Received 1 June 2024; accepted 12 August 2024. Date of publication 21 August 2024;
date of current version 13 September 2024. The review of this article was arranged by Associate Editor Luis Ribeiro.

Digital Object Identifier 10.1109/OJIES.2024.3447001

Toward Semantic Event-Handling for Building Explainable Cyber-Physical Systems

GERNOT STEINDL [1] (Member, IEEE), TOBIAS SCHWARZINGER [1], KATRIN SCHREIBERHUBER [2], AND FAJAR J. EKAPUTRA [2]

[1]Institute of Computer Engineering, Research Unit Automation Systems, Technical University Wien, 1040 Vienna, Austria
[2]Institute for Data, Process and Knowledge Management, Vienna University of Economics and Business, 1020 Vienna, Austria

CORRESPONDING AUTHOR: GERNOT STEINDL (e-mail: gernot.seindl@tuwien.ac.at).

This work was supported in part by the Austrian Research Promotion Agency FFG through research project SENSE under Grant 894802 and in part by TU Wien Bibliothek for financial support through its Open Access Funding Programme.

ABSTRACT In the context of cyber-physical systems (CPS), understanding system behaviors is crucial for ensuring reliability, efficiency, and trust. However, due to the increasing complexity of the modern CPS, gaining such understanding is becoming more challenging. In this article we provide a foundation for explaining system behavior through detected system events. To this end, the article proposes a technology-agnostic, semantic event-handling module to address the challenge of enhancing explainability within CPS. This module is designed to be part of the common architecture for Explainable CPS and, therefore, can be integrated seamlessly into existing CPS frameworks by providing different interfaces to access detected events. Two case studies in the smart building and smart grid domain are carried out to demonstrate the feasibility and efficacy of the approach, utilizing an open-source software stack for the prototypical implementation. A qualitative evaluation of the proposed approach, based on the ISO/IEC 25010:2023, was used to analyze the software design. Our evaluation result shows that the semantic event-handling module is appropriate as a generic approach to handling events within a CPS. The module becomes a foundation for incorporating explainability into the system, which is needed as a foundation to ensure human trust and enable informed decision-making.

INDEX TERMS Cyber-physical systems (CPSs), event detection, semantic web.

I. INTRODUCTION

A cyber-physical system (CPS) represents the convergence of the digital and the physical world, which is changing the way we interact and manage complex systems in various domains, such as manufacturing, energy, or buildings. These integrated systems combine computational capabilities with physical components to make those systems and processes smarter, more efficient, and easier to handle.

What differentiates CPSs from classical embedded systems that can also interact with their physical environment is the spatial distribution and networking capability within CPSs. Networking capabilities are needed to connect various components of a CPS. Frequently, this is founded on (Industrial) Internet of Things technologies. The spatial and logical distribution, together with dynamic and adaptable system behavior, increase the complexity of such CPS. Thus, for human operators, it is getting more challenging to understand the behavior of the underlying system, which increases the need for experienced operators. However, finding experienced operators can be challenging as they are not always readily available. Therefore, the increasing complexity reduces the understandability and trust in CPS, which led to the introduction of the explainable cyber-physical system (ExpCPS) concept [1].

The goal of explainability in this context is to ensure that the insights and decisions made by the CPS are interpretable, and transparent, enhancing the overall reliability and effectiveness of the system. However, explainability in this context is not an absolute characteristic of a CPS, as many intricate aspects influence the users' perception of the system. [2]

Nevertheless, the system's state and interactions with the surrounding environment must be observed, stored, and processed to provide a solid foundation for any generated explanation. Events encapsulate the occurrences or changes within the system, acting as critical indicators of its dynamic behavior. A simple definition for an event is: *"Anything that happens, or contemplated as happening"* [3].

Keeping track of the events within a system is important to provide a solid foundation for explanations of its behavior. As a result, events are crucial in ExpCPS due to their role in tracking and understanding state changes, enabling the monitoring and processing necessary for comprehending the system behavior. Thus, solutions must be provided that enable detecting and processing every relevant event within the CPS. But, only storing events in a CPS is not sufficient. To harness the full potential, adding semantic context to these events is crucial for designing, deploying, monitoring, and adapting CPSs [4], and especially for creating ExpCPS [1]. The semantic enrichment of events enhances their interpretability, allowing for a more profound understanding of the system's behavior. Semantic annotations provide meaningful labels and relationships, transforming raw event data into comprehensible information. This semantic layer forms a foundation for developing ExpCPS frameworks. However, the existing landscape of event-handling architectures within CPS still shows a notable gap due to the unavailability of simple, adaptable, and generic frameworks. Moreover, the current frameworks commonly overlook incorporating semantic context, a critical element necessary for building ExpCPS. The absence of semantics within these architectures limits their capability to provide meaningful insights and explanations for CPS behaviors. This problem hinders the development of ExpCPS because researchers have no reusable concepts for semantic event handling that would allow them to focus solely on the explainability capabilities of a system.

Thus, the overall research aim is to streamline the development of ExpCPS by providing versatile and generic concepts that facilitate the creation and management of ExpCPS frameworks. In this work, we want to establish a generic architecture for semantic event handling in ExpCPS that targets simplicity and adaptability while also integrating semantic contexts. Such an architecture forms the foundation for ExpCPS, enabling comprehensive system observations and fostering enhanced transparency, and interpretability within complex CPS environments.

To reach that goal, in this work, we answer the following research question: *What are the design characteristics of a semantic event-handling architecture capable of managing simple and complex events to establish a foundational framework for ExpCPS?*

To answer that question, a literature review is conducted to understand the principles of ExpCPS architectures and already applied event-handling architectures for CPS. Based on that, the fundamental principles, challenges, and design considerations associated with event handling in ExpCPS are identified. Those insights were used to propose a semantic event-handling architecture that is evaluated by case studies in the smart grid and smart building domain. The case studies analyzed the event-handling architecture employed, focusing on their ability to manage simple and complex events. Therefore, a prototype for the semantic event-handling framework is implemented. The case studies are used to validate the feasibility and extract insights into how these architectures contribute to the establishment of foundational frameworks for ExpCPS. In addition, a qualitative evaluation of the proposed software architecture is conducted, based on the ISO/IEC 25010:2023 standard.

With regard to the stated research question, our work helps to advance the development of ExpCPS by providing the following contributions.

1) An ExpCPS architecture including a minimal set of components essential for effective operation, based on insights from existing literature in the field.
2) Details for the "semantic event-handling module" as part of the ExpCPS architecture, designed to enhance event processing in ExpCPS.
3) A prototypical implementation of the proposed architecture, leveraging Semantic Web Technologies, to demonstrate its feasibility in real-world scenarios.
4) A qualitative evaluation of the approach based on the ISO/IEC standard, offering insights into the effectiveness and usability of the proposed ExpCPS architecture and semantic event-handling module.

II. RELATED WORK

In this section, we provide an overview of the concepts of the relatively young research area of ExpCPS and its relation to event handling. Furthermore, event-handling approaches that have been applied in CPS are described, especially those that incorporate semantic approaches, as this has been shown to be a notable requirement for providing context in ExpCPS.

A. EXPLAINABILITY IN CPS

The overarching goal of explainability is to provide answers to "why something happens" [5]. This is a key feature for humans to build trust in systems that impact their lives, which is often the case in CPS (e.g., buildings or energy systems).

Sadeghi et al. [6] discussed the growing necessity for systems to explain their behavior, particularly in the context of complex technologies, such as machine learning and self-adaptation. In this article, we introduce a taxonomy aimed at categorizing the diverse needs for explanations based on various reasons. This article's key contributions include the taxonomy of explanation needs, the associated explanation cases, and the identification of influencing factors determining the necessity of explanations. Similar to our work, this article's primary focus is enhancing user interaction with the system through improved explanations. This means that the original system behaviour is not understandable, desirable, or expected by a user as defined by Gregor and Benbasat [7].

CPS development faces different challenges in terms of complexities. In [8], various complexity facets concerning the type of system, the environment as well as organizational aspects are identified. The difficulty in understanding a system is identified as one of the consequences of these complex facets. In our approach, we offer a framework to make CPS more understandable, especially concerning the complexity facets of heterogeneity in sensors, properties, and behaviors, as well as size and computability-related facets concerning a large number of units and interactions in a complex CPS. Our framework aims to process event data effectively to make data more accessible to humans, reducing information overload and increasing understandability, considering limited memory capacity and bounded rationality as human limitations.

In the realm of interpretable machine learning, which is a related field to ExpCPS, different approaches have already been proposed. A taxonomy for these methods that enable the interpretation of machine learning models, that is structured around the outcomes of interpretation methods is provided in [5]. It distinguishes between model-specific and model-agnostic approaches, as well as local and global interpretations. Another survey on explainable anomaly detection for Industrial Internet of Things (IIoT) provides a similar taxonomy [9]. The authors specifically focus on explaining abnormal behavior within an IIoT system because, as they stated, anomaly detection is essential for real-time monitoring, including decision-making in dynamic, heterogeneous, and large-scale IIoT infrastructure. In this article, the authors only focus on approaches that apply to machine learning. They mainly distinguish between intrinsic interpretable machine learning models, such as linear regression, decision trees, or Naive Bayes Classifier, and model-agnostic methods, such as local interpretable model-agnostic explanations [10], Anchors [11], and Sapley Additive Explanations [12].

However, for enhancing the explainability of CPS, these machine learning approaches have drawbacks as they do not consider the impact of the physical and virtual context, which affects its behavior [1]. Therefore, they propose that such ExpCPS need different approaches for providing meaningful explanations. Furthermore, an enrichment with additional context information using semantic technologies is proposed. An example of how such semantic technologies can be applied in ExpCPS is presented in [13], where the authors show the essential role of causality knowledge in ExpCPS. This article introduces a two-level abstraction of causality, namely abstract and concrete causality, and outline how SPARQL queries can infer concrete causality from the query results. The method's utility is demonstrated in a smart grid scenario, illustrating its potential to streamline the incorporation of causality knowledge into CPS.

A general architecture for ExpCPS is essential due to the increasing complexity of these systems, which often involve intricate interconnections between digital and physical components. A standardized architecture provides a systematic and unified framework for incorporating explainability features across diverse CPS applications. Furthermore, a general architecture facilitates consistency in implementing explainability mechanisms, easing the integration of novel approaches, and fostering interoperability. One example of such a general architecture is presented in [13]. Aryan et al. [13] proposed a knowledge graph-based framework for creating explainable cyber-physical energy systems (CPES) in the context of smart grids. The framework utilizes an ontology to model and link data from various sources and employs graph-based algorithms to generate explanations from events. The document presents a simulated demand response scenario to demonstrate the applicability of the framework. The scenario also involves aspects outside the electricity grid, such as data from weather services and the energy market. The framework provides an explanation that traces the root causes of an event and considers the aggregation relationship of causes. Integrating various data and knowledge is identified as a key challenge in building an explainable CPES.

Another framework is shown in [14]. In this case, the authors outline a vision for addressing the challenge of understanding and comprehending the behavior and decisions of complex CPS. The proposed solution is called the Monitor, Analyze, Build, Explain (MAB-EX) framework, which is an adaption of the Monitor, Analyze, Plan, Execute, Knowledge-base (MAPE-K) approach that was introduced in [15] for self-adaptive systems. The MAB-EX framework is designed to create self-explainable systems that answer questions about their past, current, and future behavior in real time. The framework operates in four stages: monitor and analyze system behavior, build an explanation using explanation models, and convey this explanation to stakeholders in a suitable manner. Importantly, the framework acknowledges the potential for learning new explanations by updating the explanation models when the system detects new and unexplainable behavior.

Looking at both approaches, it is apparent that proper event handling is crucial in such ExpCPS frameworks for several critical reasons as follows.

1) Events represent dynamic changes, occurrences, or triggers within the system, serving as essential indicators that demand an explanation.
2) In an ExpCPS, understanding the causes and effects of events is foundational to providing transparent and interpretable explanations for the system's behavior.
3) Effective event handling ensures that a framework is able to monitor, analyze, and respond to events in real-time.
4) Event handling is crucial for identifying anomalies, learning from new behaviors, and updating explanation models accordingly.

Thus, by providing proper event-handling capabilities, a framework can offer various stakeholders a clear and contextually rich explanation, fostering trust, usability, and reliability in these complex and dynamic systems.

For this reason, we focus in our work on the event-handling part in such systems to provide a solid foundation for implementing ExpCPS frameworks. In the next section, we elaborate on existing event-handling approaches in CPS and how they have to be adapted for ExpCPS.

B. EVENT HANDLING ARCHITECTURE IN CPS

Molina et al. [16] presented an event-driven architecture for CPS. They identified several requirements for CPS event-handling architecture, such as interoperability, flexibility, low coupling, and asynchrony. Consequentially, the authors advocate for an event-driven architecture applying the publish-subscribe communication pattern. The proposed architecture focuses primarily on scalability, error correction, and consistency, aligning with the complex demands of CPS. A similar event-driven architecture approach, based on a broker technology is presented in [17]. López [17] proposed a real-time event-based platform for developing digital twin applications.

However, a notable gap exists in both approaches regarding considering semantics in conjunction with events. Even if they show the suitability of broker technology for event handling and exchange within a framework, they neglect the semantic aspects, which is important to provide context for explainability [1].

An approach that takes context into consideration is presented in [18]. Verma et al. [18] introduce an innovative approach to complex event processing (CEP) in smart environments, addressing the limitations of classical complex event detection engines. They consider the spatial relationships among sensors and acknowledge the dynamic impact of temporal changes in sensor data. They show the effectiveness in enhancing smart environments by providing a more context-aware approach to CEP. Therefore, they introduced the concept of detecting so-called atomic events, which were afterward processed by the CEP engine. This two-step approach provides additional flexibility for event detection. However, the authors applied a feature extraction algorithm in combination with a convolutional neural network. This approach requires labeled training data, which is often not available especially before or during commissioning. In addition, expert knowledge cannot be incorporated to detect atomic events within the sensor data.

A more semantic-based approach is presented in [19], in which an ontology-based language and approach for event description and detection was introduced. It provides rich expressiveness as language and can act as an intermediate representation for transformations between different event description languages used in various CEP engines. However, the presented ontology focuses only on the representation of complex events and does not capture any domain knowledge. Such use of domain knowledge and and its application is presented in [20], which introduces an ontology for formalizing knowledge related to condition-monitoring tasks in manufacturing processes. The ontology is composed of three modules: manufacturing, context, and condition monitoring. The applicability of the ontology is shown with a use case of bearing conditions in rotating machinery. They applied rule-based reasoning to infer operating states and error severity, which initiates further condition-monitoring activities. The approach showcases the benefits of rule-based reasoning, that incorporates expert knowledge.

Another semantic approach is presented in [21], where the authors used stream reasoning for real-time detection of situations that may lead to failures in manufacturing processes. It enriches sensor data with contextual information to enhance real-time situation detection. The use of stream reasoning enables real-time processing and supports decision-making based on continuous data streams and background knowledge. The authors introduce a general framework architecture that integrates contextual knowledge, user experience, and semantics in manufacturing and production processes.

A follow-up paper [22] from the same authors presents an extension of the framework that provides a hierarchy of situations for decision-making, enabling the identification of actions to correct abnormal behavior to prevent process interruptions. As in their first work, their framework leverages ontologies and knowledge bases to create a machine-interpretable representation of manufacturing knowledge to enable effective condition monitoring, diagnosis, and decision-making. However, the approach is also tailored towards stream reasoning. It neglects the problem of detecting the events themselves in a discrete or continuous signal that is produced by sensors. As already mentioned, detecting such events is not trivial, but also crucial for generalizing this approach.

In summary, the literature shows that proper event detection and handling is vital for creating ExpCPS. Using event brokers is a common technique to handle events within an architecture, because of the decoupling between event producers and consumers. It has also been shown that using semantic models to capture domain knowledge for context-aware explanations and facilitating effective event detection methods is crucial. A two-stage event detection approach introduces additional flexibility to apply different techniques at certain levels, e.g., combining rule-based or model-based detection with stream reasoning. These findings are used in the following section for the design of a generic event-handling architecture that is a foundational part of an ExpCPS.

III. THE EXPCPS ARCHITECTURE

Based on the architectures found in the literature, the main components are identified that are necessary to create a ExpCPS. It should be noted, that for providing the full feedback loop to the physical entity of an ExpCPS, as suggested for the MAB-EX framework [14], more modules would be needed. However, if only explanations have to be provided and the system is not acting autonomously, the modules as depicted in Fig. 1 are sufficient. The modules are structured based on the layers defined by the Reference Architecture Model Industry 4.0 (RAMI 4.0) [23]. The RAMI 4.0 layers help to structure the different modules of such an ExpCPS based on a functional view. These modules are now explained in more detail.

Physical entity: The *physical entity* represents the physical part of an CPS. It is usually distributed in nature. The integration into the cyber-space is done by using various sensors (and actuators), which are networked by communication systems. We are not showing those integration and communication

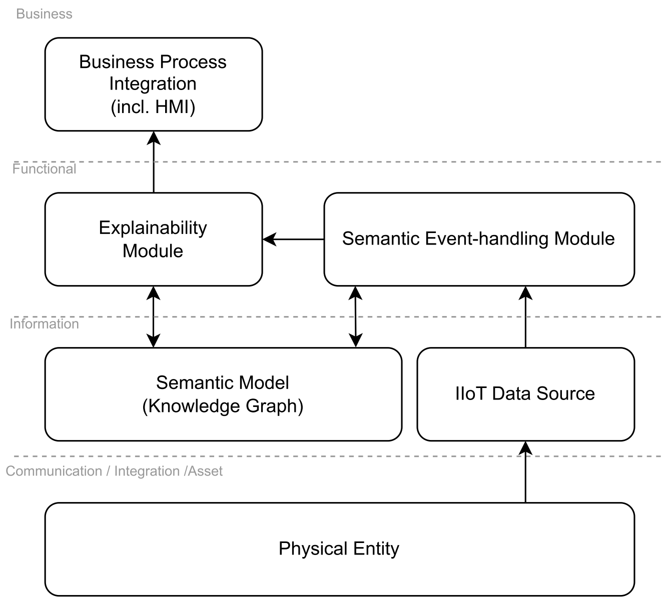
Business
Business Process Integration (incl. HMI)
Functional
Explainability Module
Semantic Event-handling Module
Information
Semantic Model (Knowledge Graph)
IIoT Data Source
Communication / Integration /Asset
Physical Entity

FIGURE 1. **Common modules for an ExpCPS architecture.**

details in Fig. 1 as they are not critical for understanding the ExpCPS architecture as a whole. Therefore, we only depicted the data flow from the physical entity to the *information layer*.

The subsequent *information layer* consists of two modules: 1) the *data source* and 2) the semantic model.

IIoT data source: Because of their distributed nature, data in a CPS is usually provided through an IIoT infrastructure. The available communication systems are often domain-specific. Some examples are OPC Unified Architecture servers or MQTT brokers in the industrial domain. Another example could be a BACnet server in the building domain. In addition, more recent concepts, like an asset administration shell instance, could also occur as a source for data. Next to these domain-specific technologies, IIoT platforms that abstract the underlying communication technology and typically provide web-based interfaces are also in place. Such platforms typically provide database connectors to store incoming values persistently. These time-series databases can also be directly used to access current and historical data. The data that is produced by these platforms or other available services can also be virtual. This means that the data are not measured but calculated or predicted. The minimal requirements for the data sources are that they provide an endpoint with an application programming interface (API) through which the data can be retrieved. Typically, the data are structured as a value with its corresponding timestamp. The contextualization of the data is performed after the extraction within the *semantic event-handling module* and with the help of the information stored in the *semantic model*. This provides a semantic abstraction layer for data integration similar to the one proposed in [24].

Semantic model: A semantic model is needed to provide context to data and events. This model describes certain aspects of the domain, like concept hierarchies and their relations, to model a virtual representation of the physical asset. The *semantic model* provides the general data schema to store system-specific assets in a knowledge graph. The model is used to retrieve information about assets, such as available entities, data sources, and mappings for the integration process. Other modules can use the information that is provided by the *semantic model*.

A variety of technologies can be employed to implement a *semantic model*. The suitability of different technologies and standards, such as graph databases, NGSI-LD [25], or a more general knowledge modeling language, such as the W3C standards resource description framework (RDF) [26] and web ontology language (OWL) [27], will depend on the semantic expressiveness that is needed and the available tooling support.

Semantic event-handling module: Within the *functional layer* various modules can be located. One essential functionality is the detection of events within the CPS and the contextualization of those events. This is performed by the *semantic event detection module*. As this module is the focus of this work, it is explained in more detail in the next section (Section IV).

Explainibility module: This module uses the *semantic model* as well as the events that are detected to provide explanations to various users of the CPS. It uses the semantic model to provide context and find the root cause for certain events. A detailed description of the module implementation is not in the scope of this article but will be provided to a limited extent in our evaluation.

Business process integration: On the *business layer* the ExpCPS has to be integrated into various business processes, e.g., a maintenance process. Usually, these processes perform some kind of user interaction with the ExpCPS where explanations are provided to the user of the system. Depending on the use case, different human–machine interfaces (HMIs) can be applied. One possible use case that we are currently working on is the utilization of a so-called chatbot that provides explanations interactively based on the findings of the *explainability module*.

It is important to note that the proposed ExpCPS architecture can be constructed on top of the legacy CPS, provided that the proper interfaces to the available IIoT data sources and other services are provided. Consequently, the explainability feature can be implemented alongside existing services within the CPS. As previously stated, if the full MAB-EX concept should be implemented, which means the system takes autonomous decisions based on the identified events and explanations, the feedback to the physical entity has to be closed via additional modules that are capable of communicating, e.g., with a CPS control service. Depending on the legacy CPS architecture, this can increase the integration effort. However, in any case the *semantic event-handling module* implements the interface to the underlying system and the rest of the modules that are needed for providing explainability. Thus, this module will be explained in more detail in the next section.

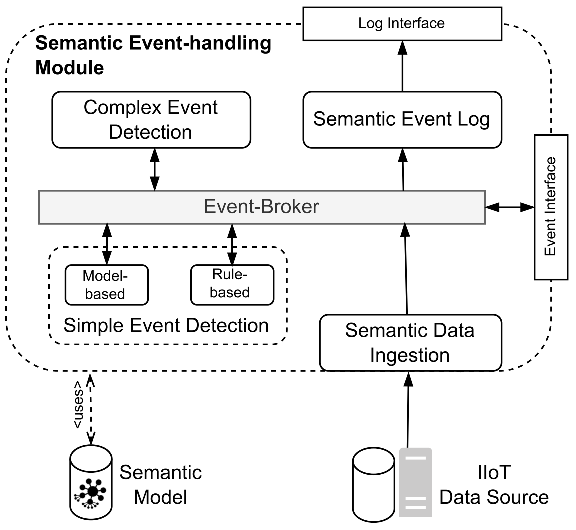

FIGURE 2. Details of the semantic event-handling module showing its internal components.

IV. SEMANTIC EVENT-HANDLING MODULE

The idea behind the *semantic event-handling module* that is presented in this section is to provide a technology-agnostic and domain-independent solution that can be seamlessly integrated with various ExpCPS architectures, offering versatility in its application. Its primary objective is to provide a semantic event-handling solution, including event detection, contextualization, and storing, that can be easily implemented in CPSs across various domains, regardless of a specific technology stack.

Fig. 2 shows the internal structure of the *semantic event-handling module* that consists of several functional components. In addition, the adjacent modules from the lower layer (*semantic model* and *IIoT data source*) are also depicted as the *semantic event-handling module* has to interact with them. The functional components from the *semantic event-handling module* are explained in more detail in the following:

Semantic data ingestion: This component has to perform two main tasks. First, depending on the data source, the module has to poll for new incoming data or register if a publish/subscribe communication mechanism is supported by the data source. Second, the data sources provide the incoming data and events in heterogeneous formats. Thus, the module has to normalize this information and perform mappings into concepts that are described in the *semantic model*. This allows a seamless integration of semantic data into further processing. Predefined mapping languages, such as the RDF mapping language (RML) [28], can simplify this process. Also, the necessary information to access various *IIoT data sources*, such as the endpoint type, its address, and how to access the information, is also stored in *semantic model*. Those data source descriptions can follow open standards, like those specified by Web of Things [29].

Event broker: Internally, the events are distributed by a so-called event broker. This approach enables decoupling the various modules within the *semantic event-handling module*. Thus, components that provide certain functionality can easily be added or removed, depending on the application use case. Furthermore, the event broker provides an easy-to-use interface that other modules within the whole ExpCPS architecture can easily connect to at run-time. They only need to register by the broker via the *event interface*. Afterward, those modules can extend the functionality of the ExpCPS.

Simple event detection: This component is responsible for detecting events within the data provided by the *data ingestion* component. There are two ways of doing that, depending on the use case. The first one is the *rule-based* approach, using logical rules that are defined externally by a domain expert or, in some cases, could be semi-automatically generated based on the information from the semantic model. Such simple rules could be, for example, a simple threshold violation or one signal exceeding another. Depending on the used rule engine and supported logic, more sophisticated rules can be defined, e.g., using signal temporal logic (STL).

The second approach is *model-based* to describe the dynamic behavior of the physical part. This is needed for use cases where simple rules are not sufficient because of the complexity of the problem. The incorporated models can be divided into three general categories: white-box, gray-box, or black-box models. *White-box models* are physics-based and represented mainly by partial differential equations. Mostly, these models are too complex and many parameters are not known exactly. *Black-box models* are purely data-driven. Machine learning is a common technique to create such models. Enough already available run-time data are crucial to capture the dynamic of the physical part. Problems can occur if they are applied under conditions that have not been captured or represented in the training data. *Gray-box models* are based on physical principles, but certain parameters are identified based on historical run-time data. Thus, they need a higher modeling effort but are usually more accurate in situations not seen in the training data or where less training data are available. Generally, every type of model can be deployed within the *semantic event-handling module* as long as an appropriate run-time environment is provided. Standards already exist that can simplify this process. The functional mock-up interface standard [30] is one example that enables the integration of third-party models as it defines the model interface and allows the encapsulation of the whole model within a so-called functional mock-up unit.

Complex event detection: This component is used to detect specific patterns within the stream of already detected simple events, with regard to time and space. An example of a such complex event is that the simple event E_A occurs within 5 min after the simple event E_B occurs, which will trigger the event E_C. CEP engines may incorporate temporal logic for detecting event sequences and causality. When predefined conditions or patterns are met, a complex event is sent to the *event broker*, which then can trigger further actions or only be logged for later analysis. The necessity of such CEP is very use-case-specific. Because of the decoupling of the modules through the *event broker*, the *complex event detection* component can be added if needed without further changes within the *semantic event-handling module*.

Semantic event log: Events that are detected by the *simple event detection* or the *complex event detection* component must be logged to be accessible for other applications. Additional functionality can be provided by other services within the CPS based on the *semantic event log*. An example, next to generating explanations within a ExpCPS, could be process mining of the events to build a behavior model of the physical system or detect discrepancies between existing models and reality. The difference between the *semantic event log* and a conventional event log is using semantic technology to provide additional context. The detected event is linked with concepts usually described in the *semantic model* of the ExpCPS. Note that this does not mean that a simple event table could not be used as an implementation (e.g., for performance or compatibility reasons). An approach of how such a simple table could also be enriched with context is the usage of ontology-based data access to map the event table at run-time [31]. Another way of creating a *semantic event log* could be linked data event streams [32]. Independent from the implementation details, a *log interface*, which is usually provided as APIs, has to be provided to other services to gain access to the information.

V. EVALUATION

This section presents a prototype implementation of the proposed event-handling service described in Section IV. In the context of this work, the implementation aims to achieve two things. First, it helps to illustrate the concepts by instantiating the architecture with concrete technologies. Second, applying the prototype to two concrete use cases will demonstrate the approach's feasibility. Lastly, the section also presents a qualitative evaluation of the architecture's quality characteristics using the ISO/IEC 25010:2023 standard [33] as an outline.

A. USE CASE DESCRIPTION

For the evaluation, two independent use cases are specified that are explained here in more detail.

1) SMART BUILDING

Fig. 3 depicts parts of a schematic drawing representing a floor in a smart office building use case. There is a large conference room (`Room_A`) with one thermometer in it (`T_A`). Occupants can control the desired room temperature using the set point `T_set`. The room houses a heating unit to react to changes in the room temperature. The controller can regulate the heating unit by setting `P_el` between 0 and 100 %. The conference room is equipped with two large windows that can be opened by occupants to allow fresh air into the room. The central server room of the building is situated on the opposite side of the conference room. This room houses the computers that operate the Building Management System (BMS) and Energy Management System (EMS). Furthermore, the EMS

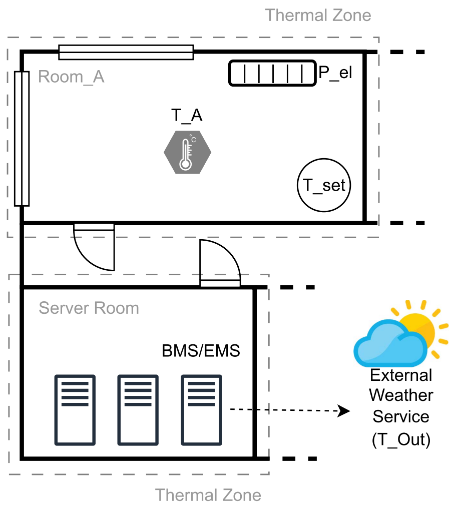

FIGURE 3. Smart building use case with available data points.

can access an external weather service via an API to obtain the current outside temperature. The aforementioned weather service is periodically queried and represented as a virtual sensor within the semantic event detection system (`T_Out`).

The objective of this use case is to detect whether the window was left open after a meeting in the conference room during the heating period. This circumstance causes the building to operate inefficiently, resulting in a higher energy bill for the operators. Different event sequences may indicate the open window. We will consider the following scenario to show the feasibility of the event detection architecture.

1) *Room A Cold* E_{RC}: The rule-based event detection triggers an event indicating the room is cold. Detecting this involves both temperature sensors and the current set point.
2) *High Power Consumption* E_{HP}: The model-based event detection triggers an event indicating that the power consumption is abnormally high. In this use case, the model estimates a maximum power input to the heater, given the set point and the outside temperature.
3) *Open Window* E_{OW}: After observing the previously mentioned events, the complex event detection triggers an event that points toward an open window. This reasoning step includes ensuring the absence of other events. For example, an increase in the set point could explain a similar pattern. However, the system can distinguish between these events because the complex event detection ensures an absence of an event that indicates a set point increase.

In addition to the open window, other events from the BMS and the EMS may be relevant to provide a suitable explanation. For example, suppose the conference room is booked for a meeting. The BMS signals an event when the meeting begins. During that meeting, attendees might want to get fresh air into the room and open the window. Consequently, an increase in power consumption from the heater is to be expected, in order to maintain the room at the desired setpoint

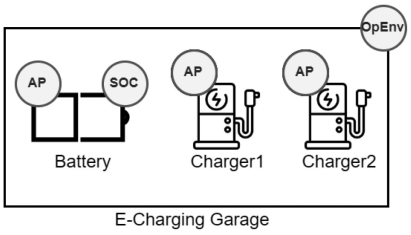

FIGURE 4. Smart grid use case with available data points.

temperature. However, if the same window open event is triggered after the EMS signals the event that the building has transitioned its operational state to night mode, it is likely that the window is not being opened intentionally. In that case, an alert should be displayed to the facility management explaining why they have to check on the conference room. Both systems, the EMS and BMS can easily hook into the semantic event-handling module via the event interface and start publishing events.

The system stores all previously mentioned events in the *semantic event log*. Each event published on the event broker contains semantic annotations that declare the source of the event. By analyzing these annotations over multiple connected events, users can identify that the open window event fired because of a drop in room temperature and high heater usage. While this use case is intentionally simple, it showcases how the different components interconnect.

2) SMART GRID

Fig. 4 shows a schematic representation of a smart e-charging garage for electric vehicles (EVs). The grid operator imposes an operating envelope on the charging facility, which is a power demand limit on the charging facility overall. The garage consists of two charge points as well as a battery, which can be used to perform peak shaving at critical times to avoid the violation of the operating envelope. Each charge point has a sensor attached to it to measure the active power (AP) consumed by the charger. The battery has a sensor measuring the AP and a sensor measuring the state of charge (SOC) of the battery. In addition, the operating envelope sensor (OpEnv)indicates the current demand limit imposed on the system.

In this use case, we aim to detect high charging events and operating envelope violations. High charging means that the charging demand of the charge points is higher than the operating envelope limit. This does not pose a problem as long as the access demand can be met by the installed battery. However, if the battery fails to provide the required extra power, an envelope violation occurs. Detecting the associated events in this use case is important to provide an explanation to a grid operator about why such a violation has occurred. In order to allow for such an explanation, the following events in the system need to be detected:

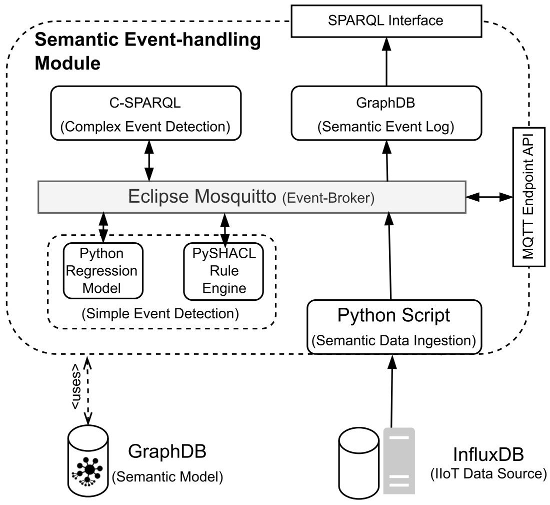

FIGURE 5. Concrete implementation of the *event-handling module* as proof of concept.

1) *Battery depleted* E_{BD}: An event indicating the battery is depleted is triggered by the event detection framework. The SOC of the battery is monitored to detect this event.
2) *HighEVCharging* E_{HC}: An event indicating that a certain e-charger is charging at higher power than the operating envelope would allow the garage to demand from the grid. The AP of the EV charger and the current operating envelope setpoint are considered to detect this event.
3) *EnvelopeViolation* E_{EV}: An event indicating the violation of the operating envelope setpoint. The violation is detected by comparing the sum of all consumers inside the e-charging garage (battery AP, charger 1 AP, charger 2 AP) to the operating envelope.

After storing all detected events in the *semantic event log*, the trace of events can be used to trigger an alert if an envelope violation was detected as well as create an explanation of why this violation occurred. Such an explanation is useful to the facility operator of the garage to examine potential weaknesses of their garage, but it could also be interesting to the grid operator to understand more in-depth why their imposed limits were violated and whether the facility operator could have prevented it.

B. PROOF OF CONCEPT IMPLEMENTATION

In this proof of concept implementation, only open-source software tools and frameworks were used to realize the proposed *semantic event-handling module*. The concrete implementation with the used tools is depicted in Fig. 5. It has to be noted that it only shows one possible realization of our generic framework to showcase the functionalities. Following our previous work in the CPS domain [34], [35], we mainly use semantic web technology standards, but the generic architecture is not limited to them. The source code of the prototype and a Docker Compose file to run the scenario, are provided on GitHub.[1]

Event broker: The event broker functionality was implemented by using MQTT with the Eclipse Mosquitto broker,[2] ensuring efficient and reliable event distribution. In the prototype, all components publish events in RDF serialized in the Turtle format to the event broker. This convention allows for transmitting semantic metadata straightforwardly, as each message is an Resource Description Framework (RDF) graph. Because these graphs should leverage ontologies defined in the *semantic model*, each involved party can understand the transmitted semantics of the data. As a result, each event can include additional context, like its source (e.g., Rule A), easily. The semantic event log can then make use of this auxiliary information. Enabling this ability makes this approach to data exchange crucial in the proposed event detection architecture.

Data ingestion: The data ingestion component obtains sensor readings from heterogeneous data sources and maps them into the common data format used on the event broker. We will demonstrate this concept by reading the sensor values from an InfluxDB[3] instance. InfluxDB is a versatile time-series database well-suited to manage timestamped data for real-time event handling and analysis efficiently. In the prototype, all sensors publish their readings to the InfluxDB. It is the job of the *data ingestion* component to query the semantic model for relevant sensors and their data source. The service must start obtaining updates from the sensor readings for each found entity. How the service facilitates these updates largely depends on the nature of the data source. This problem is purely an implementation issue, as the semantic model explicitly defines the characteristics of the data source. The data ingestion service must publish each observed data point on the event broker as RDF graph. The prototype uses the MQTT topic *events/sensors* to publish the observations. An example of such RDF data that is published for the smart building use case is shown in Listing 3. In the prototype, the mapping rules between the raw sensor readings and the RDF graph are hard-coded in the data ingestion module. To further increase the flexibility of this approach, the module could use general-purpose RDF mapping ontologies. One example is RML,[4] which is currently released as a W3C draft. This approach ensures that the data ingestion implementation can cater to the requirements of a wide range of industries.

Semantic model: The semantic web technology is an integral part of the prototype implementation. In this technology stack, concepts and entities are identified by uniform resource identifiers (URIs). Users can use prefixes to abbreviate long URI strings. Table 1 lists all namespace prefixes used in this work. In Fig. 6, the conceptual representation of the developed *SENSE Ontology* is shown. Classes and properties that are irrelevant for the proof of concept in the context of this article are excluded from the figure. The *SENSE Ontology* is an extension of the SOSA Ontology,[5] a lightweight general-purpose ontology to represent the interaction between entities in cyber-physical systems [36]. The SENSE extension of SOSA is mainly concerned with modeling events, their causal relations, and their detection. Therefore, events are modeled as results of *sosa: Observations* and *sosa: Procedures* are used to describe the applied event detection algorithm, as shown in Fig. 6.

TABLE 1. Used Namespace Prefixes

Prefix	Namespace
sh:	http://www.w3.org/ns/shacl#
xsd:	http://www.w3.org/2001/XMLSchema#
sosa:	http://www.w3.org/ns/sosa/
brick:	https://brickschema.org/schema/Brick#
ref:	https://brickschema.org/schema/Brick/ref#
unit:	http://qudt.org/vocab/unit/
s:	http://w3id.org/explainability/sense#
:	http://example.org/usecase/

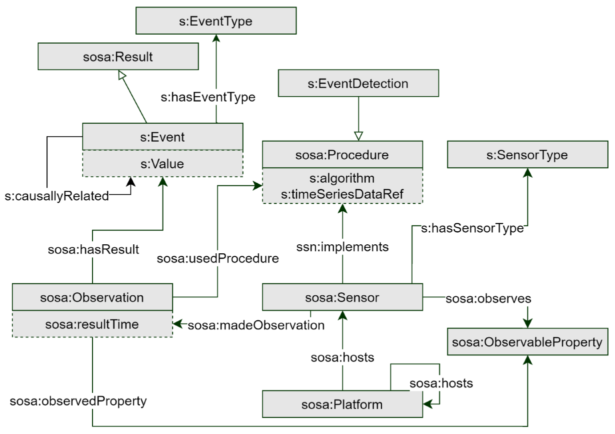

FIGURE 6. SENSE Ontology used for the event handling proof of concept.

For the proof of concept, especially the sensors and their connections are important to be represented in the semantic model. Each *sosa:Sensor* (i.e., `T_A`) in a system observes an *sosa:ObservableProperty* (i.e., temperature) and is hosted by a Platform (i.e., `Room_A`). An *sosa:Observation* is created when an event detection procedure (i.e., ruleBasedEventDetectionColdRoom), which is implemented at a sensor (i.e., `Room_A`) has detected an Event (i.e., `roomColdEvent123`).

For the storage of the semantic model and the semantic event log, GraphDB,[6] a triple store, was employed, enabling the association of events with semantic context.

Simple event detection: The simple event detection registers itself to the MQTT topic *events/sensors*. Thus, it receives incoming new values that are relevant to its operation. As these updates are RDF triples, the service can construct an

[1][Online]. Available: https://github.com/semanticsystems/semantic-event-handling
[2][Online]. Available: https://mosquitto.org/
[3][Online]. Available: https://www.influxdata.com/
[4][Online]. Available: https://rml.io/specs/rml/
[5][Online]. Available: https://www.w3.org/TR/vocab-ssn/
[6][Online]. Available: https://www.ontotext.com/products/graphdb/

RDF graph as a union of all received triples. This technique allows seamless integration into other approaches that leverage semantic web technology. The system overwrites old values from a sensor with a new observation to prevent the graph from growing indefinitely. The simple event detection service uses this graph to query the current sensor values. We will refer to this data structure as *observation graph*.

One approach to rule-based event detection, that is implemented in the prototype, is using the shapes constraint language (SHACL), as defined by the W3C. SHACL rules[7] provide sufficient expressiveness for calculations and comparing values. These capabilities allow the detection of, for example, the average temperature of a thermal zone dropping 4 °C below the set point. However, formulating more complex concepts, such as time constraints, is cumbersome. These restrictions pose no problem in this context because the prototype uses Shapes Constraint Language (SHACL) only for simple event detection. For the implementation, a Python SHACL engine[8] is used to create event triples from the observation graph. The system connects the detected event to the rule that triggered the event to enable tracing the reason behind an event instance. Furthermore, the timestamp of the sensor observation is attached to the event.

The model-based event detection component within the *semantic event-handling module* offers a versatile approach for detecting anomalies in a CPS. Various implementation strategies could be applied, with one potential method utilizing regression models for anomaly detection, which the prototype uses to demonstrate its principle functionality. The smart building use case demonstrates model-based event detection by using a simple regression model, that is trained on historical data.

Complex event detection: Contrary to simple event detection, complex event detection looks beyond the system's current state to detect trends. Furthermore, complex events sometimes rely on the absence of other events. For example, building operators expect higher-than-usual power usage if the occupants recently increased the set point of the thermal zone by a few degrees. Cao et al. [20] demonstrated that stream reasoning is an apt approach for detecting more complex event patterns while relying on knowledge from a semantic model. The presented prototype leverages stream reasoning over the simple event stream to detect the possibility of an open window. The prototype executes a C-SPARQL [37] query over the simple event stream to implement this behavior. Other implementations could also leverage CEP systems to implement this module.

Semantic event log: A separate GraphDB instance is used to implement the *semantic event log*. This instance stores all detected events sent via the event broker. The SPARQL API is used as an interface to retrieve the stored event information. The sensor values are not stored, to reduce the number of nodes in the *semantic event log* and are already available in the time series database.

```
1   :MyBuilding a brick:Building;
2       brick:hasPart :Room_A.
3
4   :ThermalZone a brick:HVAC_Zone;
5     brick:hasPart :Room_A;
6     brick:hasPoint :T_set.
7
8   :Room_A a brick:Room, sosa:Platform;
9     brick:hasPoint :T_A;
10    sosa:hosts :T_A;
11    brick:hasPoint :P_el;
12    sosa:hosts :P_el;
13    sosa:hosts :DetectOpenWindowSensor.
14
15  :T_A a brick:Zone_Air_Temperature_Sensor;
16    a sosa:Sensor;
17    brick:hasUnit unit:DEG_C.
18
19  :DetectOpenWindowSensor a sosa:Sensor;
20    sosa:implements :DetectOpenWindowQuery.
```

LISTING 1. Instantiation of the smart building use case in the semantic model.

TABLE 2. InfluxDB Database Schema With Converted Timestamp for Readability.

_meas	_time	_field	_value
room	2023-11-13 09:30:25	t_set	22.5
room	2023-11-13 09:30:30	t_a	24.5
room	2023-11-13 09:30:35	t_out	14.3
room	2023-11-13 09:40:00	t_a	19.5
...	...	...	...

1) SMART BUILDING IMPLEMENTATION

The SENSE ontology (cf., Fig. 6) provided the generic data model required to represent event handling data in the CPS domain based on the SOSA ontology [36]. For specific use cases, however, additional domain-specific ontologies might be necessary.

It is the case for the smart building use case, where the prototype uses the Brick [38] ontology to define the structure of the building in a precise manner. Listing 1 shows an excerpt of the semantic model, which is in line with Fig. 3. First, the example defines a building that consists of a single room `Room_A` (Lines 1 and 2). This room has one temperature sensor (`T_A`) (Line 9) and is part of an heating, ventilation, and air conditioning (HVAC) zone with a set point `T_set` (Lines 4–6). Furthermore, the Listing shows that the temperature sensor `T_A` published its readings in Celsius (Line 17). Lastly, there is a virtual sensor defined that is used for the complex open window event E_{OW} (Lines 19 and 20). A virtual sensor is required because the system cannot associate this event with a single regular sensor. This is contrary to, for example, associating a :RoomAColdEvent E_{RC} with the temperature sensor `T_A`. This example shows how RDF and Brick can be used to model the building use case. Naturally, operators can apply these capabilities to other domains using other ontologies.

[7][Online]. Available: https://www.w3.org/TR/shacl-af/
[8][Online]. Available: https://github.com/RDFLib/pySHACL

```
:influx_ref a ref:TimeseriesReference;
  ref:storedAt :influx_instance;
  ref:hasTimeseriesId "measurement=r,field=t_a".
:T_A ref:hasExternalReference :influx_ref.
```

LISTING 2. Mapping rule that defines the source of values for the Sensor T_A.

```
:T_set sosa:madeObservation :obs_T_set_243.
:obs_T_set_243 a sosa:Observation;
  sosa:hasSimpleResult 23.0;
  sosa:resultTime "2024-01-17T14:39"^^xsd:dateTime.
```

LISTING 3. Example of a sensor measurement published to the event broker.

Table 2 shows the data schema as the sensor values are stored in the database. Each sensor is configured to write its readings into the measurement *room*. Furthermore, each sensor has a configured unique *field*. The sensors use these two values to publish their readings into the InfluxDB.

The semantic model defines which sensors exist and where they publish their readings. In this prototype, the model links each sensor with the InfluxDB instance that hosts the sensor's readings. Listing 2 shows an example of such a link. Using a suitable ontology, the semantic model may provide additional information on the data source instance (e.g., IP address) and the published data (e.g., temperature unit). This additional information is not shown here.

The example outcome of the data ingestion is shown in Listing 3. It depicts a measurement that is published as a RDFgraph with additional context information on the event broker. This particular example shows a setpoint change of (`:T_set`) with its observation timestamp and the observed new value.

Listing 4 shows a SHACL rule (E_{RC}) from the prototype that is used for the simple event detection. The rule detects whether the sensor `:T_A` falls more than 4 °C below the set point (Line 25). This circumstance triggers an event that the room is cold. We want to highlight that SHACL can support more complex rules. The rule engine can evaluate rules not only on individuals [e.g., `:T_A` (Line 4)] but on whole classes of nodes. For example, a single SHACL rule can detect temperature drops in all thermal zones of a building. This article does not showcase such rules, as its focus is on the overall architecture of the event detection system.

Next to the rule-base approach, also a simple model-based event-detection has been implemented for that use case. Therefore, a regression model is trained with artificial data. In a real-life application, such a black-box model would have been trained with historical data instead and could use more sophisticated algorithms, such as neural networks, etc. The structure of this model is depicted in Fig. 7. It uses the values `:T_set`, `:T_out` as input variables for the regression and estimates an upper bound for the `:P_el` variable. If the actual value of `:P_el` exceeds the predicted upper bound, the system triggers an event to the event broker. While this is a toy example, more sophisticated and realistic models can similarly be integrated into the system.

```
1   :RoomIsColdRule
2     a s:EventDetection ;
3     a sh:NodeShape ;
4     sh:targetNode :T_A ;
5       sh:rule [
6         a sh:SPARQLRule ;
7         sh:construct """
8     # Prefixes
9     CONSTRUCT {
10      $this sosa:madeObservation ?newEventObs.
11      ?newEventObs a sosa:Observation ;
12        sosa:resultTime ?time ;
13        sosa:usedProcedure :RoomIsColdRule ;
14        sosa:hasResult ?newEvent .
15      ?newEvent a :RoomAColdEvent, s:Event .
16    }
17    WHERE {
18      $this sosa:madeObservation [
19        sosa:hasSimpleResult ?temp;
20        sosa:resultTime ?time
21      ].
22      :T_set sosa:madeObservation [
23        sosa:hasSimpleResult ?setpoint
24      ].
25      FILTER (?temp < (?setpoint - 4))
26          BIND(IRI(...) AS ?newEvent)
27          BIND(IRI(...) AS ?newEventObs)
28    }
29        """;
30      ].
```

LISTING 4. Simple SHACL rule for detection of the room cold event E_{RC}.

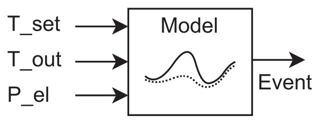

FIGURE 7. Regression model for the model-based event detection implementation, used to detect the high power consumption event E_{HP}.

```
:HeatingPowerConsumptionModel a s:EventDetection;
    sosa:hasInput :T_set, :T_Out, :P_el;
    sosa:hasOutput :PowerConsumptionNormalEvent,
                   :PowerConsumptionHighEvent.
```

LISTING 5. Interface definition of a the simple regression model.

The automated deployment of models requires information about the model's interface and the corresponding input and output values. In addition, contextual information provides additional value for model management and usage. Fortunately, the *semantic model* can incorporate both, provided there are apt ontologies. While established ontologies for describing simulation models exist, the prototype implements just a minimal version to keep the description as simple as possible.

Listing 5 shows one possible way of defining the regression model within the *semantic model*, based on the SOSA ontology. This snippet defines the model's inputs and which event shall be triggered if the actual `P_el` exceeds the predicted value. The model-based event detection instantiates and continuously evaluates the model on the observed data. How the service might implement this procedure depends on

```
1  REGISTER QUERY DetectOpenWindowQuery AS
2  CONSTRUCT {
3   :DetectOpenWindowSensor
4       sosa:madeObservation ?newEventObs.
5   ?newEventObs a sosa:Observation;
6        sosa:resultTime ?timestamp;
7        sosa:usedProcedure :DetectOpenWindowQuery;
8        sosa:hasResult ?newEvent.
9   ?newEvent s:causallyRelated ?roomAColdEvent;
10       s:causallyRelated ?powerConsumptionHighEvent.
11  }
12 FROM STREAM :simple_events [RANGE 120s STEP 10s]
13 FROM STREAM :complex_events [RANGE 120s STEP 10s]
14 WHERE {
15   ?roomAColdEventObs a sosa:Observation ;
16     sosa:resultTime ?roomColdTimestamp ;
17     sosa:hasResult [ a :RoomAColdEvent ] .
18   ?powerConsumptionHighEventObs a sosa:Observation ;
19     sosa:resultTime ?highPowerConsumptionTimestamp ;
20     sosa:hasResult [ a :PowerConsumptionHighEvent ].
21   FILTER NOT EXISTS {
22     # Room Temperature has not returned to normal
23   }
24   FILTER NOT EXISTS {
25     # Power consumption has not returned to normal
26   }
27   FILTER NOT EXISTS {
28     # No setpoint change
29   }
30   FILTER NOT EXISTS {
31     # No open window event yet triggered
32   }
33   BIND(... ?timestamp)
34   BIND(IRI(...) AS ?newEvent)
35   BIND(IRI(...) AS ?newEventObs)
36  }
```

LISTING 6. **Simplified C-SPARQL query for detecting the open window event E_{OW}.**

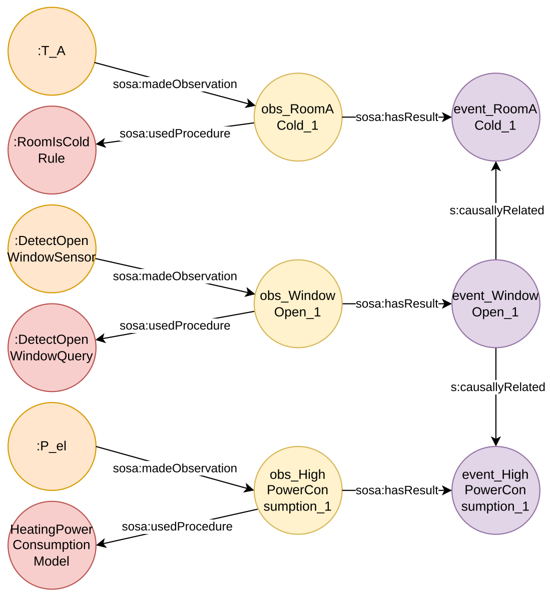

FIGURE 8. **Exploring high power consumption events.**

the model type? All events detected by the model refer to its URI in the semantic model, thus facilitating traceability. While the prototype implements the model evaluation directly, a more sophisticated implementation could define the model parameters in the knowledge base and automatically deploy the model to the event detection service.

It is important to note that the showcased regression algorithm represents just one plausible approach for model-based anomaly detection, serving as a basic demonstration in this context. It is beyond the scope of this article to investigate various approaches here rather than show the principle of a model-based event detection module.

Listing 6 depicts a simplified version of the C-SPARQL query that is used for *complex event detection*. Again, the resulting event is interconnected with the Uniform Resource Identifier (URI) of the query that triggers the event (Line 7). Furthermore, the example highlights that the query includes ensuring the absence of an event, as the system will not trigger an open window event E_{OW} if the set point was increased recently. Note that such queries may also use the static data from the semantic model, which truly showcases the power of this approach. Again, the focus of this article is to showcase its integration into the bigger picture, not to demonstrate the full potential of the employed technology. In our example, the users want to know why the power consumption of the heater was higher than expected. One explanation is the open window event E_{OW}, which was thoroughly discussed in this section. Another explanation could be that the thermostat on the heater is temporarily faulty, thus causing the heater to heat the room way beyond the set point `T_set`. The complex event detection service could detect this by looking for a hot room event E_{RH} instance in conjunction with a high power consumption event E_{HP} instance. This pattern would provide an alternative explanation for the high power consumption. To find a suitable explanation for a user, the *semantic event log* can now be explored. Fig. 8 visualizes parts of the *semantic event log* for our example. Based on the entries, the users are able to gain insights and deduce possible explanations for the behavior of an CPS, based on causality chains.

To find a possible explanation for our example, a user starts by listing all high power consumption events E_{HP} instances to gain insights into the power consumption behavior of the building. The user may see a complex event related to the event instance. For example, there is an open window event E_{OW} or alternatively a faulty thermostat eventinstance with a `s:causallyRelated` relationship. These relations within the events can be used to explain a reason why the power consumption was high in this particular instance. With this example it should be clear how the integration of automatic explanation generation algorithms can be developed that are based on this architecture. However, doing this is beyond this article's scope, as we only seek to demonstrate this architecture's ability to advance explainability in CPS.

2) SMART GRID IMPLEMENTATION

The structure for the proof of concept implementation for the smart grid use case is defined according to the *SENSE Ontology*, as shown in Fig. 6. In Listing 7, an excerpt of the semantic model is shown representing the system as shown in Fig. 4. It consists of the two e-chargers and a battery (Lines 1–5).

```
:Garage1 a sosa:Platform;
  sosa:hosts :OpEnv_Sensor1;
  sosa:hosts :Charger1;
  sosa:hosts :Charger2;
  sosa:hosts :Battery.

:Charger1 a sosa:Platform, :Charger;
  sosa:hosts :AP_Sensor1.

:Charger2 a sosa:Platform, :Charger;
  sosa:hosts :AP_Sensor2.

:Battery a sosa:Platform;
  sosa:hosts :AP_Sensor3;
  sosa:hosts :SoC_Sensor4.

:OpEnv_Sensor1 a sosa:Sensor;
  sosa:observes :OperatingEnvelope.

:AP_Sensor1 a sosa:Sensor;
  sosa:observes :ActivePower.

:SoC_Sensor4 a sosa:Sensor;
  sosa:observes :StateOfCharge.
```

LISTING 7. Instantiation of the smart grid use case in the *semantic model*.

The sensor data are sent to and stored in a timeseries database. Each sensor in the system is configured to write its measurement values to a unique field. The active power sensors send a value of the current active power to the database every minute. In the *semantic model*, all sensors of the system are defined as well as where they publish their readings. An example of a link between a sensor and its dedicated time-series database instance has been shown in the smart building use case (Listing 2, Table 2)

Listing 8 depicts a SHACL rule for detecting an envelope violation event E_{EV} of the whole charging garage. The rule implements this by gathering the AP from both chargers and the battery module (Lines 17–25). In addition, the current envelope is referenced (Lines 26–28). If the sum of the chargers and the battery is greater than the envelope, the system must issue an envelope violation event (Line 29). Note that the AP of the battery is positive while charging and negative while discharging energy. Thus, the battery can prevent an envelope violation by discharging energy to the garage.

Listing 9 depicts an envelope violation event E_{EV}, which was detected by a rule-based envelope violation detection procedure as it is published and stored in the *semantic event log*. The example shows the observation of the event, with the related rule that triggered the event, an observation timestamp and the observed value as well as the unit of the value (Lines 10 and 11).

Listing 10 shows a possible implementation of the high charging event detection E_{HC}. This event is fired if a single charger consumes more power than the envelope offers. Contrary to the other SHACL rules demonstrated in this work, this rule is a class rule. This is indicated by the sh:targetClass predicate instead of sh:targetNode (Line 3). The SHACL engine will apply this rule to every instance of the class :Charger. This feature allows users to detect all high charging events with a single rule instance, thus greatly reducing the number of rules. Class rules often require the use of the `$this` built-in so that users can refer to the current node.

```
1   :Garage1EnvelopeViolationRule
2     a sh:NodeShape;
3     a s:EventDetection;
4     sh:targetNode :Garage1;
5     sh:rule [
6       a sh:SPARQLRule;
7       sh:construct """
8     CONSTRUCT {
9     $this :madeObservation ?newEventObs.
10    ?newEventObs a sosa:Observation;
11         sosa:resultTime ?time;
12         sosa:usedProcedure :Garage1EnvelopeViolationRule;
13          sosa:hasResult ?newEvent.
14    ?newEvent a :Garage1_EnvelopeViolationEvent, s:Event.
15    }
16    WHERE {
17       :AP_Sensor1 sosa:madeObservation [
18         sosa:hasSimpleResult ?chrg1R
19       ].
20       :AP_Sensor2 sosa:madeObservation [
21         sosa:hasSimpleResult ?chrg2R
22       ].
23       :AP_Sensor3 sosa:madeObservation [
24         sosa:hasSimpleResult ?battR #negative if draining
25       ].
26       :OpEnv_Sensor1 sosa:madeObservation [
27         sosa:hasSimpleResult ?enveloperR
28       ]
29       FILTER ((?chrg1R + ?chrg2R + ?battR) > ?envelopeR)
30       BIND(... AS ?time)
31       BIND(IRI(...) AS ?newEvent)
32       BIND(IRI(...) AS ?newEventObs)
33    }
34         """;
35    ].
```

LISTING 8. SHACL rule for detecting an envelope violation event E_{EV} of the garage.

```
1   :OpEnv_Sensor1 a sosa:madeObservation
2   :obs_OpEnvViolated_1.
3
4   :obs_OpEnvViolated_1 a Observation;
5     sosa:usedProcedure :Garage1EnvelopeViolationRule;
6     sosa:hasResult :event_OpEnvViolated_1;
7     sosa:resultTime "2023-04-17T17:39"^^xsd:dateTime.
8
9   :event_OpEnvViolated_1 a s:Event.
10    s:value 15.2;
11    s:unit "kW beyond Limit".
```

LISTING 9. Example of an envelope violation event E_{EV} in the smart grid use case.

The detected events are stored in the *semantic event log* and are the basis for further evaluation of the system, such as analyzing why the envelope violation occurred. Manually, this analysis can be performed by querying all relevant event types in the system that co-occurred with the event to be analyzed in order to show a domain expert possible reasons for an event. In this use case, a user is interested in possible reasons for an envelope violation at the garage level. In Fig. 9, the events and their potential causal relation are shown. A high charging event is detected by the event detection procedure, which potentially causes an envelope violation that is detected by another procedure at the same time. The causal relation is currently based on domain knowledge.

```
:HighChargingRule
  a sh:NodeShape;
  sh:targetClass :Charger;
  sh:rule [
    a sh:SPARQLRule;
    sh:construct """
  CONSTRUCT {
    ?apSensor sosa:madeObservation ?newEventObs.
    ?newEventObs a sosa:Observation;
      sosa:resultTime ?time;
      sosa:usedProcedure :HighChargingRule;
      sosa:hasResult ?newEvent.
   ?newEvent a ?newEventClass, s:Event.
  }
  WHERE {
    ?garage sosa:hosts $this.
    ?garage sosa:hosts ?envelope.
    ?envelope sosa:observes :OperatingEnvelope;
      sosa:madeObservation [
        sosa:hasResult ?envelopeR
      ].
    $this sosa:hosts ?apSensor.
    ?apSensor sosa:observes :ActivePower;
      sosa:madeObservation [
        sosa:hasResult ?chrgR
      ].
    FILTER(?chrgR > ?envelopeR)
    BIND(... AS ?time)
    BIND(IRI(...) AS ?newEventClass)
    BIND(IRI(...) AS ?newEvent)
    BIND(IRI(...) AS ?newEventObs)
  }
     """;
  ].
```

LISTING 10. SHACL rule for detecting high charging event E_{HC} of any charger.

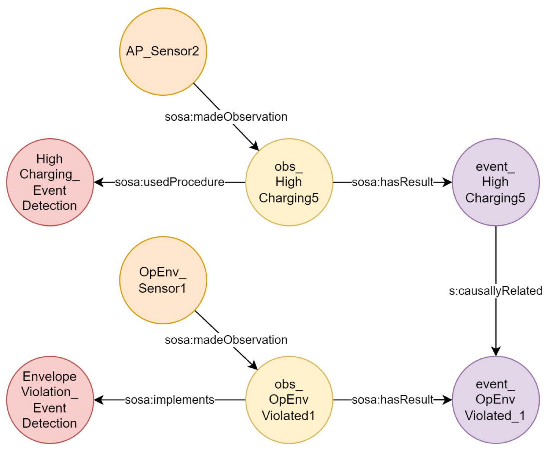

FIGURE 9. Exploring high power consumption events.

In future work, the detected events can be the input for a more advanced explanation engine to enable more complex and potentially automated explanations of events in the system.

C. QUALITATIVE SOFTWARE DESIGN EVALUATION

The ISO/IEC 25010:2023 standard [33] defines a quality model for software products and gives nine relevant characteristics. Furthermore, each characteristic is subdivided into subcharacteristics. This section discusses whether the presented architecture can promote the implementation of high-quality software products. We use the nine quality characteristics as an outline to structure the discussion. This approach aims to cover diverse quality aspects to identify potential shortcomings of the architecture. This discussion concerns the proposed architecture rather than the prototype developed in the feasibility study. In this section, phrases in cursive always refer to the mentioned quality characteristics, while phrases in quotes are direct citations from the standard.

Functional suitability covers whether the product can meet the "stated and implied needs of intended users" when the software product runs under the specified conditions. In this context, the need is i) monitoring the system for any occurring events, ii) communicating the detected events to another system (e.g., user interface), and iii) supporting another system in generating explanations. The architecture relies on the event detection services for i), thus delegating this responsibility to their implementations. The architecture supports these activities as the implementations can easily publish new events. Readers can find a discussion on the expressivity of the employed stream reasoning formalism C-SPARQL in [39]. If an even more expressive solution is required, users can employ a more powerful formalism or even a hand-crafted complex event detection solution. Point ii) is addressed by the event interface and the semantic event log. External applications can easily register to the unfiltered event stream or the semantic event log, which might do additional processing before publicizing the events. Lastly, iii) must still be evaluated by an explanation approach built on the proposed architecture. However, the implemented prototype showcases how the event graph can support explainability in CPS. The power of the RDF representation allows the published events to be self-descriptive, thus encouraging interoperability across system boundaries.

Performance efficiency contains subcharacteristics that argue about the capability to "perform its functions within specified time and throughput parameters." In the event detection context, this means detecting many events per time unit with low detection latency. By separating the detection mechanism between simple events (evaluation per relevant sensor reading) and complex events (evaluation per relevant triggered event), we aim to increase throughput as the simple event detection can use a more simple and efficient rule engine. Furthermore, the complex event detection is likely triggered less frequently as there are fewer simple events than sensor updates. However, empirical experiments with real-life systems must still provide evidence to support these hypotheses. Note that these experiments will require performance optimizations in the implementation, as the purpose of the proof of concept was to demonstrate the architecture's feasibility. Teymourian et al. [40] proposed relevant techniques for optimizing the integration of background knowledge and streaming data in semantic event processing. Furthermore, the event broker is a possible system bottleneck, as all messages must flow through it. However, to mitigate this issue, users can use scalable message brokers (e.g., Apache Kafka[9]). Horizontal scaling

[9][Online]. Available: https://kafka.apache.org/

might be relevant in event detection services when monitoring large systems. While we currently have no architectural support for distributing the rules to multiple instances, the architecture can be extended in this direction, especially if the system deploys rules automatically. Lastly, the system does not guarantee detection latencies.

Compatibility covers the "exchange of information with other products" and coexistence with other systems. The proposed architecture promotes interoperability between systems by using a formalized event representation in RDF using fitting ontologies. This technique allows the published events to be self-descriptive, thus encouraging interoperability across system boundaries.

Interaction capability refers to the exchange of "information between a user and a system via the user interface." We avoid discussing this aspect because the event detection system does not interact with users.

Reliability covers performing the "specified functions [...] for a specified period of time without interruptions and failures." Most aspects of this characteristic relate to the thoughts on scalability. Each component must be implemented with redundant instances to achieve high reliability. A mechanism of the horizontal scaling of the event detection services has yet to be considered in the architecture and is thus missing. Whether this is necessary is dictated by the context of the CPS. Lastly, one subcharacteristic is recoverability; this includes the capability to "re-establish the desired state of the system" after service interruption. Achieving the "desired state of the system," i.e., detecting events, may take a long time, as there is no central notion of state. Each event detection service builds its observation graph by listening to the appropriate topics. In the event of a service restart, a sensor that rarely publishes its value may inhibit the event detection due to an incomplete observation graph at the event detection services. Additional mechanisms would be necessary to enhance this subcharacteristic in the proposed architecture.

Security covers the capability to "protect information and data" and to "defend against attack patterns." First, the architecture assumes that (write) access to the semantic model, the event broker, the semantic event log, the event detection services code, and the sensor sources (e.g., InfluxDB) is only possible for authorized and trusted parties. Restricting access to the event broker is critical, as attackers could observe every published event and inject malicious values into the system. The event interface provided by the architecture is an additional attack vector in this context. Authentication and authorization mechanisms are therefore crucial and are commonly supported by brokers. However, in the prototype this security mechanisms are not used to reduce the additional overhead in the implementation.

Maintainability covers the "capability of a product to be modified by the intended maintainers." On the one hand, the proposed architecture promotes maintainability as it is very modular, and the components are reusable in different contexts and domains, especially if the system deploys rules and models automatically. Furthermore, the interface via the message bus allows isolated testing of modules while restricting possible side effects to other parts of the system if the sent messages are not changed. However, on the other hand, the distributed nature of the architecture makes it inherently more complex than monolithic approaches. This complexity manifests manifold. Proper integration tests require running multiple systems. Debugging the application requires tracing the message flow between various systems. While maintainers can address the latter point by looking into the semantic event log, bugs that prevent this from happening are challenging to track. Another complication when analyzing the system is the distributed state and the timing behavior of sensor data and event flows.

Flexibility is the "capability of a product to be adapted to changes in its requirements, contexts of use, or system environment." Many subcharacteristics (e.g., use on different hardware) depend on the concrete implementation. However, the communication between the modules is very flexible due to the expressiveness of the used RDF graphs. Event detection services can add supplementary information by adding triples to the published RDF graph. Furthermore, this approach is compatible with other rule engines or model-based approaches, given that there is an apt ontology for describing them. However, installing implementations of this architecture may require more work than a monolithic approach due to their distributed nature. The scalability subcharacteristic is again related to the thoughts on redundancy made earlier.

Safety covers the capability of a product to "avoid a state in which human life, property, or the environment is endangered." Such circumstances can happen if operators rely on an implementation of this architecture for safety-critical events. As the architecture does not guarantee detection latencies, implementations without further assurances are unsuitable for safety-critical contexts.

VI. DISCUSSION

The presented general architecture for ExpCPS, which follows the RAMI 4.0 layers, as shown in Section III, provides a set of minimal viable modules that are necessary to provide explainability to CPS. However, we are neglecting the feedback loop to the physical system that most CPSs have in place. Following the MAPE-K approach also for ExpCPS as proposed in the literature, we only target the Monitoring and Analysis part, in which the explanation happens. That means we are not concerned about how the system further handles the explanations, to make decisions and act on the physical system autonomously. In our case, we are just providing explanations to the user. We do that to reduce complexity and focus on the central part of our contribution, namely, the semantic event handling. Nevertheless, extending this general architecture by further components that would close the loop is possible and necessary for some use cases.

The *semantic event-handling module* provides a clear structure of the necessary components that must be implemented. It provides a technology-agnostic framework that can be followed to provide a solid foundation for building ExpCPS.

The used technology stack can be adapted to specific use cases. For example, the rule-based event detection could use simple if/then rule engines or use more sophisticated ones like, e.g., an signal temporal logic (STL) engine that provides more expressiveness if needed, or they can even be combined. The same holds for the other components within the semantic event-handling module.

Such a flexible combination is possible because of the proposed broker-based approach, which also introduces a separation of concerns between components within the *semantic event-handling module*, e.g., between simple and complex event detection. This makes our approach very flexible as components are decoupled and can be exchanged or added easily. This also enables containerization that allows horizontal and vertical scalability. However, it also introduces additional complexity that comes naturally with most distributed system approaches. Debugging can be especially tricky as developers must track messages across different processes.

One key feature of our approach is using a *semantic model* within the architecture to provide additional context to detected events. This is inherently important for deriving explanations and facilitates interoperability with other modules. Maintaining the semantic model introduces overhead. However, this additional work can be partly automated if engineering artifacts, e.g., a building information model, is available as a starting point for the semi-automatic instantiation. The *semantic model* is not intended to be used directly by users but instead used transparently in the back end.

The *semantic event-handling module* provides two main interfaces. The first one is via the *semantic event log*, which can be used for analysis of the past or explanations that are needed for certain events in retrospect. The second interface is directly provided by the event broker, which enables access to the events for the explanation module and potentially other modules at run-time.

The evaluation based on the case study with the implemented prototype showed the feasibility of the proposed semantic event-handling module. Especially the use of Semantic Web technology for the implementation, like RDF/OWL within the *semantic model* as well as in the exchanged messages provides benefits regarding extensibility and interoperability. Even if only simple approaches for rule-based and model-based event detection were implemented, they could be easily replaced with more sophisticated methods, like STL or models based on neural networks. Also, the Mosquito Broker is usually not sufficient for industrial deployment. Other enterprise-grade brokers should be used for a productive system that provides the needed performance and supports various security features we are not addressing in our prototypical implementation because of the overhead. However, these brokers typically support authentication and authorization at different granularity levels. Furthermore, our implementation lacks a solid fault and error handling strategy, as its purpose is only to show the feasibility of our approach. This holds also for performance optimization, as this is out of the scope of this work. Also, currently, we are not fully utilizing the semantic model as the single source of truth. Some of the model information that is stored in the semantic model is also hard-coded in the model implementation itself. The reason is to reduce the implementation effort. However, it does not influence the presented concept.

The qualitative evaluation of our approach was carried out based on the ISO/IEC 25010:2023 quality model and showed that our design caters to most of the nine characteristics. Here, mainly maintainability can be an issue, which is in the nature of the distributed approach that it can be more complex than in a monolithic approach.

VII. CONCLUSION AND FUTURE WORK

In this article, we propose a technology-agnostic, semantic event-handling component that can be integrated into a general ExpCPS architecture to provide a foundation for explainability. We provide the fundamental design characteristics that are necessary for such a semantic event-handling module, like a semantic model to provide context, an event broker for communication, and components for rule-based, model-based, and complex event detection. The case studies in the domain of smart building and smart grid demonstrate the feasibility of a prototypical implementation, based on an open-source software stack and are used for the evaluation of the proposed approach, in combination with the ISO/IEC 25010:2023 quality characteristics.

In future work, we will enhance the approach's functionality, efficiency, and applicability in real-world scenarios, especially in combination with algorithms for automatic explanation generation. Therefore, we focus on the semi-automatic generation of rules for the *Simple Event Detection* module, based on information derived from the semantic model, streamlining the rule generation and deployment process. In addition, the integration of more sophisticated rule logic, such as Signal Temporal Logic (STL), will be investigated to improve the system's ability to detect events accurately. In addition, the incorporation of a cosimulation runtime environment for model-based event detection would increase the flexibility and adaptability of the system. Furthermore, a comparison of various semantic technologies, with a particular focus on performance evaluation, is required. The establishment of benchmarks and recommendations would greatly facilitate the implementation of the proposed architecture. Finally, the iterative refinement of the prototype based on feedback and empirical testing in real-world applications is essential to validate its effectiveness, identify potential refinements, and ensure its suitability for diverse use cases and environments.

REFERENCES

[1] S. S. Jha, "An overview on the explainability of cyber-physical systems," in *Proc. Int. FLAIRS Conf.*, 2022, pp. 1–4, doi: 10.32473/flairs.v35i.130646. [Online]. Available: https://journals.flvc.org/FLAIRS/article/view/130646

[2] D. Bohlender, F. J. Chiyah Garcia, M. Köhl, C. Menghi, and A. Wortmann, "On explainability and its characterization," Explainable Software for Cyber-Physical Systems (ES4CPS): Report from the GI Dagstuhl Seminar 19023: Jan. 06-11 2019, Schloss Dagstuhl / Edited By: Joel Greenyer; Malte Lochau; Thomas Vogel, pp. 4–7, 2019, doi: 10.18154/RWTH-2020-01625. [Online]. Available: http://publications.rwth-aachen.de/record/782047/files/782047.pdf

[3] D. Luckham and W. R. Schulte, "Event processing glossary–Version 2.0," *Event Process. Tech. Soc. (EPTS)*, 2011. [Online]. Available: https://complexevents.com/2011/08/23/event-processing-glossary-version-2/

[4] C. Talcott, "Cyber-physical systems and events," in *Softw.-Intensive Syst. and New Comput. Paradigms*, M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, Eds. Berlin, Germany: Springer, 2008, pp. 101–115. [Online]. Available: http://link.springer.com/10.1007/978-3-540-89437-7_6

[5] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, Victoria, BC, Canada: Leanpub, 2022.

[6] M. Sadeghi, V. Klos, and A. Vogelsang, "Cases for explainable software systems: Characteristics and examples," in *2021 IEEE 29th Int. Requirements Eng. Conf. Workshops*, 2021, pp. 181–187, 2021, doi: 10.1109/REW53955.2021.00033. [Online]. Available: https://ieeexplore.ieee.org/document/9582300/

[7] S. Gregor and I. Benbasat, "Explanations from intelligent systems: Theoretical foundations and implications for practice," *MIS Quart.*, vol. 23, no. 4, 1999, Art. no. 497, doi: 10.2307/249487. [Online]. Available: https://www.jstor.org/stable/249487?origin=crossref

[8] M. Törngren and U. Sellgren, "Complexity challenges in development of cyber-physical systems," in *Principles of modeling: Essays dedicated to Edward A Lee on the occasion of his 60th birthday*, Cham, Switzerland: Springer, pp. 478–503, 2018.

[9] Z. Huang and Y. Wu, "A survey on explainable anomaly detection for Industrial Internet of Things," in *2022 IEEE Conf. Dependable Secure Comput.*, 2022, pp. 1–9, doi: 10.1109/DSC54232.2022.9888874.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?": Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1527–1535, doi: 10.1609/aaai.v32i1.11491. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11491

[12] S. M. Lundberg et al., "A unified approach to interpreting model predictions," in *Adv. in Neural Inf. Process. Syst.*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2017, [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[13] P. R. Aryan et al., "Explainable cyber-physical energy systems based on knowledge graph," in *Proc. 9th Workshop Model. Simul. Cyber-Phys. Energy Syst.*, 2021, pp. 1–6, doi: 10.1145/3470481.3472704.

[14] M. Blumreiter et al., "Towards self-explainable cyber-physical systems," in *2019 ACM/IEEE 22nd Int. Conf. Model Driven Eng. Languages Syst. Companion*, 2019, pp. 543–548, [Online]. Available: https://ieeexplore.ieee.org/document/8904796/, doi: 10.1109/MODELS-C.2019.00084.

[15] "An architectural blueprint for autonomic computing," IBM, Armonk, NY, USA, Tech. Rep., Jun. 2005. [Online]. Available: https://users.cs.fiu.edu/~sadjadi/Teaching/Autonomic%20Grid%20Computing/CIS-6612-Summer-2006/AC-Blueprint-WhitePaper-V7.pdf

[16] J. Moreno Molina, J. Ferrer Garcia, and C. Kuchkovsky Jimenez, "Archer: An event-driven architecture for cyber-physical systems," in *2018 IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2018, pp. 335–340. [Online]. Available: https://ieeexplore.ieee.org/document/8605801/, doi: 10.1109/UCC-Companion.2018.00077.

[17] C. E. B. López, "Real-time event-based platform for the development of digital twin applications," *Int. J. Adv. Manuf. Technol.*, vol. 116, no. 3-4, pp. 835–845, 2021, doi: 10.1007/s00170-021-07490-9.

[18] R. Verma, J. Brazauskas, V. Safronov, M. Danish, I. Lewis, and R. Mortier, "RACER: Real-time automated complex event recognition in smart environments," in *Proc. 29th Int. Conf. Adv. Geographic Inf. Syst.*, 2021, pp. 634–637, doi: 10.1145/3474717.3484270.

[19] M. Ma, L. Liu, Y. Lin, D. Pan, and P. Wang, "Event description and detection in cyber-physical systems: An ontology-based language and approach," in *2017 IEEE 23rd Int. Conf. Parallel Distrib. Syst.*, 2017, pp. 1–8, doi 10.1109/ICPADS.2017.00012.

[20] Q. Cao, F. Giustozzi, C. Zanni-Merk, F. D. B. D. Beuvron, and C. Reich, "Smart condition monitoring for industry 4.0 manufacturing processes: An ontology-based approach," *Cybern. Syst.*, vol. 50, no. 2, pp. 82–96, 2019, doi: 10.1080/01969722.2019.1565118.

[21] F. Giustozzi, J. Saunier, and C. Zanni-Merk, "Abnormal situations interpretation in industry 4.0 using stream reasoning," *Procedia Comput. Sci.*, vol. 159, pp. 620–629,2019, doi: 10.1016/j.procs.2019.09.217 [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050919314012

[22] F. Giustozzi, J. Saunier, and C. Zanni-Merk, "A semantic framework for condition monitoring in industry 4.0 based on evolving knowledge bases," *Semantic Web*, vol. 15, pp. 1–29, 2023, doi: 10.3233/SW-233481.

[23] R. Heidel, *Industrie 4.0: The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 Component*. Berlin, Germany: Beuth Verlag GmbH, 2019.

[24] D. Schachinger, W. Kastner, and S. Gaida, "Ontology-based abstraction layer for smart grid interaction in building energy management systems," in *2016 IEEE Int. Energy Conf.*, 2016, pp. 1–6, doi: 10.1109/ENERGYCON.2016.7513991.

[25] Context information management (CIM); NGSI-LD information model. RGS/CIM-006v121, European Telecommunications Standards Institute, Sophia Antipolis, France 2023.

[26] G. Klyne, "Resource description framework (RDF): Concepts and abstract syntax," 2004. [Online]. Available: http://www.w3.org/TR/rdf-concepts/

[27] "OWL 2 web ontology language document overview (second edition) - W3C recommendatio," *World Wide Web Consortium*, 2012. [Online]. Available: https://www.w3.org/TR/owl-overview/

[28] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R.van de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *Proc. 7th Workshop Linked Data Web*, Apr. 2014. [Online]. Available: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf

[29] Q.-D. Nguyen, S. Dhouib, J.-P. Chanet, and P. Bellot, "Towards a Web-of-Things approach for OPC UA field device discovery in the industrial IoT," in *2022 IEEE 18th Int. Conf. Factory Commun. Syst.*, 2022, pp. 1–4, doi: 10.1109/WFCS53837.2022.9779181.

[30] "Functional mock-up interface specification, STD." Accessed: Sep. 2, 2024. [Online]. Available: https://fmi-standard.org/docs/3.0.1

[31] O. Corcho, F. Priyatna, and D. Chaves-Fraga, "Towards a new generation of ontology based data access," *Semantic Web*, vol. 11, no. 1, pp. 153–160, 2020, doi: 10.3233/SW-190384 [Online]. Available: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-190384

[32] P. Colpaert, "Linked data event streams," 2023. [Online]. Available: https://w3id.org/ldes/specification

[33] ISO Central Secretary, "Systems and software engineering–systems and software quality requirements and evaluation (SQuaRE) – Product quality model, international qrganization for standardization Std.," Nov. 2023. [Online]. Available: https://www.iso.org/standard/78176.html

[34] M. Sabou, S. Biffl, A. Einfalt, L. Krammer, W. Kastner, and F. J. Ekaputra, "Semantics for cyber-physical systems: A cross-domain perspective," *Semantic Web*, vol. 11, no. 1, pp. 115–124, 2020.

[35] G. Steindl and W. Kastner, "Transforming OPC UA information models into domain-specific ontologies," in *2021 4th IEEE Int. Conf. Ind. Cyber- Phys. Syst.*, 2021, pp. 191–196.

[36] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "SoSA: A lightweight ontology for sensors, observations, samples, and actuators," *J. Web Semantics*, vol. 56, pp. 1–10, 2019.

[37] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, "C-SPARQL: A continuous query language for RDF data streams," *Int. J. Semantic Comput.*, vol. 4, no. 1, pp. 3–25, 2010.

[38] B. Balaji et al., "Brick: Towards a unified metadata schema for buildings," in *Proc. 3rd ACM Int. Conf. Syst. Energy-Efficient Built Environments*, 2016, pp. 41–50.

[39] H. Stuckenschmidt, S. Ceri, E. D. Valle, and F. V. Harmelen, "Towards expressive stream reasoning," in *Semantic Challenges Sensor Netw.*, 2010.

[40] K. Teymourian, M. Rohde, and A. Paschke, "Fusion of background knowledge and streams of events," in *Proc. 6th ACM Int. Conf. Distrib. Event-Based Syst.*, 2012, pp. 302–313, doi 10.1145/2335484.2335517.

GERNOT STEINDL (Member, IEEE) received the B.S. and first M.S. degrees in electrical engineering from Technical University Wien, Vienna, Austria, in 2013, the second M.S degree in building technology from the University of Applied Science Burgenland, Eisenstadt, Austria, in 2016, and the Ph.D. degree in computer science from Technical University, in 2021.

From 2015 to 2018, he was a Researcher with Research Burgenland GmbH. In 2018, he joined the Institute of Computer Engineering, Technical University Wien as a Predoc Researcher. Since 2022, he has been a Postdoc Researcher with the Research Unit Automation Systems, Technical University Wien. His research interests include Industrial Internet of Things, semantics in automation systems, and digital twins.

TOBIAS SCHWARZINGER received the B.Sc. and M.Sc. degrees in computer science in 2020 and 2023, respectively, from TU Wien, Vienna, Austria, where he is currently working towrad th Ph.D. degree in informatics with the Institute of Computer Engineering.

Since 2023, he has been an University Assistant with TU Wien. His research interests combine ideas from knowledge engineering and formal methods in the context of cyber-physical systems.

KATRIN SCHREIBERHUBER received the bachelor's degree in business, economics, and social sciences from WU Wien, Vienna, Austria, in 2019, and the master's degree in data science with TU Wien, Vienna, in 2023.

There, she focused on Machine Learning, Visual Analytics and Semantic Technologies. Since 2023, she has been a Research Assistant with WU Wien. Her research focuses on semantics-enabled explainability of cyber-physical systems in the context of smart grids and smart buildings.

FAJAR J. EKAPUTRA received the bachelor's degree in informatics and master's degree in electrical engineering from the Institut Teknologi Bandung, Bandung, Indonesia, and the doctorate degree in informatics from TU Wien,Vienna, Austria, in 2018.

He is currently an Assistant Professor with the Institute for Data, Process, and Knowledge Management, WU Wien, Vienna. He has coauthored more than 70 peer-reviewed scientific publications and actively involved in research communities as scientific and organizing committees in various top-tiered conferences, such as ISWC, ESWC, SEMANTiCS, and EKAW. His research interest is mainly in the semantic web, knowledge (graphs) engineering, and their interaction with machine learning approaches, and applying such hybrid semantic web and machine learning systems in various application domains, including cyber-physical systems and materials engineering.