Cyber-Physical Topology Language: Definition, Operations, and Application

Carmen Cheh, Gabriel A. Weaver, and William H. Sanders
Coordinated Science Laboratory, Electrical and Computer Engineering Dept., and Computer Science Dept.
University of Illinois at Urbana-Champaign, Urbana, IL 61801
Email: {cheh2, gweaver, whs}@illinois.edu

***Abstract*—Maintaining the resilience of a large-scale system requires an accurate view of the system's cyber and physical state. The ability to collect, organize, and analyze state central to a system's operation is thus important in today's environment, in which the number and sophistication of security attacks are increasing. Although a variety of "sensors" (e.g., Intrusion Detection Systems, log files, and physical sensors) are available to collect system state information, it's difficult for administrators to maintain and analyze the diversity of information needed to understand a system's security state. Therefore, we have developed the *Cyber-Physical Topology Language* (CPTL) to represent and reason about system security. CPTL combines ideas from graph theory and formal logics, and provides a framework to capture relationships among the diverse types of sensor information. In this paper, we formally define CPTL as well as operations on CPTL models that can be used to infer a system's security state. We then illustrate the use of CPTL in both the enterprise and electrical power domains and provide experimental results that illustrate the practicality of the approach.**

Keywords*—*Cyber-Physical Topology Language, graph theory, description logics, target system model, system state

I. Introduction

To be resilient to attacks, a system must be able to detect that its state has been corrupted and respond in a way that ensures proper operation. An accurate view of the system's cyber and physical state is thus key to achieving system resiliency. Views should be comprehensive; i.e., they integrate together information about state from diverse sensors into a coherent whole. However, the diversity of sensor data can be overwhelming, and when combined with a siloed approach to storage, makes it impossible to draw inferences from knowledge of the connections between the different types of sensor information. For example, automated tools such as *Intrusion Detection Systems* (IDSes) are used to reduce the burden on humans by sifting through data and presenting a condensed representation of the data. According to a 2012 report, even purpose-built IDSes, however, may detect just 5% of known intrusions against larger organizations and similarly, manual log review may detect only 8% of known intrusions [1].

To further illustrate the need, consider the huge amount of system data (one trillion events per day for an enterprise like HP [2]) that might be stored in a traditional database. Administrators can query the database for data of interest to obtain insights about the system's security state. The system state, however, relies not only on the state of individual entities in a system, but also on the relationships among those entities. One natural way to represent such relationships is by using graphs. There are currently many graph databases as well as query languages available for storage and querying of system data. Some of the graph databases support reasoning over the data in addition to traditional graph queries. We agree with that approach, and have explored more formally the combination of graphs and formal semantics to drive holistic analysis of a system's security state.

In particular, we have developed the *Cyber-Physical Topology Language* (CPTL), which organizes heterogeneous system data in a manner that facilitates useful analyses about the system's security state. We hypothesize that multiple views of the same system and operations upon those views will provide administrators with more insight into the system's security state, given the diversity of data. As argued earlier, a unified view of the system's security state is key to determining proper responses to maintain system resiliency. In this paper, we focus on two specific contributions. First, we define the CPTL data model as a way to organize and understand the relationships among diverse sensor data that have been collected. Second, we define a set of operations to process those views in order to obtain insights about the system's overall security state.

CPTL Data Model: A CPTL data model represents information about the target system using graphs and description logic. In that manner, it can integrate event information across a variety of architectural layers of the target system. CPTL can represent not only typical network and host-level information, but also user behavior, as we will see in Section V-D3. Furthermore, interactions between devices and the physical environment may also be represented and reasoned about using our framework, as shown in Section V-D1.

CPTL Operations: CPTL operations integrate graph-theoretic operations with semantics from description logics to derive meaning from a CPTL data model. We have developed the *CPTL View Analyzer* (CVA) tool that maintains instances of CPTL views and implements those CPTL operations. We illustrate the use of these operations in the context of a small electrical power substation and a small enterprise network. For the first example, we used two CPTL operations to drive analyses about the interaction between cyber and physical components of the substation. For the second example, we integrated information about printers and version control systems with user behavior to detect abnormalities in behavior on an enterprise network.

II. Related Work

We will now describe the existing work in representing system data and performing analyses on those data to provide

insights about a system's security state.

In 2013, the Cloud Security Alliance reported that an estimated 1 trillion events per day, or 12 million events per second, were produced by enterprises like HP [2]. There are multiple different ways to present that huge amount of data to users. We hypothesize that a better representation of the data would help to diversify the types of analyses possible.

State of the Practice: Most state-of-the-practice approaches use tables to represent data in a traditional database [3]. However, queries that relate multiple relationships among entities are difficult to perform under the traditional database approach because of the large overhead of joining multiple tables.

Instead of representing data as rows in tables, some approaches have moved towards representing data as a graph so as to expand the possible types of analyses that can be performed on the data. For example, the graph database Neo4j [4] has been utilized to detect bank, insurance, and e-commerce fraud by analyzing the connections made by different entities.

The Semantic Web is a framework for data sharing among organizations, and the standardized model for data exchange is the *Resource Description Framework* (RDF) [5]. RDF encodes data in the form of a subject-predicate-object triple that can be viewed as two vertices representing the subject and object and an edge between them representing the predicate. A collection of RDF triples can encode the information about a system and is represented as a labeled, directed multigraph.

Existing graph databases, such as GraphDB [6], AllegoGraph [7], Neo4j [8], and Stardog [9], support RDF triples, making them suitable for representing various types of system data. GraphDB, AllegoGraph, and Stardog support inference of new facts from the existing data, but only Stardog supports checking of constraints and updating of the knowledge base represented by the ontology. There are a number of query languages, e.g., SPARQL, that are designed for graph databases. However, most of these query languages are mainly focused on extracting graph patterns and do not involve reasoning [10]. The exception is SPARQL-DL [11], but it involves only fundamental semantic queries, like checking of subclass properties. So none of the query languages support analyses that combine both graph theory and logic.

State of the Art: State-of-the-art approaches also construct specific graphs for their own use cases or represent system data in terms of formal logics.

Joslyn et al. [12] represented IPFLOW data as a graph, and detected botnets and exfiltration through identification of graph patterns. Furthermore, Brdiczka et al. [13] extracted semantic entities (e.g., topics, keywords, users, and objects) to construct activity and topic graphs of user behavior in an organization. Patterns in graph structure and graph features are used to detect malicious insider behavior.

Data can also be represented in terms of formal logic. Some approaches [14], [15] represent data in an ontology and infer system state by checking the data automatically against a formal specification of the states of interest. KnowLang [16], [17] also uses ontologies to reason about the system state. In addition, KnowLang enforces policies and actions to be taken by the system. Similarly, ontologies were used by [18] and [19] to specify attack signatures and collate data from sensors. CHAMPION [20] is a framework that uses a semantic Knowledge Base and a hierarchy of reasoners to integrate diverse sources of information to achieve situational awareness. It was applied to detect insiders in enterprise systems [21] and anomalies in power consumption [22]. Morin et al. [23] use first-order logic to describe information about certain aspects of a system (hosts, subnets, gateways, and services), vulnerabilities, attack classes, IDSes, and events. The data can be queried for interactions among IDS alerts, events, and the system. Temporal logic was used by Viswanathan et al. [24] to describe behavior models of network traffic.

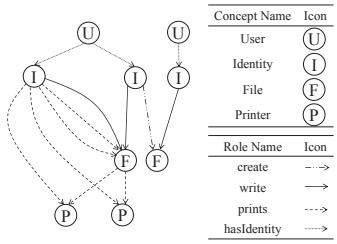


Fig. 1. A directed multigraph $G(V, E, h)$ representing employees' accesses to files and printing activity in an enterprise system.

Our Approach: Our approach merges graph theory and ontologies such that we can utilize the advantages of both techniques. Analyses of data can utilize graph theory, and ontologies lend themselves to support heterogeneous information.

III. Cyber-Physical Topology Language for System Modeling

In Section I, we explained the need for a better representation of a target system. We will now define different views of the target system that represent the state of the target system. A *view* of the target system is formalized using a CPTL data model.

A **CPTL data model** $(G, \mathcal{K})_{\mathcal{I}}$ consists of a directed multigraph $G = (V, E, h)$, an ontology $\mathcal{K}$ expressed in a description logic (DL), and an interpretation $\mathcal{I}$ of G. In the following sections, we will describe each of those three components.

A. Directed Multigraph

The CPTL data model uses a **directed multigraph** $G = (V, E, h)$, in which V is a set of vertices, E is a set of edges, and $h : E \rightarrow V \times V$ is a function that maps each edge to an ordered pair of vertices. For example, Fig. 1 represents a small portion of an enterprise system.

The vertices V and edges E have sets of **vertex attributes** and **edge attributes**, respectively. We define the vertex attributes mapping of vertex $v \in V$ as $A_V(v)$ and the edge attributes mapping of edge $e \in E$ as $A_E(e)$. For example, a vertex representing a **Printer** has a vertex attribute **location** with value *#2N* (second floor, north wing). An edge representing a **Print** action has an edge attribute **numPages** with value 10.[1]

[1]Such details are not shown in Fig. 1 because of space constraints.

B. Ontologies and Description Logics

We use ontologies to specify the semantics of assets (vertices V) and their interactions (edges E). The semantics is unambiguous for humans and machine-actionable for algorithms. Ontologies are expressed using a *description logic*.[2]

Description Logics: A **description logic** (DL) is a knowledge representation language that is a tuple of four sets $DL(\mathbf{N_I}, \mathbf{N_C}, \mathbf{N_R}, \mathbf{N_F})$:

- A set of individual names $\mathbf{N_I}$ that describe the identifiers of individuals (e.g., *Alice*, *Document1.txt*),
- A set of concept names $\mathbf{N_C}$ that define classes of individuals (e.g., **Printer**, **User**, **File**, **Identity**),
- A set of role names $\mathbf{N_R}$ that describe the set of possible relations among individuals (e.g., **create**, **write**, **hasIdentity**, **prints**), and
- A set of feature names $\mathbf{N_F}$ that describe the set of possible values in a particular concrete domain $\mathcal{D} = (\Delta^{\mathcal{D}}, \mathcal{P}^{\mathcal{D}})$. $\Delta^{\mathcal{D}}$ is a set of individuals, and $\mathcal{P}^{\mathcal{D}}$ is a set of predicate names such as strings and integers (e.g., **location**, **numPages**).

Additional concept and role names can be defined within DL.

Ontology: An **ontology** is a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. The set of terminological (*TBox*) axioms ($\mathcal{T}$) describes relationships among concepts. The set of assertional (*ABox*) axioms ($\mathcal{A}$) describes the semantics of individuals as well as relationships among those individuals [26].

For example, an *ABox* axiom for a vertex in the graph in Fig. 1 would assert whether that vertex name (or *individual*, in DL terminology) is an instance of the **User**, **Identity**, **Printer**, or **File** concept. Another *ABox* axiom for an edge in the graph in Fig. 1 would assert whether that edge represented an action that a user can perform (**create**, **write**, or **prints**), or it represented a user's possession of a credential (**hasIdentity**).

C. Interpretation of Graphs

We want to present different views of a target system for users in different roles, and for algorithms interested in different aspects of a system. DL uses an *interpretation* to map components of an ontology to a graph. An **interpretation** $\mathcal{I}$ maps an ontology $\mathcal{K}$ to a domain of discourse, $\Delta^{\mathcal{I}}$, via an interpretation function, $\cdot^{\mathcal{I}}$, that is defined as follows:

- For concepts $A \in \mathbf{N_C}$, $\cdot^{\mathcal{I}}(A) = A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.
- For roles $r \in \mathbf{N_R}$, $\cdot^{\mathcal{I}}(r) = r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.
- For individuals $a \in \mathbf{N_I}$, $\cdot^{\mathcal{I}}(a) = a^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.
- For features $f \in \mathbf{N_F}$, $\cdot^{\mathcal{I}}(f) = f^{\mathcal{I}}$ where $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow \bigcup_{1 \leq i \leq n} \Delta^{\mathcal{D}_i}$.

The interpretation $\mathcal{I}$ maps the ontology to G, so $\Delta^{\mathcal{I}} = V(G)$. In addition, $r \in \mathbf{N_R}$ maps to $r^{\mathcal{I}} \subseteq E(G)$.

[2]The enterprise use case examples are defined in the ontology available in [25].

IV. Operations in CPTL

We can perform traditional graph operations and formal reasoning on an instance of a CPTL data model. With the combination of graphs and semantics, however, we can introduce operations that can simultaneously involve both graphical and semantic changes.

An operator is characterized by the (1) topological, (2) attribute, and (3) semantic changes applied to the data model instances. Topological changes involve modification of the structure of the graph G; attribute changes involve modification of the vertex and edge attributes; and semantic changes involve modification of the ontology $\mathcal{K}$ and/or the interpretation $\mathcal{I}$. Table I shows the changes made by the operators that we will introduce in the following subsection.

TABLE I. The check marks indicate the set of changes associated with each operation.

Operation	Component of Data Model Affected		
	Topological	Semantic	Attribute
Abstract		✓	
Contract	✓	✓	✓
Join	✓	✓	✓

A. Unary Operations

Unary operations take in a single CPTL data model instance and output a new CPTL data model instance. We define two unary operations, **Abstract** and **Contract**.

1) Abstract: A view can contain extremely detailed information about a system. Sometimes, it is necessary to operate on a more coarse-grained view, for example, to preserve privacy when publishing the view or to conduct high-level analyses. The **Abstract** operation extracts a higher-level description of a graph by modifying the interpretation.

Definition: The interpretation is modified by replacing the concept (C), role (R), and feature names (F) with more general concept ($C \sqsubseteq C'$), role ($R \sqsubseteq R'$), and feature names ($F \sqsubseteq F'$).[3] Those replacements are called the *ancestors* of the original names because the replacements are located higher than the original names in the hierarchy defined in the ontology. The level of generality is specified with respect to the level in the hierarchy tree defined in the ontology.

Example: In an electrical power substation, the operation of a breaker is controlled by one or more relays. We may want to perform analyses on the connections between different brands of relays and breakers. In Section V-D2, we calculate the diversity of relays controlling a breaker using **Abstract** to remove low-level details of the brand number of the relay.

2) Contract: We can derive higher-level features of a view by aggregating related entities. The **Contract** operation extracts higher-level features of the graph using the graph-theoretic vertex contraction operation and enforcing a consistent semantics on the resultant CPTL data model instance.

Definition: Given a set of vertices $V_C = \{v_1 \dots v_n\}$ in the view instance, **Contract** merges the set into a single vertex (or supernode) v'. That supernode v', since it represents

[3]The notation $A \sqsubseteq B$ means that B subsumes A, i.e., all elements of A are elements of B.

a set of vertices, inherits a summary of their properties $f(A_V(v_1), \ldots, A_V(v_n))$ and is adjacent to all vertices incident to/from the set ($\{x | e_i(x, v_i) \in E(G)$ or $e_i(v_i, x) \in E(G), x \in V(G) - V_C\}$). The concept name of the supernode and the role names of the edges incident to it are determined by a set of *TBox* axioms.

Example: Studies have shown that a significant number of insiders exfiltrate data physically through stealing or printing of documents, and it has been suggested that organizations should monitor the printing behavior of employees [27], [28]. In an enterprise system, employees may use printers other than their usual printer choices; such anomalous behavior may signify data exfiltration. However, that assumption could be false, because an employee may use another printer for many reasons (e.g., the employee's usual printer might be malfunctioning). Our hypothesis is that if a person prints a file F to printer P_1 and then soon after prints F to another printer, P_2, then P_1 was unavailable at that time. We then use **Contract** to infer the state of the printers.

For each printer, we contract the set of files sent to it and the users who use it. If the supernodes share common (user, file) pairings, then a user must have printed a file to two printers. That knowledge is represented by the presence of an edge with a particular role name. We use the aggregated edge attributes to infer the sequence of print events and thus identify the malfunctioning printer based on the hypothesis. Now, if a user prints to a second printer at a time when the default printer is malfunctioning, then we interpret that behavior as reasonable. However, if the user prints to a second printer when the default printer is working, we interpret that behavior as suspicious.

B. Binary Operations

Binary operations take in two CPTL data model instances and output a single new CPTL data model instance. We define a binary operation **Join**.

1) Join: A single view by itself can yield important details. However, combining two related views allows us to study the interactions between the entities in the views. The **Join** operation combines two views to produce a single comprehensive view by merging related entities and adding edges representing the relationship between the views.

Definition: **Join** merges vertices and edges that refer to the same individual, where the notion of "same individual" is given by a function of vertex attributes and concept name $g(A_V(v), C(v))$. The merged vertices and edges inherit the properties of their individual components. Finally, edges between vertices in the two view instances are added based on a specification of the relationship between the views.

Example: In Section V-D1, we show how to use **Join** to relate two views of an electrical power substation.

C. Checking Consistency of Operations

After an operation has been applied to a CPTL data model instance, we run a consistency check on the result, using a *reasoner* to ensure that the interpretation of the graph satisfies the ontology. That ensures that the semantics of the CPTL data model instance is preserved as a result of the operations.

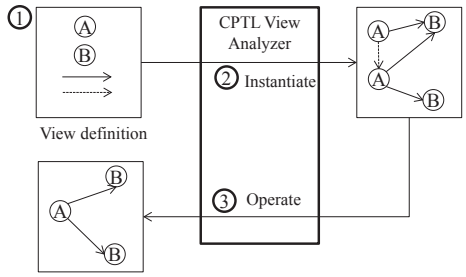


Fig. 2. Workflow of CVA. The numbers in the circle represent the order in which the steps are performed. First, practitioners define the CPTL view. Second, CVA instantiates a view with system data based on the definition. Finally, CVA operates on the view instance to produce a new view instance.

A **reasoner** is an engine that infers new facts based on the asserted axioms. In particular, we utilize three reasoning tasks: *consistency*, *retrieval*, and *realizaton*. The **consistency** reasoning task is used for verifying that the data represented as *ABox* axioms are consistent with respect to the *TBox* axioms. The **retrieval** reasoning task is used by **Abstract** to determine the ancestor concept, role, or feature names for a given vertex or edge. The **realization** reasoning task is used by **Contract** to determine the concept and role names of the new supernode and its edges based on *TBox* axioms.

V. Application of CPTL

In this section, we demonstrate how we can specify different views, instantiate those views with system data, and analyze those views using operations discussed in Section IV. We describe our CVA tool that maintains instances of views and implements CPTL operations. In particular, we used CVA (1) to check for design issues in an electrical power substation, and (2) to identify suspicious user behavior in an enterprise system. We now describe the overall three-step workflow shown in Fig. 2 to organize and operate upon diverse system data using CPTL.

A. General Workflow

Given a target system, we want to use CPTL to represent information about that system and apply CPTL operations on that representation to reason about the security state.

First, we need to specify how we want to represent system information using a CPTL data model. We have to formally indicate (1) the vertices and edges in the directed multigraph, (2) the vertex and edge attributes, and (3) the ontology and the mapping of the elements of the ontology to the directed multigraph. This definition is similar to a database schema and describes how the view should be instantiated with data.

Second, we create an empty view and instantiate it with data according to the view definition. This can be done with CVA in an offline manner using data files or in an online manner such that the data are constantly being fed into CVA. The view instance can then be validated for consistency with respect to the ontology in terms of syntax and semantics.

Finally, we conduct analyses on the view instance using CPTL operations by specifying the parameters that feed into the operation. CVA runs the operations over the view instance to obtain new view instances, which are then used to infer properties of the system state.

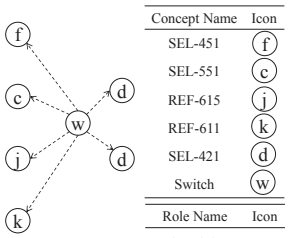


Fig. 3. The substation network view instance.

B. View Definition

We specify how the information is represented in a CPTL data model in terms of the mathematical constructs introduced in Section III.

Bulk Electric System: In this use case, we have a model of a substation based on the Ogdenville substation, which is part of an 8-substation model produced by the CyPSA project [29]. The substation network includes one switch and several relays. Protection relays, such as overcurrent relays and distance relays, protect equipment in the substation yard against short circuits caused by faults. Relays will open and close the breaker in an attempt to clear the fault. Different relays monitor different areas of the electrical power substation; overcurrent relays focus on the current in the substation, whereas distance relays focus on power lines between substations.

We define two views of the Ogdenville substation: a *substation network view* and a *node-breaker view*.

The substation network, shown in Fig. 3, is an instance of a **substation network view** defined as $G = (V, E, h)$, in which $V[G] = \cdot^{\mathcal{I}}(Switch \cup Relay)$, $h(E[G]) = \cdot^{\mathcal{I}}(hasLink)$. This view describes the cyber components of the substation.[4]

The substation yard, shown in the left box of Fig. 4, is an instance of the **node-breaker view** defined as $G = (V, E, h)$, in which $V[G] = \cdot^{\mathcal{I}}(Breaker \cup Load \cup Generator \cup Bus \cup Transformer)$, $h(E[G]) = \cdot^{\mathcal{I}}(hasLine)$. This view describes the physical components of the substation.

Enterprise System: In this use case, we focus on commits to a version control system (*git*) repository, and on print jobs of employees in a research group over a period of one year. The *git* repository is a shared research resource that contains papers, reports, source code, Web documents, spreadsheets, and presentations. The *git* repository is accessed by 5 employees, and each employee can have multiple credentials. Employees print documents to printers that reside in buildings.

We define two views of the enterprise system: a *write file view* and a *print job view*.

A **write file view** is defined as $G = (V, E, h)$, in which $V[G] = \cdot^{\mathcal{I}}(User \cup Identity \cup File)$, $h(E[G]) = \cdot^{\mathcal{I}}(hasIdentity \cup write)$. This view describes the writes made by employees to files in *git*.

A **print job view** is defined as $G = (V, E, h)$, in which $V[G] = \cdot^{\mathcal{I}}(User \cup Identity \cup File \cup Printer)$, $h(E[G]) = \cdot^{\mathcal{I}}(hasIdentity \cup prints)$. This view describes the print jobs of employees.

[4]For illustrative purposes, Fig. 3 has more diverse types of devices than what is typically found in substations.

C. View Generation and Validation

Given a view definition and system data, CVA creates an empty view instance and populates it with the system data so that it conforms to the definition. The view instance is then serialized to one of several different formats (including JSON Node-Link and RDF Turtle). It is syntactically validated using a context-free grammar and semantically validated using a reasoner.

Bulk Electric System: The dataset is an architectural diagram that describes the connections between the cyber and physical components within the substation. Because of the simplicity of the diagram, we manually construct a view instance.

For the *substation network view*, we perform the following actions for each element in the diagram:

- *Switches:* Creates a vertex of type **Switch**.
- *Relays:* Creates a vertex whose type is the brand number of the relay.
- *Links between components:* Creates an edge of type **hasLink** between the corresponding vertices in the view.

For the *node-breaker view*, we perform the following actions for each element in the diagram:

- *Breakers:* Creates a vertex of type **Breaker**.
- *Loads:* Creates a vertex of type **Load**.
- *Generators:* Creates a vertex of type **Generator**.
- *Buses:* Creates a vertex of type **Bus**.
- *Transformers:* Creates a vertex of type **Transformer**.
- *Lines between components:* Creates an edge of type **hasLine**.

Each of the vertices has a unique individual name so as to identify the respective component in the architectural diagram. Finally, the views are validated against the ontologies associated with the CyPSA project [29].

Enterprise System: The dataset for the enterprise system consists of commits to the *git* repository obtained through *git* `log` commands, and data about print jobs collected from a Web interface. The *git* logs contain the commit number, list of files modified, number of insertions and deletions made to the file in terms of lines, timestamp of commit, employee credentials, and employee name. The print job data consist of the employee credentials, employee name, file name, timestamp of print, number of pages printed, and printer location.

For the *write file view*, the parser performs the following actions for each commit:

- *Credentials:* Creates a vertex of type **Identity** with an **email** attribute having the value of the credential.
- *Employee name:* Creates a vertex of type **Person** with a **name** attribute having the employee's name. Also creates an edge of type **hasIdentity** from this vertex to the **Identity** vertex mentioned above.

- *File:* Creates a vertex of type **File** with a **name** attribute having the file's name and a **type** attribute having the file's extension. Also creates an edge of type **write** from the **Identity** vertex mentioned above to this vertex. The edge has a **numIns** attribute having the number of insertions made to the file, a **numDel** attribute having the number of deletions made to the file, and a **timestamp** attribute having the timestamp of the commit.

For the *print job view*, the parser performs the following actions for each print job:

- Same as above for credentials and employee name.
- *File:* Creates a vertex of type **File** with a **name** attribute having the file's name and a **type** attribute having the file's extension. Also creates an edge from the **Identity** vertex mentioned above to this vertex of type **prints** and with a **numPages** attribute having the number of pages printed and a **timestamp** attribute having the timestamp of the print job.
- *Printer:* Creates a vertex of type **Printer** with a **name** attribute having the printer's name and a **location** attribute having the printer's location. Also creates an edge of type **prints** from the **Identity** vertex mentioned above to this vertex and a similar edge of type **prints** from the **File** vertex mentioned above to this vertex. Both edges have a **numPages** attribute, whose value is the number of pages printed, and a **timestamp** attribute, whose value is the timestamp of the print job.

Just as in the bulk electric system use case, each of the vertices has a unique individual name so as to identify the respective entities. Finally, the views are validated against the associated ontologies.

D. Analyses

We now describe the analyses that we conducted on the views. For the bulk electric system, we performed a simple check of the design of cyber and physical components within a substation. For the enterprise system, we detected abnormal user behavior when an employee wrote an abnormal amount of modifications to a file or printed to a distant printer.

1) Join: The substation network is designed to protect the substation yard. We run **Join** over the substation network view and node-breaker view instances. Since we know that the two view instances do not share the same individuals, we specify that no vertices and edges be merged. From the architectural diagram, we know of the connections between the relays and breakers that control breaker operation. So we can specify the edges to be added between relays and breakers. Fig. 4 illustrates the resulting view instance with edges from relays to breakers. **Join** can be used to generate composite views on a per-need basis, e.g., if cyber-physical connections are rewired.

2) Abstract: In the composite view instance, we see that some breakers are connected to one or more different brands of relays (SEL and ABB). One simple analysis that uses **Abstract** involves diversity of relays controlling a breaker. We hypothesize that the more diverse the brands of relay, the more reliable the operation of the breaker to which those relays are connected. This diversity metric can inform the practitioner's design decisions in securing the system. Whether the hypothesis holds true remains to be seen; nonetheless, CPTL operations provide a simple way to calculate this metric. Thus, we use **Abstract** to mask the details of the brand number of the relay.

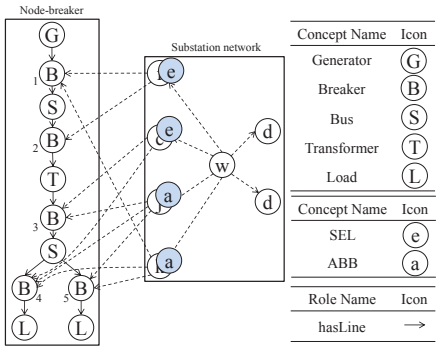


Fig. 4. A view of cyber and physical components within the substation. The node-breaker view instance is shown in the left box; the substation network view instance is shown in the right box. The result of the **Join** is the edges between the two boxes. The result of the **Abstract** is the shaded circles that replace the original non-shaded circles. The breakers are numbered 1 to 5.

We run **Abstract** over the composite view instance that resulted from **Join**. We specify the granularity level to be {*Switch, SEL, ABB*}, as shown in Fig. 5. The resulting view instance is given in Fig. 4, where the shaded circles represent the abstracted vertices.

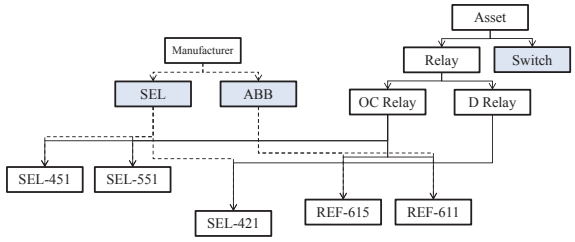


Fig. 5. The concept hierarchy of the ontology. The granularity level specified for **Abstract** is reflected by the shaded boxes.

Using the view instance that resulted from the running of **Abstract**, we used a simple metric $\frac{\text{Number of brands}}{\text{Total number of relays}}$ to calculate the diversity for each breaker in terms of the set of relays connected to it. So the diversity (or, correspondingly, reliability) of breakers 1, 2, and 3 is 1; that of breaker 4 is $\frac{2}{3}$; and that of breaker 5 is $\frac{1}{2}$. So breakers 4 and 5 are the least reliable with respect to diversity. A practitioner can decide to change the links to those breakers such that both relay brands are connected to breakers 4 and 5, so that reliability is increased. In addition, the result of **Abstract** can be used to share architecture diagrams among utilities and other organizations at a higher level of granularity so that only less detailed information is shared.

3) Contract: Suspicious user behavior may be the result of insider attacks. Insider attacks result in greater financial damage [27], [28] and pose a greater threat to systems than external attacks do, because of the intimate knowledge of the system and its defenses possessed by insiders [30].

We can identify suspicious user behavior by looking for

deviations from a user's normal behavior. So we develop baselines of each employee's writes to files. An anomalous amount of writes to files could be considered suspicious. We also look at situations in which employees print to distant printers; that can be considered suspicious if a nearby printer is working.

Application to Write File View: Our aim is to identify as suspicious behavior an anomalous number of modifications made by a person to a file.

We want to express a baseline that limits a person's writes to within three standard deviations of the mean number of modifications made to a file per person. Thus, we perform two contractions: one to obtain the average number of modifications, and one to obtain the standard deviation.

We want to collapse the distinction between people and their credentials, so we specify the sets of vertices to be contracted as $V_C = \{(P_k, I_1, \ldots, I_s) | P_k \in \texttt{User}^{\mathcal{I}}, \exists e_r(P_k, I_q) \in E(G), e_r \in \texttt{hasIdentity}^{\mathcal{I}}$ for $q \in [1, s]\}$. Each set consists of a **User** (represented as P) and his or her associated credentials (represented as I). After contraction is performed, each set will be represented as a single supernode. In our example, we are developing baselines based on a person's credentials. We can easily extend this to baselines of groups of people, or roles in an organization, by selecting the appropriate vertices for contraction. We can also contract *similar* files together based on shared attributes, e.g., those generated by compilation, like the files generated by LaTeX.

We also specify that edges that are instances of the **write** role should be grouped together and that the edge attributes of the resulting **write** edge are the average numbers of insertions and deletions made by a person. The result of the first contraction is a view instance called `avgV`.

We perform a similar second contraction in which we instead specify the edge attributes as the standard deviation of the number of insertions and deletions made by a person. The result of the second contraction is a view instance called `devV`.

We can create a variety of rules using the results of the two contraction operations. In our example, we will assert that P_k can make between $(\texttt{numIns}_{\text{avgV}} + 3(\texttt{numIns}_{\text{devV}}))$ and $(\texttt{numIns}_{\text{avgV}} - 3(\texttt{numIns}_{\text{devV}}))$ insertions, and between $(\texttt{numDel}_{\text{avgV}} + 3(\texttt{numDel}_{\text{devV}}))$ and $(\texttt{numDel}_{\text{avgV}} - 3(\texttt{numDel}_{\text{devV}}))$ deletions.

The baseline is implemented as a class $\texttt{P}_\texttt{k}\texttt{WriteF}_\texttt{i}$ in the ontology that describes the baseline of modifications that person P_k makes to file F_i.

After CVA updates the view instance automatically with write events, we run a reasoner to check whether the ontology is satisfiable (or, more specifically, whether the baselines are satisfiable). If not, then there was an anomalous number of modifications, which is indicative of suspicious user behavior.

Application to Print Job View: Our aim is to identify abnormal printing behavior by noticing print events that involve use of a printer other than the user's default printer choice, when the unusual use cannot be explained by printer state (i.e., malfunctioning).

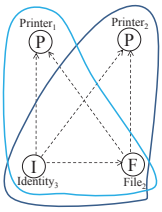


Fig. 6. The vertices in each bounded region are selected for contraction.

First, we infer each person's default choice of printer by comparing the frequencies with which the person's print jobs are sent to various printers.

Next, we want to infer the state of a printer based on our hypothesis mentioned in Section IV. We can easily restrict our hypothesis further by requiring at least n people to have printed to P_1 and subsequently to another printer. In our example, however, we demonstrate only the more general hypothesis. To illustrate the application of **Contract**, we look at a small snippet of the instance of the print job view given in Fig. 6.

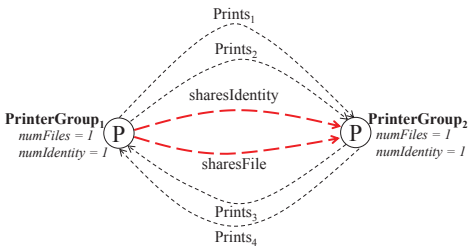
We specify the concept name of the supernodes as an axiom $(= 1)$**numFiles**. $\geq_0 \sqcap (= 1)$**numIdentity**. $\geq_0$. CVA will then infer the concept name of the supernode to be **PrinterGroup**, because the definition of that concept is exactly the axiom described above.

In the implementation of **Contract**, if the vertices selected from the two groups for contraction overlap, we add an edge of role name **shares**<C> between the two groups, where C is a template for the concept name of the original vertex. Since the two groups of vertices in Fig. 6 overlap, both **sharesFile** and **sharesPrinter** exist. We know that if both of those edges exist between two printer groups, then it is evidence that two printers have been used by the same user to print the same file. So, we specify that edges that are instances of **sharesFile** and **sharesIdentity** should be grouped, and the role name of the resulting edge is given by the axiom **sharesFile** $\sqcap$ **sharesPrinter**.

Then, CVA will infer the role name of the resulting edge to be **changePrinter** when both **sharesFile** and **sharesIdentity** edges exist; if only **sharesFile** or only **sharesIdentity** exists, then the role name of the resulting edge takes on the existing edge's role name. In our example, the role name of the resulting edge will be inferred to be **changePrinter**.

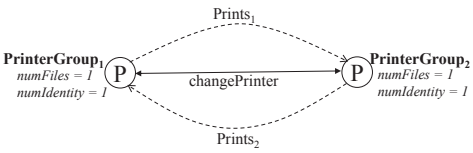
At the moment, we can only infer from the **changePrinter** edge that one of the printers was unavailable. We can determine the order of the printing events by looking at the times of the first print jobs that were sent to the two printer groups. Then, we can identify the printer that was unavailable as the head vertex of the edge with the lowest time. So, we additionally specify that edges that are instances of **prints** should be grouped, and that the **timestamp** attributes of the **prints** edges are the minimum **timestamp** values in the group. If we used the stricter hypothesis mentioned earlier, we could instead determine the times of the last print jobs so as to infer which printer was the last one functioning.

As a result of the contraction, the $\textbf{Prints}_1$ edge from $\textbf{PrinterGroup}_1$ to $\textbf{PrinterGroup}_2$ has a lower timestamp than the $\textbf{Prints}_3$ edge from $\textbf{PrinterGroup}_2$ to $\textbf{PrinterGroup}_1$. Therefore, we deduce that a file was printed to $\textbf{PrinterGroup}_2$



	$Prints_1$	$Prints_2$	$Prints_3$	$Prints_4$
Timestamp	1PM	1PM	2PM	2PM

(a)



	$Prints_1$	$Prints_2$
Timestamp	1PM	2PM

(b)

Fig. 7. The edges are grouped based on their role names. Partial attribute values for each **Prints** edge are given in the table at the bottom of (a). (We show only the hour of the timestamp in the table.) (b) shows the resultant graph after the edge attributes have been computed for the sets of edges. The table shows partial attribute values of the final edge (hours only).

and subsequently to $\textbf{PrinterGroup}_1$. Thus, we can infer that $\textbf{Printer}_2$ is likely not working.

The inferred set of printers that a user is likely to use for printing is implemented as a class $\texttt{S}_\texttt{k}\texttt{Print}$ in the ontology in which S_k is the user.

After the view instance has been updated automatically with print events through CVA, we run a reasoner to check whether the ontology is satisfiable. If not, then there has been abnormal user behavior in that the user printed to a distant printer although the default printer was functioning.

VI. Evaluation

In this section, we evaluate the performance of a CPTL data model instance, CPTL operations, and the application of those operations in the context of the two use cases.

The CPTL data model and operations were implemented in Java. The ontology is expressed using the *OWL 2 Web Ontology Language* (OWL 2) serialized to an RDF format. All experiments were conducted on a Windows 7 machine with a 2.7 GHz Intel CPU core and 4 GB of RAM.

A. Implementation Performance

1) CPTL Model: We measured the size of the instance of each view that we defined in the electrical power substation use case and the enterprise system use case. The size of the view instance is defined in terms of the size of the graph (number of vertices and edges) as well as the size of the ontology (number of concepts, roles, and RDF triples). Table II shows the resulting size of each view instance.

2) Operations: We evaluate the average running time of each operation that we defined in Section IV.

TABLE II. The size of the instance in terms of the graph and ontology representation for each view.

View	Size of CPTL Data Model				
	Graph		Ontology		
	Vertices	Edges	Concepts	Roles	Triples
Bulk electric system					
Node-breaker	13	12	7	1	39
Substation network	6	5	13	1	19
Composite	19	23	20	2	80
Enterprise system					
Print job	956	3,620	9	3	45,242
Write file	1,805	318	7	3	4,733

a) Abstract: **Abstract** operates on the vertices and edges of a view, so we evaluate the running time of the operation **Abstract** with respect to the number of vertices and edges in the view instance. Since the power substation has a small, fixed number of vertices and edges, we use a subset of the enterprise system example and monotonically increase the size of the subset in terms of vertices and edges. Fig. 8(a) shows that the running time is linear in the number of vertices and edges.

b) Contract: In our examples, the selected number of vertices for **Contract** ranged between 8 and 35. Thus, the running time of **Contract** was more dependent on the number of selected edges, which ranged between 0 and 960. We evaluated the running time of **Contract** with respect to the number of edges in the view instance. The time complexity of **Contract** is on the order of $p \log p$ with respect to the number of selected edges, p, which tallies with Fig. 8(b).[5]

c) Join: **Join** operates on the vertices and edges in the view instance, so we evaluate the running time of **Join** with respect to the number of vertices and edges in the view instance. Since the power substation has a small, fixed number of vertices and edges, we use a subset of the enterprise system example and monotonically increase the size of the subset in terms of vertices and edges. Fig. 8(d) shows that the running time is linear in the number of vertices and edges.

d) Checking Consistency of Operations: The running time of the HermiT reasoner [31] to perform verification is on the order of a second-order polynomial, as shown in Fig. 8(e).

B. Enterprise Use Case Performance

In this section, we evaluate the performance of CPTL in the context of the detectors that we implemented in Section V.

1) Write File View: In Section V-D3, we described an approach to flag abnormal writes to files on a per-person basis.

In our dataset, we assume that all writes represent normal user behavior. So we designed a procedure to generate synthetic test data that simulate abnormal user behavior. First, we randomly chose a user-file pair and injected 10 abnormal writes over a period of 10 weeks. Five user-file pairs were chosen, giving rise to 50 abnormal writes in total. We limited the files to *.tex* and *.pdf* files, because our repository consists mainly of these two file types. We constructed the abnormal writes as follows. For each selected user and file (U_1-F_1) pair, we randomly chose another file, say F_2, that had the same extension as the selected file F_1. Then, we randomly selected a write made to F_2 and noted down the number of modifications

[5]More details are provided in [25].

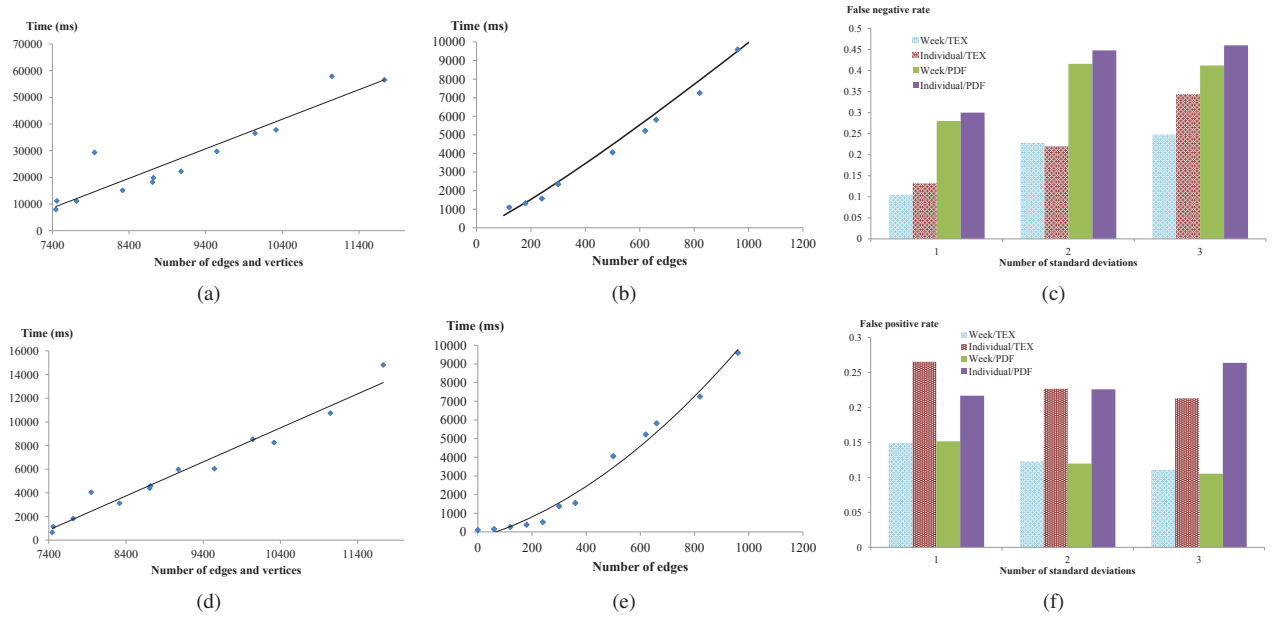


Fig. 8. (a) The time taken for **Abstract** vs. number of vertices and edges in view instance. (b) The time taken for **Contract** vs. number of edges incident to selected vertices for contraction. (c) The false negative rate in detecting anomalous writes. (d) The time taken for **Join** vs. number of edges and vertices in view instance. (e) The time taken for verifying validity of composite write file and print job view vs. number of edges in the view instance. (f) The false positive rate in detecting anomalous writes.

made to F_2. That number of modifications was used as the value for the abnormal write for the U_1-F_1 pair.

We modified the number of standard deviations in the baselines to be either more specific or more general. If the number of standard deviations is higher, the range of possible modifications is larger, and thus the baseline is more general.

We ran the procedure five times and obtained the average false positive and false negative rates.

Fig. 8(c) shows that as the baseline is made more specific, the false negative rate decreases, and therefore more abnormal writes are caught. The *.tex* files' abnormal modifications were caught more often than the *.pdf* modifications because of the *.pdf* files' higher variance in modifications. In general, the weekly update rate outperformed the individual write update rate, which indicates that the modifications were relatively steady over a week but varied greatly from write to write.

On the other hand, Fig. 8(f) shows that as the baseline was made more specific, the false positive rate generally increased. That trend is typical of approaches that use baselining to detect abnormal activity. However, the false positive rates across all configurations are within 5% of each other, which implies that we should focus on the false negative rate when choosing the specificity of the baseline. We can see that the weekly update rate greatly outperforms the individual write update rate in terms of false positives. Therefore, the best baseline would be a weekly update rate with a standard deviation of 1.

2) Print File View: In Section V-D3, we described an approach to identifying abnormal printing behavior that is based on a clearly defined specification. So any print jobs that do not fulfill the specification are caught by the axiom. Thus, we discuss only false positives. There were 20 violations from a total of 3,594 print jobs. Out of the 20 violations, 14 were print jobs that were sent to a printer in a different building from the default printer, from a machine that was located close to the printer used. That evidence indicates that we could reduce the number of false positives if we had more information (i.e., machine location). The other six violations were print jobs that were sent to a neighboring printer close to the default printer even though the default printer was inferred to be working. In the six violations we found, the employee happened to find that the default printer was not working and subsequently printed to the neighboring printer. Thus, there was no occurrence of a file's being printed to both printers. However, we only infer the printer state based on our *hypothesis* of human behavior. In fact, someone may print a file to two different printers for legitimate reasons. For example, a secretary could want to save time by printing multiple copies of a report to two different printers. We can corroborate the results of our hypothesis with printer logs that indicate whether a printer is working.

VII. Future Work

We discuss areas for future work that include extending CPTL and applying CPTL to other research topics.

First, this paper introduces the use of CPTL in both static and dynamic fashions. Our aim is for CPTL, in an ongoing manner, to analyze the security state of a system. As data flow in, we update the view instance, and after a set period of time, we re-run the analyses on the view instance. However, our analyses could be slowly poisoned by malicious entities to gradually accept abnormal behavior [32]. This is an ongoing research area, and we want to further explore this avenue and apply it to the way in which we are using CPTL.

Second, we have used DL in the definition of a CPTL data model to assign semantics to the graph in a human-readable and machine-interpretable way. Although DL is expressive for the needs that we have described here, it also has its limitations. For example, DL alone is not sufficient to express mathematical formulae. On the other hand, the *Semantic Web*

Rule Language (SWRL) [33] is able to handle mathematical formulas as well as composition of relations.

Third, we do realize that using CPTL is not a simple task and that a certain amount of expertise and familiarity is needed to specify the views and parameters for operations. The reason is that there is a typical trade-off between expressivity and usability. However, we believe that with the proper training and validation of the views and operations, practitioners would be able to use CPTL for their systems.

Finally, we want to extend the application of CPTL to other areas, such as access control. Role-based access control and attribute-based access control have both benefited from the use of logics, and CPTL is a natural extension to that framework. We can use CPTL operations to analyze the system and continuously update the policies at runtime.

VIII. Conclusion

In this paper, we explained the need for a formalism that describes views that maintains information about the security state of a target system. We introduced the Cyber-Physical Topology Language as a formal specification of a target system and defined operations on views that provide insight about the security state of the target system.

To demonstrate the utility of CPTL, we applied our language to both the electrical power grid and enterprise settings. In the electrical power grid setting, we used CPTL to perform design checks of the system. In the enterprise setting, we profiled user behavior to identify abnormal behavior. The results show that CPTL presents an overarching approach to presenting system data at a variety of levels of abstraction.

References

[1] J. Flynn. (2012) Intrusion along the kill chain. [Online]. Available: https://media.blackhat.com/bh-us-12/Briefings/Flynn/bh-us-12-Flynn-intrusion-along-the-kill-chain-WP.pdf

[2] Cloud Security Alliance. (2013) Big data analytics for security intelligence. https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf.

[3] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, "Analyzing system logs: A new view of what's important," in *Proc. 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007, pp. 6:1–6:7.

[4] G. Sadowski and P. Rathie. (2015) Detect fraud in real time with graph databases. [Online]. Available: http://neo4j.com/resources/wp-fraud-detection/

[5] W3C Recommendation. (2014) Resource Description Framework (RDF). [Online]. Available: http://www.w3.org/RDF/

[6] Ontotext. (2015) Ontotext GraphDB TM. [Online]. Available: http://ontotext.com/products/ontotext-graphdb-owlim-new-2/

[7] Franz Inc. (2015) AllegoGraph. [Online]. Available: http://franz.com/agraph/allegrograph/

[8] neo4j. (2015) Neo4j. [Online]. Available: http://neo4j.com

[9] Stardog. (2015) Stardog. [Online]. Available: http://stardog.com

[10] P. T. Wood, "Query languages for graph databases," *SIGMOD Record*, vol. 41, no. 1, pp. 50–60, Apr. 2012.

[11] E. Sirin and B. Parsia, "SPARQL-DL: SPARQL query for OWL-DL." in *Proc. 3rd Workshop on OWL: Experiences and Directions*, 2007.

[12] C. Joslyn, S. Choudhury, D. Haglin, B. Howe, B. Nickless, and B. Olsen, "Massive scale cyber traffic analysis: A driver for graph database research," in *Proc. First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, pp. 3:1–3:6.

[13] O. Brdiczka, P. Mahadevan, and R. Shi, "Method and system for thwarting insider attacks through informational network analysis," 2014, U.S Patent App. 13/709,940. [Online]. Available: http://www.google.com/patents/US20140165195

[14] N. Baumgartner, S. Mitsch, A. Müller, W. Retschitzegger, A. Salfinger, and W. Schwinger, "A tour of BeAware–A situation awareness framework for control centers," *Information Fusion*, vol. 20, pp. 155–173, 2014.

[15] M. M. Kokar, C. J. Matheus, and K. Baclawski, "Ontology-based situation awareness," *Information Fusion*, vol. 10, pp. 83–98, 2009.

[16] E. Vassev and M. Hinchey, "Towards a formal language for knowledge representation in autonomic service-component ensembles," in *Proc. 3rd International Conference on Data Mining and Intelligent Information Technology Applications*, 2011, pp. 228–235.

[17] E. Vassev and M. Hinchey, "Knowledge representation for adaptive and self-aware systems," in *Software Engineering for Collective Autonomic Systems*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 221–247.

[18] M. L. Mathews, P. Halvorsen, A. Joshi, and T. Finin, "A collaborative approach to situational awareness for cybersecurity," in *Proc. 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2012, pp. 216–222.

[19] S. More, M. Matthews, A. Joshi, and T. Finin, "A knowledge-based approach to intrusion detection modeling," in *Proc. IEEE Symposium on Security and Privacy Workshops*. IEEE, 2012, pp. 75–81.

[20] R. E. Hohimer, F. L. Greitzer, C. F. Noonan, and J. D. Strasburg, "Champion: Intelligent hierarchical reasoning agents for enhanced decision support," Pacific Northwest National Laboratory (PNNL), Richland, WA (US), Tech. Rep., 2011.

[21] F. L. Greitzer and R. E. Hohimer, "Modeling human behavior to anticipate insider attacks," *Journal of Strategic Security*, vol. 4, no. 2, pp. 25–48, 2011.

[22] N. Lu, P. Du, F. L. Greitzer, X. Guo, R. E. Hohimer, and Y. G. Pomiak, "A multi-layer, data-driven advanced reasoning tool for intelligent data mining and analysis for smart grids," in *Proc. IEEE Power and Energy Society General Meeting*. IEEE, 2012, pp. 1–7.

[23] B. Morin, L. Mé, H. Debar, and M. Ducassé, "A logic-based model to support alert correlation in intrusion detection," *Information Fusion*, vol. 10, no. 4, pp. 285–299, 2009.

[24] A. Viswanathan, A. Hussain, J. Mirkovic, S. Schwab, and J. Wroclawski, "A semantic framework for data analysis in networked systems," in *Proc. 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 127–140.

[25] C. Cheh, "The cyber-physical topology language: Definition and operations," Master's thesis, University of Illinois at Urbana-Champaign, 2014.

[26] M. Krötzsch, F. Simančík, and I. Horrocks, "A description logic primer," *Computing Research Repository*, vol. abs/1201.4089, 2012.

[27] M. L. Collins, D. Spooner, D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, "Spotlight on: Insider theft of intellectual property inside the united states government involving foreign governments or organizations," Software Engineering Institute, Technical Note, 2013.

[28] A. Cummings, T. Lewellen, D. McIntire, A. Moore, and R. Trzeciak, "Insider threat study: Illicit cyber activity involving fraud in the U.S. financial services sector," Software Engineering Institute, Special Report, 2012.

[29] G. Weaver. (2015) ITI/cptl-power. [Online]. Available: https://github.com/ITI/cptl-power/wiki

[30] E. Schultz, "A framework for understanding and predicting insider attacks," *Computers and Security*, vol. 21, no. 6, pp. 526–531, Oct. 2002.

[31] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient owl reasoner." in *Proc. 5th International Workshop on OWL: Experiences and Directions*, vol. 432, 2008.

[32] P. Laskov and R. Lippmann, "Machine learning in adversarial environments," *Machine Learning*, vol. 81, no. 2, pp. 115–119, 2010.

[33] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. (2004) SWRL: A Semantic Web Rule Language combining OWL and RuleML. [Online]. Available: http://www.w3.org/Submission/SWRL/