Received February 26, 2022, accepted March 6, 2022, date of publication March 14, 2022, date of current version March 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3159715

Architecture of an Artificial Intelligence Model Manager for Event-Driven Component-Based SCADA Systems

ZLATAN SIČANICA[1], STJEPAN SUČIĆ[1], AND BORIS MILAŠINOVIĆ[2], (Member, IEEE)
[1]Končar—Digital, 10000 Zagreb, Croatia
[2]Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia

Corresponding author: Zlatan Sičanica (zlatan.sicanica@koncar.hr)

This work was supported by the Advanced Tools Towards Cost-efficient Decarbonisation of Future Reliable Energy Systems (ATTEST) Project through the European Union's Horizon 2020 Research and Innovation Program under Grant 864298.

ABSTRACT This paper analyzes Hat, an open-source framework for developing event-driven component-based SCADA applications, and discusses possibilities to add various analytical tools to such platforms. As a part of the contribution, an open-source component called Artificial Intelligence Model Manager (AIMM) has been developed and integrated into a Hat-based SCADA platform. AIMM is extensible through various plugins, allowing the addition of various models for advanced analytics e.g., machine learning tools, statistical tools, etc. The paper describes AIMM architecture and provides a use case in which state estimation was performed in a medium-voltage distribution grid. This case study demonstrates that it is possible to extend component-based SCADA systems with components for advanced analytics with minimal fundamental system changes.

INDEX TERMS Artificial intelligence, power system analysis computing, SCADA systems, software architecture.

I. INTRODUCTION

Supervisory Control and Data Acquisition (SCADA) systems have undergone many changes over the various generations of software development trends. First SCADA systems were among the earliest large-scale software systems, as they typically managed critical infrastructure. Over time, new software development and architecture principles were applied to incrementally improve these systems, introducing changes and evolving the common SCADA development practices. One of these architecture principles, which is the focal point of our research, is the event-driven component-based architecture.

An event-driven software architecture consists of independent event consumers and producers communicating via a common event bus [1]. The central unit of information is an event – a data element that indicates a change in the producer. Producers send events via TCP, function calls, or some other communication method, to the event bus – an independent actor that forwards registered events to its consumers. Consumers are actors that receive and process events. Consumers and producers are not necessarily separate entities; a single entity may act as a consumer of one event type and a producer of another.

Component-based systems are systems whose main functionality is achieved by dividing the system into smaller, loosely coupled components that focus on their specific roles and are usually able to communicate with each other via method calls or message passing [2]. Each specialized component implements a small subset of the main system functionality. In the context of SCADA systems, these components could include communication or graphical user interfaces (GUI) and various data processing modules. Combining this approach with the event-driven architecture, a SCADA system developed this way would consist of several independent processes that communicate with each other via events and perform various SCADA functions.

Event-driven component-based architecture principles have seen an increase in use over recent years because they offer many benefits:

- They are distributed, which increases their robustness and allows implementation of redundant individual components.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhouyang Ren.

- The addition of new functionality amounts to adding a new component which implements that function since components are inherently less coupled than in alternative kinds of architecture.
- The increase in Internet speed enables better communication between components and the rise of trends such as cloud computing and microservices improving service availability without significantly impacting system latency or security.

Event-driven component-based architecture principles are not without drawbacks, such as higher data flow complexity and greater communicational overhead. Despite these shortcomings, they find application in various industrial systems, including the example used in our case study, power grid management.

Another group of techniques, focused on inference from historical data, has seen a large increase in usage. These mainly include methods from the field of machine learning, such as neural networks, various types of linear models, or deep learning. They find application in a variety of industries that have nothing to do with SCADA, e.g. medicine [3], computer vision [4], marketing [5], etc. The field of statistics has also seen an increase in use since the improvements in computer hardware have allowed more computationally challenging methods to be implemented. Here we refer mainly to the increasing use of Bayesian statistics, which allows inferred processes to be modeled using methods such as structured time series, or Gaussian process regression, and optimized using approaches such as Markov chain Monte Carlo or variational inference [6]. Finally, in these data-driven techniques, we also include methods from the field of soft computing, such as fuzzy logic or evolutionary computation. These are also widely used, either as stand-alone solutions to practical problems [7] or as part of hybrids with other data-driven approaches (e.g., neural networks optimized with a genetic algorithm) [8].

Hereafter, we refer to above-discussed techniques as advanced analytics. Apart from the aforementioned applications, all these techniques have also been successfully applied to SCADA systems. This mainly includes solving problems such as prediction, fault, or attack detection. However, in our work, we apply them to solve the problem of state estimation of an electrical power grid.

Electric power grids consist of a large number of different interconnected elements, that serve to transmit and transform electric current. Monitoring them and tracking measurements is an important task of the SCADA systems that manage them. Since the number of measuring devices that can be placed at different points of the system is limited, advanced analytical methods are used to estimate the most probable physical state of the devices in the network. The elements whose states are estimated are power buses, lines, and transformers, while the states of elements consist of voltage magnitudes, voltage angles, active powers, reactive powers, and currents. Performing accurate state estimation of a power system has many advantages, such as detecting outages, overloads, or erroneous measurements. It can also serve as a first step in solving the problem of load estimation and forecasting. In our case study, we use a method based on weighted quadratic loss optimization, but other advanced analytical methods such as deep learning or Bayesian statistics can be used.

A. MOTIVATION AND RESEARCH CHALLENGES

In this paper, we analyze an existing event-driven component-based SCADA system developed using an open-source platform, Hat [9]. The Hat platform is publicly available and we use it as an example of how a generic distributed component-based SCADA system would work, focusing mainly on existing components and how they communicate with each other. The SCADA system is configured to monitor the measurements of a medium-voltage distribution network. Further on, we propose the addition of a new component, Artificial Intelligence Model Manager (AIMM), which specializes in advanced analytics. The component is also open-sourced and publicly available [10]. This component connects to the SCADA's event interface, receives events containing reported measurements, and performs state estimation, returning the estimated measurements to the SCADA system so they can be shown on its user interface. For the most part, AIMM is seamlessly integrated and respects all protocols and interfaces of the event-driven component-based system.

Event-driven SCADA systems with distributed, component-based architecture have the advantage of new components being easily added without changing the rest of the system. Since there is motivation to extend such tools with advanced analytics, the question becomes how such a requirement can be met and what are the architectural implications of such a solution. We analyze the data model of the original SCADA system to show what additional components and event types should be added. Another point of interest is the communication overhead that results from the fact that the AIMM component communicates via Transmission Control Protocol (TCP).

An additional major research challenge is the development of the AIMM system as a whole, its components, and the setup of the state estimation case study. We are trying to create a general-purpose tool with interfaces that are as generic as possible and not just tailored to the specific case study. This means that we leave options open for advanced analytics applications other than state estimation.

The final challenge is the case study itself, as it serves as a proof-of-concept for our solution. We are trying to implement it with minimal changes to the underlying system. Our goal is to obtain estimations in soft real-time in response to measurement changes in the system.

B. MAIN CONTRIBUTIONS

There are several contributions presented in this article. The first one is the design of a modular architecture for integrating advanced analytics into SCADA systems. It shows the main entities a component should contain and how they should be interconnected. Components implementing this architecture can be used in any SCADA system that has an interface allowing access to the data of the monitored process.

The second contribution is the implementation of such an architecture. We describe an existing SCADA system based on the Hat platform and show how such a system can be extended with AIMM, a concrete implementation of the previously mentioned architecture. In addition, AIMM is integrated into an event-driven component-based SCADA system, and there is little research analyzing the integration of advanced analytics into such an architecture to solve problems the power industry faces.

The rest of the paper is structured to describe the component development approach. First, we provide an overview of the relevant research on event-driven component-based SCADA systems, the application of advanced analytics in the energy industry, concrete approaches to integrating these methods into SCADA systems, and we finish with an overview of Hat's architecture. The second section describes the AIMM component. We cover its internal architecture and how it can be integrated into the Hat environment. The third section focuses on the specific state estimation case study and describes the problem setup, the way the measurements are simulated, the results, and the discussion of potential improvements. Finally, we present conclusions and topics for potential future research.

II. RELATED WORK

Many successful research use cases are demonstrating the use of advanced analytics with SCADA systems. In this section we provide an overview of the research, focusing on the following:

- successful application of event-driven component-based SCADA systems
- the application of advanced analytics to data derived from SCADA systems, with a focus on power systems
- algorithms used to solve the state estimation problem
- examples of the integration of intelligent agents into SCADA systems

A. EVENT-DRIVEN SCADA SYSTEMS

Event-driven component-based SCADA is based on the use of events as general, central data structures that transport information and indicate changes between system components. The specialized components implement various subsets of the SCADA functionality and communicate with each other using these events. This concept has seen previous use, Beck *et al.*, for instance, provide an overview of a LabVIEW-based control system that is event-driven and has SCADA capabilities [11]. Approaches that implement SCADA as a distributed event-driven system are also known. For example, an actor-based framework has been presented that allows software engineers to create modal functional blocks that follow the event-driven principles and communicate with other components via labeled messages [12]. There is also research that addresses the security of such systems and proposes a security implementation scheme of a wide-area event-driven SCADA system [13].

B. APPLICATION OF ADVANCED ANALYTICS TO POWER SYSTEMS

There are many examples of the use of advanced analytics on data coming from SCADA systems. In this paper, we focus on SCADA systems used in the context of power systems. One problem that is often solved by advanced analytics is the short-term prediction of a measurement. Several review articles show how advanced analytics can be used to predict energy consumption in smart grids [14], [15], buildings [14], or power systems in general [16]. Power generation is also frequently predicted, usually in less reliable renewable energy systems, such as wind farms [17]. Common algorithms mentioned in these reviews are neural networks, support vector machines (SVM), or various hybrid models.

Another common application of advanced analytics in SCADA systems focuses on improving the resilience of the system to failures and cyber-attacks. There is research showing machine learning can improve fault detection in wind farms [18] and substations [19]. These reviews show the use of neural networks, machine learning, and soft computing. Another heavily researched topic is the security of SCADA systems. This involves analyzing how advanced analytical algorithms can be used to prevent data injection [20], execution of unauthorized commands [21] or man-in-the-middle, denial-of-service (DOS), and replay attacks [22]. Ferrag *et al.* provide a comprehensive review of cyber-security solutions specifically for fog-based SCADA systems [23]. The proposed solutions mostly utilize neural networks, SVMs, and various hybrid models. These are the most common use cases for advanced analytics in SCADA. However, some publications also show successful applications of these techniques in various specialized case studies that do not fall under the previously described categories, including the focus point of the case study of this paper, state estimation.

C. CURRENT TRENDS IN STATE ESTIMATION

In this paper, we see the advanced analytics component we implemented integrated into an existing, event-driven SCADA system solving the state estimation problem. State estimation is a process of using a set of available measurements in a system to approximate the state of system components for which exact measurements are not available [24]. This approach can also be used to indicate potentially inaccurate measurements, such as when a measurement differs greatly from the estimate of the same element. Some of the methods used are more traditional, such as the weighted least squares (WLS) algorithm or projection statistics [24], [25], but there have also been successful applications of more computationally intensive approaches such as autoencoders [26], Markov chain Monte Carlo [27], and convolutional graph networks [28].

D. INTEGRATION OF ADVANCED ANALYTICS TO SCADA SYSTEMS

A topic separate from the domain-specific problems and the methods used to solve them, and an important focal point of our research, is the integration of these solutions into SCADA systems. Previous research typically took the approach of viewing SCADA as a multi-agent system and treating the advanced analytic operations as actions of intelligent agents [29]. They specify which interfaces these agents can access and which actions they can perform. Standardized multi-agent architecture such as Condition Monitoring Multi-agent System and Protection Engineering Diagnostic Agents have also been analyzed, highlighting the advantages and disadvantages of both systems and examining their interoperability. Recent research also shows similar conclusions on what data sources the agents should be able to access. These are usually the database and the process data received from devices [30], [31]. However, some approaches attempt to integrate intelligent agents in a more seamless way, where the underlying system is unaware that it is being analyzed by an agent and the agent is unaware of the domain of the system it is observing but only reacts to changes it perceives, pointing out anomalies [32]. The approach in our research is somewhere in the middle, where the advanced analytics component is not completely independent of the semantics of the data it receives but is still separated from the SCADA system by a web interface, which is more common in IoT-based SCADA systems [33], [34].

Cloud and fog-based SCADA systems also have great potential for easy integration of advanced analytics. One of the applications explored is the integration of intrusion detection systems into such an architecture. Numerous research results show the successful integration of supervised learning agents into the fog and cloud-based SCADA systems [23], [35]. There are also examples of more autonomous components based on reinforcement learning used in cloud-based SCADA systems [36]. Cloud systems are typically tasked with processing large amounts of data, and a review by Pliatsios *et al.* lists how Big Data analytics can be integrated into such systems, especially in the context of intrusion detection [37].

When it comes to presenting the results of advanced analytics to the users of the system, different approaches are proposed. Some advocate for the development of a separate application, while others, including our research, propose solutions that use the existing SCADA user interface to display predictions and alerts as various graphs or system alarms.

These papers show that advanced analytics can be integrated into SCADA systems in a variety of ways, and in Table 1 we compare the aspects of interest to our research. In the last row, we also list our method.

In summary, related research shows the following:

- Event-driven SCADA systems have a practical application.
- Power grid data, collected by SCADA systems, can be used to add value through advanced analytic techniques.
- The state estimation problem can be solved with advanced analytics.
- Advanced analytics are integrated into SCADA systems as intelligent agents, with access to the database and process data.

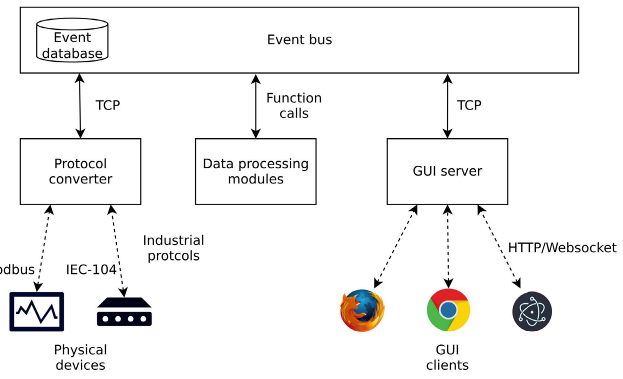


FIGURE 1. Interpretation of a Hat-based application architecture.

E. EVENT-DRIVEN COMPONENT-BASED SCADA SYSTEM

One of the main focuses of this paper is Hat – an open-source framework for development of event-driven component-based SCADA applications [38]. SCADA systems developed using this framework have a distributed architecture with components connected via an event bus. Figure 1 shows what the architecture of a generic Hat-based application could look like if it used all of the available components. It can be divided into four abstractions – event bus, data processing modules, protocol converter, and GUI server.

Figure 2 also shows an example of interaction between the three primary logical components. In this example, the system is configured to receive data from two separate devices, one of which communicates via the IEC 104 protocol [39], the other via Modbus [40], and assign their values to separate abstract process points. The process point values are then manipulated by the data processing modules in various ways. Finally, the process point values all end up on the GUI server, which notifies its clients of the changes using the WebSocket protocol [41]. In the following subsections, we describe the primary components in more detail, often referring to the sequence diagram as an example. We also compare the architecture to the Integrated Devices Open Management (IDOM) SCADA, as it is an example of a practically used and documented SCADA system [42].

1) EVENT BUS

The event bus is a central component that allows other components to connect, subscribe to, and register events. Its event data structures consist of the following properties:

- Event type – list of string values used to determine what the event refers to

TABLE 1. Overview of related work demonstrating practices for integrating advanced analytics.

Reference	SCADA architecture	Problem	Advanced analytics workflow	Integration method	Result presentation
Catterson et al. [29]	Monolith	Fault detection	Machine learning	Internal component	System interface
Uraikul et al. [30]	Monolith	Monitoring, control, diagnostics	Machine learning, fuzzy logic	Internal component	System interface
Leahy et al. [31]	Monolith	Fault detection	Machine learning	Internal component	SCADA interface
Baldoni et al. [32]	Generic distributed	Anomaly detection	Statistical method	Separate component	Separate interface
Yadav et al. [33]	IoT-based	Intrusion detection	Machine learning	Separate component	Separate interface
Fazlollahtabar [34]	IoT-based	Assembly automation	Probabilistic method	Autonomous devices	Implicit, device actions
Ferrag et al. [23]	Fog-based	Intrusion detection	Supervised learning	Separate component	Separate interface
Khorsand et al. [35]	Cloud-based	Resource provisioning	Fuzzy logic	Separate component	Implicit, agent actions
Ghobaei-Arani et al. [36]	Cloud-based	Resource provisioning	Reinforcement learning	Internal component	Implicit, agent actions
Pliatsios et al. [37]	IoT-based	Intrusion detection	Big Data analytics	Separate component	SCADA interface
Our method	Event-driven	Power grid state estimation	Statistical method	Separate component	SCADA interface

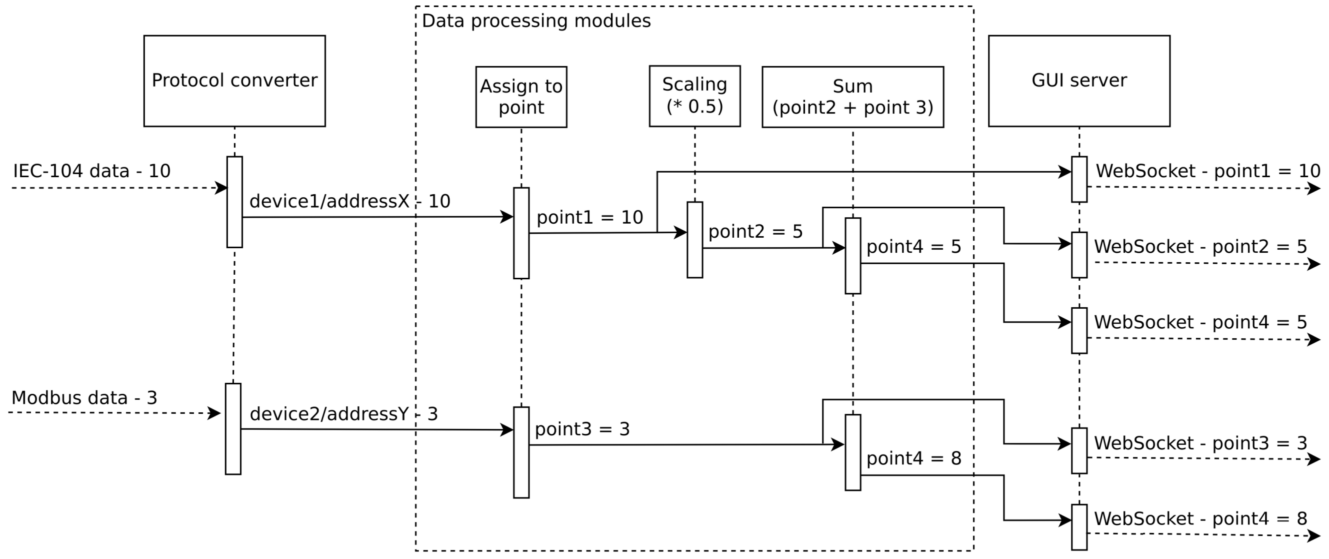


FIGURE 2. Sequence diagram of messages that can traverse a Hat-based system. Dashed arrows represent direct messages in various protocols, while full arrows represent events that originate from their creator, traverse the event bus, and reach their consumers.

- Timestamp – the time when the event was registered (set by the event server)
- Source timestamp – optional additional timestamp that can be passed by the event creator
- Payload – JSON serializable data structure used to represent additional data

When connecting to the event bus, a component specifies which event types it wants to subscribe to. When other components register events of that type, the component is notified, receives the event, and can use it to perform its function (for example, the GUI server receives data to serve on its web interface). The event bus stores the registered events into a database and provides an interface for components to query these events. This provides the components with access to historical data and allows the state of the SCADA system to be persisted while it is shut down.

Concerning the example shown in Figure 2, although the event bus is not explicitly drawn, it still affects the behavior of the system. Every full arrow between sequence actors in the diagram means that the source of the arrow has registered an event and that its targets have subscribed to events with that event type. In this context, the event bus serves more as a communication medium than as a participant in the data sequence.

2) PROTOCOL CONVERTER

The main function of SCADA systems revolves around communication with different types of devices which are used in the managed industrial processes, such as meters, switches, or transformers. The logic implementing this communication varies from one device to another and often does not even use the same communication protocol. For this reason, it is common practice to separate the implementation of this logic into a separate component. The component unifies all available communication drivers, the entities that use them to control remote devices, and provides an interface to the business logic layer of the SCADA system. This interface can be accessed in a variety of ways. One common approach is to call functions or use standard object-oriented principles [42]. In event-driven SCADA systems, the method invocation is replaced by sending and receiving events. The component receives events describing what data to send to remote devices, and it converts datagrams containing process data into events. In Hat-based SCADA systems, this is done by the protocol converter component.

This can also be seen in Figure 2, where the protocol converter is connected to two different devices – one communicating with the IEC-104 protocol and the other with Modbus. The protocol converter receives information packets from those devices and converts them into events. It is important to note that it is possible to reverse the direction of communication, from SCADA to the devices, to write data or execute commands, even though this is not shown in the sequence diagram.

3) DATA PROCESSING MODULES

Business logic is usually the core of any system, and in Hat-based systems, it is implemented with data processing modules. These modules are intermediate components that subscribe to events and generate new ones. While in non-event-driven systems, business logic is usually implemented by modeling the process using object-oriented programming [42], Hat achieves it with a data flow. A module acts as an independent component that defines the types of events to which it subscribes. Once these events are registered, the component registers new events based on them (referred to as processing). The modules differ from each other depending on the exact kind of processing they do. One module might register events indicating that a switch position has changed, but then other modules might receive that event and register new events based on it, for example, indicating that the change is dangerous and triggering an alarm. By chaining different modules in such a way that the output events of one module are inputs of another, the data flow, and thus the business logic, is implemented.

In the sequence diagram in Figure 2 these modules are placed in the middle. There are three different modules, distinguished by how they process events. The one labeled 'Assign to point' converts events registered by the protocol converter that are protocol-specific into process points. The 'Scaling' module multiplies any process point by 0.5 and creates a new process point with a scaled value. The 'Sum' module receives two process points and creates a new one with a value equal to the sum of values of the input process points.

4) GUI SERVER

Graphical user interfaces (GUI) of Hat applications are web-based applications deployed on a component called the GUI server. It represents the presentation layer of the SCADA system and is implemented similarly to SCADA systems such as IDOM, consisting of a web-based frontend and a backend that serves it [42]. The backend acts as an intermediary between the frontend and the business logic contained in the data processing modules. It communicates with the frontend using HTTP and WebSocket protocols.

The GUI server connects to the event server, receives events, and notifies its clients when relevant changes occur. A simple example of a change would be the GUI server receiving an event stating a measurement has taken a new value. The server would then notify all its clients of the new measurement value that needs to be displayed. Since GUI clients must be authenticated before accessing any data, the GUI server would notify only clients with sufficient access rights.

Reverse data flow is also supported: GUI clients can send requests to the server, which can lead to the creation of new events. This is necessary whenever a client needs to make a change that affects the global state of the system, i.e. the change should somehow affect other GUI clients or external devices. This could, for instance, be manual entries, alarm confirmations, or commands. The same rules regarding authentication apply to this direction of the data flow, i.e. the GUI server filters out all requests for which the logged-in user does not have sufficient rights.

The sequence diagram in Figure 2 illustrates most of these concepts (all except the opposite data flow). The GUI server subscribes to all process point events and notifies its clients of their changes via a WebSocket connection.

III. ARTIFICIAL INTELLIGENCE MODEL MANAGER

In addition to standard components described in the previous chapter, we add a new component to the Hat environment whose function is to apply advanced analytics to Hat events: Artificial Intelligence Model Manager (AIMM). The component is open-source and publicly available [10]. The repository is also documented with examples, one of which shows the case study described in the following chapter,

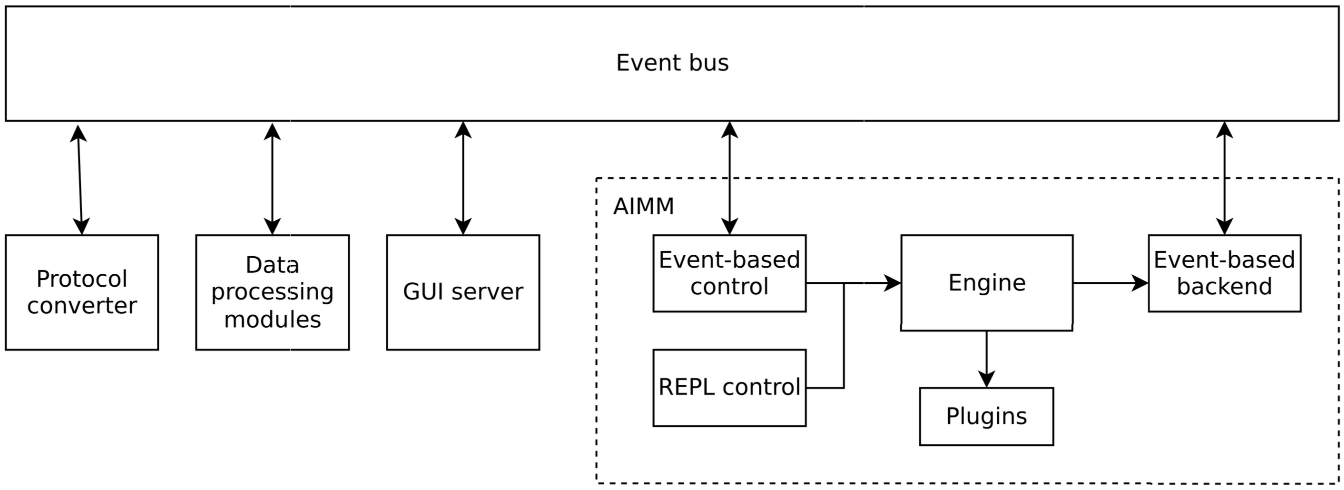


FIGURE 3. AIMM's integration into the Hat environment.

with a simplified SCADA and publicly available power grid data [43].

The component can be viewed as a model manager, where models are implementations of various analytical algorithms. These models can be instantiated (an instance is a concrete configuration of a model), trained, and executed. The AIMM component provides interfaces that allow its users to create and execute models, access data, and implement control and storage interfaces. There are four independent components – plugins, controls, backend, and the engine. Plugins allow users to create their models and data access functions, control interfaces serve as entry points to system functions, the backend component manages the storage of model instances, and the engine acts as the central component that manages actions and resources created by the other three components. Control and backend instances can also connect to Hat's event server and use its infrastructure. The interaction of all these components and the way they are integrated into the Hat environment is shown in Figure 3. Following subsections describe each subcomponent in greater detail.

A. PLUGINS

Plugins are the most extensible part of the AIMM component. Their function is to register entry points for various workflows specific to analytical algorithms, such as fitting or prediction functions, instantiation of machine learning models, or data access and preprocessing. The plugins are implemented as various functions and classes that follow interfaces defined by the AIMM component. These implementations are registered on the AIMM server so that other components can use them in their actions.

This approach has several advantages – it provides flexibility when it comes to adding new algorithms to the system and it allows the plugins to be used in offline mode so that integrators can try different configurations of models before deploying them (if they use the same plugins locally as on the server). Also, plugin implementations can report progress status, which gives users more insight into longer-running operations and provides a better user experience. One drawback is the need to implement plugins to use the component, which could be a problem for users who are not programmers. This problem could be mitigated by developing a standard plugin library that includes generalized plugins.

```
interface Model:

    def constructor(*args)

    def fit(*args)

    def predict(*args) -> result

    def serialize() -> bytes

    @classmethod
    def deserialize(model_bytes) -> Model
```

LISTING 1. Pseudocode of the model interface.

The current implementation of AIMM supports implementations of two types of artifacts used by the engine: models and data access points. Models are objects that contain implementations and various parameters of advanced analytics. Their public methods are used by the rest of the AIMM objects to create, serialize, deserialize, train, and use the models (at the moment, only the supervised learning workflow is supported). The current implementation can still be modified, but the basic necessary methods need to be specified as shown in Listing 1. Some method arguments are only loosely specified since a precise specification might limit the needs of the plugin implementer. Arguments for progress reporting callback functions can be specified here, which can then be used by the rest of AIMM to inform the user of the progress of those calls.

Data access points are functions that are called to access datasets. They handle long-lasting file reads or downloads of

```
def data_access(*args) -> data
```

LISTING 2. **Function signature of the data access plugin.**

datasets, and any optional data preprocessing while retaining the previously mentioned benefits of plugin calls. A data access plugin must match the signature in Listing 2. As it is shown, it does not have to meet precise requirements for its arguments, for the same reason as the model interface.

B. BACKEND

The purpose of the backend component is to persist model instances. At startup, the backend is queried for all persisted model instances so that they can be included in the server state. Subsequently, the engine can request storage of model instances at various times during runtime. There is no single implementation of the backend, it is an interface, and the specific implementation is defined in the component configuration.

AIMM allows the implementation of backends that can access the event bus and use events to store and load model instances. Such a backend implementation is also used in our case study for state estimation. Since events can contain binary data, the backend can serialize each model instance, register the event with the instance bytes and query the event server for any stored instances on subsequent launches. This has the advantage of having a single source of truth for all persisted data since all persisted data are stored on the event server. If the AIMM component is running in redundant mode and the primary component is shut down, the secondary component can access any model instances that the primary component may have created. The disadvantage is the loss of control over how the instances are stored. This could be problematic if the event server storage method is suboptimal for this purpose.

```
interface Backend:

    def constructor(conf, event_bus)

    def get_models() -> List[Model]

    def save_model(model)

    def update_model(model, model_id)
```

LISTING 3. **Pseudocode for the backend interface.**

The interface that each backend must implement is the same as previous descriptions, as shown in Listing 3. In addition, there is a certain implementation detail related to accessing the event interface: the backend receives an instance of an object representing access to the event bus that it can use to create events containing serialized model instances.

C. CONTROL

The control component is used to host various interfaces that serve as entry points for actions that the AIMM system can perform. There may be several different implementations of these interfaces running in parallel. The control interface is also needed because its implementations are the ones that communicate with external systems. This means that, in order to integrate AIMM into a different SCADA system, the only requirement is the development of new control implementation, one specific to the external interfaces of that SCADA system.

```
interface Control:

    def constructor(conf, event_bus)
```

LISTING 4. **Pseudocode for the control interface.**

The control interface is simpler than other interfaces since its implementations only need to be instantiated at startup and then act independently thereafter. This is reflected in the interface pseudocode in Listing 4.

One interface used in the case study was nicknamed the REPL interface, REPL referring to the term Read–eval–print loop. This is a programming environment in which users can enter commands from a scripting language and the results of the command are immediately printed to standard output. AIMM's REPL interface is designed to allow users to perform server actions by running a Python REPL, invoking the command to connect to the server, and executing the functions that correspond to desired actions. The motivation is to provide an interface that allows users to quickly upload new model instances, update old ones, access existing instances, and perform model training and executions, without hiding these actions deep in the business logic of the server's runtime.

The second implementation of the control interface served as a bridge between the event server and the AIMM system. This interface subscribes to various events, performs the appropriate actions specified by those events, and registers new events that contain the model execution results. The interface waits for events informing it what action to perform with which model instance and optional parameters or configurations. This information is passed using the event's event type and payload and is generic concerning plugins, meaning that the event control is not aware of which plugin functions exist or the exact semantics of their arguments and configuration parameters. The results of triggered actions are then also registered as events to which other components of the SCADA system can react.

This way, AIMM can either receive historical data as inputs from its clients (i.e. as an event payload in the case of an event-based control implementation), or it can implement control with access to the event bus and an interface that allows its users to specify query parameters when requesting model fitting, which would then be used to retrieve the data. The first option is easier to implement because it is up to the caller of the control interface to specify the data, while the advantage of the second option is that it avoids the creation of bloated events (a disadvantage of the first option) since one event can be interpreted as one entry of historic data in the query result.

D. ENGINE

The engine is the central component of the AIMM architecture. It is used to synchronize all other components to implement expected behavior of the entire system. Workflow actions like fitting or executing model instances are executed in separate processes since they usually take a long time. Management of these processes is also handled by the engine.

The workflow of a generic server action could be represented with the following steps:

1) A control starts a new engine action.
2) The engine loads resources required to execute the action.
3) The engine accesses configured plugins and uses them to execute the action.
4) The action result is returned to its caller but is also available in the engine state.
5) When an action creates or modifies a model instance, the engine uses the backend to persist it.

Currently supported actions in the engine allow controls to create, update and upload new model instances, fit the instances to the data and run models on unseen data.

IV. STATE ESTIMATION OF A MEDIUM-VOLTAGE GRID

State estimation is a problem that is often solved using various methods from fields of statistics or artificial intelligence since there is no analytical solution. The goal is to use a small subset of all available measurements in a power grid to estimate states of various elements in the system. This subset is not sufficient to accurately evaluate the entire grid using laws of physics, so various approximation mechanisms must be used.

In the case study, a state estimation was performed for a medium-voltage grid. The only available measurements were active and reactive power loads on all buses. The objective was to estimate voltage values and their angles on the buses.

A. SYSTEM SETUP

1) DEVICES

In the case study, all configured devices are used to report measurements. These measurements show active and reactive power values on buses in the distribution network. The GUI is configured to display the measurements next to the corresponding buses. The complete setup of the scheme is shown in Figure 4. The scheme represents the medium-voltage grid of a Croatian county and was provided by the project ATTEST. We also created a publicly available demonstration using the IEEE 14-bus grid, but with a simplified SCADA implementation that would not have been used in practice [43].

2) UNIFIED DATA MODEL

As mentioned earlier, separate data processing modules are used to transform and manipulate data as it travels from

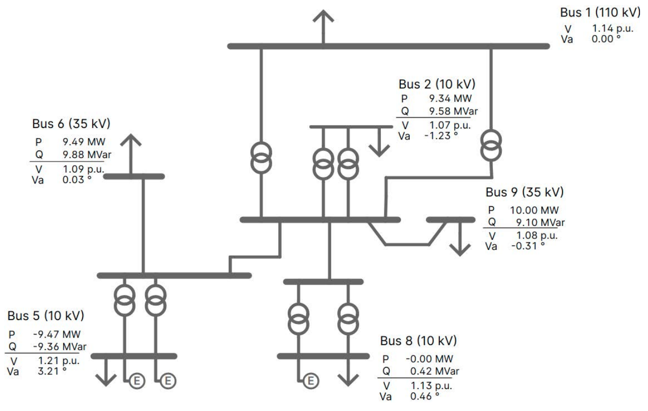


FIGURE 4. Electrical schematic for the case study showing measurements (P and Q) and state estimates (V and Va).

devices to user interfaces. Since there are many different types of devices, data transformations, and user actions that the SCADA system must handle, a separate unified data model (UDM) was introduced that synchronizes the events originating from these sources under the same event type and payload structure. The UDM represents all readings from devices as process points, similar to the example in Figure 2. Process points are identified by their event types and contain information such as measurement values, quality of data, or other protocol-specific information in the event's payload data. When a device reading changes its state, e.g. a measurement changes or a switch changes its position, an event representing that change is mapped onto a UDM event, performing any additional transformations. The process point event is then transformed into a GUI-specific event that should be consumed by the GUI server component. The reverse direction works similarly, where user actions are registered as GUI-specific events that target process points and are converted into device-specific events only when needed.

3) HAT-AIMM INTERFACE

A new module has been added to the event server's data processing modules whose function is to register events for the AIMM component to respond to. The module processes UDM's process point events and generates AIMM-specific events containing data necessary to perform state estimation. To avoid simultaneous registration of too many events requesting state estimates in cases where measurements change rapidly, a buffer time of 0.5 seconds was introduced. This way, measurement changes would accumulate in these 0.5 seconds and a single request for a state estimate would be registered after this time had elapsed. The module would also handle the reverse communication direction, i.e. it would subscribe to all events registered by the AIMM component and use them to update the UDM with estimation data.

From a functionality standpoint, this module was not necessary – the AIMM component has access to the event bus and could have used events that the module would normally handle to directly generate state estimation events for the GUI server component. Nevertheless, the specialized module for state estimation was added to the event server, mostly to ensure separation of concerns.

4) AIMM CONFIGURATION

AIMM was configured to connect to the event server and receive events that trigger state estimates. A plugin was developed that uses the *pandapower* package to perform estimation [44]. It uses the Gauss-Newton method to find the most likely estimates while modeling the problem as a WLS regression. Incoming events, sent by the newly developed module contain information about available measurements. These measurements are passed to the plugin, which processes them and attempts to estimate the states of the remaining measurements in the system. The estimates are then sent back to the event server module. The module translates them into UDM events enabling their visualization as regular measurements in the remaining components.

An important concept to point out is that AIMM's event server control, engine, and backend are not aware of semantics of incoming and outgoing events. Their purpose is infrastructural – they connect incoming data to the plugins and provide evaluation results to whoever is listening, in this case, Hat's event server. This means that AIMM's only development requirement is plugin development, which should normally depend on the context of its use. An alternative approach could have involved development of a specialized control implementation that is aware of exact functions the AIMM component provides. This way, it would not have been necessary to implement a new module of the event server, since the functions it provides are covered by the specialized control. Nevertheless, the case study opted for the first approach, i.e. developing a specialized module and using the generic event control, mainly because the authors considered this to be an architecturally "cleaner" solution since the SCADA system is aware of the AIMM component to some extent and this awareness does not affect its core functions.

B. SIMULATION

Since the communication with all configured devices is done via TCP, readings can be simulated by hosting a server for each device (or connecting with a client if the SCADA system is to act as a server). The connections are then used as communication channels that send simulated data to the protocol converter. IEC 104 protocol was used to transmit the data [39].

Since real measurements were not available in large numbers, the simulated data were created using the available measurements as a reference. Nevertheless, simulated inputs did not contain values that would not have been available in the real measurements. This means that the model did not have access to additional data that it would not have had in a real scenario.

C. RESULTS

The state estimation was successfully deployed with minimal changes to the preexisting SCADA system. The only modification was the addition of an event server module that communicated with the AIMM component, which could have also been avoided using the alternative method described at the end of subsection A. Overall, AIMM was successfully integrated into the Hat environment, had access to all necessary data, and was able to contribute its estimates.

Since the AIMM component was running in a separate process, there was no impact on SCADA performance, and even if there had been, the component could have been moved to a separate device since events are transmitted over TCP. A potential negative side-effect of implementing the state estimator in a separate component of a distributed system is an increase in response time from reading the measurements to displaying the estimates on the user interface. Since the components ran on the same physical device, this effect was not particularly pronounced when the case study was conducted but could increase in configurations where they are connected over the network. Figure 5 shows the flow of messages through the system and the times at which they were received by the components processing them. The times were measured in over 4500 simulation scenarios where measurements change their values, AIMM performs estimation and the changes are displayed in the SCADA GUI. An additional reason why events requesting estimation are registered after measurement events are available on the GUI is the buffer time mentioned in the Hat-AIMM interface section. Overall, the measured times show that the biggest factor in the delay between measurement change times and estimate availability is the calculation of the estimate itself.

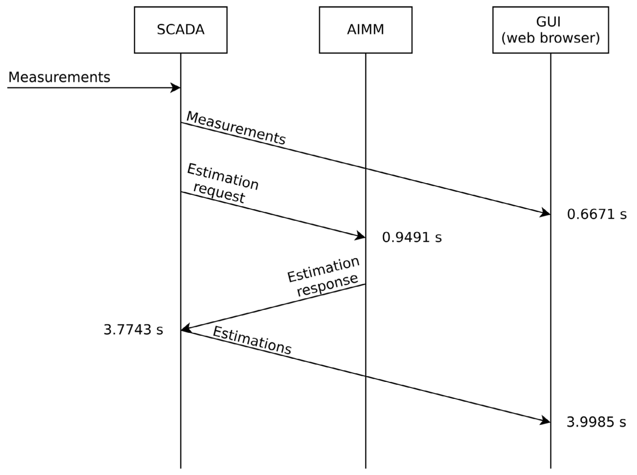


FIGURE 5. Component messages and their average arrival times over 4500 simulation scenarios.

D. DISCUSSION

The solution in the case study has managed to provide state estimates in soft real-time. Nevertheless, some improvements can be added in the future. For example, since most analytical algorithms this paper refers to are based on approximation, the question of quantifying uncertainty rises. It is common for estimators to not only produce estimates but to express how likely they are to be incorrect. This is usually expressed by a probability or a confidence interval. The quantification of uncertainty may also depend on the type of method used, e.g. Bayesian methods always provide distributions of estimates rather than individual values. Therefore, methods such as Gaussian process regression are useful in use cases where such quantification is important, as shown in [45]. Uncertainties from non-probabilistic methods can also be quantified, but usually using separate workflows, independent of the process of estimation, e.g. with probabilistic collocation [46].

To reduce the possibility of inaccurate measurements, another potential improvement to the solution is noise handling. Measurement noise is partially handled by the estimation method itself. *Pandapower's* estimators take into account the standard deviation of the measurements since it is a configurable parameter. However, the implemented workflow could be improved by adding a separate step to analyze noise before calculating the estimate. In *pandapower*, this is usually done using chi-squared tests to calculate the probability that a measurement is correct. Another option would be to not use *pandapower's* methods and implement our own. However, such a method would still be based on statistics, it would just use a different test. Adding the noise analysis step would affect the estimation workflow in such a way that a SCADA alarm could be triggered if the estimation is not possible due to noisy measurements.

Another problem associated with unstable measurements is the convergence of the algorithm. The estimator requires a large number of continuous input variables, which means it may not be able to compute a solution for every possible combination. The statistical test mentioned earlier could be used to check measurements before the estimation. In addition, the case where an estimate could be made must be handled in the AIMM control, probably by raising an alarm in the SCADA system. Another approach could be to use a model based on Bayesian statistics, which would provide a distribution of estimates and enable determining not only the estimated measurements but also their reliability. This approach could converge more often but would classify estimates based on unrealistic measurements as highly unreliable.

Finally, the speed of the estimation method itself could be improved. The WLS estimator has an imprecise computational complexity, that depends mainly on the convergence speed of the Gauss-Newton optimization. In practice, this is acceptable for the presented case study only if the estimates have to be computed at least every 3 seconds for this case study's power grid. If the metering changes consistently arrive faster, the estimates cannot adjust quickly enough. A solution to this problem is buffering where the event server module responsible for generating events for estimation requests does not generate events every time a measurement change occurs, but only every 0.5 seconds, as mentioned in the case study. For the simulated use case, the 0.5 seconds were sufficient to produce acceptable results, but in a real-life scenario, that interval might need to be increased as changes could occur much more randomly.

Another approach to solving the estimation speed problem could be to change the estimation algorithm and use a less complex solution that does not perform optimization during the calculation of estimates. For example, one could use a neural network already optimized for the power grid on which it performs estimates. Another improvement is shown in [47] and [48] where using sparse models, Bayesian methods, and special feature importance quantification methods, irrelevant measurements are detected and excluded from calculation. A potential drawback is that such models are generalized and not coupled in any way with the state estimation problem, which could lead to a loss of accuracy in the estimates, as the model would lose insight into physical aspects of the power grid. *Pandapower*'s implementation of the state estimator could also use a general method but is specific to the state estimation problem – its models are aware of the complete grid topology, all elements within it, and their states, e.g., which switches are opened or closed. Arguably, reducing the number of features or using a generalized model such as neural networks could lead to a loss of estimation accuracy and poor ability to generalize, if these details are ignored. However, this should be further confirmed in a separate study.

V. CONCLUSION

In this paper, we presented an architecture for a new component of an event-driven, component-based SCADA system that provides capabilities for advanced analytics. We configured the SCADA to monitor measurements of a medium-voltage network and used the new component to solve the state estimation problem. The addition of the component was mostly seamless, requiring only minimal changes to the SCADA system (and even those changes were not necessary, but were added because they made architectural sense).

The work shows that a generalized approach to integrating advanced analytics into event-driven component-based SCADA can be achieved. The interface between the central data processing component and the AIMM component was not specific to the state estimation problem, nor was it limited to processing power grid measurements. This suggests that the component could have been applied to any SCADA-related problem that requires advanced analytics. A more formal generalization could be a topic of future research. As for specific goals of this work, we believe they were successfully achieved.

REFERENCES

[1] B. M. Michelson, "Event-driven architecture overview," *Patricia Seybold Group*, vol. 2, no. 12, p. 1057, 2006.

[2] B. J. Cox, *Object Oriented Programming: An Evolutionary Approach*. Reading, MA, USA: Addison-Wesley, 1986.

[3] A. Rajkomar, J. Dean, and I. Kohane, "Machine learning in medicine," *New England J. Med.*, vol. 380, no. 14, pp. 1347–1358, 2019.

[4] A. I. Khan and S. Al-Habsi, "Machine learning in computer vision," *Proc. Comput. Sci.*, vol. 167, pp. 1444–1451, Oct. 2020.

[5] D. T. Senthil Kumar, "Data mining based marketing decision support system using hybrid machine learning algorithm," *September*, vol. 2, no. 3, pp. 185–193, Aug. 2020.

[6] R. van de Schoot, S. Depaoli, R. King, B. Kramer, K. Märtens, M. G. Tadesse, M. Vannucci, A. Gelman, D. Veen, J. Willemsen, and C. Yau, "Bayesian statistics and modelling," *Nature Rev. Methods Primers*, vol. 1, no. 1, p. 1, Dec. 2021.

[7] M. Abdel-Basset, G. Manogaran, A. Gamal, and V. Chang, "A novel intelligent medical decision support model based on soft computing and IoT," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4160–4170, May 2020.

[8] R. Zhang and J. Tao, "A nonlinear fuzzy neural network modeling approach using an improved genetic algorithm," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5882–5892, Jul. 2018.

[9] B. Kopić, J. Krstulović Opara, Z. Sičanica, and A. Trstenjak. (Dec. 2021). *Hat-Open—About*. [Online]. Available: https://hat-open.com

[10] Z. Sičanica. (Mar. 2022). *Introduction—AIMM Documentation*. [Online]. Available: https://aimm.readthedocs.io/en/latest/introduction.html

[11] D. Beck, K. Blaum, H. Brand, F. Herfurth, and S. Schwarz, "A new control system for ISOLTRAP," *Nucl. Instrum. Methods Phys. Res. A, Accel. Spectrom. Detect. Assoc. Equip.*, vol. 527, no. 3, pp. 567–579, Jul. 2004.

[12] C. Angelov, X. Ke, and K. Sierszecki, "A component-based framework for distributed control systems," in *Proc. 32nd EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2006, pp. 20–27.

[13] Y. Zhang and J. L. Chen, "Wide-area SCADA system with distributed security framework," *J. Commun. Netw.*, vol. 14, no. 6, pp. 597–605, Dec. 2012.

[14] M. Q. Raza and A. Khosravi, "A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings," *Renew. Sustain. Energy Rev.*, vol. 50, pp. 1352–1372, Oct. 2015.

[15] C. Deb, F. Zhang, J. Yang, S. E. Lee, and K. W. Shah, "A review on time series forecasting techniques for building energy consumption," *Renew. Sustain. Energy Rev.*, vol. 74, pp. 902–924, Jul. 2017.

[16] A. Baliyan, K. Gaurav, and S. K. Mishra, "A review of short term load forecasting using artificial neural network models," *Proc. Comput. Sci.*, vol. 48, pp. 121–125, Jan. 2015.

[17] T. Hong, P. Pinson, and S. Fan, "Global energy forecasting competition 2012," *Int. J. Forecasting*, vol. 30, no. 2, pp. 357–363, Apr. 2014.

[18] A. Stetco, F. Dinmohammadi, X. Zhao, V. Robu, D. Flynn, M. Barnes, J. Keane, and G. Nenadic, "Machine learning methods for wind turbine condition monitoring: A review," *Renew. Energy*, vol. 133, pp. 620–635, Apr. 2019.

[19] M. Kezunovic, "Substation fault analysis requirements," in *Proc. Innov. Smart Grid Technol.*, Jan. 2010, pp. 1–6.

[20] Y. He, G. J. Mendis, and J. Wei, "Real-time detection of false data injection attacks in smart grid: A deep learning-based intelligent mechanism," *IEEE Trans. Smart Grid.*, vol. 8, no. 5, pp. 2505–2516, Sep. 2017.

[21] R. C. B. Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, "Machine learning for power system disturbance and cyber-attack discrimination," in *Proc. 7th Int. Symp. Resilient Control Syst. (ISRCS)*, Aug. 2014, pp. 1–8.

[22] W. Gao, T. Morris, B. Reaves, and D. Richey, "On SCADA control system command and response injection and intrusion detection," in *Proc. Ecrime Res. Summit*, Dallas, TX, USA, Oct. 2010, pp. 1–9.

[23] M. A. Ferrag, M. Babaghayou, and M. A. Yazici, "Cyber security for fog-based smart grid SCADA systems: Solutions and challenges," *J. Inf. Secur. Appl.*, vol. 52, Jun. 2020, Art. no. 102500.

[24] A. Abur and A. G. Exposito, *Power System State Estimation: Theory Implemention*. Boca Raton, FL, USA: CRC Press, 2004.

[25] L. Mili, M. G. Cheniae, N. S. Vichare, and P. J. Rousseeuw, "Robust state estimation based on projection statistics," *IEEE Trans. Power Syst.*, vol. 11, no. 2, pp. 1118–1127, May 1996.

[26] P. N. P. Barbeiro, J. Krstulovic, H. Teixeira, J. Pereira, F. J. Soares, and J. P. Iria, "State estimation in distribution smart grids using autoencoders," in *Proc. 8th Int. Power Eng. Optim. Conf.*, Mar. 2014, pp. 358–363.

[27] J. Du, S. Ma, Y.-C. Wu, and H. V. Poor, "Distributed hybrid power state estimation under PMU sampling phase errors," *IEEE Trans. Signal Process.*, vol. 62, no. 16, pp. 4052–4063, Aug. 2014.

[28] A. S. Zamzam and N. D. Sidiropoulos, "Physics-aware neural networks for distribution system state estimation," *IEEE Trans. Power Syst.*, vol. 35, no. 6, pp. 4347–4356, Nov. 2020.

[29] V. M. Catterson, E. M. Davidson, and S. D. J. McArthur, "Issues in integrating existing multi-agent systems for power engineering applications," in *Proc. 13th Int. Conf. Intell. Syst. Appl. Power Syst.*, 2005, pp. 1–6.

[30] V. Uraikul, C. W. Chan, and P. Tontiwachwuthikul, "Artificial intelligence for monitoring and supervisory control of process systems," *Eng. Appl. Artif. Intell.*, vol. 20, no. 2, pp. 115–131, Mar. 2007.

[31] K. Leahy, C. Gallagher, P. O'Donovan, K. Bruton, and D. O'Sullivan, "A robust prescriptive framework and performance metric for diagnosing and predicting wind turbine faults based on SCADA and alarms data with case study," *Energies*, vol. 11, no. 7, p. 1738, Jul. 2018.

[32] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Gener. Comput. Syst.*, vol. 45, pp. 123–132, Apr. 2015.

[33] G. Yadav and K. Paul, "Architecture and security of SCADA systems: A review," *Int. J. Crit. Infrastruct. Protection*, vol. 34, Sep. 2021, Art. no. 100433.

[34] H. Fazlollahtabar, "Internet of Things-based SCADA system for configuring/reconfiguring an autonomous assembly process," *Robotica*, vol. 4, pp. 1–18, Jun. 2021.

[35] R. Khorsand, M. Ghobaei-Arani, and M. Ramezanpour, "FAHP approach for autonomic resource provisioning of multitier applications in cloud computing environments," *Softw., Pract. Exper.*, vol. 48, no. 12, pp. 2147–2173, Dec. 2018.

[36] M. Ghobaei-Arani, A. Souri, T. Baker, and A. Hussien, "ControCity: An autonomous approach for controlling elasticity using buffer management in cloud computing environment," *IEEE Access*, vol. 7, pp. 106912–106924, 2019.

[37] D. Pliatsios, P. Sarigiannidis, T. Lagkas, and A. G. Sarigiannidis, "A survey on SCADA systems: Secure protocols, incidents, threats and tactics," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1942–1976, 3rd Quart., 2020.

[38] B. Kopić, J. Krstulović Opara, Z. Sičanica, and A. Trstenjak. (Mar. 2021). *Hat-open/hat-core-Končar—KET*. [Online]. Available: https://github.com/hat-open/hat-core

[39] *IEC International Standard 2.1*, document IEC 60870-5-104:2006+AMD1:2016 CSV, Jun. 2016.

[40] A. Swales, "Open modbus/tcp specification," *Schneider Electr.*, vol. 29, pp. 3–19, 1999.

[41] I. Fette and A. Melnikov, "The websocket protocol," Internet Eng. Task Force (IETF), Wilmington, DE, USA, Tech. Rep. RFC 6455, Dec. 2011.

[42] A. Monaco, "Analysis and development of supervisory control and data acquisition system for industry 4.0," Ph.D. dissertation, Dept. Control Comput. Eng., Politecnico di Tori, Turin, Italy, 2019.

[43] Z. Sičanica. (Mar. 2022). *Getting started—AIMM documentation*. [Online]. Available: https://aimm.readthedocs.io/en/latest/getting_started.html

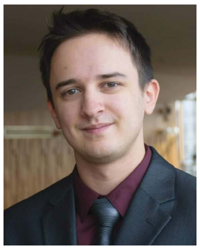
[44] L. Thurner, A. Scheidler, F. Schafer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower—An open-source Python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 6510–6521, Nov. 2018.

[45] K. Liu, Y. Shang, Q. Ouyang, and W. D. Widanage, "A data-driven approach with uncertainty quantification for predicting future capacities and remaining useful life of lithium-ion battery," *IEEE Trans. Ind. Electron.*, vol. 68, no. 4, pp. 3170–3180, Apr. 2021.

[46] G. Lin, N. Zhou, T. Ferryman, and F. Tuffner, "Uncertainty quantification in state estimation using the probabilistic collocation method," in *Proc. Power Syst. Conf. Expo. (PSCE)*, Mar. 2011, pp. 1–8.

[47] K. Liu, X. Hu, H. Zhou, L. Tong, W. D. Widanage, and J. Marco, "Feature analyses and modeling of lithium-ion battery manufacturing based on random forest classification," *IEEE/ASME Trans. Mechatronics*, vol. 26, no. 6, pp. 2944–2955, Dec. 2021.

[48] K. Liu, Z. Wei, Z. Yang, and K. Li, "Mass load prediction for lithium-ion battery electrode clean production: A machine learning approach," *J. Cleaner Prod.*, vol. 289, Mar. 2021, Art. no. 125159.

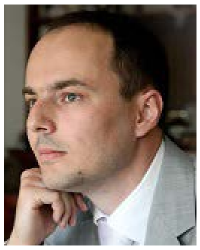


ZLATAN SIČANICA was born in Zenica, Bosnia and Herzegovina, in 1993. He received the B.S. and M.S. degrees in computing from the University of Zagreb, Zagreb, Croatia, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in computing with the Faculty of Electrical Engineering and Computing.

He works as a Software Engineer at Končar—Digital, where he works in the department for real-time applications development. He has authored several conference papers with a focus on the artificial intelligence applied to SCADA systems.

STJEPAN SUČIĆ received the master's and Ph.D. degrees in electrical power engineering from the University of Zagreb, in 2008 and 2013, respectively.

He works as the Head of software products development at Končar—Digital. His research interests include middleware technologies for optimized and adaptable smart grid automation. His main research interests include middleware analysis, architectural application design paradigms, service-oriented integration, and M2M applications. He is the National Representative at IEC for several TC57 working groups related to international standard IEC 61850.



BORIS MILAŠINOVIĆ (Member, IEEE) graduated from the Department of Mathematics, Faculty of Science, University of Zagreb, in 2001. He received the M.Sc. and Ph.D. degrees in computing from the Faculty of Electrical Engineering and Computing, in 2006 and 2010, respectively.

He is currently an Associate Professor at the Department of Applied Computing, Faculty of Electrical Engineering and Computing, University of Zagreb. His main research interests include software development methodologies and workflow management. He has been a member of the Editorial Board of *Computer Science and Information Systems* journal, since 2018; a member of the Editorial Board of *CIT. Journal of Computing and Information Technology*, since 2021; and a program committee member of several international conferences.