computer Society, 2011, pp. 315–318, http://dx.doi.org/10.1109/ISADS.2011.46.

[60] A. Calabrò, F. Lonetti, E. Marchetti, Business process execution analysis through coverage-based monitoring, ERCIM News 2016 (105) (2016) 40–41.

[61] A. Calabrò, F. Lonetti, E. Marchetti, KPI evaluation of the business process execution through event monitoring activity, in: International Conference on Enterprise Systems, ES 2015, Basel, Switzerland, October 14-15, 2015, IEEE, 2015, pp. 169–176, http://dx.doi.org/10.1109/ES.2015.23.
[62] A. Calabrò, F. Lonetti, E. Marchetti, Access control policy coverage assessment through monitoring, in: S. Tonetta, E. Schoitsch, F. Bitsch (Eds.), Computer Safety, Reliability, and Security - SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Trento, Italy, September 12, 2017, Proceedings, in: Lecture Notes in Computer Science, vol. 10489, Springer, 2017, pp. 373–383, http://dx.doi.org/10.1007/978-3-319-66284-8_31.
[63] A. Calabrò, F. Lonetti, E. Marchetti, Monitoring of access control policy for refinement and improvements, in: D. Winkler, S. Biffl, J. Bergsmann (Eds.), Software Quality: Methods and Tools for Better Software and Systems - 10th International Conference, SWQD 2018, Vienna, Austria, January 16-19, 2018, Proceedings, in: Lecture Notes in Business Information Processing, vol. 302, Springer, 2018, pp. 17–36, http://dx.doi.org/10.1007/978-3-319-71440-0_2.
[64] A. Calabrò, S. Daoudagh, E. Marchetti, Towards enhanced monitoring framework with smart predictions, Log. J. IGPL 32 (2024) 321–333, URL: https://api.semanticscholar.org/CorpusID:268501704.
[65] A. Calabrò, E. Marchetti, Motef: A testing framework for runtime monitoring infrastructures, IEEE Access 12 (2024) 38005–38016, http://dx.doi.org/10.1109/ACCESS.2024.3376239.
[66] A. Calabrò, E. Marchetti, Transponder: Support for localizing distressed people through a flying drone network, Drones 8 (9) (2024) http://dx.doi.org/10.3390/drones8090465, URL: https://www.mdpi.com/2504-446X/8/9/465.
[67] A. Calabrò, S. Daoudagh, E. Marchetti, Towards enhanced monitoring framework with smart predictions, Log. J. IGPL 32 (2) (2024) 321–333, http://dx.doi.org/10.1093/jigpal/jzae010, arXiv:https://academic.oup.com/jigpal/article-pdf/32/2/321/58499069/jzae010.pdf.
[68] J. Joyce, G. Lomow, K. Slind, B.W. Unger, Monitoring distributed systems, ACM Trans. Comput. Syst. 5 (2) (1987) 121–150, http://dx.doi.org/10.1145/13677.22723.
[69] M. Mansouri-Samani, M. Sloman, Monitoring distributed systems, IEEE Netw. 7 (6) (1993) 20–30, http://dx.doi.org/10.1109/65.244791.
[70] M. Mansouri-Samani, M. Sloman, Gem: a generalized event monitoring language for distributed systems, Distrib. Syst. Eng. 4 (2) (1997) 96–108, http://dx.doi.org/10.1088/0967-1846/4/2/004.
[71] C. Krieger, U. Breitenbücher, M. Falkenthal, F. Leymann, V. Yussupov, U. Zdun, Monitoring behavioral compliance with architectural patterns based on complex event processing, in: A. Brogi, W. Zimmermann, K. Kritikos (Eds.), Service-Oriented and Cloud Computing, Springer International Publishing, Cham, 2020, pp. 125–140.
[72] J. Jeston, J. Nelis, Business Process Management, Taylor & Francis, 2014, URL: https://books.google.it/books?id=8Q6pAgAAQBAJ.
[73] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg, Internet of things patterns for communication and management, Trans. Pattern Lang. Program. 4 (2019) 139–182, URL: https://doi.org/10.1007/978-3-030-14291-9_5.
[74] D. Luckham, The power of events: An introduction to complex event processing in distributed enterprise systems, in: N. Bassiliades, G. Governatori, A. Paschke (Eds.), Rule Representation, Interchange and Reasoning on the Web, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, 3–3.
[75] A. Calabrò, S. Daoudagh, E. Marchetti, MENTORS: Monitoring environment for system of systems, in: F.J.D. Mayo, M. Marchiori, J. Filipe (Eds.), Proceedings of the 17th International Conference on Web Information Systems and Technologies, WEBIST 2021, October 26-28, 2021, SCITEPRESS, 2021, pp. 291–298, http://dx.doi.org/10.5220/0010658900003058.
[76] C. Forgy, Rete: A fast algorithm for the many patterns/many objects match problem, Artificial Intelligence 19 (1) (1982) 17–37, http://dx.doi.org/10.1016/0004-3702(82)90020-0.
[77] A. Calabrò, F. Lonetti, E. Marchetti, Access control policy coverage assessment through monitoring, in: S. Tonetta, E. Schoitsch, F. Bitsch (Eds.), Computer Safety, Reliability, and Security, Springer International Publishing, Cham, 2017, pp. 373–383.
[78] S. Daoudagh, E. Marchetti, A. Calabrò, F. Ferrada, A.I. Oliveira, J. Barata, R. Peres, F. Marques, An ontology-based solution for monitoring IoT cybersecurity, in: L.M. Camarinha-Matos, L. Ribeiro, L. Strous (Eds.), Internet of Things. IoT Through a Multi-Disciplinary Perspective, Springer International Publishing, Cham, 2022, pp. 158–176.
[79] A. Calabrò, S. Daoudagh, E. Marchetti, MENTORS: Monitoring environment for system of systems, in: F.J.D. Mayo, M. Marchiori, J. Filipe (Eds.), Proceedings of the 17th International Conference on Web Information Systems and Technologies, WEBIST 2021, October 26-28, 2021, SCITEPRESS, 2021, pp. 291–298.