SENSE

SENSE: Semantics-based Explanation of Cyber-Physical Systems

Deliverable 4.1 (v3): Semantic-Based Explainability Framework

Authors	:	Katrin Schreiberhuber, Mevludin Memedi, Fajar J. Ekaputra, Marta Sabou
Dissemination Level	:	Public
Due date of deliverable	:	31.07.2024
Actual submission date	:	31.08.2024
Work Package	:	4. Semantics-based Event Explainability
Type	:	Report
Version	:	3.0

Abstract

This deliverable is the second version of D4.1 from the SENSE project. This deliverable version contains a major update (v2) to the SENSE ontology with a detailed description of core classes and properties as well as example instances for these classes. The SENSE ontology is used as a basis for the other tasks within the work package. This update complements the existing content of type causality knowledge acquisition method (T4.2), the explanation generation framework (T4.3), and an initial literature study result of the personalized and actionable explanation approach (T4.4). We plan to elaborate more on the personalized and actionable explanation approach in the last version of the deliverable, scheduled for M30.

FFG

Project Funded by FFG – IKT der Zukunft Programme

Project Number: FO999894802 **Start date: 01.02.2023** **Duration: 30 months**

History

Version	Date	Reason	Revisited by
0.1	15.11.2023	Initial draft (v1)	FJE
0.2	28.02.2024	Ch.3,4,5	MM, KSch, MS
1.0	31.03.2024	Final review (v1)	FJE, MS
1.1	26.08.2024	Update on Ch.2 (v2)	KSch
1.2	29.08.2024	Abstract and Intro update (v2)	FJE
2.0	31.08.2024	Final review (v2)	MS
2.1	06.02.2025	Update on Ch.2,4,5	KSch
3.0	03.07.2025	Update on Ch.2 + final review	KSch

Author List

Project Partner	Name (Initial)	Contact Information
WU	Marta Sabou (MS)	marta.sabou@wu.ac.at
WU	Mevludin Memedi (MM)	mevludin.memedi@wu.ac.at
WU	Katrin Schreiberhuber (KSch)	katrin.schreiberhuber@wu.ac.at
WU	Fajar J. Ekaputra (FJE)	fajar.ekaputra@wu.ac.at

Table of Content

List of Figures

List of Tables

1 Introduction

The SENSE project aims to develop Explainable Cyber-Physical Systems (ExpCPS) that allow various interested stakeholders to understand system events. The project hypothesises that providing more transparency into CPS event explanations could boost efficiency, make them more user-friendly, and affect environmental sustainability.

The WP4 of the SENSE project aims to provide semantics-based event explainability over integrated CPS data, building on the integrated CPS data collected and integrated in WP3 of the project. To this end, WP4 contains four tasks as the following:

- *T4.1 The SENSE semantic model definition*. This task focuses on designing and developing a semantic model for knowledge-driven event explainability. The SENSE semantic model will consist of: (a) an overarching model for causality knowledge, (b) a model to describe CPS and their contexts, e.g., topology and environments, and (c) domain-specific models to capture specific data and information from use cases.
- *T4.2 Type causality knowledge acquisition*, focuses on employing automatic causal discovery methods to support domain experts in identifying type causality between CPS events.
- *T4.3 Explanation generation and ranking*, aiming to provide methods to generate and rank explanations for CPS events. The following steps to achieve our goals: (a) identification of actual causality knowledge between CPS events based-on type causality knowledge, (b) generation of causality paths based on actual causality for selected events, and (c) Ranking of the identified causality paths. First results of this task are published in a paper accepted at the 2024 Energy Informatics DACH+ Conference [2].
- *T4.4 Personalized and actionable explanation*. This task includes identifications and descriptions of user profiles to allow personalized explanations and represent them as a knowledge graph.

The work on WP4 will be conducted mainly by WU with the support and contributions from other SENSE project partners. The collaboration especially important on T4.1 towards the development of the SENSE semantic model. The list of involved partners is provided in Table 1.

Table 1: List of Involved project partners and members

Project Partners	**Project Members**
Wirtschaftsuniversität Wien	Katrin Schreiberhuber, Fajar J. Ekaputra, Marta Sabou, Mevludin Memedi
TU Wien	Gernot Steindl, Thomas Frühwirth, Muhammad Bilal
Siemens AG Österreich	Konrad Diwold, Daniel Hauer, Simon Steyskal
AEE INTEC	Christoph Moser
MOOSMOAR Energies OG	Wolfgang Prüggler

This deliverable is the second version of D4.1, which will be updated periodically (we plan to deliver the next version in M30) and includes progress and results from all four tasks in WP4. The rest of the deliverable is structured as follows: Section 2 reports the result of T4.1 on the SENSE semantic model, Section 3 describes the related work and experiments conducted on type causality knowledge acquisition of T4.2, Section 4 narrates our current result on the explanation generation and ranking algorithms, as well as its implementation and evaluation result in one of SENSE use case, and Section 5 outlines the result of our initial literature study on the topic of personalized and actionable explanation.

2 Semantic Model for Explainability

This task focuses on designing and developing semantic model necessary for knowledge-driven event explainability. The goal of the SENSE semantic model is to provide a solid basis for the development of (a) causality knowledge representation, (b) models to describe CPS and their contexts, e.g., topology and environments; and (c) domain-specific models to capture specific data and information from use cases.

2.1 Related Work

In this section, we provide an overview of related ontologies, which are potentially used for the development of user-centered explanations in cyber-physical system (CPS). Explainable cyber-physical system (ExpCPS) requires the integration of domain knowledge about sensor data, to make the combined knowledge usable. Ontology-based modeling facilitates knowledge sharing, logic inference as well as knowledge reuse [3]. There are five core modules of a CPS to be integrated to unlock user-centered and context-aware explainability features as envisioned for an ExpCPS. Each of the analyzed ontologies has focused on at least one of the key areas relevant for ExpCPS. In existing approaches, there are two types of related ontologies. Either an ontology covers a single aspect of an ExpCPS for multiple domains, or it covers multiple aspects, but is restricted to a very specific domain. To the best of our knowledge, there exists no domain-independent ontology, which is able to integrate all five aspects to create semantics-enabled explainabilty of CPS. The following ontologies are relevant for the creation of an ExpCPS framework:

- SOSA [4]
- SAREF core + SAREF4SYST [5]
- COInd4 [3]
- NORIA-O [6]
- FMECA [7]
- SEM [8]
- FARO [9]
- Explanation ontology [10]
- ODP for Causality [11]

2.2 Methodology

In this task, we followed the Linked Open Terms (LOT) methods [12] to develop the SENSE ontology.

LOT consists of four main steps: (i) *Ontology Requirement Specification* step, which resulted in a conceptual Ontology Requirement Specification Document (ORSD), (ii) *Ontology Implementation* step, where the requirements are implemented in a specific ontology language and evaluated using ontology evaluation tools, (iii) *Ontology Publication* step, where the ontology developed in previous step is published and documented in a web page, typically using publishing tools such as Widoco [13], and (iv) *Ontology Maintenance* step, where issues and bugs

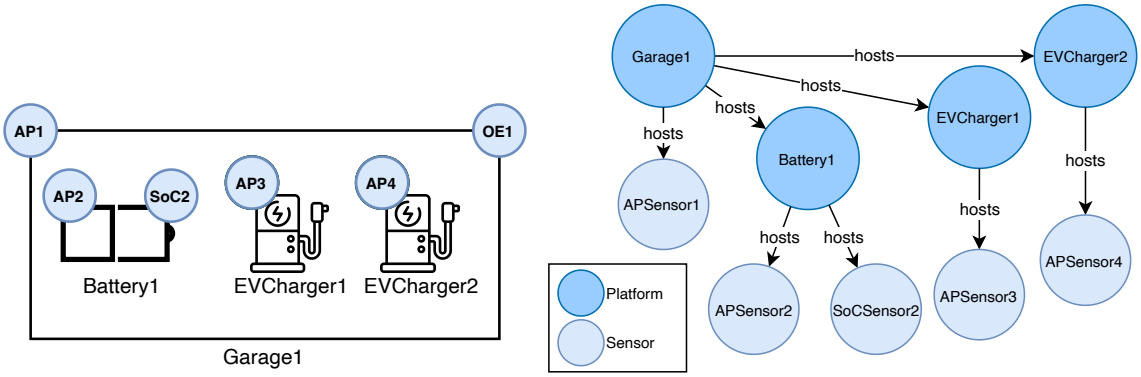

Figure 1: smart charging garage as a motivating example. On the left: schematic system setup with devices and sensors. On the right: representation of the system topology in a knowledge graph, representing devices as platforms(dark blue) and installed sensors at these platforms(light blue).

are collected, reported and used inputs to further develop and maintain the ontology in the future.

The focus of the T4.1 in this reporting period is on Step (i) and (ii). As part of the first step of Ontology Requirement Specification, we have conducted a series of workshops to gather inputs and requirements for the SENSE semantic model. The result of these workshops and the following analysis are collected in the following documents:

- The SENSE ORSD document[1], containing a high-level specification document of the SENSE ontology. This document contains links to the other documents: UC requirements document and the Modeling Data document.
- the UC requirements document[2], which reported the relevant competency questions, the origin and importance, and the subsequent answers to the questions.
- the Modeling Data (MODA) document[3]. This document contain the identified classes, properties, relations, as well as property of relations that are necessary for the SENSE ontology. Note that this document is not yet specific to specific modeling language.

In the Step (ii), we implemented the first draft of the ontology using Draw.io and Chowlk converter[14] within iterations. Furthermore, we evaluated the developed SENSE ontology using tools (e.g., OOPS [15]) and discussion with use case partners.

2.3 Motivating Example

We introduce a motivating example that will be referenced throughout the paper for clarity or our arguments and motivation. We present a small smart charging garage for electric vehicles (EVs) as an example of a CPS to clarify the intended use and context of the proposed ontology. The example in Figure 1 represents a charging garage (`Garage1`) equipped with two EV chargers (`EVCharger1` and `EVCharger2`) and a stationary battery (`Battery1`) for managing power demand peaks. The facility operator of the garage must ensure that the power demand remains within predefined limits, specified by an operating envelope (OE). Various sensors are installed across the system, measuring observable properties such as active power (`AP1`, `AP2`, `AP3`, `AP4`), state of charge (`SoC2`), and operating envelope limits (`OE1`). The main goal is to detect an

[1] "LOT_common_ORSD_v1.0.0.docx", the file is available on project repository
[2] "LOT_UC_Requirements_v1.0.0.docx", the file is available on project repository
[3] "LOT_MODA_v1.0.0.docx", the file is available on project repository

envelope violation when it occurs and to find its root cause within the system. An explanation framework needs to access multiple data sources:

- **sensor measurements** to detect violations, and other events (e.g., comparing sensor `AP1` and `OE1`).
- **system topology** to understand the connections between sensors and their measurements (e.g., which sensors are located within `Garage1` and can thus have an impact on the detected violation?).
- **causal knowledge** to know the causal implications between anomalous events (e.g., what type of events can cause a violation?).
- **user context** to only give explanations to users, who have access to the data used for explanations (is a user of an EV charger allowed to see this explanation? Which technical terms does this user understand?).
- **explanations** to be able to compare previously generated explanations and see if causes of an anomaly have changed.

2.4 The SENSE Ontology v2

We propose a new ontology, which serves as a data integration layer for various heterogeneous data sources to create semantics-enabled explanations in CPS. It facilitates the integration of diverse data sources for comprehensive system understanding and enables root cause analysis, as well as user-centered, context-aware and actionable explanations. The instantiation of the ontology with data points from a concrete CPS creates a knowledge graph (KG), which can be queried to identify relevant relations between events, states, devices, etc.

In alignment with the five key areas to enable context-aware, user-centered explanations of CPS we identified, the proposed ontology consists of five modules, focusing on different aspects of an ExpCPS: (i) System Topology (ii) Observations (iii) Causal Knowledge (iv) User Context (v) Explanations. In Fig. 2, the ontology and its interconnected modules are shown. As a design decision we used explicit typing, as defined in the ontology design pattern library MODL [**?**]. The main benefit of this modeling approach is the possibility to easily and flexibly introduce new types of sensors/events/states based on the use case. New types can be defined without extending the core ontology, as types can be defined in A-box. Therefore, this ontology can be flexibly reused for different use cases of CPS. Additionally, this decision allows the definition of properties, which are related to types of objects. As the list of object types is different for each use case, these properties can be defined in the use case data instead of the ontology directly, allowing for more flexibility. Especially since CPS domains are very diverse, this functionality is crucial to define a reusable and domain-independent CPS ontology. Moreover, defining types as separate classes allows the definition of object properties as relations between concepts, independent of the population of the dataset.

In the next sub-sections, we will provide a detailed information on each aspect of the SENSE ontology. Furthermore, we will provide an example instantiation of each aspect (i.e., Listing 1-5) based on the motivating example.

2.5 Topology Knowledge

The *System Topology* contains information about the system setup, all the devices in the system as well as the sensors, which provide a continuous data stream of system measurements at

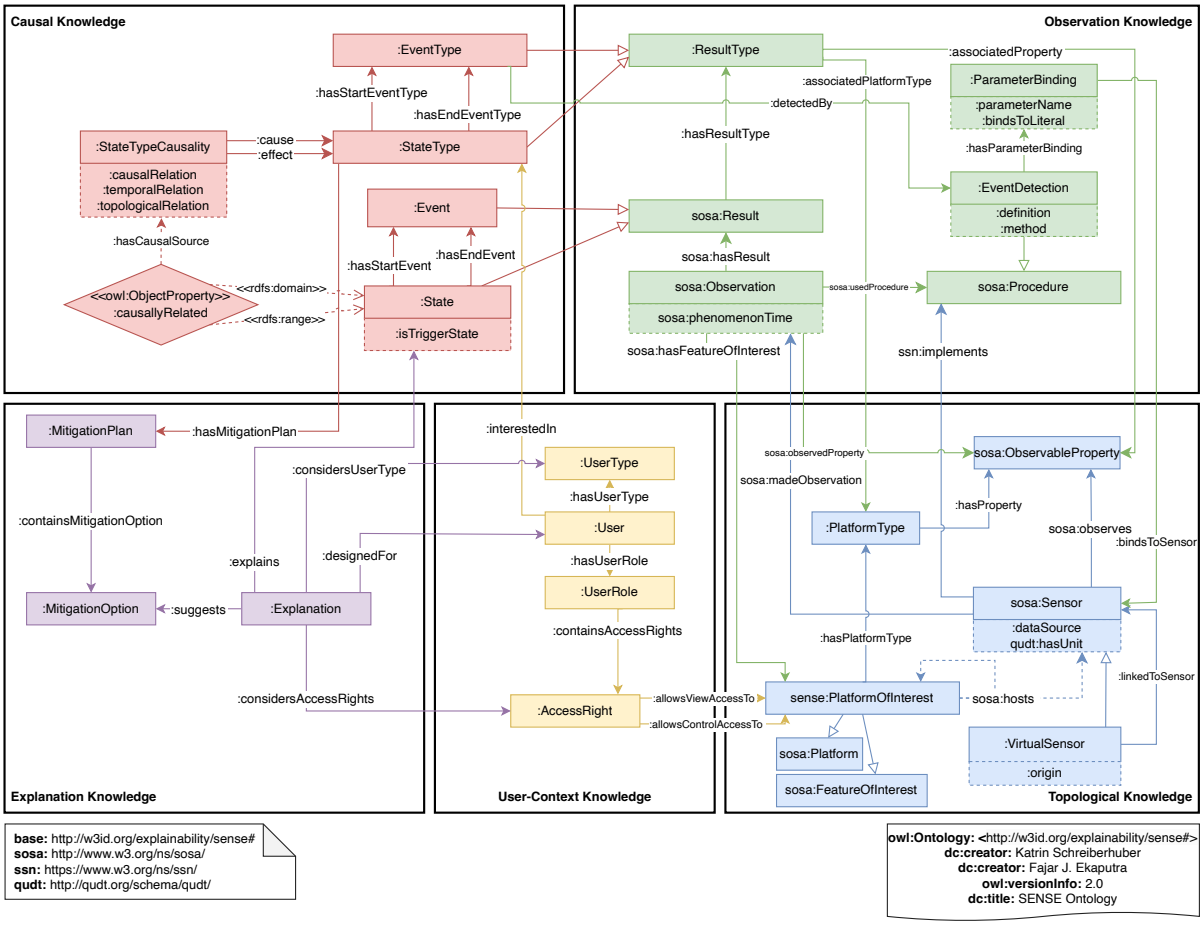

Figure 2: The SENSE ontology.

runtime. The topology information specifies the positions of the devices and their physical and logical connections. To represent the topology of a system, we use three core concepts: sensor, and observable property directly derived from the SOSA ontology[4]. Additionally, we introduce the concept of platform of interest, which is a subclass of the SOSA classes Platform and FeatureOfInterest.

A **PlatformOfInterest** is a subclass of *sosa:Platform* and *sosa:FeatureOfInterest*. A *sosa:Platform* is an entity that hosts other entities, particularly sensors and other platforms (e.g. EVCharger, battery, garage). Platforms are devices or facilities within a system, where sensors are located. A Platform can *sosa:host* other platforms or sensors. Using the relation *sosa:hosts*, a hierarchical relation between platforms and sensors can be represented.

As Platforms and their hierarchies define the topology of the system, they are the main features of interest in an ExpCPS according to the SENSE ontology. Any user question connected to *WHERE* anything happened is essentially asking *at which platform* a certain event occurred. Therefore, the combined class *PlatformOfInterest* allows the focus to be on Platforms as the main feature of interest in an ExpCPS.

In the motivating example, *Battery1*, *EVCharger1* and *Garage1* are platforms of interest. *Garage1* hosts *Battery1* and *EVCharger1*, as they are installed inside the garage (see lines 8-10 in Listing 1).

A **PlatformType** is defined as a separate class for explicit typing to define common properties of platform types independently of a specific instance of a platform.

A **Sensor** is a device or agent, which collects measurements of an observable property (e.g. AP). Sensors are hosted by a platform, which means a sensor is located at/within a Platform and its measurements correspond to this platform (e.g. *Garage1* hosts *OperatingEnvelope_Sensor1* in Fig. 1). Each sensor stores information on how to access its sensor measurements in the dataSource property (e.g. access token to a timeseries database).

A **Virtual Sensor** is defined as a subclass of sensor. A virtual sensor can be defined additionally to existing sensors in the system.

A virtual sensor is defined when a platform has an observable property that can be determined using data from multiple sensors, even though there is no single sensor directly attached to the platform to measure that property (e.g. there might not be a sensor at Garage1 measuring active power, but a virtual sensor can be defined, which sums up all active power measurements within the garage).

An **Observable Property** is an observable quality (property, characteristic) in the system. Each sensor observes an observable property (e.g. `APSensor3` observes `ActivePower`).

These concepts (PlatformOfInterest, Sensor, Observable Property) and their corresponding relations allow the definition of topological relations between system components to facilitate event explainability. It enables the representation of sensor locations at platforms, sensor measurements, where to access relevant timeseries data and how platforms are connected to each other. An example of an instantiation for the toplogy knowledge according the the SENSE ontology is shown in Listing 1.

[4] https://www.w3.org/TR/vocab-ssn/

Listing 1: topology knowledge instantiation excerpt for the motivating example (cf. Fig 1). It shows the definition of a garage as a platform and a sensor as well as a virtual sensor installed at the garage.

```
@prefix : <http://w3id.org/explainability/sense#> .
@prefix ex: <http://example.org/explainability/motivatingexample> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Garage1 a sense:PlatformOfInterest ;
    rdfs:label "Garage1" ;
    sense:hasPlatformType :Garage ;
    sosa:hosts ex:Battery1,
        ex:EVCharger1,
        ex:EVCharger2,
        ex:OE1,
        ex:AP1 .

ex:OE1 a sosa:Sensor ;
    rdfs:label "OperatingEnvelope_Sensor1" ;
    sense:dataSource "timeseriesDB reference" ;
    sosa:isHostedBy ex:Garage1 ;
    sosa:observes :OperatingEnvelope .

:AP1 a :VirtualSensor ;
    rdfs:label "ActivePower_Sensor1" ;
    sosa:isHostedBy ex:Garage1 ;
    :linkedToSensor ex:AP2, ex:AP3, ex:AP4 ;
    :origin "sum of linked sensors" .
```

2.6 Observation Knowledge

To reduce computational overload and the storage of irrelevant data, we propose an intermediate event detection step to detect events of interest in the data streams, allowing to store relevant historical data, while reducing the overall size of the knowledge base.

The *Observation Knowledge* module contains concepts related to the system's runtime data, including sensor measurements and detected events. Furthermore, it also contains knowledge about procedures for creating these runtime data.

The central concept of this module is an **Observation**, which represents the act of measuring or estimating a particular property of the CPS, including detected events, and resulting state changes. Each observation is associated with a Sensor, which is crucial for relating observation knowledge to topological knowledge. Furthermore, observations always refer to an observable property observed by a sensor. This additional link is necessary because a single sensor can have multiple observable properties (e.g. Temperature, Humidity, and CO2 concentration).

Each observation produces a **Result** as the outcome of an observation. This can be a detected event, or a new state. Section 2.7 discusses these concepts in detail.

A **Procedure** defines the steps necessary for obtaining an observation. A procedure can define a measurement process that must be carried out to obtain comparable measurements across different sensors.

An **EventDetection**, a subclass of procedure, describes events the system can observe and how to detect them. Each event detection instance describes how to detect a particular Event Type for a particular Sensor in the CPS. Each event detection is characterized by two data type properties - *:definition* and *:method*. The **method** property defines the event specification

method employed, allowing different event detection methods to be defined. Examples of event specification methods range from formalisms such as signal temporal logic (STL) [16] to custom Monitoring Scripts written in Python.

In addition to specifying the event specification method, the definition property contains the specification itself. The event specification method of the event definition dictates the syntax and semantics of the definition. For example, the definition could contain a concrete temporal logic formula when the specification method is STL. In contrast, it could contain the name of a Python class when a custom monitoring script is employed.

Using a class hierarchy to define event detection methods was discussed in the ontology design phase. However, this design decision would constrain the ontology to only represent event detection procedures implemented in the context of current use cases, which is not the intended scope of this ontology. We aim to have a generic representation, flexible enough to be extended to multiple use cases.

Using strings for the method and the definition of event detection procedures allowed us to exclude the set of supported event specifications from the ontology design. While this worked well for the event specifications methods implemented in this research project, a more sophisticated ontological description may be required for more expressive specifications. As a result, this decision may change in future iterations of the SENSE ontology, providing a separate class for each supported event specification method.

ParameterBindings allow users to bind parameters to an event detection class. This class is needed to be able to represent the full event detection mechanism in the knowledge graph. Suppose a possible event can be detected if a signal exceeds a certain threshold. One can formalize this circumstance with a formula similar to $signal > threshold$. Once this formula becomes true, the SENSE system shall detect this circumstance as an event. By relating parameter bindings to an EventDetection procedure, implemented by a sensor, it is sufficiently defined, which signal (i.e., the signal measured by the respective sensor) should be evaluated. The bound value can be either a sensor measurement, i.e., a time series with a changing value over time, or a literal in the knowledge base. Each event detection procedure may require a different set of bound parameters. Usually, there are multiple event detection procedures for multiple Sensors, as each procedure has a different set of parameter bindings.

A full defintion of an event detection procedure, and its parameter bindings is shown in Listing 2, lines 1-9. An example of an observation as a result of the event detection is then shown in lines 11-19.

Beyond this circumstance, materializing all event detection procedures in the knowledge base can be beneficial for other reasons. Firstly, it allows for event detection implementations that can be reused across different systems as the knowledge base contains the event definitions, i.e., they are not hard-coded in the program. In addition, engineers can partially automate this parameterization by leveraging Semantic Web technology, as demonstrated in [17]. Lastly, in a sophisticated implementation of the event detection system that synchronizes with the knowledge base, adding, removing, and changing the event detection behavior is as simple as updating the knowledge base.

Listing 2: observation knowledge instantiation excerpt for the motivating example. It shows the definition of an event detection procedure and one observation, which was detected by the defined procedure.

```
@prefix : <http://w3id.org/explainability/sense#> .
@prefix ex: <http://example.org/explainability/motivatingexample> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:OEViolated_EventDetection a :EventDetection, :Procedure ;
    :definition "always[0:0](signal > threshold)" ;
    :hasParameterBinding [ a :ParameterBinding ;
            :bindsToSensor ex:AP1 ;
            :parameterName "signal" ],
        [ a :ParameterBinding ;
            :bindsToSensor ex:OE1 ;
            :parameterName "threshold" ] ;
    :method "STL" .

ex:AP1_Observation1 a sosa:Observation ;
    sosa:hasResult ex:OEViolatedEvent1 ;
    sosa:observedProperty ex:ActivePower ;
    sosa:phenomenonTime "2023-04-03T06:54:00+00:00"^^xsd:dateTime ;
    sosa:usedProcedure ex:OEViolated_EventDetection ;
    sosa:hasFeatureOfInterest ex:Garage1.

ex:OEViolatedEvent1 a :Event ;
    :hasResultType ex:OEViolatedEvent .
```

2.7 Causal Knowledge

The *causal knowledge* module covers the representation of causal relations in a system. Two types of information are represented in this module: *type knowledge* about predefined, general knowledge about types of events and states, as well as *actual knowledge* about instances of events and states, detected at runtime [18].

Event detection methods, defined as stream reasoning tasks, are focused on detecting phenomena at a certain time point. Causal relations, as defined by domain experts, however, are defining relations between phenomena that occur over a period of time. We discovered in expert workshops, when discussing causal relations, human reasoning is mostly concerned with phenomena, which can be discovered - which means they usually occur over a longer time period, phenomena at single time points are only the starters of a state that is discussed by experts. To bridge the gap between these two notions of phenomena, we propose the following definition of events and states in the ontology:

- An **event** is a phenomenon that occurs at a certain point in time, at a specific place and was detected by an event detection procedure.
- In contrast, a **state** is a phenomenon that occurs over a period of time. A state transition is triggered by an event. Therefore, each state is started (:hasStartEvent) and ended (:hasEndEvent) by an event.

Events and states are both results of an observation. The type of event or state is defined by the relation *:hasResultType* to a *:ResultType*.

ResultTypes are event types and state types, which are concepts of these phenomena. Each event type represents a transition between state types, which is defined a priori (e.g.

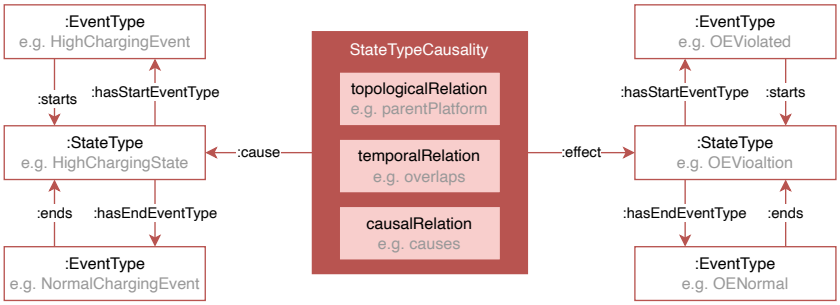

Figure 3: State Type Causality definition in detail, featuring an example definition of causality in grey.

`HighChargingEvent` starts `HighChargingState`, and `NormalChargingEvent` ends `HighChargingState`). Each StateType has a number of event types which are responsible for starting (s:hasStartEventType) or ending (s:hasEndEventType) the stateType. In Figure 3, this relation is shown as part of the presented causal relation, the turtle definition of event types and their corresponding state types is depicted in lines 1-14 of Listing 3. A state derivation query can use this information combined with topology data to identify which state has been changed at which sensor by a certain event (e.g. `HighChargingEvent4` at `APSensor3` starts `HighChargingState3` at `APSensor3` as a result of the event-to-state mapping indicating `HighChargingEvent` starts `HighChargingState`).

Causal Knowledge builds on the notion of states and state types. It is defined as type causality, which is a relation between state types (e.g. *HighChargingState*), and actual causality is a relation between state instances (e.g. `HighChargingState3` at `APSensor3`).

Type causality is represented in the ontology via the class **State Type Causality**, which is a relation between State Types. Each causality relation consists of a cause state type and an effect state type as well as three parameters (causal, temporal, topological) that give detailed information on the nature of the causal relation (as shown in Figure 3). State Type Causality between state types is to be defined as a prerequisite for detecting actual causal relations between states at runtime. We rely on the definition of the Causal and Temporal Relation Scheme (CaTeRS) [19] for causal and temporal relations. In the FARO ontology [9], even more types of causal relations are defined to capture the intricacies of causality in an ontology. Since the causalities in the SENSE ontology are especially defined to represent expert knowledge from interviews and workshops, the definition of causal relations is heavily based on linguistic causal research, such as the CaTeRS scheme, where causal relations are extracted from text. A detailed definition of State Type Causality in the SENSE ontology, including an example relation is shown in Figure 3. Additionally, the turtle representation of the class is exemplified in lines 16-22 of Listing 3.

State Type: A state type is the concept of a state. It has a description of the characteristics that this type of state exhibits (e.g. `HighChargingState` is a state when an `EVCharger` charges more than 50kW). A state type is not related to a specific time or place. It is a concept of a state that can occur at runtime.

causalRelation: The causal relation defines a more detailed view of the type of causality as compared to a simple "caused by" relation. There are three types of causal relations that can occur between two states ($stateA \rightarrow stateB$) - based on the CaTeRS scheme on causal relations:

- *cause:* If stateA occurs, stateB most probably occurs as a result.
- *enable:* If stateA does not occur, stateB most probably does not occur (not enabled to occur).

- *prevent:* If stateA occurs, stateB most probably does not occur as a result.

temporalRelation: The temporal relation captures the temporal aspects of a relation between two states. There are two options for temporal relations ($stateA \rightarrow stateB$) that showed to be relevant for representing causal knowledge in use cases so far (these options can be extended based on future use case requirements):

- *before:* stateA starts and ends before stateB.
- *overlaps:* stateA starts before stateB, but ends after stateB has started.

The temporalRelation of a StateTypeCausality defines a restriction on the temporal proximity of two states to be counted as a valid causal relation. Some causal relations only hold if the cause and effect states overlap (e.g. *HighCharging -¿ Overload* only if the states happen at the same time), while some relations also hold if there is a temporal difference between the states (e.g. *HighCharging -¿ LowBatterySoC*).

topologicalRelation: The topological relation considers the physical/logical relation between two states as a constraint for the causal relation to hold. We consider a hierarchical representation of platforms and their sensors. A state is always associated with a sensor, hosted by a platform, via an observation. In Fig 1, the topology representation of the motivating example is depicted on the right side. Platforms are shown in light blue, while sensors are shown in dark blue. Three types of topological relations are defined as causal constraints. each of the relations is described below and exemplified in red in Fig. 1:

- *samePlatform:* stateA and stateB are associated with a sensor on the same platform (e.g.`SoCSensor2` and `APSensor2` are associated to the same platform `Battery1`).
- *parentPlatform:* stateA is associated with a sensor on a platform that is hosted by the platform of stateB (e.g. `APSensor1` is associated with `Garage1`, which is the parent platform of `EVCharger1`. `APSensor3` is associated with `EVCharger1`, forming a `parentPlatform` relation from `APSensor3` to `APSensor1`). Note that this relation is the only non-symmetric topological relation.
- *siblingPlatform:* stateA is associated with a sensor on a platform that is hosted by the same platform as the platform associated with stateB (e.g.`APSensor3` is associated with `EVCharger1` and `APSensor4` is associated with `EVCharger2`. Both platforms are directly hosted by `Garage1`, so the two sensors are hosted by `siblingPlatforms`).

In the ontology definition, these relations are defined as strings. We provide SHACL shape definitions to ensure the conformitiy of the relations to the defined options. A more detailed description of these shapes and their definitions can be found in Section **??**. The definition of these relations can be extended if needed for specific domains or use cases, simply by updating the SHACL shapes that define the accepted options for the relations. The current choice on the representation of causality in the SENSE ontology is based on a tradeoff between expressivity and efficiency, as well as considering how to translate expert knowledge from workshops into a usable format for automated explanation frameworks.

A thoroughly crafted causal knowledge dataset is crucial for an explanation engine to provide meaningful and correct explanations of states. This knowledge not only provides information about causality, but it also provides information on which types of states are relevant. Therefore, it serves as an indication to create corresponding event detection procedures in the event detection module of the ExpCPS framework.

The relation **causallyRelated** defines the causal relation between two states (i.e. phenomena over a period of time, related to a specific time and place). Based on causal type knowledge, stored in the *StateTypeCausality* class, and states detected in the system at runtime, actual causality between states can be derived. This relation can be created automatically at runtime. For each state, which is added to the knowledge base, the *StateTypeCausality* between the state type of the newly added state and state types of existing states is compared, filtering the causal relations based on the properties of the *StateTypeCausality* (temporal, topological relation). If a causal relation between the newly added state and any existing state holds, they are causally related.

For each relation *causallyRelated*, an RDF-Star relation between the triple and the corresponding StateTypeCausality in the form *(¡¡:CauseState, :causallyRelated, :EffectState¿¿ , :hasCausalSource, :StateTypeCausalityA)* needs to be added to preserve the causality knowledge related to the relation.

Based on this knowledge, users can query for potential root causes of specific system states. More information on the identification of a root cause for a trigger state is provided in a separate paper [2].

Listing 3: causality knowledge instantiation excerpt for the motivating example. It shows the definition of event types (responsible for state transitions), state types, and the definition of a causal type relation between state types.

```
@prefix : <http://w3id.org/explainability/sense#> .
@prefix ex: <http://example.org/explainability/motivatingexample> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:OEViolated a sense:EventType ;
    rdfs:label "Operating Envelope Violated" ;
    sense:detectedBy ex:OEViolated_EventDetection ;
    sense:startsStateType ex:OEViolation .

ex:OENormal a sense:EventType ;
    rdfs:label "Operating Envelope Normal" ;
    sense:detectedBy ex:OENormal_EventDetection ;
    sense:endsStateType ex:OEViolation .

ex:OEViolation a :StateType ;
    rdfs:label "Operating Envelope Violation State" ;
    :hasStartEventType ex:OEViolated ;
    :hasEndEventType ex:OENormal .

ex:StateTypeCausality1 a :StateTypeCausality ;
    rdfs:label "EVHighCharging causes OEViolation at the parentPlatform
        , when they overlap temporally" ;
    :cause ex:EVOverallHighCharging ;
    :effect ex:DemandEnvelopeViolation ;
    :causalRelation "causes" ;
    :temporalRelation "overlaps" ;
    :topologicalRelation "parentPlatform" .
```

2.8 User-Context Knowledge

The *user context* module contains information connected to a user requiring explanations. Each user has a certain role in the system, preferences, access rights, and states they are interested

in, which influence the explanation they require. Any information, which is required for providing a personalized explanation to a certain user is stored in this module. Personalization in the context of system explanations are mainly considered in two aspects: user rights, determining the type of knowledge a user is allowed to see, user type, defining preferences on how an explanation should be presented to a user.

User: A user who is interested in system explanations is to be registered (see Listing 4, lines 1-4). Each user, who requires explanations from the system, is stored as a user in the knowledge base, with their respective features:

- **UserType** defines a user in terms of their preferences for a specific explanation medium, or technicality level of an explanation (e.g. lay user vs technical user, visual explanation vs text-based explanation).
- **UserRole** defines a user in terms of the system context. A user role defines the access rights of a user within the system. View Access Rights define what Platforms a user can know about, thus which platforms are included in an explanation for this user, while Control Access Rights define the Mitigation Options for an anomaly in the system, which are presented to the user (e.g. unplugging an EV can only be a valid mitigation option for the owner of the EV).

Access Rights are always connected to a platform of interest. View access rights define the rights to view a platform's state, while control access rights define the rights to change a platform's state. An excerpt of an access right definition is shown in lines 6-8 of Listing 4.

Listing 4: user context knowledge instantiation excerpt for the motivating example. A user and their access rights in the system is defined.

```
1  @prefix : <http://w3id.org/explainability/sense#> .
2  @prefix ex: <http://example.org/explainability/motivatingexample> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5
6  ex:UserA a :User ;
7      :hasUserType ex:layUser ;
8      :hasUserRole ex:EVChargingUser ;
9      :containsAccessRights ex:EVChargerUser .
10
11 ex:AccessToAll a :AccessRight;
12     :allowsControlAccessTo ex:EVCharger1 ;
13     :allowsViewAccessTo ex:EVCharger1 .
```

2.9 Explanation Knowledge

The *explanation* module stores explanations of states that have been given to users. Each explanation is customized according to the user for which it is designed. A user's type and role is considered for creating an explanation of a state, including mitigation options.

MitigationPlan: For each state type, a mitigation plan is defined. It lists a set of options to avoid/mitigate a certain state type. A mitigation plan contains multiple **MitigationOptions**, each option requiring specific control access rights to be able to conduct the mitigation option. Therefore, the user role, including a user's access rights, is relevant to propose adequate mitigation options to a user.

Explanation: An explanation always explains a specific state, based in the explanation interest of a user. If a state is triggering such an explanation, it is marked as a triggerState. An

explanation stores the user-centered version of a root-cause analysis of a specific state (based in the causallyRelated relation). The personalization of this analysis depends on the *UserType*, *ViewAccessRights* and *ControlAccessRights* of a user. Each explanation is designed for a user of the system and explains a *state* instance. Based on the user and the causal path, mitigation options are presented to a user. An example for an explanation and respective mitigation options is shown in Listing 5.

Listing 5: explanation knowledge instantiation excerpt for the motivating example. For each state in an explanation, mitigation options can be defined in a mitigation plan.

```
@prefix : <http://w3id.org/explainability/sense#> .
@prefix ex: <http://example.org/explainability/motivatingexample> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:EVHighCharging :hasMitigationPlan ex:EVHighChargingMitigationPlan .

ex:EVHighChargingMitigationPlan a :MitigationPlan ;
    :hasMitigationOption ex:unplugEV .

ex:Explanation1 a :Explanation ;
    :designedFor ex:UserA ;
    :explains ex:OEViolation1 ;
    :suggests ex:unplugEV ;
    :considersAccessRights ex:EVChargerUser .
```

3 Type Causality Knowledge Acquisition

3.1 Towards causality identification problem: From time series to causality relations

Understanding how a set of variables are related is becoming and important aspect when developing an information system that makes decisions using large amount of observational data. Having knowledge about causal relationships between variables help in comprehending the dynamics behind complex systems. Data-driven methodologies using statistics and machine learning (ML) allow processing vast amounts of data and applying algorithms to find causal relationships among variables with the aim of establishing causality.

Causality of $X \rightarrow Y$ ($\rightarrow$ meaning X causes Y) can be defined if and only if an intervention or manipulation in X affects Y, where X is known as independent, or cause variable and Y is known as dependent or effect variable. In complex systems, the effects are known events and are time specific. One approach is to calculate correlation coefficient between the variables. Nevertheless, the measure of correlation provides the degree of association and does not consider the directionality of the relationship. Therefore, more advanced measures should be used to provide the directionality among the variables.

Identifying causality relations among multiple variables in complex systems is crucial for understanding the underlying characteristics and dynamics of the system. Such exploratory data analysis as suggested by [20] would help in getting insights on how the different system components interact and influence each other over time, which in turn could be helpful making informed decisions. Another benefit is gaining knowledge on cause-and-effect relationships that can be used for predicting future states or events within the system and regulate the system more effectively. However, the task of inferring causality relation among variables comes with a set of challenges. In complex systems, interactions between variables can be nonlinear and there can be time delays between the variables. There can also be confounding variables that may influence both the cause and effect and can complicate the process of causal inference. Additionally, causal relationships among variables within a system can change over time and failing to capture such changes may result in misleading conclusions.

Time series data of complex systems describe traces of systems over time and can be used as inputs to quantitative methods for inferring causality relations between them. Time series can be derived as observations recorded during experiments or sensors collected over time. The goal of time series analysis for causal identification in complex and dynamic systems is to statistically and reliably estimated causal links, including their time lags [21]. The causality identification problem is related to the challenge of extracting causal relationships between variables based on observed time-series data. For instance, consider a set of variables denoted as $X_1, X_2, \ldots, X_n$, where each X represents a time-varying variable within a complex system M (figure 4, [1]). The main aim of the analysis is to determine whether there exists a causal relationship between a variable X_j (the cause) and a target variable X_k (the effect). The causality identification problem involves detecting whether changes in the values of X_j influence changes in the values of X_k. The two aims of such analysis would be to detect general causal interactions among the components of M, including time lags and to quantify the strength of causal interactions within M in a well-interpretable way.

The challenge during causality identification from time series data is in estimating a function that represents the causal relationship and determining the appropriate time lag or delay between the cause and effect. Various methods such as Granger causality tests and structural equation modeling, which are statistical methods, and Transfer Entropy (TE), which is an

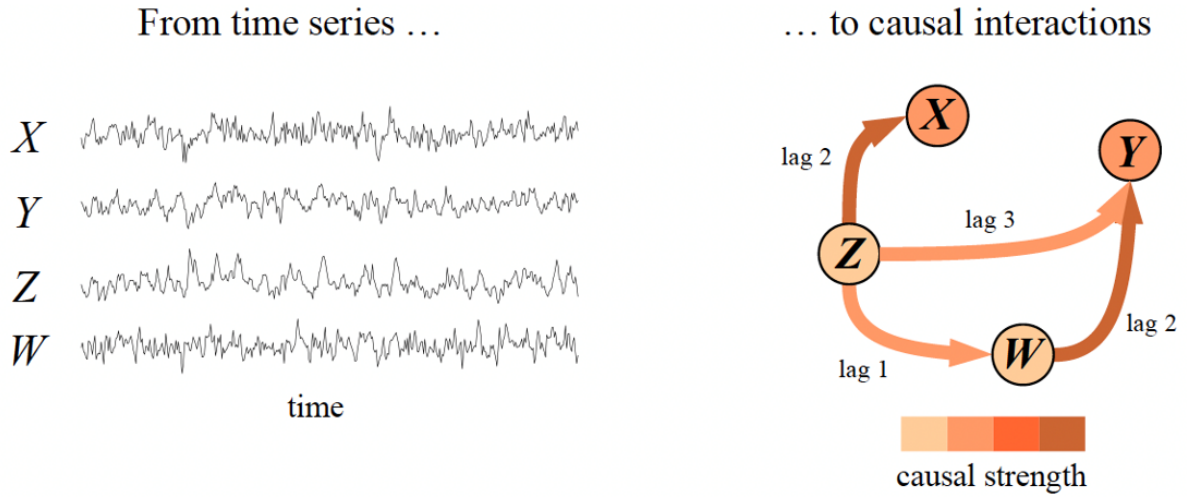

Figure 4: The process of analyzing different time series within a complex system to extract causality relationships between them [1]

information-theoretic method, are employed to address this problem.

3.2 Related Work

Time series are present in various forms and can be generated in different domains, for instance in healthcare through monitoring systems using devices and sensors, industry through predictive maintenance systems, and other application domains by using data such as images, acoustic signals, etc.

In literature, many studies have focused on identifying causality problem and many techniques have been introduced. The very first work on causality inference was presented by Clive Granger in 1969 [22] where the Granger causality test was introduced. This method has been widely used studies dealing with, but not limited to, econometrics time series data. The idea with the method is to determine if a time series can be used to predict the future values of another time series. This method was to study brain connectivity using neuroimaging time series [23]. In the paper by Sugihara et al. [24] causality methods were used to infer causalities within climate systems based on analysis of time series data. The paper addressed the challenges of processing time series where there are nonlinear and non-stationary dynamics. When determining causation in time series there is a need to model the delayed causation between them and this need inspired the development of vector autoregression, which involves analyzing previous data samples with current ones to determine delayed causal links [25]. One drawback of this method is that when applied in datasets with high dimensionality it can lead to low detection power. In addition, when dealing with time series data one important aspect is to consider methods that are robust to detect non-linear interactions and operate without any priori model. Granger causality test does not address nonlinearity in time series.

Another method that has been applied in literature is TE. TE is like Granger causality [26]. Nevertheless, Granger causality is not good for modelling non-linearity in the time series. This drawback of Granger causality is related since it is based on linear models and is not appropriate for nonlinear systems. TE has been proved as effective measure of capturing dynamical features among different components in time series. TE has been applied in different application domains. TE has been used by in a real-world dataset consisting of financial time series to characterize the information flow among different stocks [27]. Additionally, it has been applied in neuroscience for identifying causal interactions in brain networks [28]. TE has been used as

a metric to detect presence of nonlinearity in health monitoring systems [29]. Even TE has its own disadvantages. It is more challenging to be automated, computationally intensive, and more complex to interpret [30]. To overcome these issues and at the same time account for non-linearity in time series some methods have been proposed utilizing ML methods. For instance, Tank et al. [31] suggested a new, extended Granger-based model using Structured Multilayer Perceptrons or Recurrent Neural Networks. Similarly, Nauta et al. [32] has proposed a new method for discovering causal relationships in time series data using attention-based Convolutional Neural Networks. Rossi et al. [33] presented a software tool implementing an architecture of timed-delayed neural networks (TDNNs) for causality-detection in physical time dependent systems. TDNNs has been shown to perform well and does not require an extensive amount of data to provide robust conclusions.

3.3 Causality detection algorithms implemented in the SENSE project

In this section of the report, we will provide a summary of the methods that were implemented within the SENSE project to infer causal relations from time series. The dataset that we used was about a smart grid use case and simulated using Siemens Bifrost simulation engine. The dataset contained data about a month with a range of time series representing different properties of a smart grid system. Some of the variables that were included in the dataset were: `TRAFO_SECONDARY_VOLTAGE@TRAFO_BUILDING, VOLTAGE_ANGLE3P@GRID_NODE`, following the naming format `VARIABLE@SYSTEM_COMPONENT`. The methods were applied on the time series selected based on expert knowledge and tested to quantify the causality relations among them.

The aim was to infer causal relations between different properties in a cyber-physical system e.g. smart grid. Such systems are very complex in nature with a large number of interactions between its properties. A challenge would be to design a custom ML model (e.g. neural network) that captures key aspects of the underlying system processes and that can generalize to different situations that could occur within a smart grid. In addition, the process for designing and validating an ML model requires sufficient domain knowledge and understanding of the mechanisms within a smart grid system to correctly specify the underlying model. Not being able to map the underlying relations within the system would also impact the possibility to check the validity of our methods [24]. An ML approach would also be time consuming to design since it would require mapping of the time series of the different properties of a system to different events. This type of work would need sufficient labeled data for training and validation and involvement of domain experts.

An alternative that we have undertaken in the project was to employ model-free approaches, which typically rely on statistical learning [34]. By doing this, our models would not need to focus on specific dynamics of a target system (in our case a smart grid system) but instead examine the characteristics (“Information flow”) among the time series of the system and cross map them. In view of this, we rationalized for the use of the following model-free approaches, including Granger causality and TE, that have been commonly used in literature to infer causal relations in time series data in different fields of studies. As a complement to the statistical methods, the pre-trained TDNN tool [33] was implemented and tested.

Below follows an overview of the methods that were implemented.

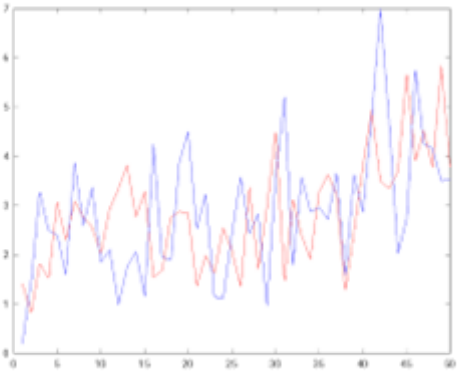

Figure 5: An example with two time series X (red-colored) and Y (blue-colored)

3.3.1 Method 1: Granger causality

Granger causality is based on prediction. If a signal X is causal for another Y, then past values of X should contain information that helps to predict Y better than merely using the information contained in past values of Y. Alternative definition: "If discarding X reduces the power to predict Y then X contains unique information about Y and thus can be concluded that X Granger-causes Y". The presence of this relation between X and Y can be referred as "X Granger causes Y".

Mathematically, this relationship can be formalized by a vector autoregressive (VAR) model where future value of Y is modeled as a linear combination of historical values of X and Y (figure 5). Each VAR coefficient is then tested for its significance whether it is helpful for predicting Y via t-tests followed by testing the final model via an F-test. If any coefficient is significant then we can conclude "X Granger causes Y". The concept of including an additional variable for better prediction of another variable is related to the linear regression model where an independent variable is declared significant if the full model predicts the dependent variable better than the model without this variable. The underlying assumption is that the time series should not be stationary and that their properties should not depend on the time they are observed. For instance, time series with trends or seasonality are not stationary since the former will affect the values in the time series at different times.

A Granger Causality test would return an output where

- 0 – indicates that the null hypothesis of noncausality should not be rejected. In other words, there is not enough evidence to suggest that the time series X Granger-causes the time series Y.
- 1 – indicates rejection of the null hypothesis of noncausality, which means that there is enough evidence to suggest that the time series X Granger-causes the time series Y.

3.3.2 Method 2: Transfer Entropy (TE)

TE is a measure used to quantify the directed information flow between two time series [35, 36]. TE provides means for quantifying how much information from the past of one time series can be used to predict the future of another time series. In other words, TE describes the uncertainty of an information in time series to quantify the directed information flow from one time series to another.

In contrast to Granger causality method which is regression-based, TE is an information-theoretic method and can capture both linear and non-linear dependencies between time series, making it suitable for analysis of causality in complex systems. The larger the value of TE, the more information is transferred from one time series to another, suggesting a stronger influence of a time series on another one. A value of 0 indicates no influence or information transfer between the time series.

3.3.3 Method 3: Pre-trained TDNN

Disadvantages when using Granger causality and TE:

- Can give contradictory results and can perform better in certain specific application than others
- Not appropriate for problems with many time series. Granger causality and TE cannot represent the nonlinearity when using multiple time series at the same time.

We have used the toolbox implemented by Rossi et al. [33] and has the following functions:

1. Causality Detection: performs a statistical test to evaluate if the variance of a time series is statistically different from the variance of another time series. If a p-value is smaller than a threshold (e.g. 0.05), a time series influences another time series.
2. Time Horizon Detection: measures the time horizons of the influence, that is the time difference between the influencing event of X and the influenced event in Y.

Next, we will provide technical details about implementation of the three methods. Methods 1 and 3 run on Matlab and method 2 runs on Python environments. Each method requires the data in specific formats. For instance, methods 1 and 3 require the data to be formatted as a Matlab catalogue with ".mat" extension. The method 2 requires the data to saved in a CSV format. To run methods 1 and 3 the Matlab toolboxes likl Econometrics, Deep Learning Toolbox, Statistics and Machine Learning Toolbox, should be installed on the computer running the code. The method 2 requires the computer to have Jupyter environment to be set up along with libraries such as Panda and PyIF to be installed.

3.4 Evaluation of the methods

In this section we will provide details on the experiments that were performed (Table 3). For all the experiments simulated data provided by our industrial partner Siemens were used.

The following results were derived for experiment 1. When hypothesis 1 was tested we found out that the methods 1 and 2 could detect causality. Nevertheless, for other hypotheses the results were mixed showing contradicting causality relations and/or two-way causality relations (that is the time series cause each other in both ways).

For experiment #2, which focused on specific events that were pre-identified through observations in a visualization tool and during the time starting from the event and an hour later, the results are displayed in Table 4.

The causality results from the methods are different to each other. In other words, when one method detects causality, the other methods do not detect any causality. The results are not either consistent within the methods.

After segmenting the time series 2 hours before the event and an hour after the event according to experiment #3 the results showed no changes. Lastly, as per experiment #4 the

Experiment#	Goal	Description of tests and hypotheses
1	To test causality detection algorithms using time series measured during the whole time.	Set of cases were derived from a project member who has expert knowledge in smart grids. Data segments: full period of the measurements that is one month. Hypotheses: • PV Power Production causes Batter State of Charge • Car Charging Demand causes Grid Demand • Car Charging Demand from Fast Charger causes Battery State of Charge (Reduction) • Grid Demand causes Battery Demand (or the other way around)
2	To test causality detection algorithms focusing on specific events happening in the smart grid system	Eight events in the dataset were identified using a visualization tool developed as part of another task in the project. The tool visualized the different time series and the time when a violation in the system occurred. A set of cases were derived by a colleague. Data segments: To improve the power of the methods to detect causality we decided to focus on a time segment related to the events. We decided to consider data for analysis from the start of the event till an hour after the event. Hypotheses: • Grid Demand causes Total Charging • Total Charging causes Battery Charge/Discharge
3	Test the methods with different time intervals.	The assumption was that when an event happens the characteristics and links between different time series are changed and reflected before the actual event. For this reason, we decided to focus on the same scenarios as in experiment #2 but include 2 hours of data before the event. Same scenarios and hypotheses were tested as in experiment #2.
4	Test refined versions of the methods.	The methods were refined by preprocessing time series before using them as inputs to methods. The assumption was the time series had non-linearity features. To eliminate nonlinear trends from the time series a low (6)-order polynomial filter was fitted to them. Same scenarios and hypotheses were tested as in experiment #2.

Table 3: Experiments and hypotheses tested.

	Grid demand causes Total charging	Total charging causes Grid demand	Both cause each other	No causality	Not relevant
GC	2	2	3	1	
TE	2	3	0	2	
TDNN	3	2		1	1

Table 4: Results for the 8 system events for each method and combination of causality.

time series were initially pre-processed using a filter to remove the non-linearity features in them and then used as inputs the three methods. Similarly, no changes in the results were observed.

3.5 Conclusions and Future work

The findings from the conducted experiments yield both insightful and complex results.

In experiment 1, the detection of causality through hypothesis 1 demonstrated promising results, especially, for methods 1 and 2. However, the presence of mixed and contradictory causality relations for other hypotheses, including two-way causality, highlights the complex nature of the relationships within the time series. Experiment 2 focused on specific pre-identified events. Again, significant divergence in causality results among the employed methods was observed. There was a lack of consistency within methods and cases where one method detected causality while others did not establish reliable causality links. Contrary to expectations, experiments 3 and 4, which involved time series segmentation around events and pre-processing time series by removing non-linearity features, did not yield observable changes in the results. These results suggest that temporal proximity to events might not significantly impact the causality relations, at least within the investigated time series frame.

As future work, one might consider the following steps. A more nuanced examination of causality during specific events could provide valuable insights. For instance, discussing the cases with domain experts may help in ensuring the existence and magnitude of causality relations between time series during a specific time frame. Additionally, testing data from diverse real-world data sources (e.g. smart buildings in cooperation with the industrial project partner AEE INTEC) could offer a comprehensive understanding of the relationships between time series, which in turn could shed light into validity and reliability of the methods. Finally, there would be a need to conduct extensive validation and robustness testing of the causality detection methods against known ground truth scenarios to provide confidence in their applicability and reliability.

4 Explanation Generation and Ranking

Explainability of a system is the ability to explain its behavior to the user. In a world, where increasingly complex systems are helping users in their everyday lives, explainability is becoming a crucial feature to be implemented in any system. A system, which can explain its decisions and reason about the root cause of a situation increases user trust, reduces maintenance time and costs and even helps users to make the right decisions when interacting with the system.

By definition, an explanation seeks to make a situation, object or event more understandable. However, creating understandable explanations is a tricky task. There is no one-fits-all solution on building an explanation that makes sure the explanation seeker improves their understanding of a certain situation. To achieve understandable explanations, multiple factors have to be considered, such as the knowledge of the explanation seeker, the context they are in as well as the actual reason for a situation to occur. In a Cyber-Physical System (CPS), an explanation of a situation or decision of the system can be created by analysing the system data. However, such a trace-based explanation is only one possible type of explanation that could satisfy user needs.

We are proposing an integrated explanation engine for Cyber-Physical, which can derive user-centered explanations from system topology data, time series sensor inputs and causality knowledge from domain experts. For a good representation of causality, we propose a multi-faceted causality input. This more intricate representation allows us to create more accurate and precise explanations, catering explanations to user's needs and context.

In this chapter, we are presenting the current version of the explanation engine in the SENSE project. In figure 6, the envisioned workflow from input data to explanations is shown. The first part, getting from raw time series data to Events, is part of Workpackage 3, in Task 3.3 and wil not be elaborated further here. The focus of this task (explanation generation and ranking is the center of the figure, where Events are converted to Causal Paths with the help of additional input, such as topology information of the system as well as Event type causality knowledge. The output of this task would be a list of causal paths, which constructs the basis for user-centered and actionable explanations (to be tackled in T 4.4). In this first version, we have focused on the SeeHub PoC. It contains a public garage, which offers 8 EV charging stations. The garage further contains a battery as well as a PV System, which help to balance the power needs of the electric vehicles charging at the facility. The local grid operator has imposed an operating envelope on the facility operator of the garage, determining the minimum and maximum capacity the garage is allowed to demand or feed into the local grid. This is necessary to prevent the local power transformer from overloading. If the envelope is violated, an explanations is required for the facility operator of the garage. This is the explanations we are aiming for.

In section 4.1, we will discuss the literature on events and causality that forms the basis of our approach. In section 4.2, we will focus on the methodology of the approach to get from Events to Causal Paths and will then show first results of applying this approach to the SeeHub PoC. Finally, we will discuss our findings and future improvements of the work in section 4.3.

4.1 Related Work

Explanations and Explainability has been discussed and researched in many domains. In the context of the SENSE project, we will focus on explainability in Cyber-Physical Systems as well as explainability in AI. While explainable AI is not the focus of the project, there are many parallels between the two domains, especially since some form of AI, either in the form of

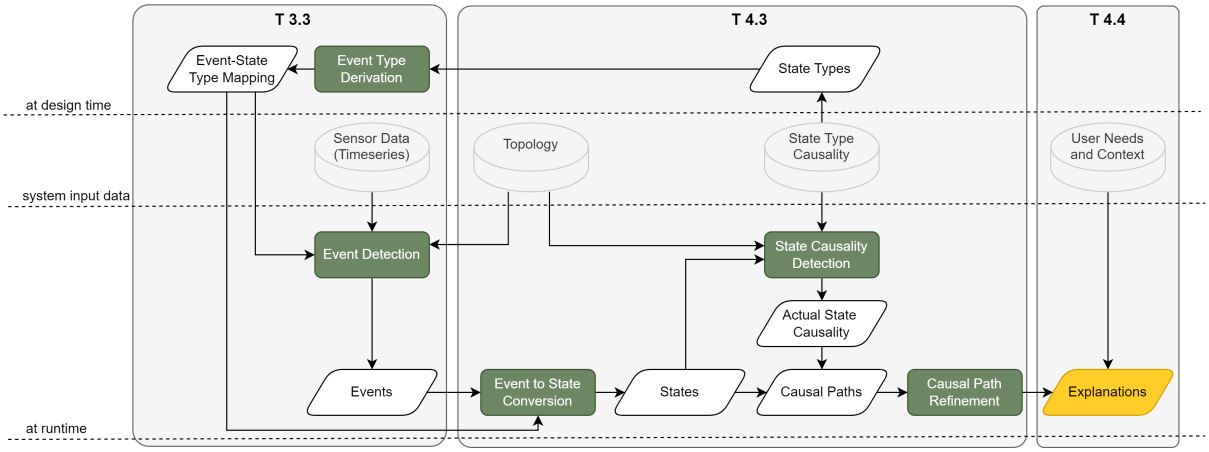

Figure 6: Workflow from input data to Explanations

decision support or data analysis processes, is implemented in almost every CPS. Therefore, we will analyse the research of explainability in both domains before we conclude with an analysis of the most important aspects to consider from either of them.

As AI becomes a regular part of our daily lives, the fact that most AI models are ”black-boxes” in terms of how they come to their decisions has resulted in increased research on explainable AI. Making AI models explainable can help in increasing user trust as well as establish accountability in model decisions. However, current Machine Learning models mostly rely on statistical inference, which is very different from human-level reasoning. While statistics is based on probability values based on pre-seen data, human reasoning is based on causal models, where we use our knowledge of cause and effect relations between events to explain a situation or decision. For a user to comprehend the decisions of a model, it is important to enhance statistical inference with causal knowledge. These causal cognitive models have only recently been ”algorithmitized” to enable mathematical exploration of causal relations [37].

On another note, research on explainability has shifted from mechanistic explanations to user-centric explanations, which are context-aware and consider the comprehensibility of an explanation [38]. While mechanistic explanations provide provenance information and justifications that are objectively valid, they might be insufficient to meet the user’s needs, especially in the face of increasingly complex systems that need explaining. User-centered explanations are thus an extension of the provenance-enabled explanations as they incorporate domain-knowledge and user’s context to generate comprehensible explanations of the functioning of a system.

Depending on the user, the type of explanation that is suitable for them might differ. In [39], an overview explanation types including their definitions and sufficiency conditions for each explanation type is presented. The authors argue that explanations for users need to include multiple types of explanations as each type serves a different purpose.

4.2 Methodology and initial Results

We have established a workflow, where causal relations between events in the system can be derived based on their position in the system, temporal occurrence as well as their event types. Descriptions of data types, classes and properties of the Semantic Data Model are described in detail in section 2.

As Input Data for the Semantic Explanation Engine, the following Data Sources are required:

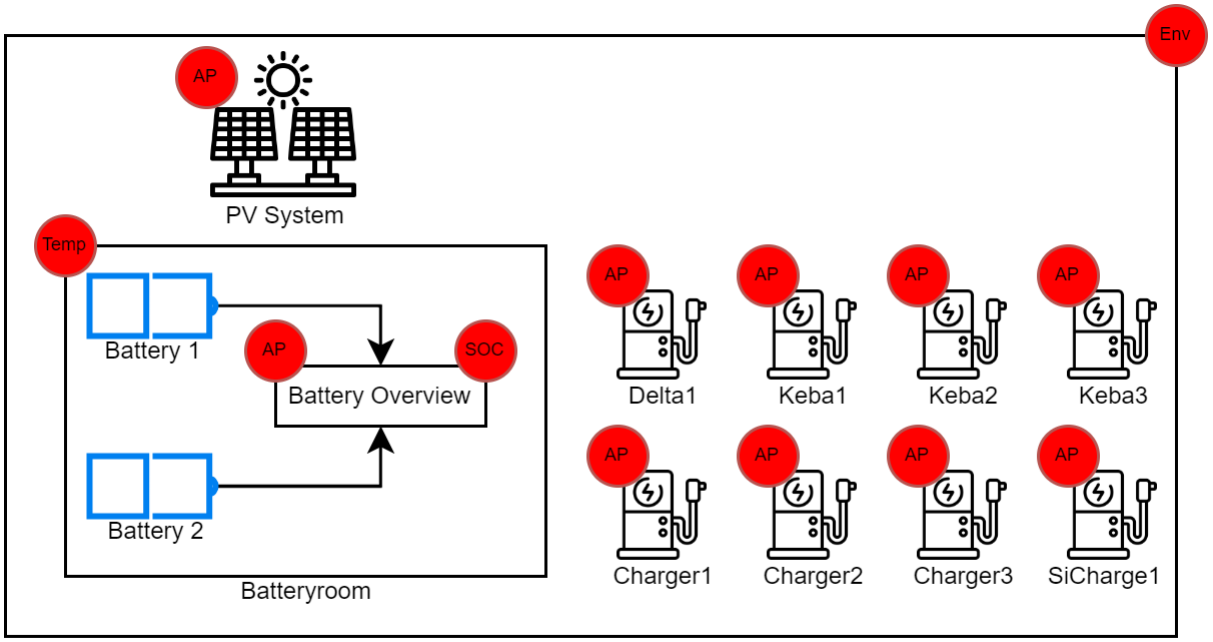

Figure 7: SeeHub topology

- **Topology Data** topology data contains static data about the topology of the system. It describes the platforms, sensors and connections between them, what type of properties the Sensors measure.
- **State Types and State Type Causality** These Datasets contain a list of potential States, which are of interested within the system. A State is an occurence of interest, which happens over a period of time. Each StateType has a type of Sensor it can be associated with. It is possible that a StateType can be associated with multiple types of sensors.
- **Event Detection Data Stream** At runtime, a continuous data streams of detected events provides the basis for the creation of states and further to detect causal paths for explanations.

We have used data from the SeeHub use case for creating and testing a first version of the Explanation Generation Framework. In figure 7, the topology of the SeeHub use case is shown. In this use case, there is an operating envelope imposed on the facility operator, which defines a range of maximum electricity power demand or feed-in of the facility to the grid. Violations of the envelope range need to be detected and explained. A more detailed description of the use case can be found in Deliverable 2.1, where the SENSE use cases are defined in detail. Components of the system (i.e. PV System, Chargers, Batteries) can have Sensors attached to them, which measure some observable property (shown as red circles). Any component within the outline of another component is considered a sub-component of the bigger system. This setup constructs the topology input for the engine.

StateType	Definition	associated SystemType
Envelope Violation	The active power consumed/produced within the platform is exceeding the envelope limits imposed by the operatin envelope of the platform	EnvelopeViolation_Garage_Sensor
High Charging	active power consumed by the EVcharger is exceeding the operating envelope of the platform above.	AP_EVCharger_Sensor
Low Battery SOC	state of charge of the battery is within the lowest 5% ever recorded	SOC_Battery_Sensor
Battery Dip	active power provided by the battery has dropped for a time instance	AP_Battery_Sensor
Battery Discharge Loading	active power provided by the battery is increasing, but has not reached its high point yet	AP_Battery_Sensor
High Charging Difference	active power consumed by the EVcharger is increasing very fast	AP_EVCharger_Sensor
Battery Unused	battery is neither consuming any active power nor providing active power to the system	AP_Battery_Sensor

Table 6: SeeHub State Types

State Types and State Type Causality was defined based on the Use Case requirements and example explanations. In table 6, the list of state types that are of interest in the SeeHub system are listed. Each state type has a definition as well as a type of sensor it can be associated with.

In addition to the state types in the system, we need to define the relations between them. The causal relation between types of states are defined in the state type causality table. It contains the information about how events are connected to each other in as many facets as possible. In the CaTeRS scheme [19], a temporal and causal relation scheme was introduced, considering not only a simple "caused by" relation, but dividing the relation further into the temporal aspect as well as the causal aspect of the relation. We are using the terms for temporal and causal relations as defined in [19]. For temporal relations, there are four options (explained in the form $eventA \rightarrow eventB$):

- **before** eventA starts and ends before eventB
- **overlaps** eventA starts before eventB, but ends after eventB has started
- **contains** eventA starts before eventB and ends after eventB has ended
- **identity** eventA starts and ends at the same time as eventB

Furthermore, there are three options for causal relations [19]:

- **cause** If eventA occurs, eventB most probably occurs as a result.
- **enable** If eventA does not occur, eventB most probably does not occur (not enabled to occur).
- **prevent** If eventA occurs, eventB most probably does not occur as a result.

Cause	causal	temporal	topological	Effect
High Charging Difference	enables	overlaps	siblingPlatform	Battery Discharge Loading
Battery Discharge Loading	causes	overlaps	parentPlatform	Envelope Violation
High Charging	causes	before	siblingPlatform	Low Battery SOC
Low Battery SOC	causes	before	parentPlatform	Envelope Violation
Battery Dip	causes	overlaps	parentPlatform	Envelope Violation
High Charging	causes	overlaps	parentPlatform	Envelope Violation
High Charging	enables	overlaps	samePlatform	High Charging Difference

Table 7: State Type Causality input

Additionally, we have added a third category to the type causality definition: topological. This relation indicates the topological relation between two events in the system. These types are defined:

- **samePlatform** eventA and eventB are associated with a sensor on the same platform
- **parentPlatform** eventA is associated with a sensor on a platform that is hosted by the platform, where eventB is detected
- **siblingPlatform** eventA is associated with a sensor on a platform that is hosted by the same platform as the platform associated with eventB is.

In table 7, the state type causality for the SeeHub use case is shown. This table is derived based on domain expert knowledge for now. Future work could focus on different options to derive this information in a more automated way.

Based on the input data on topology, state types and state type causality, the explanation generation algorithm can derive a set of causal paths for any state that requires an explanation. In a first step, actual causality is derived from type causality through reasoning. The algorithm to create actual causality relations is desrcibed in algorithm 1. In the next step, a causal path needs to be found in the system through exploratory search for actual causality relations. The function to find a full causal path is described in algorithm 2. It is a recursive function, querying the causal path for actual causal relations between events up to the point, when no more causes are detected in the system. The result is a tree of causal paths, where all branches lead to a potential root cause. The final root cause in the system is not necessarily the actual root cause, but it is the last step of the cause, where reliable data is available. Any further causes would be events happening outside of our defined system and thus we do not have data about them.

4.3 Conclusion and Future Improvements

We have presented the SENSE explanation engine in its current version. We have established its usefulness in the SeeHub use case, which is a small real-life e-charging facility in Aspern. Future work will focus on implementing the engine for other use cases in the smart grid and smart building domain. Additionally, the refinement of the explanations and the evaluation of their correctness will be tackled in the next months within the project. We have written up and refined this approach in a paper accepted at the 2024 Energy Informatics DACH+ Conference [2].

Algorithm 1 derivation of state causality from state type causality

for each $typeCausality$ in $StateTypeCausality$ **do**
  for each $state$ in the data stream that has $eventType = typeCausality.Effect$ **do**
    $effectSensor \leftarrow$ sensor associated with $state$
    $effectPlatform \leftarrow$ platform hosting $effectSensor$
    $effectTimeInterval \leftarrow$ time between $start$ and end of $state$
    $causePlatforms \leftarrow$ list of platforms which have $typeCausality.topological$ relation to $effectPlatform$
    $causeSensors \leftarrow$ list of sensors hosted by $causePlatforms$
    $causeTimeInterval \leftarrow$ limits for $start$ and end of a potential $causeState$ according to $typeCausality.temporal$
    for each $state$ in the data stream fulfilling $causeTimeInterval$ associated with $causeSensors$ **do**
      add Causality relation ($causeState$,$typeCausality.causal$,$effectState$)
    end for
    Check for cause event types in the potential systems
    Return for now
  end for
end for

Algorithm 2 causal path identification

function findCause(effectState)
  query $causeStates$ in the relation ($causeState$, $causalrelation$, $effectState$)
  for each $state$ in $causeStates$ **do**
    save $state$ as a cause of $effectState$
    findCause(state)
  end for
end function

```
------------------------------------------------------------
Envelope Violation 114 @ Garage1
- is caused by Battery Discharge Loading 44 @ BatteryOverview
        Battery Discharge Loading 44 @ BatteryOverview
        - is enabled by High Charging Difference 229 @ SiCharge1
                High Charging Difference 229 @ SiCharge1
                - is enabled by High Charging 103 @ SiCharge1
                        High Charging 103 @ SiCharge1
                        - has no detected causes.
- is caused by High Charging 103 @ SiCharge1
        High Charging 103 @ SiCharge1
        - has no detected causes.
------------------------------------------------------------
```

Figure 8: example of a causal path detected by the explanation engine

5 Personalised and Actionable Explanation

Explanations in the SENSE-architecture aim to be catered to a user's needs. Additionally, we will also focus on the access rights and context of a user who is requesting an explanation of the system. All the information surrounding the user will be represented in the semantic model. In a recently submitted paper to the Neurosymbolic AI journal about the importance of ontologies in explainable AI, Confalonieri and Guizzardi [40] explain the importance of considering a user's context to create good explanations.

User-centered explanations are becoming a increasingly important in the era of explainable AI. The importance of a user-centered design of explanations has already been mentioned in 2007 by [41]. Explanations are a broadly used term, but explanations can have different aims, such as transparency, scrutability, increase of trust, effectiveness, persuasiveness, efficiency or satisfaction. In a user studies on movie recommendations and explanations of the recommendations, they have found that explanation features need to be tailored to the user and the context.

In another paper, which has been accepted to PerCom 2024 [42], a user-centered explanation engine has been proposed in the smart home domain. They have developed a system, where the user profile as well as the user state and role of the user is considered at the time of occurence of an event. All of these layers are important aspects which will be considered in the personalised and actionable explanations of the SENSE system.

On another note, the definition of a *good* explanation remains an open topic until today. Since explanations are generated with different goals in mind (as mentioned above), their evaluation is tends to be application-specific as well. In [43], an explainability factsheet is proposed to evaluate an explanation in a structured way. The evaluation is to be done in five dimensions:

- functional - type of problem to be addressed
- operational - type of interaction with the end user
- usability - how comprehensible an explanation is to the end-user
- safety - robustness and security of an approach, echting if there is information leaks and consistent information
- validation - process to evaluate and prove the effectiveness of the explanation

While an explanation should aim to perform well in all dimensions, there is usually a trade-off happening between some of the dimensions. Which dimension is more important is finally an application-specific decision to be made.

In a first step to tackle this research problem, we have already integrated user context into the SENSE Ontology, as described in section 2.8.

5.1 Implementation of User-Centered Explanations in the SENSE stack

As an addition to the data input of topology data, event detection rules and causalities, we have now added user data as well as mitigation plans to the system-specific data input in the form of an excel file.

This input data is added to the Knowledge base according to the user-knowledge and explanation knowledge of the SENSE ontology.

User	UserType	UserRole	AccessRights
OperatorUser1	technical	FacilityOperator	ALL
EVUser1	lay	EVResponsible	seeAllcontrolEV
BatteryUser1	technical	BatteryResponsible	seeAllcontrolBattery

Figure 9: Input of different users in the sense system.

AccessRight	allowsAccessToPlatformOfInterest	control/viewAccessRight
seeAllcontrolEV	Garage1	view
seeAllcontrolEV	EVChargerDelta1	control
seeAllcontrolEV	EVChargerKeba1	control
seeAllcontrolEV	EVChargerKeba2	control
seeAllcontrolEV	EVChargerKeba3	control
seeAllcontrolEV	EVCharger1	control
seeAllcontrolEV	EVCharger2	control
seeAllcontrolEV	EVCharger3	control
seeAllcontrolEV	EVChargerSiCharge1	control
seeAllcontrolEV	Battery1	view
seeAllcontrolEV	Battery2	view
seeAllcontrolEV	BatteryOverview	view
seeAllcontrolEV	PVSystem1	view
seeAllcontrolEV	AllEVChargers1	control
seeAllcontrolBattery	Garage1	view
seeAllcontrolBattery	EVChargerDelta1	view
seeAllcontrolBattery	EVChargerKeba1	view
seeAllcontrolBattery	EVChargerKeba2	view
seeAllcontrolBattery	EVChargerKeba3	view
seeAllcontrolBattery	EVCharger1	view
seeAllcontrolBattery	EVCharger2	view
seeAllcontrolBattery	EVCharger3	view
seeAllcontrolBattery	EVChargerSiCharge1	view
seeAllcontrolBattery	Battery1	control
seeAllcontrolBattery	Battery2	control
seeAllcontrolBattery	BatteryOverview	control
seeAllcontrolBattery	PVSystem1	view
seeAllcontrolBattery	AllEVChargers1	view

Figure 10: definition of access rights in the sense system.

5.1.1 User Input

In Figure 9, an example input to the SENSE system about user data is shown. For each user, one needs to define the user type in terms of explanation preferences, the user role, defining their responsibilities and thus their access rights in the system.

5.1.2 Access Rights Input

In Figure 10, the access rights granted by a specific definition of access rights is shown. For each platform in the use case system, either control or view access rights can be defined. Importantly, if a user has control access rights for a platform, they automatically also have view access rights for this specific platform. Currently, the definition of these access rights is very manual, but this can be improved by an advanced access control system in the future.

5.1.3 Mitigation Plan

In Figure 11, an example input for different mitigation options are shown. For each state type, that is defined in the system, a mitigation plan can be defined. A mitigation plan is a list of options, which can be conducted to reduce, or avoid the state type. Each mitigation option requires control access rights to the platform, where the state occurs to be able to perform this option.

StateType	MitigationPlan	OptionID	MitigationOption
BatteryDip_State	BatteryDip_State_Mitigation	MitigationOption1	call Battery Manufacturer about the incident
BatteryDischargeDifference_State	BatteryDischargeDifference_State_Mitigation	MitigationOption2	
BatteryUnused_State	BatteryUnused_State_Mitigation	MitigationOption3	
BatteryDischarging_State	BatteryDischarging_State_Mitigation	MitigationOption4	
BatteryCharging_State	BatteryCharging_State_Mitigation	MitigationOption5	
BatteryLowSOC_State	BatteryLowSOC_State_Mitigation	MitigationOption6	
BatteryLowSOC_State	BatteryLowSOC_State_Mitigation	MitigationOption7	change battery management
EVIndividualHighCharging_State	EVIndividualHighCharging_State_Mitigation	MitigationOption8	limit charging capacity
EVIndividualChargingDifference_State	EVIndividualHighCharging_State_Mitigation	MitigationOption9	unplug EV
EVIndividualChargingDifference_State	EVIndividualChargingDifference_State_Mitigation	MitigationOption10	change ev charging behaviour/capacity
EVOverallHighCharging_State	EVOverallHighCharging_State_Mitigation	MitigationOption11	
EVOverallChargingDifference_State	EVOverallChargingDifference_State_Mitigation	MitigationOption12	
DemandEnvelopeViolation_State	DemandEnvelopeViolation_State_Mitigation	MitigationOption13	turn of power supply

Figure 11: A list of potential mitigation options for different state types

5.1.4 User-centered and personalised explanations

Based on the input data in Sections 5.1.1-5.1.3, a SPARQL query can be defined to get user-centered explanations with mitigation options based on their control access rights. The SPARQL query to get such explanations is shown in listing 6

Listing 6: SPARQL rule to retrieve personalised and actionable explanations

```
PREFIX sense:  <http://w3id.org/explainability/sense#>
PREFIX s: <http://w3id.org/explainability/sense#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX : <http://example.org/seehub#>

SELECT distinct ?mitigationoptionlabel ?causesensor ?cause ?cset ?
   ceet ?relation ?effectsensor ?effect ?set ?eet
WHERE {
 ?cause sense:causallyRelated ?effect .
 ?effect (sense:causallyRelated)* :
    DemandEnvelopeViolation_State_90bd7500-7827-4706-893b-
    eab29062732b .
 <<?cause sense:causallyRelated ?effect>> sense:hasCausalSource ?
    stc .
 ?stc sense:causalRelation ?relation .
 ?causeobservation sosa:hasResult ?cause .
 ?causesensor sosa:madeObservation ?causeobservation .
 ?effectobservation sosa:hasResult ?effect .
 ?effectsensor sosa:madeObservation ?effectobservation .
 ?effect sense:hasStartEvent ?se .
 ?effect sense:hasEndEvent ?ee .
 ?seo sosa:hasResult ?se .
 ?seo sosa:phenomenonTime ?set .
 ?eeo sosa:hasResult ?ee .
 ?eeo sosa:phenomenonTime ?eet .
 ?cause sense:hasStartEvent ?cse .
 ?cause sense:hasEndEvent ?cee .
 ?cseo sosa:hasResult ?cse .
 ?cseo sosa:phenomenonTime ?cset .
 ?ceeo sosa:hasResult ?cee .
 ?ceeo sosa:phenomenonTime ?ceet .

    # filter on user view access rights
    ?causeplatform sosa:hosts ?causesensor .
```

```
32      ?user rdfs:label "EVUser1" .
33      ?user sense:hasUserRole ?userrole .
34      ?userrole sense:containsAccessRights ?accessright .
35      ?accessright sense:allowsViewAccessTo ?causeplatform .
36      OPTIONAL {
37       ?accessright sense:allowsControlAccessTo ?causeplatform .
38          ?cause sense:hasStateType ?causeType .
39       ?causeType sense:hasMitigationPlan ?mitigationplan .
40       ?mitigationplan sense:containsMitigationOption ?
            mitigationoption .
41       ?mitigationoption rdfs:label ?mitigationoptionlabel .
42          }
43  }
```

References

[1] J. Runge, “Detecting and quantifying causality from time series of complex systems,” 2014.

[2] K. SCHREIBERHUBER, F. J. EKAPUTRA, M. SABOU, D. HAUER, K. DIWOLD, T. FRÜHWIRTH, G. STEINDL, and T. SCHWARZINGER, “Towards a state explanation framework in cyber-physical systems,” in *DACH+ Energy Informatics 2024, Lugano, Switzerland, Proceedings*. ACM SIG Energy, 2024.

[3] 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*, no. Preprint, pp. 1–29, 2024.

[4] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, “Sosa: A lightweight ontology for sensors, observations, samples, and actuators,” *Journal of Web Semantics*, vol. 56, pp. 1–10, 2019.

[5] L. Daniele, F. den Hartog, and J. Roes, “Created in close interaction with the industry: the smart appliances reference (saref) ontology,” in *Formal Ontologies Meet Industry: 7th International Workshop, FOMI 2015, Berlin, Germany, August 5, 2015, Proceedings 7*. Springer, 2015, pp. 100–112.

[6] L. Tailhardat, Y. Chabot, and R. Troncy, “Noria-o: an ontology for anomaly detection and incident management in ict systems,” in *European Semantic Web Conference*. Springer, 2024, pp. 21–39.

[7] A. Zhou, D. Yu, and W. Zhang, “A research on intelligent fault diagnosis of wind turbines based on ontology and fmeca,” *Advanced Engineering Informatics*, vol. 29, no. 1, pp. 115–125, 2015.

[8] W. R. Van Hage, V. Malaisé, R. Segers, L. Hollink, and G. Schreiber, “Design and use of the simple event model (sem),” *Journal of Web Semantics*, vol. 9, no. 2, pp. 128–136, 2011.

[9] Y. Rebboud, P. Lisena, and R. Troncy, “Beyond causality: Representing event relations in knowledge graphs,” in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2022, pp. 121–135.

[10] S. Chari, O. Seneviratne, D. M. Gruen, M. A. Foreman, A. K. Das, and D. L. McGuinness, “Explanation ontology: a model of explanations for user-centered ai,” in *International Semantic Web Conference*. Springer, 2020, pp. 228–243.

[11] U. Jaimini, C. Henson, and A. Sheth, “An ontology design pattern for representing causality,” 2023.

[12] M. Poveda-Villalón, A. Fernández-Izquierdo, M. Fernández-López, and R. García-Castro, “LOT: An industrial oriented ontology engineering framework,” *Engineering Applications of Artificial Intelligence*, vol. 111, p. 104755, May 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197622000525

[13] D. Garijo, “Widoco: a wizard for documenting ontologies,” in *The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16*. Springer, 2017, pp. 94–102.

[14] S. Chávez-Feria, R. García-Castro, and M. Poveda-Villalón, "Chowlk: from uml-based ontology conceptualizations to owl," in *European Semantic Web Conference*. Springer, 2022, pp. 338–352.

[15] M. Poveda-Villalón, A. Gómez-Pérez, and M. C. Suárez-Figueroa, "Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 10, no. 2, pp. 7–34, 2014.

[16] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Springer, pp. 152–166.

[17] T. Schwarzinger, G. Steindl, T. Frühwirth, and K. Diwold, "Semi-automated event specification for knowledge-based event detection," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, ISSN: 1946-0759. [Online]. Available: https://ieeexplore.ieee.org/document/10710936

[18] J. Y. Halpern, *Actual causality*. MiT Press, 2016.

[19] N. Mostafazadeh, A. Grealish, N. Chambers, J. Allen, and L. Vanderwende, "CaTeRS: Causal and Temporal Relation Scheme for Semantic Annotation of Event Structures," in *Proceedings of the Fourth Workshop on Events*. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 51–61. [Online]. Available: https://aclanthology.org/W16-1007

[20] J. W. Tukey *et al.*, *Exploratory data analysis*. Reading, MA, 1977, vol. 2.

[21] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science advances*, vol. 5, no. 11, p. eaau4996, 2019.

[22] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[23] A. K. Seth, A. B. Barrett, and L. Barnett, "Granger causality analysis in neuroscience and neuroimaging," *Journal of Neuroscience*, vol. 35, no. 8, pp. 3293–3297, 2015.

[24] G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems," *science*, vol. 338, no. 6106, pp. 496–500, 2012.

[25] E. E. Leamer, "Vector autoregressions for causal inference?" in *Carnegie-rochester conference series on Public Policy*, vol. 22. North-Holland, 1985, pp. 255–304.

[26] L. Barnett, A. B. Barrett, and A. K. Seth, "Granger causality and transfer entropy are equivalent for gaussian variables," *Physical review letters*, vol. 103, no. 23, p. 238701, 2009.

[27] X. Mao and P. Shang, "Transfer entropy between multivariate time series," *Communications in Nonlinear Science and Numerical Simulation*, vol. 47, pp. 338–347, 2017.

[28] M. Rubinov and O. Sporns, "Complex network measures of brain connectivity: uses and interpretations," *Neuroimage*, vol. 52, no. 3, pp. 1059–1069, 2010.

[29] J. Nichols, M. Seaver, S. Trickey, M. Todd, C. Olson, and L. Overbey, "Detecting nonlinearity in structural systems using the transfer entropy," *Physical Review E*, vol. 72, no. 4, p. 046217, 2005.

[30] B. Lindner, L. Auret, M. Bauer, and J. W. Groenewald, "Comparative analysis of granger causality and transfer entropy to present a decision flow for the application of oscillation diagnosis," *Journal of Process Control*, vol. 79, pp. 72–84, 2019.

[31] A. Tank, I. Covert, N. Foti, A. Shojaie, and E. B. Fox, "Neural granger causality," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4267–4279, 2021.

[32] M. Nauta, D. Bucur, and C. Seifert, "Causal discovery with attention-based convolutional neural networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, p. 19, 2019.

[33] R. Rossi, A. Murari, L. Martellucci, and P. Gaudio, "Netcausality: A time-delayed neural network tool for causality detection and analysis," *SoftwareX*, vol. 15, p. 100773, 2021.

[34] A. R. Coenen, S. K. Hu, E. Luo, D. Muratore, and J. S. Weitz, "A primer for microbiome time-series analysis," *Frontiers in genetics*, vol. 11, p. 310, 2020.

[35] J. T. Lizier, M. Prokopenko, and A. Y. Zomaya, "Local information transfer as a spatiotemporal filter for complex systems," *Physical Review E*, vol. 77, no. 2, p. 026110, 2008.

[36] M. Lindner, R. Vicente, V. Priesemann, and M. Wibral, "Trentool: A matlab open source toolbox to analyse information flow in time series data with transfer entropy," *BMC neuroscience*, vol. 12, pp. 1–22, 2011.

[37] J. Pearl, "The seven tools of causal inference, with reflections on machine learning," *Communications of the ACM*, vol. 62, no. 3, pp. 54–60, Feb. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3241036

[38] S. Chari, D. M. Gruen, O. Seneviratne, and D. L. McGuinness, "Directions for Explainable Knowledge-Enabled Systems," Mar. 2020, arXiv:2003.07523 [cs]. [Online]. Available: http://arxiv.org/abs/2003.07523

[39] S. Chari, O. Seneviratne, D. M. Gruen, M. A. Foreman, A. K. Das, and D. L. McGuinness, "Explanation Ontology: A Model of Explanations for User-Centered AI," Oct. 2020, arXiv:2010.01479 [cs]. [Online]. Available: http://arxiv.org/abs/2010.01479

[40] Roberto Confalonieri and Giancarlo Guizzardi, "On the Multiple Roles of Ontologies in Explainable AI," *Neurosymbolic Artificial Intelligence Journal*, 2023. [Online]. Available: https://www.neurosymbolic-ai-journal.com/paper/multiple-roles-ontologies-explainable-ai

[41] N. Tintarev and J. Masthoff, "Effective explanations of recommendations: user-centered design," in *Proceedings of the 2007 ACM conference on Recommender systems*, 2007, pp. 153–156.

[42] M. Sadeghia, L. Herbolda, M. Unterbuscha, and A. Vogelsanga, "SmartEx: A Framework for Generating User-Centric Explanations in Smart Environments."

[43] K. Sokol and P. Flach, "Explainability fact sheets: A framework for systematic assessment of explainable approaches," in *Proceedings of the 2020 conference on fairness, accountability, and transparency*, 2020, pp. 56–67.