SENSE

SENSE

SENSE: Semantic-based Explanation of Cyber-physical Systems

Deliverable 3.2:
Semantic Data Integration Methods

Authors	:	Mohammad Bilal, Thomas Frühwirth
Dissemination Level	:	Public
Due date of deliverable	:	30.06.2024
Actual submission date	:	Oct. 2024
Work Package	:	WP3.2
Type	:	Report
Version	:	1.0

Abstract

Deliverable 3.2 first presents a comprehensive methodology for extracting tacit knowledge from domain experts, called the Tacit Knowledge Tree (TKT) methodology. This is important, as tacit knowledge can be a valuable resource but is generally hard to articulate and often not explicitly documented. However, tacit knowledge is a primary source for generating explanations of events within Cyber-Physical Systems (CPSs). The TKT methodology results in a semantic description of the system, which has yet to be linked to the corresponding data streams observed by sensors within the system. Thus, the deliverable next focuses on integrating semantic and time-series data based on concepts provided by the Sensor, Observation, Sample, and Actuator (SOSA) ontology. It results in a concept for the data ingestion module, which reads sensor values primarily from a connected time-series database, enriches the data with SOSA concepts, and provides these semantically enriched sensor readings to the remaining SENSE modules.

History

Version	Date	Reason	Revised by
1.0	20.09.2024	Initial draft	TF

Author List

Project Partner	Name (Initial)	Contact Information
AEE INTEC	Dagmar Jähnig (DJ)	d.jaehnig@aee.at
AEE INTEC	Christoph Moser (CM)	c.moser@aee.at
TU	Thomas Frühwirth (TF)	thomas.fruehwirth@tuwien.ac.at
TU	Mohammad Bilal (MB)	Mohammad.bilal@tuwien.ac.at
SIE	Juliana Kainz (JK)	juliana.kainz@siemens.com
SIE	Daniel Hauer (DH)	daniel.hauer@siemens.com
SIE	Konrad Diwold (KD)	konrad.diwold@siemens.com
SIE	Alfred Einfalt (AE)	alfred.einfalt@siemens.com
SIE	Gerhard Engelbrecht (GE)	gerhard.engelbrecht@siemens.com
SIE	Simon Steyskal (SS)	simon.steyskal@siemens.com
WU	Katrin Schreiberhuber (KS)	katrin.schreiberhuber@wu.ac.at
WU	Marta Sabou (MS)	marta.sabou@wu.ac.at

Executive Summary

The SENSE project aims to explain events occurring in technical systems from the area of Smart Grids and Smart Buildings. The goal is to contribute to Austria's sustainability goals by making complex systems that underlie key (and often highly polluting) infrastructures more efficient and user-friendly through explanations of (anomalous) events occurring in those systems. The SENSE system to be developed in this project aims to make complex Cyber-Physical Systems (CPSs) more transparent and thereby improve the performance and user acceptance of such systems.

In the rapidly advancing landscape of modern technology, the seamless integration of time-series data with domain knowledge has become essential for creating sophisticated systems such as Digital Twins (DTs), which provide the underlying methodological framework for SENSE. DTs are virtual representations of physical entities, which require a comprehensive framework for semantic data integration and storage to mirror real-world scenarios accurately.

The document outlines a methodology for formalizing the necessary tacit knowledge based on structured interviews between knowledge engineers and domain experts, called the Tacit Knowledge Tree (TKT) methodology. The methodology is a top-down approach structured into four stages: Stage 1: identify objects, Stage 2: identify states, Stage 3: identify causes, and Stage 4: identify impacts. Thereby, Stages 2 and 3 build upon a fishbone diagram to illustrate the dependencies of objects, states and causes with the main problem to solve. Stage 4 utilizes a cause-and-effect table to rate the impact of each cause.

After identifying the necessary knowledge using the TKT methodology, the deliverable investigates the semantic interlinking of different storage types in a hybrid data storage system for Digital Twins (DTs). These storage types are relational databases, graph-based databases, semantic-based databases, and time-series databases. Each of these storage types has advantages and drawbacks. The deliverable then focuses on integrating semantic information with time-series data streams, the primary requirement for implementing the SENSE use cases. It builds upon concepts of the Sensor, Observation, Sample, and Actuator (SOSA) ontology as this was the part of the SENSE ontology which focuses on topological knowledge, a widely adopted ontology in the context of CPSs. Specifically, sensor readings, i.e., time-series data streams, are annotated with SOSA concepts and encoded in RDF before being made available to other SENSE components, such as the event detection module.

Table of Content

List of Figures & Tables

1 Introduction

1.1 Purpose and Scope of the Document

The SENSE architecture, described in Deliverable D3.1 [5], provides the infrastructure for explainability and integration of time-series data. This integrated system can be leveraged in tasks such as chatbot development for semantic explanations and ranking events and causes.

Overall, integrating time-series data with domain knowledge, enriched through explicit and tacit knowledge, ensures robust, flexible, and efficient data integration and storage solutions. This approach supports the sophisticated needs of modern Digital Twins (DTs), enhancing their accuracy and relevance by leveraging domain expertise and advanced semantic techniques.

This document presents the results of WP3, Task 3.2 of the SENSE project: Semantic Data Integration & Storage (cf. Figure 1). The semantic integration of data is essential in future tasks such as Task 3.3, where event detection algorithms can use this integrated data to produce better results while detecting events. Task 3.2 is critical for WP4 as the semantic data integration process will help provide event explainability and help develop the SENSE semantic model in Task 4.1. The data is also crucial for Task 4.2, where causality knowledge acquisition can be more efficient, with semantic data providing a structure for acquiring new knowledge.

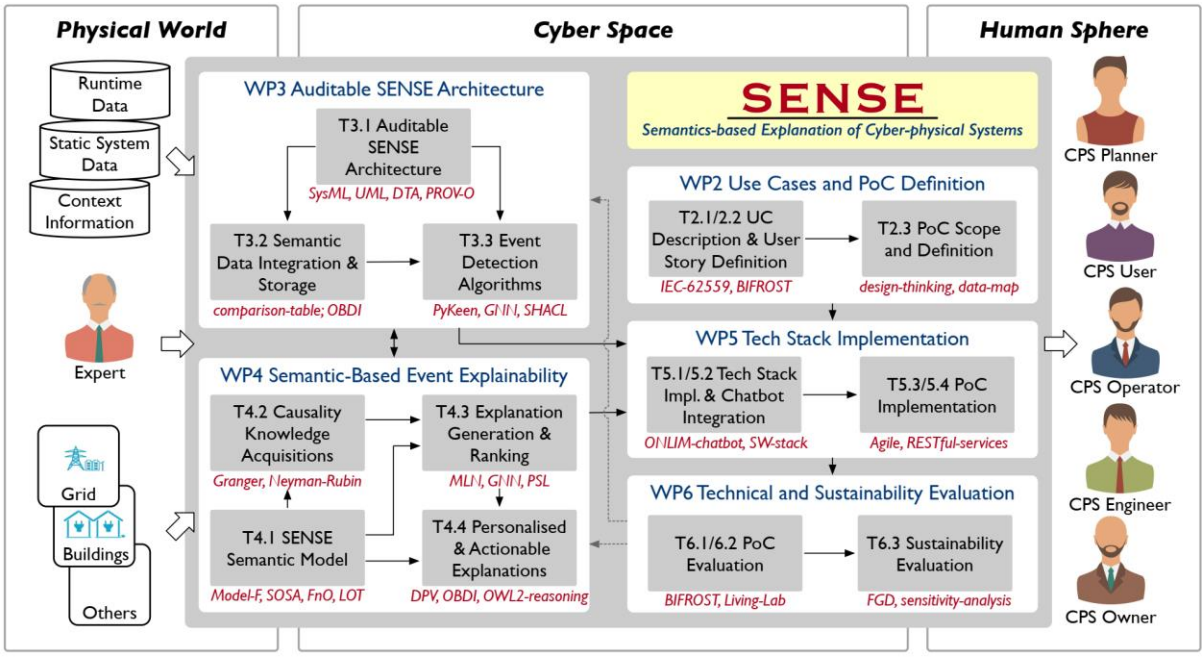

Figure 1 – SENSE Conceptual Components, Their Connections, and Relevant WPs

1.2 Structure of the Document

The document is structured into four main sections. Section 1 serves as an introduction, outlining the document's purpose, scope, and structure. Section 2 delves into the methodology for extracting tacit knowledge. Section 3 is dedicated to discussing various techniques and methodologies for time-series and semantic data integration paradigms. Finally, the document concludes with a summary in Section 4.

2 Methodology for Extracting Tacit Knowledge

From a Semantic Web perspective, ontology enables the user to describe and share reusable domain-specific knowledge [1]. Engaging with domain experts to extract tacit knowledge is essential in the ontology creation process [2]. Tacit knowledge, acquired through experience, intuition, and implicit learning, can be a valuable resource but is generally hard to articulate and often not explicitly documented. To make this knowledge available in technical systems, it must be formalized to the largest extent possible, requiring a sophisticated methodology.

For this reason, we introduce the Tacit Knowledge Tree (TKT) as a methodology aiming to address the issue of knowledge loss within the scope of ontology development. The proposed solution involves engaging with domain experts to extract tacit knowledge, formalizing this knowledge in an ontology, and linking the ontology concepts with the corresponding time-series data points. This integration presents a significant challenge in the rapidly evolving field of data science, necessitating a nuanced approach that leverages explicit and tacit knowledge to enhance data usability and semantic richness. This section outlines the TKT methodology for achieving this integration, emphasizing the role of domain experts and advanced semantic techniques.

2.1 Overview of the TKT Methodology

Extracting tacit knowledge following the TKT methodology is a top-down approach, the TKT methodology is a novel methodology which targets the extraction of tacit domain knowledge from domain experts. The stages, as well as the corresponding techniques/tools applied in each stage, are illustrated in Figure 2. It starts with the problem the resulting ontology shall describe and solve.

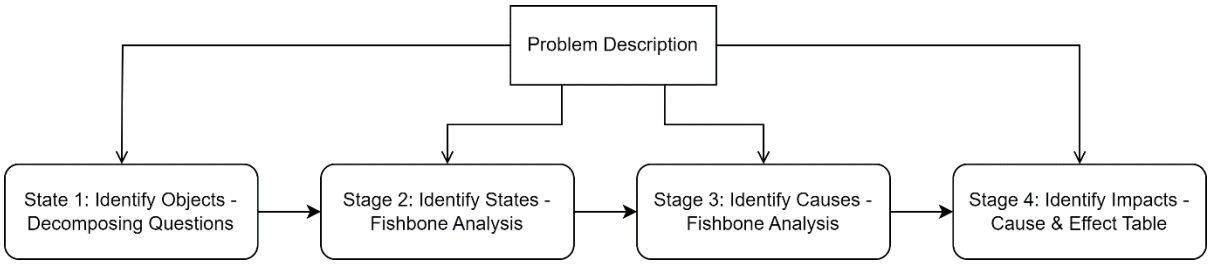

Figure 2 – 4 Stage Process Overview

The main influences are identified at each stage, and this knowledge is then detailed in the next stage. The order of knowledge extraction is firstly to formulate the problem and then work top down to identify objects. However, multiple iterations may be required. The process builds upon semi-structured interviews that are structured into four main stages. The primary purpose and tools for documenting each stage are briefly explained in the following and further refined in the next section.

- **Stage 1: Identify Objects:** Through decomposing questions, this stage identifies the various objects that play a major role in defining the problem.

- **Stage 2: Identify State:** Using a fishbone diagram for Root Cause Analysis (RCA), this stage explores the objects identified to extract the states of those objects.
- **Stage 3: Identify Cause:** By detailing the fishbone diagram, this stage extracts the different causes that can change the states of the objects.
- **Stage 4: Identify Impact:** Using a cause-and-effect table, this stage identifies the impact of each cause on the problem and rates them using a multi-value metric considering severity, occurrence, and probability.

2.2 Detailed Discussion of the TKT Methodology Stages and Steps

Each state briefly introduced in the previous section consists of several steps that need to be taken by either a domain expert or a knowledge engineer. These steps and the person primarily responsible for these are depicted in Figure 3. First, we clarify the role of the domain expert and the knowledge engineer. Afterward, we discuss each of the steps in more detail.

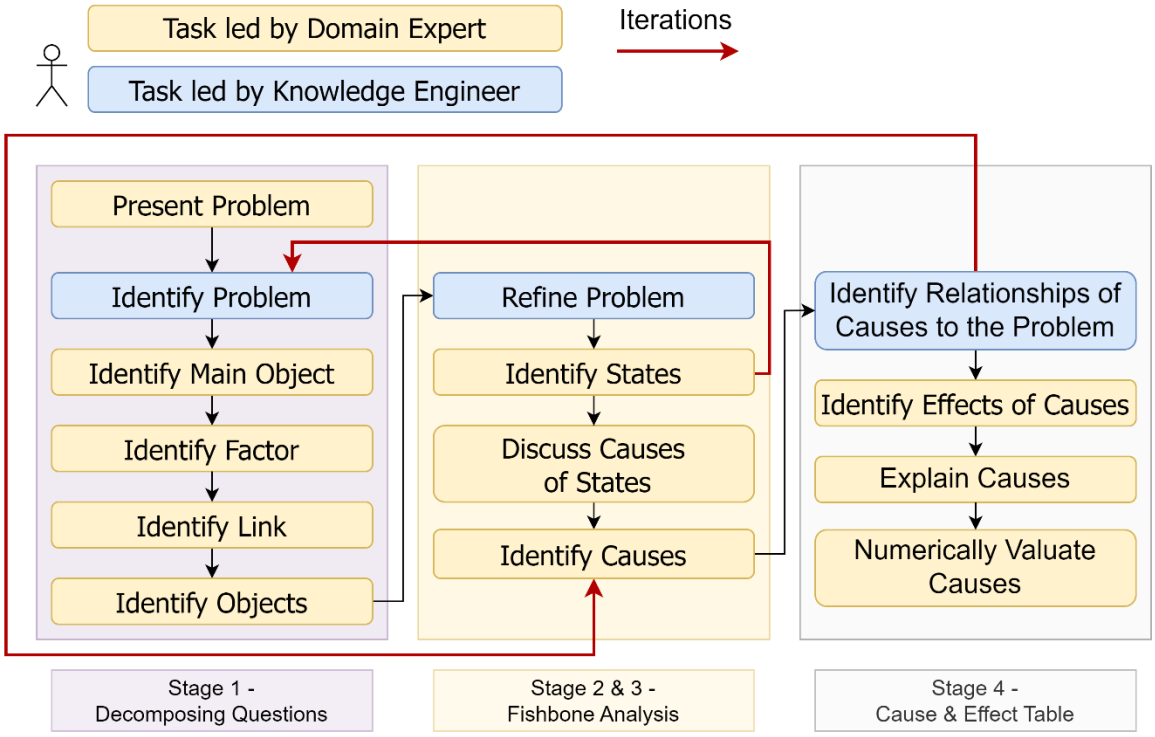

Figure 3 – Knowledge Management Diagram

2.2.1 Involved Roles: Domain Expert and Knowledge Engineer

Domain Expert: A specialist with extensive, often implicit, knowledge of a particular domain. While they understand the nuances and intricacies of the problem, they may not have the expertise to translate this into a structured, machine-readable format. The domain expert plays a significant role in identifying the main object, the main factor, the link of the objects, and other objects influencing the main factor. Which for example in this case would be the HVAC unit if high energy consumption is the problem we are targeting.

Knowledge Engineer: A professional who works with the domain expert to capture and formalize their knowledge into a structured ontology. This enables the knowledge to be used

in computational models, such as AI systems, ensuring scalability, consistency, and reusability. In the context of this document, specifically, the knowledge engineer is responsible for capturing, organizing, and representing the knowledge of domain experts with appropriate ontology concepts.

2.2.2 Stage 1: Identify Objects

This stage aims to decompose the Cyber-Physical System (CPS) into relevant objects. The order of knowledge extraction is firstly to formulate the problem and then work top down to identify objects and then filter this knowledge down to the next stage. This is now working towards developing a causality chain.

Present Problem: In this step, the domain expert describes the issue to the knowledge engineer. This involves a collaborative discussion where the expert outlines the overarching problem that they want to target, for example high energy consumption— often based on their experience, observations, and practical knowledge. The expert might reference specific incidents, challenges, or inefficiencies they've encountered (e.g., high power consumption in a system).

Identify Problem: After the problem is initially described, the knowledge engineer works to refine the problem statement. This involves asking clarifying questions to ensure that the issue is fully understood and going beyond surface-level descriptions to uncover the root causes or underlying factors that might be contributing to the problem. So initially the problem could be abnormal energy consumption, but after the first iteration the problem is refined to high energy consumption.

Identify Main Object: Once the problem is clearly defined, the domain expert and knowledge engineer collaborate to pinpoint the primary object or system component central to the problem. This object is the focal point around which the subsequent steps are structured. For example, if the problem is high power consumption, the main object might be a specific device consuming the most power which could be HVAC unit.

Identify Factor: Once the main object (or platform if the object is a more complex system) is identified, the domain expert and the knowledge engineer must determine what affects the main object. The factor directly impacts the main object, so the word factor should be understood as the primary catalyst impacting the main object. An example would be the amount of solar radiation that affects the power generated by a solar panel. Therefore, solar radiation is the main factor in this case., or room temperature

Identify Link: Now that the factor has been identified, the link follows this. This is where the domain expert will go deeper into the causality chain analysis and look at what sensors impact the main factor and, thus, should be linked to the main factor. In this step, the discovery of other objects is likely.

Identify Objects: This is the step where the causality chain of the main object, the main factor, and the link all point toward other objects that can be identified as impacting the problem. The domain expert will detail what other objects affect the problem through the causality

chain. For example, other objects like Window, Door, Sensors could also play a part in influencing the energy consumption not just the HVAC unit.

2.2.3 Stage 2 &3: Identify States & Causes

Refine Problem: At this step, the problem is revisited, and rethinking is performed to ensure that all objects other than the main object are also considered. This is where the wording of the problem is essential, and the knowledge engineer plays a part in ensuring that the problem is refined, the objects identified are scalable so multiple instantiations of the objects can be modeled and covers the project the methodology is targeting to extract tacit knowledge from in this case SENSE.

Identify States: At this step, the focus shifts to analyzing the object identified in Stage 1. The domain expert provides insight into the various states that this object can take. For example, a sensor might have active, idle, or off states. The goal is to categorize the object's different operational modes or conditions, in the case of a door the state would be open and close.

Discuss Causes of States: This step involves using a fishbone diagram, which is a technique used in root cause analysis, especially for discussion with domain experts to systematically categorize these states [3]. The fishbone structure helps to visually break down the object into possible states based on various influencing factors, ensuring all relevant states are captured for further analysis, for example the cause of a state open for the object door could be room at capacity.

Identify Causes: This step builds on the fishbone diagram and identifies the cause of each state. Here, the knowledge engineer and domain expert work together to understand why the object behaves in a certain way under different conditions. For example, what causes the sensor to transition from idle to active? Is it a temperature change, system demand, or external signal?

This analysis helps establish a causality chain, linking states of the object to their underlying causes. Each cause might be associated with multiple objects, and thus, the system's scalability can be ensured. The output of this step becomes the backbone for formalizing relationships between objects, states, and causes in the domain ontology.

2.2.4 Stage 4: Identify Impacts

Identify Relationships of Causes to Problem: In this step, the fishbone diagram is used to extract and identify how the causes are related to the problem. As each cause is identified, it acts as an input to each row of the cause-and-effect table.

Identify Effects of Causes: This part establishes the impact of each cause on the problem, which is classified as high, low, or both. The domain expert will play a more significant role in this as the causality chain and the impact would be known to the domain expert so they can identify the effect.

Explain Causes: At this step, the domain expert explains each cause. The knowledge engineer's job is to ensure that if the explanations are similar, they are re-worded to be

reusable. This can provide semantic structure and scalability to domain experts and knowledge engineers so the extracted knowledge can be reused. These explanations will later be used to give the facility manager a possible reason for the problem.

Numerically Valuate Causes: Once the causes of the objects' states are understood, the next step is to assess the impact of these causes on the problem. Based on their experience, the domain expert rates each cause's significance. For example, how much does a prolonged "active" state of the sensor contribute to the overall problem of high-power consumption?

This impact analysis helps prioritize the causes based on their influence on the problem, enabling targeted interventions. The knowledge engineer documents these ratings and ensures they are captured formally and scalable.

2.2.5 Iterations

Figure 2 is a proposed knowledge management diagram for extracting tacit knowledge, which can interlink with time-series data by engaging with domain experts. This iterative process helps expand and fine-tune the domain ontology based on expert input. The idea of having iterations is to ensure that knowledge loss is at a minimum and that the output is in a machine-readable format. The iterative workflow includes ingesting and normalizing sensor data, semantically annotating it with ontology terms, and continuously validating the alignment with real-world practices.

2.3 Smart Building Example

To evaluate the TKT methodology, we applied it to the Smart Building use case [4]. We selected a smart meeting room with sensors throughout the room, providing objects to form causality chains and sensor values for event detection of abnormal energy consumption.

2.3.1 Stage 1: Identify Objects

The first step is to formulate the problem and identify the objects within the system which affect this problem. For our demonstration, we use the example of abnormal energy consumption. The formulated problem is added in the first line of Table 1.

Table 1 – Steps and Outcomes – Stage 1

Step	**Result**
Identify Problem	Energy consumption is abnormal (too high)
Identify Main Object	Heating, Ventilation and Air Conditioning (HVAC) unit
Identify Factors	Change of room temperature
Identify Links	Domain expert acknowledges other objects affecting the factor
Identify Objects	Weather, window, door, room sensor

Once the problem is formulated, follow-up questions are asked to extract further tacit knowledge from the domain expert's perspective on why the problem occurred.

The domain expert thinks that HVAC workload is the main reason for high energy consumption, as documented in line 2 of Table 1. Thus, the main object is identified. One explanatory factor that influences the energy consumption of the HVAC unit is a change in room temperature, which is documented in line 3 of the table. The next part is essential as this is seeking a causality link between the factor and other objects. The domain expert will need to identify what objects affect the temperature. The decomposing process reveals objects which affect the temperature of the room. These objects are documented in line 5 of the table.

2.3.2 Stage 2&3: Identify States & Causes

In this step, a fishbone analysis is conducted that uses the identified objects from Stage 1 as categories to extract further knowledge and identify the states of every object. So, the categories in this fishbone diagram are door, window, weather, HVAC unit, and room sensor, as identified in Stage 1. The number of states depends on the object. For example, windows and doors can only be in states open and closed, whereas an HVAC unit can be in multiple states. The thinking behind identifying only the state in this section is that, firstly, we need to organize this so it can be scalable semantically, so the semantics that are extracted and modeled can be reused in multiple instances. The result of Stage 2 is illustrated in Figure 4. Additionally, iteration is implemented to ensure minimal knowledge loss occurs, and an opportunity is provided to both the domain expert and the knowledge engineer to ensure all the objects involved are documented.

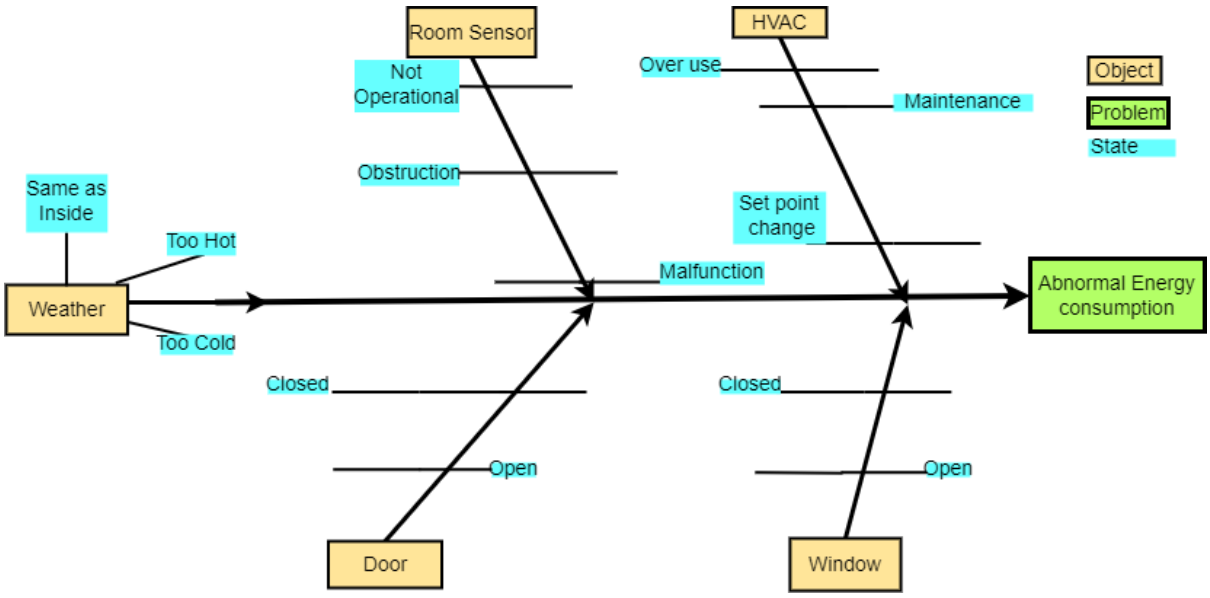

Figure 4 – Fishbone Diagram – Stage 2

Stage 3 extends the fishbone analysis depicted in Figure 4 by identifying the cause that will bring each object to the identified state. For example, the object window may be in the state open due to the cause of "bad air quality". The number of causes for each state is not limited and is directly related to the use case. For example, a door being open and closed can have many causes. While the room being at capacity has no direct impact on the problem of abnormal energy consumption, bad air quality can lead to states of objects changing, which

impacts power consumption. It is, therefore, added to the fishbone diagram at the appropriate location. Figure 5 depicts the fishbone diagram resulting from Stage 3 of the TKT methodology applied to the Smart Building use case.

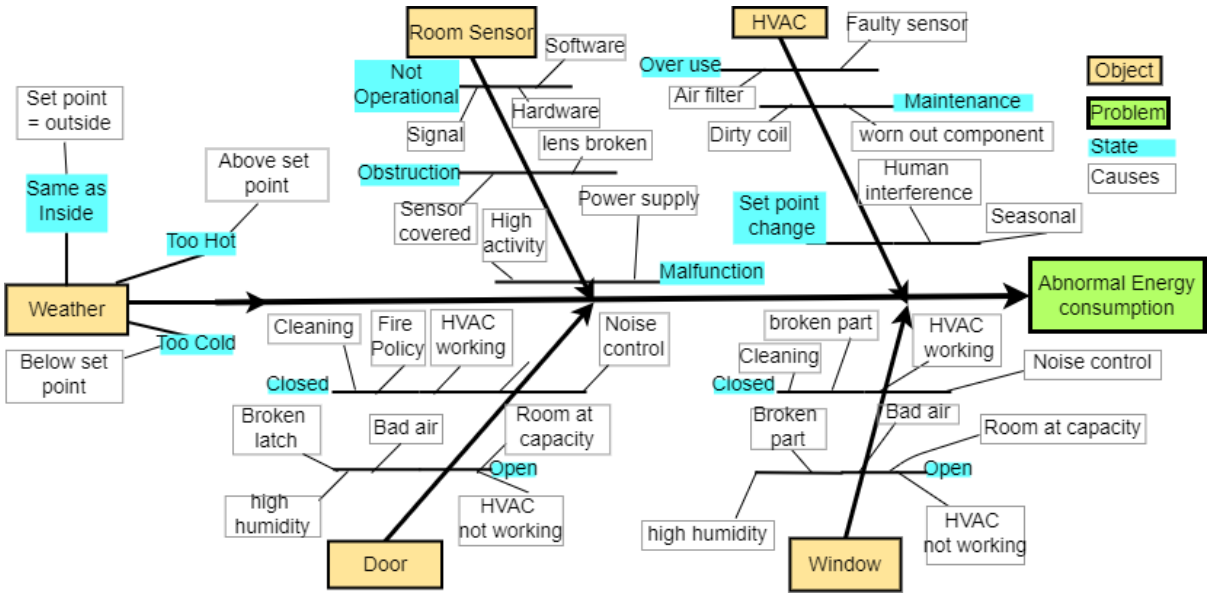

Figure 5 – Fishbone Diagram – Stage 3

2.3.3 Stage 4: Identify Impacts

As a preparation for this stage, the objects, states, and causes are transferred from the fishbone diagram in the previous stage to a cause-and-effect table. The layout of the table is illustrated in Table 2. The domain expert then rates the impact of each cause on the abnormal energy consumption problem. The columns represent various aspects of tacit knowledge captured during the TKT methodology. All the numbers of Severity (S), Occurrence (O), and Detection (D) are populated using a 0-5 scale, where 5 is the most impactful. The Risk Priority Number (RPN) is then calculated as the product of S*O*D. In the final system, these RPNs can be used to rank explanations if there are multiple possibilities.

Table 2 – Cause-and-Effect Table – Stage 4

Object	State	Cause	Effect	Explanation (HVAC unit power consumption)	Explanation HVAC State	S	O	D	RPN
Door	Open	Room at capacity	High	HVAC / Cooling unit working harder	Working harder	3	5	4	60
Window	Open	Room at capacity	High	HVAC / Cooling unit working harder	Working harder	3	5	3	45
Door	Closed	Noise control	Both	HVAC / Cooling unit working consistently	As expected	1	5	5	25
Door	Open	Broken Lock	High	HVAC / Cooling unit working harder	Working harder	5	1	5	25
HVAC	Set point change	Seasonal	Both	HVAC / Cooling unit working consistently	As expected	1	5	5	25
Room Sensor	Malfunction	Power Supply	High	HVAC unit working harder	Working harder	5	1	5	25
HVAC	Set point Change	Human interference	Both	HVAC / Cooling unit working harder	Working harder	3	2	4	24
Window	Open	Broken Lock	High	HVAC / Cooling unit working harder	Working harder	5	1	4	20
HVAC	Overuse	Faulty Sensor	High	HVAC unit working harder	Working harder	5	1	3	15

Door	Open	Bad Air	High	HVAC / Cooling unit working harder	Working harder	3	4	1	12
Window	Open	Bad Air	High	HVAC / Cooling unit working harder	Working harder	3	4	1	12
Room Sensor	Not Operational	Signal	High	HVAC unit working harder	Working harder	4	1	3	12
Room Sensor	Not Operational	Software	High	HVAC unit working harder	Working harder	4	1	3	12
Room Sensor	Obstruction	Sensor Covered	High	HVAC unit working harder	Working harder	3	2	2	12
Room Sensor	Obstruction	High Activity	High	HVAC unit working harder	Working harder	2	3	2	12
Weather	Too Cold	Below setpoint	High	HVAC / Cooling unit working harder	Working harder	1	2	5	10
Weather	Too Hot	Above setpoint	High	HVAC unit working consistently	Working harder	1	2	5	10
HVAC	Overuse	Air filter	High	HVAC unit working harder	Working harder	5	1	2	10
Room Sensor	Obstruction	Lens Broken	High	HVAC unit working harder	Working harder	5	1	2	10
Door	Open	high humidity	High	HVAC / Cooling unit working harder	Working harder	3	3	1	9
Window	Open	high humidity	High	HVAC / Cooling unit working harder	Working harder	3	3	1	9
Room Sensor	Not Operational	Hardware	High	HVAC unit working harder	Working harder	4	1	2	8

3 Hybrid Data Storage for Digital Twins

In the rapidly evolving technological landscape, DTs have emerged as a critical innovation, enabling the creation of virtual replicas of physical systems. These replicas simulate real-world behavior, facilitating monitoring, diagnostics, and predictive maintenance. One of the primary challenges in implementing DTs is managing the complex and diverse types of data they generate and consume, including real-time sensor data, domain knowledge from ontologies, and causal relationships. To address these challenges, a hybrid data storage approach is essential, leveraging the strengths of multiple storage paradigms.

This hybrid approach offers several distinct benefits. First, it provides flexibility by accommodating a wide variety of data types, from structured metadata to unstructured sensor data and ontological knowledge. Each database component is optimized for specific data types, ensuring efficient handling and processing across the system. Secondly, the architecture is highly scalable, with time-series databases managing the real-time demands of sensor data while graph and semantic databases handle complex relationships and reasoning. This scalability is essential for DTs, which often evolve and require a data architecture that can grow alongside them.

Furthermore, the hybrid system delivers deeper insights and more actionable intelligence by combining time-series data with semantic reasoning and causal analysis. This integration enables better decision-making, predictive maintenance, and overall system optimization, enhancing the value of the DT.

This section provides an overview of existing data storage paradigms focusing on relational database in section 3.1.1, graph-based in section 3.1.2, semantic database in in section 3.1.3, and time-series databases in section 3.1.4. Next, we narrow the solution to combining semantic and time-series databases in section 3.2, as these are required within the SENSE

architecture. We discuss general concepts for this integration and present existing approaches in section 3.2. Finally, we present a solution for hybrid data storage for SENSE by linking time-series data streams to Sensor, Observation, Sample, and Actuator SOSA ontology concepts in section 3.3.

3.1 Types of Databases in Hybrid Data Storage Systems for Digital Twins

The first database type is relational, which is crucial in managing structured metadata about the DT's assets, sensors, and historical logs. Relational databases, such as PostgreSQL, ensure data consistency and enable complex querying for structured data, including well-defined information about the DT's physical components, sensor configurations, and recorded events. This reliable storage and querying of static and historical data provide a stable foundation for the DT's operations. Despite challenges, relational databases remain essential in DT architecture, especially for storing structured metadata and transactional data. They offer consistency and efficiency, making them an integral part of hybrid data storage systems designed to support the functionality and precision of DTs.

In addition to the relational component, a graph database such as Neo4j[1] is incorporated to handle the complex relationships between the components and systems within the DT. Graph databases excel at modeling interconnected entities, making them ideal for representing the dependencies, causal relationships, and interactions between various elements of the DT. This relational mapping enables detailed system analysis, dependency tracking, and the tracing of causal chains. In the context of DTs, graph databases are particularly beneficial for managing ontologies and modeling complex interactions. They are a vital component of hybrid storage solutions for understanding and analyzing the intricate web of connections within modern DTs.

The hybrid system also includes a semantic database, such as Apache Jena[2], to manage ontologies and enable reasoning over sensor data, enriching raw sensor data with contextual and domain-specific knowledge for advanced analytics and inference. By integrating ontological reasoning, the system can infer events, detect anomalies, and provide explanations based on predefined relationships, adding an additional layer of intelligence to the DT for predictive and diagnostic analytics. Semantic databases, well-suited for managing domain ontologies and reasoning over time-series data, enhance DT's ability to make context-aware decisions, supporting the sophisticated analytical capabilities required for more intelligent and context-aware DT implementations.

The final piece of the hybrid architecture is the time-series database, such as InfluxDB or TimescaleDB, essential for managing the continuous real-time sensor data streams feeding into the DT. Optimized for handling large volumes of time-stamped data, time-series databases provide efficient storage and fast querying for real-time analytics, making them crucial for monitoring the DT's ongoing performance, detecting anomalies, and performing trend analysis. Although not general-purpose, their specialized capabilities in managing high-frequency data make them indispensable for supporting the real-time and predictive analytics required for modern DTs.

[1] https://neo4j.com/
[2] https://jena.apache.org/

So far, we provided a brief overview of the role of each storage paradigm in the context of a DT. In the following, the unique strengths and weaknesses of each storage paradigm will be discussed.

3.1.1 Relational Databases

Relational databases represent one of the most widely adopted data storage systems, primarily structured around tables, rows, and columns. Popular systems like MySQL, PostgreSQL, and Oracle are designed to store structured data within well-defined schemas. Over the years, relational databases have served as the backbone of enterprise data management, renowned for their robustness and reliability in managing data across various industries.

One of the critical advantages of relational databases is their maturity as a technology. Being well-established, they benefit from extensive support for standardized querying, mainly through SQL (Structured Query Language), allowing users to manage and retrieve data quickly. Another strength lies in their data consistency, as relational databases are built on the Atomicity, Consistency, Isolation, and Durability (ACID) principles, which ensure reliable data handling and maintain data integrity even during system failures. Additionally, relational databases are optimized for handling complex queries that involve joining multiple tables, making them highly effective when working with structured data where relationships between data points are straightforward and well-defined.

However, relational databases also have limitations, especially in modern, dynamic applications such as DTs. One significant challenge is scalability. Relational databases can struggle to efficiently handle large data streams, which is particularly problematic when managing the massive amounts of time-series data generated by sensors in DTs. Moreover, the rigid schema requirement of relational databases can hinder their adaptability in environments where data models are subject to frequent changes, as seen in the rapidly evolving DT environments. Finally, while relational databases are suitable for superficial relationships, they become suboptimal for managing complex many-to-many relationships. These situations often require costly join operations that can degrade database performance as the volume of data grows.

3.1.2 Graph-based Databases

Graph databases, such as Neo4j, ArangoDB[3], and JanusGraph[4], store data in the form of nodes (entities) and edges (relationships), making them particularly well-suited for applications that require a deep understanding of complex, interconnected relationships. This data model is especially effective when relationships between entities are the primary focus, such as tracing dependencies, modeling interactions, or identifying causal chains.

One of the primary advantages of graph databases is their ability to handle relationships between entities efficiently. They are optimized for querying complex, many-to-many relationships, often essential in DTs where numerous sensors, components, and systems

[3] https://arangodb.com/
[4] https://janusgraph.org/

interact in intricate ways. This makes graph databases ideal for modeling and analyzing the interdependencies between various elements within the DT. Another significant benefit is the flexibility of graph databases regarding schema requirements. Unlike relational databases, graph databases do not enforce rigid schemas, allowing for more accessible adaptation and evolution of data models as the system grows and changes over time. This flexibility is essential in dynamic environments like DTs, where the structure and relationships within the data may evolve as new components are integrated or system behaviors change. Moreover, graph databases excel in graph traversal queries, where the system needs to explore the relationships between nodes. Such queries are highly efficient, making it easier to analyze interconnected systems and derive insights from complex relational data.

However, graph databases are not without their limitations. While they are highly effective at managing relationships, they may face scalability challenges when dealing with massive datasets, mainly if the system includes many data nodes and edges. This can pose difficulties in scenarios where DTs generate vast amounts of sensor data or when the system needs to scale across large, distributed environments. Additionally, graph databases are less efficient when dealing with non-relational data. Other storage paradigms like relational or time-series databases may offer better performance for data queries that do not center around relationships.

3.1.3 Semantic-Based Databases

Semantic databases are designed around ontologies, which provide a structured framework for representing domain-specific knowledge. Typical implementations of semantic databases include Resource Description Framework (RDF) triple stores and Web Ontology Language (OWL)-based stores, which facilitate ontological reasoning over data. These databases utilize RDF and OWL to define entities, relationships, and logical rules, supporting advanced reasoning and querying capabilities through SPARQL Protocol and RDF Query Language (SPARQL). In the context of DTs, semantic databases enable a deeper understanding of data by incorporating domain knowledge and contextualizing sensor data.

One of the primary advantages of semantic databases is their ability to perform ontological reasoning. By leveraging domain-specific ontologies, semantic databases allow users to derive rich insights from the data. This capability is precious in DTs, where it is essential to contextualize sensor data, trace causal relationships, and infer meaningful events or anomalies based on predefined domain knowledge. Another benefit of semantic databases is their support for linked data. These databases explicitly define the relationships between entities, enabling users to navigate, query, and analyze linked datasets quickly. Furthermore, semantic databases, like graph databases, offer schema flexibility. They do not impose rigid schema requirements, allowing them to adapt to dynamic and evolving data models. This is especially beneficial in DTs, where data structures may continuously change.

However, semantic databases come with certain drawbacks. One of the most significant is performance limitations. Querying and reasoning over data in semantic databases can be slow, mainly when dealing with large and complex datasets. This can be problematic for DTs, which generate large volumes of sensor data requiring real-time or near-real-time analysis. Additionally, scalability is another challenge. While semantic databases excel at reasoning over small to medium-sized datasets, they may struggle to scale efficiently when dealing with

high-volume data streams, as commonly encountered in modern DTs' vast and interconnected environments. Another limitation is the complexity associated with developing and maintaining ontologies. Mastering SPARQL queries and understanding the intricacies of ontological reasoning require specialized expertise, adding to the overall complexity of managing the system.

3.1.4 Time-Series Databases

Time-series databases, such as InfluxDB[5], TimescaleDB[6], and Prometheus[7], are specifically designed to manage large volumes of time-stamped data. These databases are optimized for the fast ingestion, storage, and retrieval of time-series data, making them particularly well-suited for applications that require real-time monitoring and analytics. Time-series databases are critical in ensuring efficient data management and analysis in environments like DTs, where sensors continuously generate data over time.

One of the critical advantages of time-series databases is their optimization for handling time-series data. These databases excel at storing and querying vast amounts of sensor data generated at regular intervals, allowing for fast retrieval and efficient analysis of trends over time. Scalability is another significant benefit. Time-series databases are built to handle enormous datasets, making them ideal for DTs that require storing and processing high-frequency sensor data in real-time. This ability to scale effectively ensures that time-series databases can manage the large volumes of data that typically accompany real-time monitoring and analysis. Moreover, time-series databases offer powerful querying capabilities tailored for real-time analytics. They support trend analysis, anomaly detection, and forecasting, all of which are essential for monitoring the ongoing performance of DTs. This enables users to detect irregularities, forecast future behavior, and ensure the optimal operation of the DTs environment.

However, time-series databases also have limitations, particularly in relationship modeling. While they are highly efficient at managing time-based data, they cannot handle complex relationships between entities, such as those in relational or graph databases. This narrow focus on time-series data makes them less suitable for applications that require complex data modeling beyond simple time-stamped entries. As a result, time-series databases often need to be integrated with other types of databases in more complex systems where relationships between entities must also be considered.

3.1.5 Summary

To provide a clear comparison of the different database types discussed, the table below summarizes their strengths and limitations, highlighting how each approach handles sensor data management

Database Type	Strengths	Limitations

[5] https://www.influxdata.com/
[6] https://www.timescale.com/
[7] https://prometheus.io/

Relational Databases	Structured schema, ACID compliance, strong consistency, well-established query language (SQL).	Limited scalability for large-scale sensor data, inefficient handling of time-series data, lacks semantic reasoning.
Graph Databases	Well-suited for representing relationships between entities, enables fast traversal of connected data.	Not optimized for high-frequency time-series data ingestion, lacks standard time-based indexing.
Time Series Databases	Optimized for high-ingestion rate, efficient time-based indexing and querying, supports aggregation and downsampling.	Limited support for complex queries beyond time-based analysis, lacks semantic relationships and metadata integration.
Semantic Databases	Supports semantic reasoning, rich metadata representation, interoperability via RDF and SPARQL.	Not optimized for handling large-scale time-series data, computationally expensive for high-frequency updates.
Hybrid Time series + Semantic	Combines efficient time-series data management with semantic reasoning, enables complex queries leveraging both temporal and contextual data, supports real-time RDF generation and automated linking.	Increased complexity in data integration, requires specialized tools for both time-series storage and semantic processing.

Interpretation of the Table and Justification for Hybrid Storage

Each of the four primary database types has specific strengths, but also clear limitations when handling sensor data at scale:

Relational databases provide structure and reliability but struggle with high-frequency sensor data due to scalability limitations.

Time-series databases efficiently handle large volumes of sensor readings but lack semantic awareness, limiting their ability to answer context-rich queries.

Graph databases excel at modeling relationships but are not designed for storing and querying high-frequency time-series data.

Semantic databases provide strong metadata representation and reasoning capabilities but are inefficient for handling fast, high-ingestion-rate sensor data.

Given these trade-offs, a hybrid approach that integrates time-series databases with semantic storage is the most suitable solution for sensor networks. This approach leverages:
Time-series databases for efficient ingestion, retrieval, and processing of high-frequency sensor data.

Semantic databases for rich contextual representation and reasoning, enabling complex queries that consider both time-based trends and relationships between sensor entities.

Interoperability and reasoning capabilities, allowing integration across IoT systems and enabling inference-based decision-making.

By addressing the efficiency of time-series data management and the contextual richness of semantic storage, the hybrid model provides the most effective foundation for the SENSE system.

3.2 Existing Work on Combining Time-Series and Semantic Data

The primary data storage concepts applied in the auditable SENSE architecture [5] are semantic-based and time-series databases. Therefore, this section focuses on integrating these two storage paradigms by introducing general concepts relevant to this context and then presenting related scientific work.

3.2.1 General Concepts

ETL Processes

Extract, Transform, and Load (ETL) processes are crucial for transforming time-series data into a semantically enriched format. Developing ETL workflows ensures that data extraction, transformation, and loading processes align with the ontology schema. For instance, weather data can be extracted, transformed, and loaded into an RDF triple store, enhancing its semantic richness.

Time Series Dump (Batch Integration)

To integrate time-series data with an ontology, the data must be preprocessed and structured to align with ontology concepts. This involves aggregating and organizing data in a format compatible with the ontology. For example, weather data can be aggregated and mapped to meteorological concepts, facilitating a more seamless integration process.

Ontology-Based Data Access (OBDA) Techniques

Ontology-mediated queries utilize ontology to facilitate complex queries over time-series data. It is possible to map these queries to the underlying database structures by developing SPARQL queries that leverage ontology concepts. For example, SPARQL queries can retrieve energy consumption data from an SQL database, providing a more semantically rich data retrieval process.

Linking Data Streams

The real-time integration of streaming data with ontology can be achieved through techniques such as stream reasoning. This involves continuously linking incoming data streams to relevant ontology concepts, such as linking real-time IoT Sensor data to a smart home ontology. This method enhances the system's real-time monitoring and analysis capabilities.

Semantic Contextual Enrichment

Linking time-series data with contextual information from the ontology adds another layer of meaning. This involves associating data points with relevant contextual concepts. For instance, temperature data can be linked with information about local events, providing a richer context for analysis.

3.2.2 Related Work

Ontologies are a formal, explicit form of a shared conceptualization, which can help within the domain scope to represent relational concepts. The ontology can be described as a tree or a hierarchy, as the basis of the ontology is a graph that can be used in various algorithms [6].

Time-series Sensor data consists of continuous streams of data points collected over time. These data points capture various metrics from Sensors deployed in different environments, such as temperature, humidity, energy consumption, or heart rate. However, raw time-series data lacks context and meaning, making it challenging to derive actionable insights. Ontologies, representing structured knowledge about a domain, can bridge this gap by providing context and relationships.

One of the most significant challenges is linking time-series sensor data with domain knowledge to create meaningful and actionable insights. This integration is achieved through ontologies, which provide a structured framework for representing knowledge. This section explores various existing methods for linking time-series sensor data to ontologies, enhancing the semantic richness and usability of the integrated data.

Manual annotation involves carefully tagging time-series data with relevant ontology concepts to provide labels for the data columns. This process begins by extracting domain-specific concepts from the ontology. Clear guidelines are then developed to ensure that data points or segments are consistently and accurately tagged. This method, while time-consuming, provides high accuracy and contextual relevance.

BECauSE Corpus 2.0 annotates causality and overlapping relations, creating causal constructs as a byproduct. Corpus focuses on causal language, the consistent language used to describe an event/cause that leads to a relationship. Corpus draws on the principle of construction grammar, which leads to causal relationships, albeit arbitrary [7].

Another approach that focuses more on Natural Language Processing (NLP) tasks, effectively highlighting the importance of understanding events and their relations, is the Causal and Temporal Relation Scheme (CaTeRs). CaTeRs is a scheme created for semantic annotations of event structures; the aim is to leverage such annotations to find patterns in the labeling [8].

3.3 Hybrid SOSA Ontology Implementation: Linking Time-Series Data and Semantics

With the ontology structure in place [9] and the ontology instantiation resulting from the TKT methodology to a specific use case (Section 2), the next phase involves integrating heterogeneous time-series data into a unified format that aligns with the ontology. Semantic annotation is a critical part of this process, where data is enriched with semantic metadata according to the ontology, often using RDF triples. This annotation facilitates mapping raw data to the conceptual entities and relationships defined in the ontology, thereby enabling more meaningful data analysis. This section describes a hybrid implementation integrating time-series data with the SOSA ontology part of the SENSE ontology focusing on modeling topological knowledge.

3.3.1 Integrating Time-Series Data with SOSA Ontology

The SOSA ontology provides a structured and semantic framework for modeling observations, measurements, and interactions with the physical world. In an era where sensors continuously generate massive streams of time-series data, there is a growing need to combine the raw numerical efficiency of time-series data with the rich context and meaning provided by semantic models like SOSA.

The data ingestion module [5] is at the heart of this hybrid system, which mediates between the time-series data and the SOSA ontology. For each new data point generated by a sensor, the module queries the SOSA ontology to retrieve relevant metadata, such as the sensor's location, the property being observed, and the specific feature of interest. This metadata is critical for providing context to the raw data. After retrieving this information, the ingestion module combines the time-series data with the metadata. For example, suppose a temperature sensor records a value of 22°C at a particular time. In that case, the ingestion module combines this reading with the corresponding metadata (such as the sensor's location and observed property). The result is an RDF triple that semantically represents the observation, linking raw data and its ontological description.

Once the data ingestion module has gathered the numerical data and the necessary metadata, it generates an RDF triple for each observation. RDF triples encode data in a structured format, making it possible to represent the relationships between different entities. For instance, a temperature reading might be encoded as:

```
:observation_1234 rdf:type sosa:Observation ;
   sosa:hasResult "22"^^xsd:float ;
   sosa:resultTime "2024-09-28T12:00:00Z"^^xsd:dateTime ;
   sosa:madeBySensor :sensor_temperature_101 ;
   sosa:observedProperty :temperature ;
   sosa:featureOfInterest :room_101 .
```

In this example, the RDF triple describes an observation (identified as :observation_1234) that includes the raw temperature reading (22°C) and its associated metadata. The triple specifies that a specific sensor observed a particular property (temperature) and did so at a specific time (result time) in a certain location (room_101). This structure encodes the numerical data and contextualizes it, providing a semantically rich description that machines can process and understand.

Once the RDF triples are generated, they are forwarded to the semantic data and event broker module, where other modules can subscribe to and process the semantically enriched observations. Furthermore, observations contributing to events and possibly containing relevant information for explanations can easily be stored in a graph database such as Apache Jena or GraphDB[8], as they are already encoded in RDF. Users can interact with the data using SPARQL, a query language designed explicitly for RDF data. SPARQL enables sophisticated queries beyond simple data retrieval by exploiting the semantic relationships defined in the SOSA ontology. For example, users can retrieve all events triggered by a particular sensor, find all observations related to these events, or analyze trends across different periods.

3.3.2 Benefits

By combining time-series data with the SOSA ontology, this hybrid system enables a deeper level of analysis. It captures the raw data and provides a structured, machine-readable format that supports advanced querying and reasoning. This approach opens the door to more powerful and flexible applications, such as smart cities, environmental monitoring, and the Internet of Things (IoT), where numerical values and contextual meaning are critical for decision-making and insight generation.

As new sensor data is continuously collected and semantically enriched by the data ingestion module, the raw data and its contextual meaning are always synchronized, reducing the need for manual intervention and enhancing the system's overall efficiency. Each new observation is dynamically annotated with the relevant metadata, enabling immediate processing and query execution.

Another crucial advantage is the system’s ability to generate real-time RDF triples. As sensor readings are captured, the data ingestion module immediately transforms these readings into RDF triples, which encode both the raw sensor values and the associated semantic context. This real-time RDF generation allows for immediate semantic querying of fresh data, a critical feature for use cases such as smart cities or environmental monitoring, where real-time insights are vital for making informed decisions. For example, real-time temperature and air quality data could be semantically analyzed in an intelligent building to optimize energy usage or trigger alerts in case of anomalies.

By leveraging standards such as SOSA and RDF, the hybrid approach ensures that the sensor data is inherently interoperable. RDF triples are designed to be easily shared and understood across different systems, platforms, and domains. This standardization facilitates the integration of sensor data in heterogeneous environments, such as the Internet of Things (IoT), where diverse systems and devices must communicate and exchange information.

[8] https://graphdb.ontotext.com/

Furthermore, the interoperability of RDF triples makes it possible to share data across cross-domain applications.

One of the most potent aspects of combining time-series data with SOSA semantics is the ability to apply semantic reasoning and infer new knowledge from the data. By representing sensor observations as RDF triples, the system can use reasoning techniques to derive additional insights that would be impossible to obtain from raw time-series data alone. For example, if multiple temperature sensors in different rooms report unusually high readings, the system might infer a heating malfunction in the building. This type of reasoning would require knowledge about the layout of the building, the locations of the sensors, and the properties being measured, all of which are encoded by the SENSE semantic model [9].

4 Summary

This document outlined a comprehensive approach to integrating time-series data with domain knowledge by engaging with domain experts and employing advanced semantic techniques.

Future work can also include Machine Learning (ML) and Artificial Intelligence (AI) integration with ontology-guided machine learning. This will entail using ontologies to guide ML model training and interpretation to ensure the models are contextually relevant. This involves developing models with ontology-based labels and interpreting the outputs in the context of the ontology. For example, anomaly detection models can be trained with labels guided by an ontology, enhancing their accuracy and relevance.

5 List of Abbreviations

Short	Description
CaTeRs	Causal and Temporal Relation Scheme
CPS	Cyber-Physical System
DT	Digital Twin
ETL	Extract, Transform, and Load
HVAC	
NLP	Natural Language Processing
OBDA	Ontology-Based Data Access
OWL	Web Ontology Language
RCA	Root Cause Analysis
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
SOSA	Sensor, Observation, Sample, and Actuator
TKT	Tacit Knowledge Tree

References

[1] M. C. Klein and D. Fensel, "Ontology versioning on the Semantic Web," in *SWWS*, 2001.

[2] R. Ouriques, K. Wnuk, T. Gorschek and R. B. Svensson, "The role of knowledge-based resources in Agile Software Development contexts," *Agile Software Development contexts. Journal of Systems and Software, 197, 111572,* 2023.

[3] H. Park and S. Sangyun, "A Proposal for Basic Formal Ontology for Knowledge Management in Building Information Modeling Domain," *Applied Sciences 13.8,* 2023.

[4] D. Jähnig, C. Moser, T. Frühwirth, K. Schreiberhuber, J. Kainz, D. Hauer, K. Diwold and M. Sabou, "SENSE Deliverable 2.1: Definition of Use Cases and User Stories," 2023.

[5] T. Frühwirth, G. Steindl, T. Schwarzinger and F. Ekaputra, "SENSE Deliverable 3.1 Auditable SENSE Architecture," 2024.

[6] M. Ben Messaoud, P. Leray and N. Ben Amor, "Integrating ontological knowledge for iterative causal discovery and visualization," in *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Berlin, 2009.

[7] J. Dunietz, L. Lori and J. G. Carbonell, "The BECauSE corpus 2.0: Annotating causality and overlapping relations," in *Proceedings of the 11th Linguistic Annotation Workshop*, 2017.

[8] 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*, 2016.

[9] M. Sabou, M. Memedi, K. Schreiberhuber and F. J. Ekaputra, "Deliverable 4.1 (v2): Semantic-Based Explainability Framework," 2024.