Knowledge Core: An SCEP Architecture for Smart Installations with Organization-Defined Policies

Ian Riley, Logan Quirk, Allen Marshall, Jacob Brue, Joe Shymanski, Vincent Gonzales, Rose Gamble
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
ian-riley@utulsa.edu, gamble@utulsa.edu

Abstract

With an increase in smart applications and ubiquitous IoT computing, it has become increasingly necessary to investigate available approaches to processing sensor data. Complex event processing provides a means of associating sensor measurements to categories of observed events, such as entry and occupancy, which are more meaningful to smart application developers. Recent advancements in semantic complex event processing (SCEP) provide an opportunity to evaluate matched events within the context of organization-defined policies defined as semantic ontologies. However, research into the application of SCEP to smart applications is currently scarce. In this paper, we introduce the Knowledge Core SCEP architecture which extends the traditional SCEP architecture to a constellation of smart installations with organizational policies. The application of this architecture can provide smart application developers and maintainers with a novel means of processing the data generated by their smart application within a context of what matters to their organization.

Keywords: Software architecture, CEP, SCEP, Smart installation, Ontological monitoring.

1. Introduction

The last decade has seen an explosion of smart applications built on vast sensor networks with the number of sensors only increasing. This rise in smart applications provides an opportunity for novel methods and architectures that can utilize the available data to produce meaningful alerts. For the last decade, researchers have been investigating the use of complex event processing (CEP) (Luckham, 1996) (Schaaf et al., 2012) to match sensor data to events that are interesting to application developers and domain experts. Additional contributions have come in the form of semantic CEP (SCEP) (Schaaf et al., 2012) (Teymourian et al., 2009) which can match observed events to pre-defined ontologies with human-understandable meaning. SCEP engines give application developers and their maintainers the opportunity to identify events spread throughout their data within a context that matters to their organization.

Despite recent advancements in SCEP, research into its application to smart domains is scarce. Over the past decade, few efforts have been made to apply SCEP to smart applications other than recent publications concerning smart homes (Vassiliades et al., 2020) and basic IoT applications (Daoudagh et al., 2022). These applications are purportedly successful but not followed by additional research. In addition, they lack resource and access control policies that are relevant to a great deal of smart applications that are subject to security, privacy, and accessibility concerns that can be monitored by available sensors. With the growing number of smart applications that include VR digital twins and semi-autonomous robotic platforms, it is necessary to investigate novel methods into extracting semantically meaningful data from sensor networks.

In this paper, we introduce the Knowledge Core SCEP architecture which extends the traditional SCEP architecture to a *smart constellation*. We define a smart constellation as a set of distinct *smart installations* that can share a limited set of resources with varying costs-of-use relative to those resources. Smart installations include any building or residence that deploys sensors on a monitored network for safety, security, performance, and accessibility. For the work of this paper, a smart constellation would include a collection of two smart university buildings (indoor) and one smart industrial building (outdoor) that are intended to have similar sensor networks and monitoring needs. The installations can share data, personnel, and equipment, but the availability and timeliness of these resources vary across locations due to the physical distances between the installations. The Knowledge Core SCEP engine has the capability of assessing matched events within the context of organizational policies, their significance, and the resource cost of available responses.

2. Background

CEP was first introduced by Luckham (1996) as part of the development of Rapide. The work of Luckham and later collaborators has culminated into the CEP architecture shown in Figure 1. The CEP architecture consists of four major components: event specification, event processing rules, an event processing engine, and an enterprise integration backbone. Event specification describes all events that can take place as well as their interactions with each other. Event processing rules handle the events by re-formatting, aggregating, or even generating new events. The event processing engine executes the rules for the defined events. Lastly, the enterprise integration backbone acts as a mediator between the CEP architecture and the enterprise system to which it's deployed.

The original CEP architecture was designed around raw sensor data being stored as event data in a relational database. Custom specifications had to be defined by hand and were often specified in a SQL derivative. In the two decades since Luckham's original paper, several commercial CEP systems have become available, each touting their own custom SQL derivative (Alaghbari et al., 2022). These commercial applications have minor differences between one another, but none stray too far from Luckham's original concept of an event. Although, event processing has since expanded to include events related to other IoT initiatives and analytics over textual, graphical, and video data streams (Dayarathna & Perera, 2019). Teymourian et al. (2009) later extended the CEP architecture to utilize a resource description framework (RDF) based on then current ideas of the semantic web. This initial development of SCEP introduced a novel relationship between events and semantic ontologies that could provide context to those events. However, event specification remained a key challenge that only increased in complexity with the introduction of semantic ontologies, such as OWL. The initial approach to this challenge was yet again another SQL-like specification language called SPARQL, which is the domain specific language for querying RDF databases.

At the same time, other authors were pursuing formal notations for representing observable events. Broda et al. (2009) developed SAGE, a monitoring and control system that employed event calculus to design a specification of events to be monitored by the system. Event calculus extends first order logic to model temporal relationships of events as time points and fluents, which hold values that can change over time. This in turn allows an event or sequence of events to be modeled as changes to a fluent over successive time points. Skarlatidis et al. (2015) extended these notions into probabilistic event calculus by introducing a mapping between propositions in event calculus and statements of conditional probabilities. In their work, the likelihood of each proposition was evaluated using a Markov Logic Network.

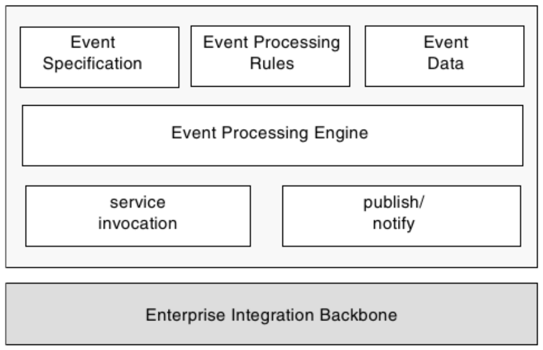

Figure 1. CEP architecture (Schaaf et al., 2012)

More recently, Patkos et al. (2016) have employed these efforts within SCEP to model the likelihood of complex events using an enhanced Bayesian network. Their SCEP architecture was deployed within the context of a smart home to assist physically challenged individuals with daily activities. Their efforts show that probabilistic event calculus and belief networks can be employed within an SCEP architecture to ascertain the likelihood that an event is occurring. It is one of few efforts that have been taken to incorporate SCEP within the wider context of IoT. Daoudagh et al. (2022) introduce DAEMON, an ontological approach for monitoring IoT cyber-security. Their architecture includes an SCEP engine to match against known malicious and anomalous events that threaten the security of their IoT devices.

Based on our research into the literature, it is our opinion that there are no readily available surveys of recent applications of SCEP architectures or applications apart from efforts taken to survey general applications of event processing (Dayarathna & Perera, 2019). It is therefore our understanding that SCEP has not been applied to a smart installation other than a smart home. Furthermore, many smart installations deploy sensors as a means of assuring the security, privacy, and accessibility of the facility. These properties can be encoded as organizational policies, such as those published by the National Institute of Standards and Technology (NIST). NIST publishes guidelines for families of cyber-physical controls with organization defined enhancements. The published guidelines on the family of physical controls (National Institute of Standards and Technology [NIST], 2020) serves as an appropriate model for smart installation policies and is utilized within US critical infrastructure. SCEP is a highly applicable technology for processing observed events that impact organizational policies but research to that end is scarce. Further research is needed to extend the SCEP architecture to smart installations with organizational policies, which is the aim of this paper.

3. Knowledge Core Architecture

The SCEP architecture introduced by Teymourian et al. (2009) can be seen in Figure 2. This architecture was intended to be a general improvement upon the CEP architecture both in its more modern design (for 2009) and its inclusion of a semantic framework. The architecture is broken into two parts, bridged together by middleware, which could utilize any transport protocol. The semantic CEP engine is broken into three different parts. The state processor determines the initial state of the event processing, based on the simple event stream. This is used by the rule engine to determine what subsequent information is needed to warrant the generation of a complex event. These event requirements are handled by a query processor which crafts queries responsible for retrieving the necessary information. The queries are handled by the triple store adapter which consults two components: the event history and the knowledge base ontologies/rules. The event history contains the previous simple events while the knowledge base contains information over the rules themselves, represented in a composite ontology.

The SCEP architecture, like the CEP architecture before it, is highly applicable to applications that involve a sensor network (Broda et al., 2009) (Daoudagh et al., 2022) in which notable (or anomalous) events are drawn from a combination of sources rather than just one individual sensor. In the domain of CEP and SCEP, raw sensor data is categorized into a discrete set of *simple events*. These simple events are the atomic units of the SCEP architecture and are stored as triples in the SCEP *triple store*. As seen on left-hand side of Figure 2, the original SCEP architecture proposes that sensors would transmit data on an integrating middleware bus connected to the triple store (bottom-right) and SCEP engine (top-right). To better accommodate smart applications today, this integrating middleware bus requires support from load balancers that can reduce the load on the bus from the array of available sensors. Otherwise, the bus becomes clogged and the SCEP engine cannot process queries against the triple store in a timely manner.

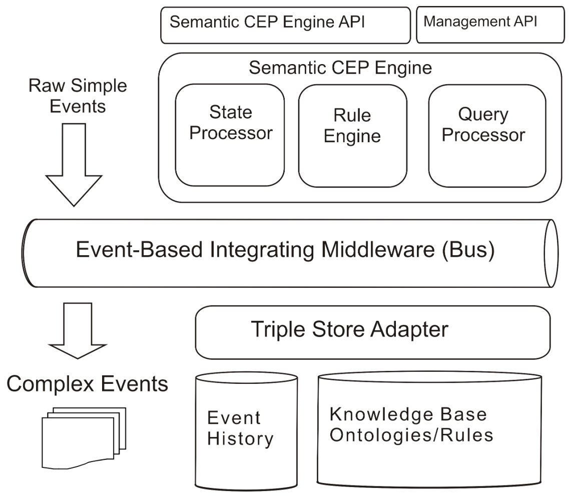

Figure 2. SCEP architecture (Schaaf et al., 2012)

Smart installations, such as those employed in this paper, are commonly cited as a use case for both the CEP and SCEP architectures. However, the number of demonstrable use cases in related literature is very limited. Notable examples include smart homes (Vassiliades et al., 2020) and other basic IoT applications (Daoudagh et al., 2022). The goal of these previous applications was to determine if a monitored user is engaging in one of a set of pre-defined activities. The applications lack resource and access control policies that govern how users engage with the installation as well as the function of the SCEP system. Without these policies, applications of SCEP architectures are limited to the detection of events without an awareness of the greater significance that those events might hold given the context in which they occur.

When policies are applied to an installation, the same event can have a different meaning depending on which policies it impacts. Entry into a room can be insignificant ingress, a violation of fire codes, an intrusion, or a point-of-interest depending on instituted policies. While policies may differ depending on the installation, a constellation still requires consistency in its representation of events across connected installations to allow for potential adaptations and the inferred extension of policies from one installation to another. Our approach to the SCEP architecture has the potential to capture the significance that policies bear in the monitoring and maintenance of a smart installation within a smart constellation.

Our research use cases work with smart installations that target more realistic deployments of sensor networks to allow for intelligent decision making. Thus, the smart installations are subject to policies that dictate installation use and performance, such as the security policies introduced by NIST (2020). We introduce an extended SCEP architecture capable of evaluating whether monitored activity complies with or contradicts an installation's policies, with an intent to generate alerts when the latter occurs. One of the primary benefits of these alerts is the ability to connect sensors and their data to the wider context of the organization's goals and the relative significance of those goals. For example, the occupancy of a room has more significant meaning if room activity is viewed under the lens of a possible fire or an on-campus intruder. Allowing for variation in policy significance is the foundation of a system that can justify autonomous or semi-autonomous responses, such as those outlined in the NIST (2021), within a context that humans can understand. When such responses involve a variation of resource cost, human-explainable justification should be required so that an appropriate cost is paid for an equitable and revisable outcome.

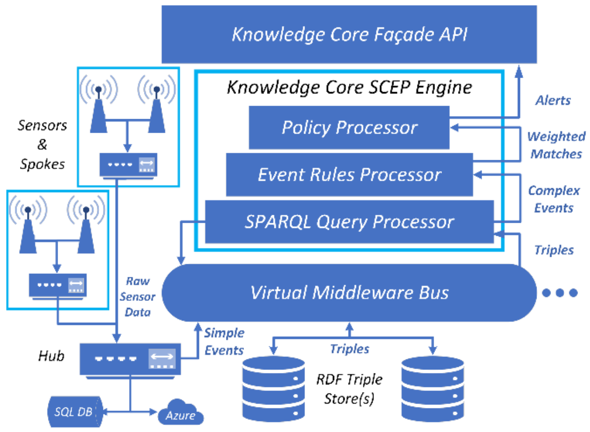

Figure 3. Knowledge Core SCEP architecture

Figure 3 shows our extended Knowledge Core SCEP architecture deployed for our smart constellation use case. In our initial investigation into applying the original SCEP architecture (Figure 3), we encountered several challenges that needed to be addressed from an architectural perspective. The most significant changes from Figure 2 to Figure 3 are how sensors are integrated into the architecture, the shift to an explicit distributed virtual middleware bus, and the introduction of a *policy processor*. These changes reflect the modern demands of a smart application both in terms of the volume of data that sensors output and the distribution of load required to handle that volume. While Figure 3 shows a single instance of the Knowledge Core SCEP engine, additional instances can be constructed and connected via the virtual middleware bus. The relocation of events and the inclusion of a policy processor further illuminates the role that events have in the engine and their eventual association with policies that dictate their significance.

The edge of the Knowledge Core SCEP architecture as shown in Figure 3 consists of a wide array of *sensors and spoke networks* (top-left). These networks are intended to be customizable and unfettered by the demands of the Knowledge Core SCEP engine. Sensors in these networks can be installed and configured in whatever manner is most appropriate for the sensor and what it is monitoring. The only requirement is that sensors in the same sensor network be connected to a single spoke, which bridges the sensor network to the Knowledge Core SCEP engine via a *hub* (bottom-left). This topology is commonly known as a hub-and-spoke network. The hub acts as a mediator between the sensor networks and the rest of the Knowledge Core SCEP architecture. The hub is also responsible for rate limiting, filtering, and caching sensor data as appropriate for the installation. Raw sensor data collected within the hub is then output to the *virtual middleware bus* (middle-right) as a collection of simple events. These simple events travel along the virtual middleware bus to a collection of *triple stores* (bottom-right), which store simple events in a semantic framework as semantic subject-predicate-object triples. It is expected that each installation would have one hub and one triple store, but additional hubs and stores can be added to accommodate larger and/or more demanding installations.

The virtual middleware bus connects instances of the *Knowledge Core SCEP engine* (top-right) to the triple stores that possess the semantic data. Triples in the RDF triple store are pulled into the engine via the *SPARQL query processor*. SPARQL, developed for use in the semantic web, employs SQL-like syntax to describe semantic queries to the RDF to extrapolate over which triples should be extracted (see Section 4.1). Triples drawn into the query processor are then restructured as complex events (see Section 4.1) and output to the *event rules processo*r.

The event rules processor is a consolidation of the state processor and the rules engine from the original SCEP architecture shown in Figure 2. The event rules processor matches complex events to rules specified in probabilistic event calculus (Skarlatidis et al., 2015) (see Section 4.2). This notation in combination with a belief network (Patkos et al., 2016) allows for complex events to be assigned a degree of certainty that can be further combined to assign a certainty to the rule itself. These weighted rules are then utilized by the *policy processor* (see Section 4.3) which matches the rules against related policies to determine the certainty with which policies have been satisfied or violated. The violation of policies results in alerts that can be pushed out from the Knowledge Core engine through its *façade API* (top).

3.1. Sensor Networks

Smart applications typically involve a large volume of data due to the number of sensors employed and the frequency at which those sensors report. For our application, we deploy sensors that combine humidity, temperature, infrared-motion, and LEDs into one Arduino device that communicates over Wi-Fi or a USB connection. We employ these sensors in combination with others, such as magnetic door sensors and O2/CO2 sensors. A single room of a smart installation might contain at least a dozen sensors that report twice per second via Wi-Fi or to a nearby machine over USB. An installation can contain dozens of rooms or regions that would need to be monitored this way. To accommodate the load that these sensors would place on an integrating middleware bus (Figure 2), we've connected them to the Knowledge Core SCEP architecture via a hub-and-spoke network topology as seen on the left of Figure 3.

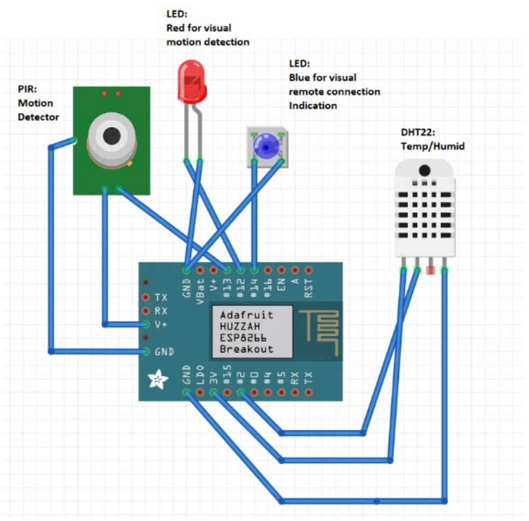

Figure 4. Arduino sensor configuration

A diagram of the custom configuration of our sensors and their pin connections to an Adafruit Breakout board (middle) are shown in Figure 4. The Arduino sensor includes a DHT11/DHT22 sensor (right) to measure temperature and humidity, a Pyroelectric infrared (PIR) motion sensor (left), and LEDs (top) for visual indicators such as when motion is detected (red) and when the device is connected (blue). The sensor does not include a sufficiently accurate onboard timer, so the Arduino must be synced with a network time protocol (NTP) server via another machine that can assign timestamps to readings. For our experiments, we have written a simple Python script to monitor serial connections on a nearby Linux machine that reports timestamped measurements to the spoke.

3.2. Sensor Spokes and Hubs

A *spoke* is a component that is responsible for accepting, combining, and retransmitting sensor data. Spokes are useful for distributing the number of required connections across separate machines. They also serve as an endpoint to minimize the volume of edge data. The latter case is common in smart city applications and fog computing where a high degree of sensors exist. The spokes in our architecture are responsible for directly interfacing with regions of sensors. A single installation could have one or more spokes depending on the size of the installation, where a spoke might be responsible for a single room, section, department, or network, depending on the needs and topology of the installation. Unlike in the original SCEP architecture (Figure 2), sensor networks are not connected directly to the middleware but are instead bridged to the Knowledge Core SCEP architecture via hubs.

A *hub* is a component that is responsible for interfacing with one or more spokes. To support an array of hubs across different installations, the bus is both virtual and distributed as described in Figure 3. The hub serves as an outlet to further reduce the volume that sensors might otherwise place on the bus as not every sensor reading needs to be reported. Smart installations are notorious for having a sparseness of interesting data. For example, sensors monitoring activity within a conference room might report two times per second every second of every day. However, if the conference room is only used when booked and that only occurs for a portion of the workday, then these sensors only report observed activity about 10% of the time. This is not to say that the other 90% of sensor readings can be ignored. That the conference room is not in use during those other times is pertinent if the conference room is to be booked or to assure it against intrusion. The sparseness of activity implies instead that not all data from the same sensors can be weighed equally over time, which is especially important when attempting to extrapolate meaning from the data.

For our application, we developed a set of Python Flask web services as a proof-of-concept that are deployed via Docker containers. Spokes collect sensor data via a RESTful API and then transmit that data to their respective hub. Hubs also serve a key role in caching sensor data. One of the features in our smart constellation that stretches the triple store is the need to both maintain sensor data over a large period of time and support the reasoning capabilities of the Knowledge Core SCEP engine, which are often at odds with each other due to the performance overhead of RDF. To alleviate these concerns, hubs cache incoming sensor data to a local PostgreSQL database before reporting data on the virtual middleware bus. Each hub has a limited cache of data that only lasts long enough to ensure that relevant sensor data is not lost. For long term storage, data from the PostgreSQL database is uploaded to a cloud database hosted in Microsoft Azure. This upload allows sensor data to be retrieved later in its raw form. Any sensor data accepted into the triple store is first converted to an XML triple, which is not required to be a lossless conversion. As such, the triple store would not be the appropriate place to preserve raw sensor data and should be viewed instead as a database for event processing rather than as a storage solution.

3.3. Virtual Middleware BUS

The Knowledge Core SCEP architecture relies on a decentralized, distributed virtual bus for the middleware between Knowledge Core SCEP engine instances, hubs, and triple stores. The core idea behind a virtual bus is that the bus is implemented over an application protocol rather than as a single physical network or lower-level transport protocol. This in turn allows for devices to join and interact with the middleware through a simple API that abstracts away the topology of the middleware. In our case, we implement the virtual middleware bus as a series of HTTP endpoints that are deployed via Docker containers organized into a connected set of ring networks, one for each smart installation. Simple events and queries that are emitted on the bus are passed to the triple stores while triples are passed to the Knowledge Core SCEP engine that requested them.

3.4. RDF Triple Store

The semantic framework that's typically employed in an SCEP architecture RDF which stores semantic triples. RDF was first introduced as part of the development of the semantic web. It is a non-relational graph database that represents triples in graph form rather than as columns in a table. For the purposes of event processing, each individual triple represents a semantic relationship between an event and one feature of a sensor measurement, such as the sensor's device identifier or its sensor reading. The dimensions of an event are represented as a collection of related triples composed into a graph, such as the graph presented in Figure 5. For example, the triple (sosa:Observation, sosa:madeBySensor, "device1") indicates the existence of a sensor observation made by "device1", where the terms sosa:Observation and sosa:madeBySensor are members of the public SOSA ontology. Ontological terms can be roughly thought of as human-readable tags with pre-defined semantic meaning. An advantage of this format is that it allows for triples to be extended into other ontologies via any of its three members.

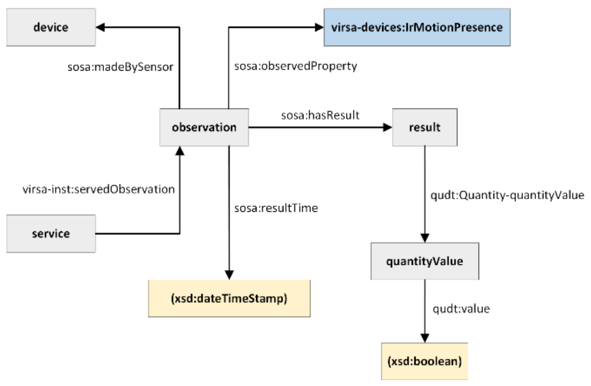

Figure 5. PIR motion presence as stored in RDF

Figure 5 shows a graphical example of a PIR motion sensor reading stored in RDF. The root of this graph is the *service* node (left), which exists as a part of a triple connecting it to *observation* (middle) via the ontological term *virsa-inst: servedObservation*. The term virsa-inst: servedObservation is an ontological term that we've defined to reflect that this data is an observation from our installation dubbed VIRSA. The observation is associated with sensor values by way of other predicates. We use predicates from three public ontologies SOSA, QUDT, and XSD that can be integrated with our own newly defined predicates. The *sosa:madeBySensor* predicate (top-left) identifies the *device* that performed the reading while the *sosa:observedProperty* (top-right) identifies the kind of sensor that reported. For a PIR motion sensor, the kind of device is indicated by *virsa-inst:MotionPresence*, which is a unique term that we introduced to differentiate between kinds of sensors. The *time* (bottom) at which that observation was taken can be predicated by the *sosa:resultTime* and stored in the *xsd:dateTimeStamp* format. The *result* (right) of the observation is identified by the *sosa:hasResult* predicate and is typified through the QUDT ontology. In this case, the result is a *qudt:value* stored as an *xsd:boolean*. This graph can be instanced for each PIR motion sensor reading by assigning value literals to the device, xsd:dateTimeStamp, and xsd:boolean nodes.

The contribution of this event specification is an ontological representation of a sensor reading that can be applied to any sensor within a smart installation. The description shown in Figure 5 is particular to a motion sensor that outputs Boolean flags. However, this graph can be templated to any type of sensor by replacing the virsa-inst:MotionPresence with a custom tag that identifies the kind of sensor and the XSD type that best matches the type of the sensor's output. This specification can be connected to any RDF by connecting its service node to the root of the RDF or any other appropriate node within its graph. It can also be extended into any other ontology by using the observation node as the subject of semantic triples that predicate into that ontology. We exemplify this by extending the PIR motion presence specification into a new NIST ontology in Section 4.3.

The downside to RDF is that queries over the data require a graph walk which is significantly slower than performing table lookups in a relational database. For example, identifying the time of PIR motion sensor readings would require walking through a graph composed of millions of instances of Figure 5. This in turn influences the need to have separate supporting relational databases if large amounts of data need to be extracted from the database at one time as would be common for traditional database access. For our implementation, we use the Apache Jena Fuseki database which provides a native high performance triple store that aims to alleviate the performance concerns introduced by RDF.

4. Knowledge Core SCEP Engine

The Knowledge Core SCEP engine is where event processing takes place within the Knowledge Core SCEP architecture. It is partitioned into three layers which include the SPARQL query processor, the event rules processor, and the policy processor. Each layer is responsible for applying a transformation to the output of the layer underneath it. The input to the Knowledge Core SCEP engine consists of triples like those shown in Figure 5. Its final output is a stream of alerts that signal violations of organizational policies. In our implementation, triples are polled from the RDF store every 30 seconds, and events are emitted by the Knowledge Core SCEP engine via a web socket, which accepts subscriptions through the Knowledge Core façade API.

The novelty of the Knowledge Core SCEP engine is its ability to utilize a unique feature of RDF for the evaluation of organizational policies. Unlike relational databases, RDF supports the open world assumption which permits the inferencing of data not currently represented within the triple store. The *open world assumption* states that not all data which is valid is known. As such, events that are unobserved might still have validly occurred and can be inferred. For example, when people use conference rooms to host meetings, they will often close the door to ensure that the meeting is uninterrupted by outside noise and to be similarly courteous to others. However, if the same conference room is booked for an office party, the door may be left open for the full duration of the party. In the case of the meeting, attendees would need to open and then close the door when joining the meeting, which would not be the case in the event of the office party. If we assume that door is being monitored, then under the open world assumption, it is possible to infer that door activity occurred. In smart installations, the ways in which people interact with buildings, rooms, resources, etc. are dynamic and the absence of data should not be held as an indication of the non-existence of events.

By leveraging rules of inference, such as the inference API available to the Fuseki server, the Knowledge Core SCEP engine can better match observations to complex events and ultimately to organizational policies. By definition, inferred data is not known with complete certainty. As such, the Knowledge Core SCEP engine couples inference rules with a belief network that can assign belief to complex events whether or not they include inferred data. The result is a set of beliefs in complex events that can be combined with the significance of affected organizational policies to perform an assessment of installation compliance.

4.1. SPARQL Query Processor

```
(1) SELECT DISTINCT ?dev ?prop ?val ?time
  WHERE {
(2) ?service <virsa-inst:servedObservation>
    ?obs .
(3) ?obs <sosa:madeBySensor> ?dev .
(4) ?obs <sosa:observedProperty> ?prop .
(5) ?obs <sosa:hasResult> ?result .
(6) ?result <qudt:Quantity-quantityValue>
    ?qVal .
(7) ?qVal <qudt:value> ?val .
(8) ?obs <sosa:resultTime> ?time .
(9) FILTER(?time >=
      "..."^^<xsd:dateTimeStamp>)
(10) }
```

Figure 6. PIR motion sensor SPARQL query

The SPARQL query processor is responsible for emitting pre-defined SPARQL queries that are used to elaborate over triples stored in RDF. Once extracted, triples are then composed into *complex events*, which are compositions of simple events. Figure 6 shows a trimmed snippet of a SPARQL query used for extracting triples associated with motion sensors. This query has been modified to fit within the available margins of this paper. The first line (1) of the SPARQL query lists which variables are being selected. Each variable begins with a question mark, such as ?dev and ?prop. In this case, we're looking for unique pairing of device name, sensor property, reading value, and observation time. The following statement (2-8) give constructions of triples that can be matched against the RDF graph to construct the values listed in the select statement (1).

For example, the device name is extracted in statement (3) using a triple whose subject is ?obs and whose predicate is <sosa:madeBySensor> (see Figure 5). ?obs is a new variable introduced by the prior statement (2). With respect to device name, statements (1-3) can be interpreted as, select all device names that are the name of a sensor whose observation was taken within the VIRSA installation. Statement (9) provides a filter so that only recent observations are returned. The literal "…" is a placeholder for the oldest timestamp that should be accepted.

To construct the complex event for room entry, we use the query for motion presence and join its result with a similar query for door activity over the same time interval. Simple events, triples, and complex events are represented in XML. It's possible that the queries might not find any matches, that only one of the two queries would find a match, or that available observations don't indicate the existence of an entry event. The SPARQL query processor is not required to produce complex events that are valid for a given rule. The SPARQL query processor emits the result of the conjoined queries for the event rules processor to evaluate.

4.2. Event Rules Processor

Complex events emitted by the SPARQL query processor are then matched against pre-defined event rules inside of the event rules processor. Given the open world assumption, it is possible that complex events generated by SPARQL queries that are not well formed can still be used to infer missing data. Going back to our earlier example, a conference room that's hosting an office party would include several room entry events, but only a few of these events might involve interaction with the door. As such, complex events for room entry might only contain matches against motion presence but not door activity. It might also be the case that no room entry occurred. The responsibility of the event rules processor therefore is to match complex events against event rules with an associated degree of certainty. To compute the degree of certainty of complex events, event rules are described in probabilistic event calculus (Skarlatidis et al., 2015).

Using our example, we can describe a room entry event as a proposition of simple events involving motion exterior to the room, the opening of the door, motion interior to the room, and the closing of the door. The closing of the door would then be followed by interior motion and room activity, assuming that there are other sensors in the room. These would be the most ideal circumstances for an entry event. Similarly, an exit event can be described in the same terms but with interior motion occurring before exterior motion. This idea can be formally represented in probabilistic event calculus as follows.

$$\begin{aligned}&happensAt(entry(R1, R2), T2)\\&\Leftarrow [initiatedAt(motion(R1), T1)\\&\wedge happensAt(doorOpen(R1, R2), T1)\\&\wedge terminatedAt(motion(R1), T2)\\&\wedge initiatedAt(motion(R2), T2)\\&\wedge happensAt(doorClose(R1, R2), T2)]\end{aligned}$$

The proposition above describes room entry into room R2 from room R1 using a connecting door between time T1 and time T2. This event is implied by the conjunction of the following terms. Each of those terms is a separate event that can be observed by a sensor. These events are motion detected in R1 at T1 along with the door between R1 and R2 being opened at T1, which is to say that motion is observed outside the room being entered while the door is being opened. These two events are succeeded by cessation of motion in R1 at T2 along with observed motion in R2 at T2 while the door between R1 and R2 is being closed. This is to say that motion outside of the room being entered ends once the person enters the room and closes the door. The whole proposition can thus be read as "an entry event into a room follows from someone outside the room opening the door, entering the room, and then closing the door." An exit event from room R2 could likewise be modeled as an entry event into room R1 from R2.

$$\begin{aligned}&P\big(entry(X = R2)\big)\\&= P\big(door(X = close), motion(X = R2)\big)\\&\cdot P(door(X = open), motion(X = R1))\end{aligned}$$

Given the proposition of an entry event into room R2, we can then compute its likelihood from the joint probability of the observed events. The above statement shows how that probability is calculated. The likelihood of entry into room R2 follows from the product of the joint likelihood of the door being closed with motion in R2 and the joint likelihood of the door being opened with motion in room R1. Specifically, the joint probability of the door being closed while in room R2 is conditioned on the other joint probability, but this has been left off for brevity.

However, it is unclear from the above statement how to calculate the likelihood of a room entry event should the complex event be ill-formed. If the door is not opened or closed, the natural choice would be to compute the likelihood of room entry as 0, which would not be correct if someone entered the room while the door was open. Therefore, we propose to include a belief network within the event rules processor that can infer the likelihoods of each event from relevant evidence. For example, the door open and door close events could be inferred with higher certainty should there be a prior complex event with motion around the door, the door being opened, and then left opened. Markov Logic Networks or enhanced Bayesian networks can be used to infer the certainty of rule matches. The computed certainty can then be utilized by the policy processor to assess compliance with organizational policies.

4.3. Policy Processor

By employing a belief network, it would be possible to infer complex events with some degree of certainty. Part of the inference would not only be to infer whether a complex event might exist despite absent simple events but also whether the complex event might relate to a pre-defined set of policies. Smart installations are often subject to a governing set of policies that determine their security requirements. Since we are applying the Knowledge Core SCEP architecture to a constellation of smart installations, we utilize physical access control policies published by NIST (2020). This family of policies dictate controls that can be further elaborated and applied to a physical location. One such relevant policy is “PE-6: Monitor Physical Access”, which states that physical access to the facility where the system resides should be monitored to detect and respond to physical security incidents. It includes four control enhancements. We use control enhancement 4, which states that physical access to a controlled system should be monitored in addition to the room that contains it or any of its components. To our knowledge, no author working with SCEP has sought to employ these controls or similar controls.

To incorporate this family of policies within our smart installation, we introduce a simplified NIST ontology, which includes predicates for <nist:assuresPolicy> and <nist:pe6-4>. We then extend the event specification from Figure 5. by predicating over the observation node with <nist:assuresPolicy> where the object of the predicate is <nist:pe6-4>. We similarly expand the SPARQL query to capture all <nist:assuresPolicy> triples so that they can be included in the construction of complex events. The policy processor then assesses compliance with organizational policies given the certainty in matched complex events that exist within their ontology.

The contribution of this effort lays a foundation that could provide human experts with a novel way to interact with smart installations. The application of policies would allow an expert to visualize their data in terms of its impact on potential policies that are or would be used to govern the security of their installation. For example, entry and exit events go from a description of people entering or leaving a room to a description of whether or not that room is observing fire codes or has been the subject of intrusion. Violations of such policies then become alerts generated by the Knowledge Core SCEP architecture and available for consumption by human operators or as justification for autonomous responses.

5. Future Work

There are several efforts that have been left for future work. One notable challenge within the domain of SCEP is rule generation and the relationship between formal event rules and SPARQL queries. Given the grammatical structure, it might be possible to transcompile event rules to SPARQL queries and vice versa. Furthermore, given a propensity of data, rules could be inferred from the RDF databases across the constellation. Similarly, the use of machine learning techniques such as LLM's can possibly be used to investigate new relationships between simple and complex events. Efforts are underway to use the ontology of organization policies as it has been applied to one smart installation to infer application of that ontology to all other installations within the same constellation. This would allow for a dynamic application of organizational policies that could prove extremely useful within smart applications and especially digital twins.

The additional inclusion of organizational policies into SCEP presents a significant opportunity to assess responses and mitigation strategies in the event of failed compliance. It is our intention to apply temporal reasoning and risk assessment to the policy processor to evaluate possible responses that a system could take when organization policies aren’t met. In addition, we aim to perform a quantitative analysis of novel applications of the proposed Knowledge Core SCEP architecture in comparison to other published applications of SCEP architectures, such as those cited in this paper.

6. Conclusion

In this paper, we introduce the Knowledge Core SCEP architecture which is an extension of the traditional SCEP architecture to a constellation of smart installations with organizational policies. The architecture that we propose includes improvements to meet the demands of modern sensor networks via a distributed, virtual middleware bus with connected hub-and-spoke networks. It also includes a novel construction of the SCEP engine via an inclusion of a policy processor that evaluates complex events within the context of pre-defined organizational policies. The Knowledge Core SCEP engine includes a belief network that can assign a degree of certainty to matched complex events that exist within an ontology of organization-defined policies. The degree of certainty in matched complex events is used to assess compliance with organizational policies and generate alerts when compliance is not met. With the inclusion of future work, this architecture would provide a novel way for domain experts to interact with the data generated by their smart installations.

Acknowledgement. This material is based upon work supported by the Engineer Research and Development Center - Information Technology Laboratory (ERDC-ITL) under Contract No. W912HZ23C0009.

7. References

Alaghbari, K. A., Saad, M. H. M., Hussain, A., & Alam, M. R. (2022). Complex event processing for physical and cyber security in datacentres - Recent progress, challenges and recommendations. *Journal of cloud computing*, *11*(1), Article 65. https://doi.org/10.1186/s13677-022-00338-x

Broda, K., Clark, K., Miller, R., & Russo, A. (2009). SAGE: A logical agent-based environment monitoring and control system. In Tscheligi, M., Ruyter, B., Markopoulus, P., Wichert, R., Mirlacher, T., Meschterjakov, A., & Reitberger, W. (Eds.), *Lecture notes in computer science: Vol. 5859. Ambient intelligence* (pp. 112–117). Springer. https://doi.org/10.1007/978-3-642-05408-2_14

Daoudagh, S., Marchetti, E., Calabrò, A., Ferrada, F., Oliveira, A. I., Barata, J., Peres, R., & Marques, F. (2022). An ontology-based solution for monitoring IoT cybersecurity. In Camarinha-Matos, L. M., Ribeiro, L., & Strous, L. (Eds.), *IFIP advances in information and communication technology: Vol. 665. Internet of things. IoT through a multi-disciplinary perspective* (pp. 158–176). Springer. https://doi.org/10.1007/978-3-031-18872-5_10

Dayarathna, M., & Perera, S. (2019). Recent advancements in event processing. *ACM computing surveys*, *51*(2), Article 33. https://doi.org/10.1145/3170432

Luckham, D. C. (1996). *Rapide: A language and toolset for simulation of distributed systems by partial orderings of events*. Stanford University. http://infolab.stanford.edu/pub/cstr/reports/csl/tr/96/705/CSL-TR-96-705.pdf

National Institute of Standards and Technology. (2020). *Security and privacy controls for information systems and organizations*. (NIST Special Publication 800-53, Revision 5). U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-53r5

National Institute of Standards and Technology. (2021). *Developing cyber-resilient systems: A systems security engineering approach*. (NIST Special Publication 800-160, Volume 2 Revision 1). U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-160v2r1

Patkos, T., Plexousakis, D., Chibani, A., & Amirat, Y. (2016). An event calculus production rule system for reasoning in dynamic and uncertain domains. *Theory and practice of logic programming*, *16*(3), 325–352. https://doi.org/10.1017/S1471068416000065

Schaaf, M., Grivas, S. G., Ackermann, D., Diekmann, A., Koschel, A., & Astrova, I. (2012). Semantic complex event processing. In Mastorakis, N., Mladenov, V., & Bojkovic, Z. (Eds.), *Proceedings of the 5th WSEAS congress on applied computing conference, and proceedings of the 1st international conference on biologically inspired computation* (pp. 38–43). World Scientific and Engineering Academy and Society. http://www.wseas.us/e-library/conferences/2012/Algarve/BICA/BICA-05.pdf

Skarlatidis, A., Paliouras, G., Artikis, A., & Vouros, G. A. (2015). Probabilistic event calculus for event recognition. *ACM transactions on computational logic*, *16*(2), Article 11. https://doi.org/10.1145/2699916

Teymourian, K., Streibel, O., Paschke, A., Alnemr, R., & Meinel, C. (2009). Towards semantic event-driven systems. In Al-Agha, K., Badra, M., & Newby, G. B. (Eds.), *2009 3rd international conference on new technologies, mobility and security* (pp. 347–352). Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/NTMS.2009.5384713

Vassiliades, A., Bassiliades, N., Gouidis F., & Patkos, T. (2020). A knowledge retrieval framework for household objects and actions with external knowledge. In Blomqvist, E., Groth, P., de Boer, V., Pellegrini, T., Alam, M., Käfer, T., Kieseberg, P., Kirrane, S., Meroño-Peñuela, A., Pandit, H. J. (Eds.), *Lecture notes in computer science: Vol. 12378. Semantic systems. In the era of knowledge graphs* (pp. 36–52). Springer. https://doi.org/10.1007/978-3-030-59833-4_3