Identifying Heterogeneous Infrastructure Interdependencies through Multiverse Simulation

Craig Poulin
Interdisciplinary Engineering
Northeastern University
Boston, USA
c.poulin@husky.neu.edu

Michael Kane
Civil and Environmental Engineering
Northeastern University
Boston, USA
michael.kane@northeastern.edu

***Abstract*—When an infrastructure system responds to a disruption, its adaptive behavior can be described by the interactions between elements. These interdependencies are heterogeneous, multi-domain, and span temporal scales. Coupled with infrequent and limited historical data, it is challenging to identify and model key interdependencies. To address this, we propose a novel modeling methodology and iterative simulation approach. Starting with discrete event simulation and simulation graph models, we introduce dependency networks (DN), counterfactual event graphs (CEG), and multiverse simulation. Dependency networks link discrete events and model states to specific infrastructure elements. Counterfactual event graphs expand discrete events to multiple possible outcomes. Multiverse simulation considers each event outcome by duplicating and diverging the simulation into multiple timelines. Through iterative multiverse simulation and focused validation by stakeholders, the model is revised to identify and incorporate interdependencies that impact the system's post-disruption performance. This approach provides a framework for new modeling techniques and implementation in future case studies.**

Keywords—modeling, simulation, applications, resilience, critical infrastructure, discrete event simulation, failure analysis, network theory (graphs), interdependencies

I. Introduction

On July 13, 1977, two high-voltage power lines in New York were struck by lightning. Due to poorly maintained and configured equipment, the system responded by disconnecting these lines. The subsequent shift in energy flows caused the Indian Point nuclear plant to automatically shut itself down. With the addition of spare capacity, the system would have restored balance—if not for the second lightning strike 20 minutes later. Another set of transmission lines were taken out of commission, and another generating station shut itself down. The subsequent cascading failure was not stopped and, one hour after the initial lightning strike, a majority of New York City descended into darkness [1].

This historical scenario highlights the need for infrastructure resilience. Using the resilience definition "the ability to prepare for and adapt to changing conditions and withstand and recover rapidly from disruptions," [2] this event illustrates how changing conditions—the cascade of failures in the electrical system—impact the ability of the system to provide its intended function. The term "operational resilience" is often used to describe the adaptive behaviors that seek to maintain continuity of function [3], [4]. In this scenario, the system's adaptive behavior could not be fully described with a physics-based model. During the event, the Con Edison system operator did not take the necessary steps to maintain system function; a post-event report would state that the system would have been rescued by "alert, well-trained operating personnel" [1]. Faced with an antiquated, confusing control room and incomplete information, the operator did not have the confidence to disconnect loads. Instead, he called his supervisor at home—and the supervisor did not make the correct decision. Later, the Federal Energy Regulatory Commission (FERC) would identify that there was a "lack of preparation for major emergencies such that operating personnel failed to use the facilities at hand to prevent a system-wide failure" and that it could have been prevented "by a timely increase in Con Edison's in-city generation or by manual load-shedding" [5].

With this additional context, understanding resiliency becomes more complicated. A system's adaptive behavior can be described by its components' dependencies—in which "the state of one infrastructure influences or is correlated to the state of the other"—or interdependences—in which two systems are dependent on the other [6]. In the scenario above, interdependencies can be found between: electrical components and other electrical components; the electrical system and power plant operating procedures; the electrical system and the system operator; the system operator's decisions and the training he received prior to the event; the system operator and the communication system; and more, especially when considering how the scenario would have proceeded with slight changes in the events.

More recently, our anecdotal experience has shown us that understanding resiliency requires understanding non-obvious interdependencies. On a campus or military installation, a critical function may have a backup generator connected to the facility through an automatic transfer switch. When the utility power is lost, the switch commands the generator to start and transfers the load when the generator is operating. During this transitional period, the critical loads are often supported by the batteries of an uninterruptible power supply. In this process, two common failures are for the generator to not start or for the switch to not transfer the load. In either case, critical loads do not fail immediately—they are still supported by the batteries. To determine if or when the system fails, we must be able to answer new questions, including: Will the facility occupants identify that the system did not operate correctly? Will they be able to contact a technician with the necessary skill to solve the problem? Will the technician arrive and fix the problem before the batteries are empty? Are the other requirements competing

for the technician's time? Each of these links provides a dependency—and a potential system vulnerability.

From anecdotes such as this, it is clear that campus-level resilience cannot be analyzed without confronting hidden element-level interdependencies. Yet the nature of these interdependences—heterogeneous, multi-domain, poorly-modeled, and across temporal scales—make it difficult to determine which are important prior to modeling the system. With this in mind, we propose a modeling framework and iterative simulation approach that is designed to identify element-level interdependencies that impact the energy resilience of military installations or similarly sized campuses. Our goal is not to create a perfect system model but instead to identify critical interdependencies through expert-in-the-loop iterative improvements.

II. Modeling Goals

Assessing infrastructure resilience requires modeling the interactions between interdependent systems. Across modeling techniques, two common weakness are limited availability of data for model development and difficulty in validating results [7]. We address both of these challenges by focusing on a more modest, transitional goal: identifying element-level interdependencies that influence possible scenarios of post-disruption failure and recovery. From this goal, the core of our approach is iterative refinement of a model through stakeholder engagement. Considering stakeholders to be experts on a subset of the system, their knowledge and perspective provides data, and we approach qualitative validation by ensuring the simulation produces sequences they expect to be possible.

We have adopted Brown's definition of a model: "an abstraction that emphasizes certain aspects of reality to assess or understand the behavior of a system under study" [8]. To this end, our focus is on the possible interactions between elements in a post-disruption infrastructure system. The term "element" is deliberately broad, and we include non-technological dependencies in our scope.

Resilience modeling and analytics are prone to "fundamental surprise," in which the world behaves in ways that were not anticipated by the modeler or user. This is in contrast to "normal events"—expected behavior—and "situational surprise"—events that are unexpected but still within the model boundaries [9]. Fundamental and situational surprise can also be framed as unknown-unknowns and known-unknowns. For resilience modeling to be effective in real-time, real-world events, it must provide a feedback mechanism to incorporate fundamental surprises and improvise new models [9]. Our iterative approach seeks to improve baseline interdependency modeling by prompting system stakeholders to identify key interactions that must be included in the model. Stakeholders—such as facility managers, utility operators, mission owners, and logisticians—have unique perspectives that enable identification of interdependencies from anecdote, experience, or foresight that might not otherwise be identified by purely technical models. While this does not eliminate the risk of fundamental surprise, it seeks to incorporate surprises that may be fundamental to the modeler, but situational to a specific stakeholder.

This iterative approach is inspired by the "modeling cycle" used to determine a sufficient level of complexity in agent-based modeling: formulate the question; assemble hypotheses for essential processes and structures; choose scales, entities, state variables, process, and parameters, implement the model; analyze, test, and revise the model [10], [11]. This "pattern-oriented modeling" approach was designed to enable rigorous and comprehensive bottom-up modeling while addressing complexity and uncertainty. In our approach, each stakeholder would be concerned with the patterns of performance related to their scope of knowledge, and we deliberately focus on the revision step of the cycle.

Returning to our framework's goal—identifying possible post-disruption interactions between elements—we must first highlight goals that are explicitly outside our scope. One, we do not seek to develop or incorporate highly-detailed, dynamical models of individual system components. To the contrary, our goal is to represent behaviors and interactions in the simplest methods possible while still representing the critical system-level behavior: the possible sequences of post-disruption interactions. Two, we do not attempt to estimate the probability of any specific system outcome. This is deliberate; limiting exploration of interdependencies to the known and easily quantified can overlook rare but high-impact interdependencies. From Eisenberg et al: "successful resilience analysis extends decisions from probabilities to possibilities" [12]. With this in mind, our modeling approach focuses on representing possible interactions across heterogeneous components.

III. Simulation Methodology

Our simulation methodology has three novel elements. We will model interactions via discrete event simulation, which can be represented as a simulation graph model [13]. To relate events in the simulation to infrastructure elements, we introduce the *dependency network* (DN). To consider the space of possible system behavior, we expand the simulation graph model into a *counterfactual event graph* (CEG). Finally, we introduce our *multiverse simulation* approach, which enables consideration of competing event outcomes by branching simulated timelines at their point of divergence.

A. Discrete Event Simulation and Graph Representation

Discrete event simulation is commonly used to simulate stochastic systems with distinct events and resources, such as manufacturing, service queues, and transportation [14]. In a discrete event simulation, simulation time-steps do not move continuously or at fixed increments, but jump between events. Between events, system states do not change. At any point in simulated time, the system can be fully described by the system clock, current set of states, the calendar of scheduled events, and the underlying logic [15]. Our modeling goals focus on interactions between elements, which can be represented as events.

Event graphs and simulation graph models provide a simple and robust method to represent discrete event simulation [13]. Both approaches describe the relationships between each event, its state updates, and the future events it schedules. With some additional features, such as canceling edges and passing parameters, event graphs have been shown to be flexible and

extensible, despite their simplicity [16], [17]. As an alternative to canceling edges, a more general approach is to include execution conditions. This reduces modeling clutter and provides computational advantages [13].

Combining notation from event graphs and simulation graph models [13], [17], we adopt the following notation for M, a general simulation graph model. These relationships can be represented graphically, as illustrated in Fig. 1.

$$M = (V, E, S, F, C, X, T, \Gamma, A, B) \quad (1)$$

where
V is the set of *event vertices*
$E = \{(v_o, v_d) \mid v_o, v_d \in V\}$ is the set of *scheduling edges*
S is the *state space*
$F = \{f_v : v \in V\}$ is the set of *state transition functions* for each event
$C = \{c_e : e \in E\}$ is the set of *scheduling-edge conditions*
$X = \{x_e : e \in E\}$ is the set of *execution-edge conditions*
$T = \{t_e : e \in E\}$ is the set of *edge delay times*
$\Gamma = \{\gamma_e : e \in E\}$ is the set of *event execution priorities*
$A = \{A_e : e \in E\}$ is the set of *edge attributes*, if any, associated with each scheduling edge
$B = \{B_v : v \in V\}$ is the set of *event parameters*, if any, associated with each event vertex

This model provides the underlying logic for a discrete event simulation. In addition to this logic, a simulation requires a *global simulation time*, τ, and an *event calendar*, Λ. The event calendar is an ordered set of *event notices*:

$$\Lambda = \{\lambda_1, \lambda_2, \dots\} = \{(\tau_1, v_1, e_1, a_1), (\tau_2, v_2, e_2, a_2), \dots\} \quad (2)$$

where
λ_i is the event notice
τ_i is the execution time
v_i is the event vertex to be executed
e_i is the scheduling edge for the event
a_i are values assigned to the edge attributes

A simulation begins with the set of initial states, an initial time, and an event calendar with one or more initializing events. Simulation proceeds with the following process:

Step 1: Update the global simulation time to the first event notice: $\tau = \tau_1$

Step 2: For the first event notice's scheduled edge, check the execution-edge conditions c_{e_1}. If the conditions pass, execute the event notice. If not, skip the following step.

Step 3: Execute the first event notice, λ_1. Pass attributes a_1 to the event vertex's parameters B_{v_1}. Evaluate the state transition function f_{v_1} to update states. Schedule future events: for each scheduling edge, e_{ij}, originating at v_1, evaluate the scheduling edge conditions $c_{e_{ij}}$; if the conditions pass, evaluate edge delay time $t_{e_{ij}}$, evaluate attributes $A_{e_{ij}}$, and generate a new event notice. Insert new event notices into the event calendar Λ based on their execution times. For ties, reference event priorities, Γ.

Step 4: Remove the first event notice from the list and shift the remaining event notices.

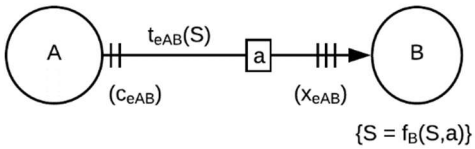

Fig. 1. Illustration of the general simulation graph model relationships. When scheduling condition c_{eAB} is true, event *A* schedules event *B* with attributes *a* at time delay t_{eAB}, which is a function of the current system state. Before executing event *B*, execution condition x_{eAB} is evaluated. If it passes, event *B* updates system states based on the passed attributes and current system states.

Repeat these steps until reaching a terminal condition: either an empty event list ($\Lambda = \emptyset$) or predetermined terminating event ($\lambda_1 = \lambda_{terminate}$).

The flexibility of discrete event simulation enables modeling interactions across time, space, and heterogenous infrastructure elements. In our application, an event takes place at a specific infrastructure element. When an event is executed, state updates are limited to that element or its local neighborhood. Additionally, each infrastructure element may have multiple possible events. From these considerations, we need another modeling convention to define the relationship between infrastructure elements, states, and events.

B. Dependency Network

We define a dependency network (DN) as a directed graph:

$$DN = (N, D) \quad (3)$$

where
N is the set of *infrastructure element* vertices
$D = \{(n_o, n_d) \mid n_o, n_d \in N\}$ is the set of *dependency* edges

The dependency network is developed in parallel with the simulation graph. Each event and each state in the simulation graph is associated with a specific infrastructure element. This relationship is identified by function $\mathcal{N}$, mapping all events and states onto specific infrastructure elements. Correspondingly, each infrastructure element is mapped by functions $\mathcal{V}$ and $\mathcal{S}$ onto unique subsets of events and states, respectively—i.e., an event or state cannot be associated with more than one infrastructure element.

$$\mathcal{N}: \begin{cases} v_i \in V \to n_j \in N \;\middle|\; \bigcup_{k=1}^{|V|} \mathcal{N}(v_k) = N \\ s_i \in S \to n_j \in N \;\middle|\; \bigcup_{k=1}^{|S|} \mathcal{N}(s_k) = N \end{cases} \quad (4)$$

$$\mathcal{V}: n_i \in N \to V_j \subset V \;\middle|\; \bigcap_{k=1}^{|N|} \mathcal{V}(n_k) = \emptyset \quad (5)$$

$$\mathcal{S}: n_i \in N \to S_j \subset S \;\middle|\; \bigcap_{k=1}^{|N|} \mathcal{S}(n_k) = \emptyset \quad (6)$$

The dependency network edges are determined by the scheduling edges in the simulation graph. Through function $\mathcal{D}$, infrastructure element B is said to be dependent on infrastructure element A if an event associated with element A schedules any event associated with element B. Parallel edges and self-loops are permitted.

$$\mathcal{D}: (v_o, v_d) \in E \to (n_o, n_d) \in D \;\middle|\; \mathcal{N}(v_o) = n_o, \; \mathcal{N}(v_d) = n_d, \bigcup_{k=1}^{|E|} \mathcal{D}(e_k) = D \quad (7)$$

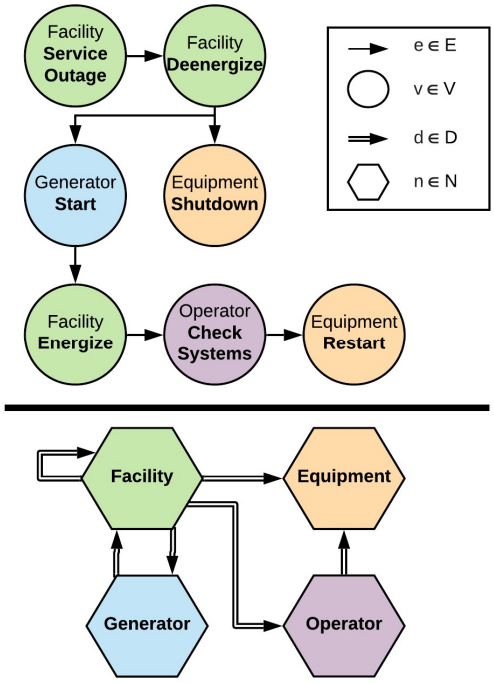

Fig. 2. (Top) example simulation graph model labeled with $\mathcal{N}(v)$ and v (for simplicity does not include time delays, conditions, attributes, and state updates); (bottom) corresponding Dependency Network (DN)

From these relationships, a dependency network can be generated entirely from a simulation graph. This provides a compact method to summarize all simulation graph relationships as dependencies between infrastructure elements. An example is provided in Fig. 2.

The dependency network is also used to bound the discrete event simulation to local information and influence. Via function $\mathcal{H}$, an element's local information is determined by its relationships in the dependency network—the element itself, its in-neighbors, and its out-neighbors:

$$\mathcal{H}(n_i) = n_i \cup \{n_j | \{n_i, n_j\} \in D\} \cup \{n_k | \{n_k, n_i\} \in D\} \quad (8)$$

The states known to nodes n_i can thus be identified with $\mathcal{S}(\mathcal{H}(n_i))$. Implemented in the general simulation model, M, local information is used to limit the inputs for scheduling-edge conditions, edge delay times, and edge attributes stored with an event notice. Execution-edge conditions and state transition functions may only access the event notice attributes—denoted as a—and local information. State transition functions can only update the states for the infrastructure element at which the event takes place. Equations (9)-(12) describe these restrictions. By avoiding global states and parameters, we force our simulation to incorporate all dependency relationships.

$$F = \left\{f_v : \left\{a, \mathcal{S}\left(\mathcal{H}\left(\mathcal{N}(v)\right)\right)\right\} \to \mathcal{S}\left(\mathcal{N}(v)\right), v \in V\right\} \quad (9)$$

$$C = \left\{c_e : \mathcal{S}\left(\mathcal{H}\left(\mathcal{N}(v_o)\right)\right) \to \{0,1\}, (v_o, v_d) \in E\right\} \quad (10)$$

$$X = \left\{x_e : \left\{a, \mathcal{S}\left(\mathcal{H}\left(\mathcal{N}(d)\right)\right)\right\} \to \{0,1\}, (v_o, v_d) \in E\right\} \quad (11)$$

$$T = \left\{t_e : \mathcal{S}\left(\mathcal{H}\left(\mathcal{N}(v_o)\right)\right) \to \mathbb{R}_{\geq 0}, (v_o, v_d) \in E\right\} \quad (12)$$

This extension of simulation graphs is useful for modeling sequences of events across infrastructure elements. Once completed, a simulation can be described by its history of executed event notices and the states following each event. In many discrete event applications, the simulation is performed to provide insights into converging performance metrics. For our purposes, insights are also obtained from the sequences of events themselves.

C. *Counterfactual Event Graph and Multiverse Simulation*

We want to extend our approach and examine the space of possible system behavior by allowing for alternative outcomes at events in our event graph. In this section, we introduce *counterfactual event graphs* and apply them in *multiverse simulation*.

Counterfactual risk analysis is an approach used to analyze possibilities that extend from historical cases—"alternative realisations [sic] of the past" [18]. By considering outcomes worse or better than the historical sequence of events (downward and upward counterfactuals, respectively), simulated event paths are compared to the baseline case. With various approaches, counterfactual simulation has been used to model driver behavior and vehicle crashes [19], select medical treatment options [20], estimate the economic impact of cyberattack on the electrical grid [21], and assess the impact of farmland preservation policies [22].

We extend the counterfactual approach to the simulated, hypothetical sequences of events. For example, the event "start" for a generator could be described as having at least two possible outcomes: "success" and "failure". Refinement to the model—as described in later sections—may result in additional outcomes such as "malfunction" or "short circuit". Describing these possible outcomes and their impact on the system's emergent behavior is fundamental to our approach.

We modify our general simulation graph model, M, to define a *counterfactual event graph* (CEG). The set of event vertices is divided into disjoint sets: prompting events (or *prompts*), V_P, and outcome events (or *outcomes*), V_O. We limit event scheduling such that prompts may only schedule outcomes and outcomes may only schedule prompts. This provides two disjoint sets of scheduling edges and classifies our CEG as a bipartite graph.

$$V = \{V_P, V_O\} \quad (13)$$

$$E = \{E_{PO}, E_{OP}\} \quad (14)$$

$$E_{PO} = \{(v_o, v_d) \mid v_o \in V_P, v_d \in V_O\} \quad (15)$$

$$E_{OP} = \{(v_o, v_d) \mid v_o \in V_O, v_d \in V_P\} \quad (16)$$

Prompting events (e.g., generator "start") are scheduled as described in the previous section. With this extension, each prompt event has one or more possible outcomes (e.g., "success" and "failure"). A prompt event does not update any states ($f_v = \emptyset, v \in V_P$) and only serves to schedule an outcome event. During a simulation, only one outcome event is scheduled (e.g., a generator cannot both "start" and "fail").

Outcome events are given the highest priority in the model ($\gamma_{e_i} > \gamma_{e_j}, e_i \in E_{PO}, e_j \in E_{OP}$) so the selected outcome is always the next event notice in the simulation. Edges scheduling outcome events are provided with the attributes from the edge that scheduled the prompting event. During the outcome event, states are updated and additional prompting events are scheduled. The CEG is related to a dependency network as described in the previous section, with two additions. One, each prompting event and its successor outcome events must be associated with the same DN infrastructure element. Two, DN edges are determined by the subset of CEG edges linking outcomes to prompts; this modifies (7) into:

$$\mathcal{D}: (v_o, v_d) \in E_{OP} \rightarrow (n_o, n_d) \,|\, \mathcal{N}(v_o) = n_o, \\ \mathcal{N}(v_d) = n_d, \bigcup_{k=1}^{|E|} \mathcal{D}(e_k) = D \qquad (17)$$

The example from Fig. 2 is expanded into a CEG in Fig. 3. Previously, all outcomes described the ideal, expected, or typical response of the system—in software, this is often referred to as the "happy path" [23]. The expansion into a CEG include alternative outcomes for three events. Even in this limited example, this change incorporates dependencies and infrastructure elements that were not identified previously.

To consider the impact of alternative outcomes, we establish a *multiverse simulation* approach. Our multiverse simulation has two layers: a *watcher* routine and a set of alternative *timelines*. Each of the timelines is an "alternate reality" [24] that simulates the system after a point of divergence. Until this point, the discrete event simulation uses the calendar and model to update states and schedule events—this is the behavior of a single timeline. With the multiverse simulation, when the CEG provides competing event outcomes, the watcher routine branches the competing outcomes into distinct and separate timelines. Both timelines have the same history leading up to the point of divergence. Because simulation does not adjust the underlying CEG model, each timeline simply manages its own history, global time, set of states, and event calendar.

For the history of a simulated timeline, we identify each prompt-outcome set of events with the *outcome index*, ε. From this index, we can identify the infrastructure element ($\mathcal{N}(v_\varepsilon)$), the prompting event ($v_i \,|\, \{v_i, v_\varepsilon\} \in E_{PO}$), and the outcome event ($v_\varepsilon \in V_O$). Because a prompting event does not update any states and is always succeeded by an outcome event, we need not record prompting events separately. With visibility of the entire timeline and all states, the watcher can calculate and record a performance measure, θ, for the entire system. The progression of a timeline, L, can thus be represented as an ordered set of five-tuples:

$$L = \{ (\varepsilon, \tau, S, \Lambda, \theta)_1, (\varepsilon, \tau, S, \Lambda, \theta)_2, \dots \} \qquad (18)$$

where
ε is the outcome index
τ is the global clock time at the time of the event
S is the set of states after the outcome takes place
Λ is the event calendar after the outcome takes place
θ is the performance measure after the outcome takes place
$L(k) = (\varepsilon, \tau, S, \theta)_k$ is the five-tuple at event index k

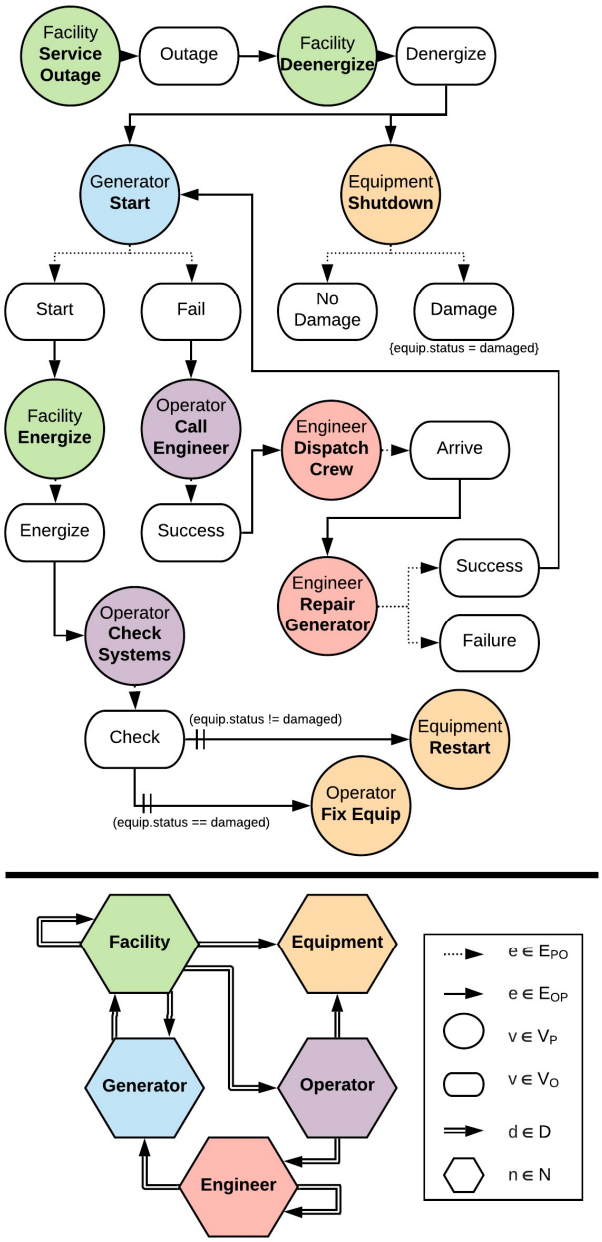

Fig. 3. (Top) example Counterfactual Event Graph (CEG) labeled with $\mathcal{N}(v)$ and v (for simplicity only includes illustrative conditions and state updates); (bottom) corresponding Dependency Network (DN)

As the watcher routine branches timelines, the entirety of the multiverse simulation is a set of timelines:

$$\mathcal{L} = \{L_1, L_2, \dots\} \qquad (19)$$

In each timeline, the progression of the system's performance measure over time can be plotted to provide a *recovery curve* [25]. The relationship between branching timelines and subsequent recovery curves is illustrated in Fig. 4.

Consider simulated timeline i at event index k with two possible outcome events: ε_α and ε_β. Timeline i selects ε_α. The timeline calculates new states and the system performance measure, updates the event calendar, and continues.

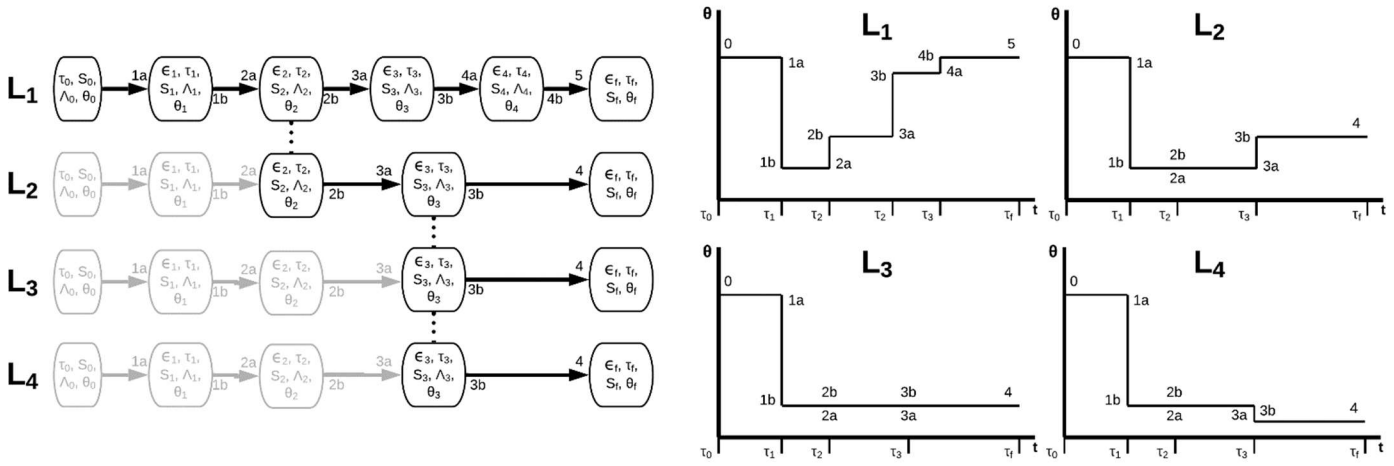

Fig. 4. For an unspecified model, illustration of branching timelines (left) and subsequent recovery curves (right); history copied upon timeline creation from a diverging outcome is shown grayed-out

To consider outcome event ε_β, the watcher creates a new timeline. This new timeline is a duplicate of its initiating timeline up until the point of divergence. Diverging from timeline i, this new timeline has different states, event calendar, and performance measure at event index k. A different event calendar means that this timeline will progress in a different direction, which still using the underlying logic of same CEG.

$$L_i = \{L_i(1), \dots, L_i(k-1), (\varepsilon_\alpha, \tau, S_\alpha, \Lambda_\alpha, \theta_\alpha), \dots\} \quad (20)$$

$$L_{|\mathcal{L}|+1} = \{L_i(1), \dots, L_i(k-1), (\varepsilon_\beta, \tau, S_\beta, \Lambda_\beta, \theta_\beta), \dots\} \quad (21)$$

A standard discrete event simulation injects variation through random variates, particularly when scheduling events [15]. In our approach, variation is introduced by adding outcomes. Consider, within a larger simulation, modeling a generator repair that historically takes between 1-3 hours. A standard discrete event simulation will generate a repair time from this distribution, and insights are gleaned from the convergence of the system performance measure across may runs of the simulation. In our multiverse simulation, "begin repair" is the prompting event, and the modeler must implement distinct outcomes—perhaps "slow" and "fast"—each of which schedules "complete repair" at a distinct time delay—one and three hours, respectively. Each of these possibilities provides a point of divergence for branching timelines.

By removing all stochasticity within our approach, the results of a given simulation run are deterministic and based on: the system model, CEG and DN; initial conditions, S_o simulation start time, τ_0; and an initializing calendar, Λ_0. In a standard discrete event simulation, the model's emergent behavior is identified by simulating many independent runs. In our multiverse simulation, we identify the model's emergent behavior by comparing the set of timelines produced within a single multiverse simulation run. This approach supports our goal—identifying possible post-disruption interactions between elements—by providing a framework upon which simulation results can be examined, the model updated, and simulation repeated.

IV. Iterative Simulation

With the multiverse simulation methodology in place, we propose iterative simulation to incorporate stakeholder knowledge into the model. Within a session, simulation results—with multiple timelines—are reviewed by the modeler and user. Based on these results, the underlying CEG and DN are revised. Simulation is repeated with the revised model, and the process continues until the modeler and user are satisfied with the model's level of detail. This process is illustrated in Fig. 5. The output of this session is a better representation of the real-world system's fundamental interactions and infrastructure dependencies.

A. Output Analysis and Model Revision

The output of a multiverse simulation (the set of simulated timelines, $\mathcal{L}$) is determined by the underlying model (CEG and DN) and scenario (initial conditions, simulation start time, and initializing calendar). At the simplest level, the underlying model can be validated by asking: for the given scenario, is each expected system response represented in a timeline? When considering infrastructure resilience, we choose to focus on the possibility of edge cases, not their probability and not the average or expected system performance. Correspondingly, the focus is the sequence of events within each timeline. Historical events and case studies of failure and recovery should be used as primary references in validating an output—this is useful for incorporating the inherent messiness of real-world scenarios [5], [26]. For example, if a recent power outage recovery was delayed due to the availability of technicians, then those considerations should be incorporated into the model. Stakeholder conjecture and historical examples from similar systems should also drive output validation.

When the simulation output is deemed insufficient—either lacking a possible sequence of events or describing a timeline with inadequate nuance—the modeler must revise the underlying model. We acknowledge this provides an impossible end-state without specific boundaries; boundaries on sufficiency should be established by the modeler and users' specific modeling goals. The counterfactual event graph and dependency

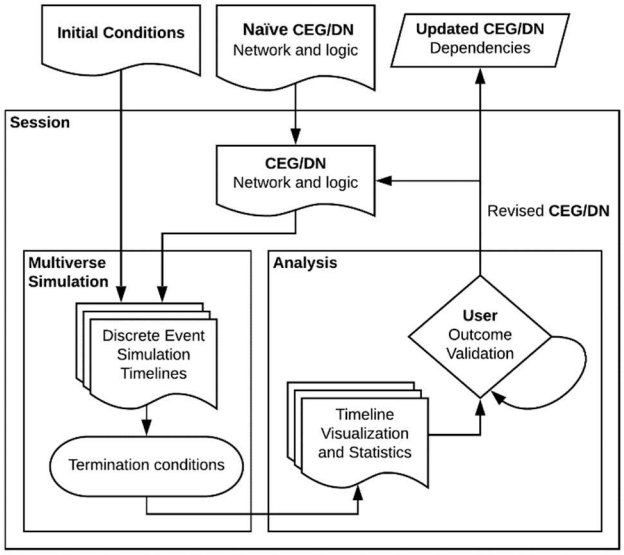

Fig. 5. Iterative modeling process

network can be revised by adding new vertices, adding new edges, and dividing vertices. Adding vertices and edges must take into consideration their relationship to the existing CEG and DN. Dividing a vertex (outcome, prompt, or element) involves allocating or modifying the relationships and properties to the new subdivisions. Division of vertices is used to add fidelity (e.g., event "start generator" is divided into "run generator" and "energize generator output" events) or to specify distinct subsystems (e.g., element "facility electrical system" is divided into "critical subpanel" and "non-critical subpanel"). There are seven possible modifications to the CEG and DN model, as shown in Table I.

With the multiverse simulation approach, we assume that results are never wrong, just insufficient. If a known or expected timeline does not appear in the simulation results ($L_{known} \notin \mathcal{L}$), then the results are insufficient. For example, simulation results are not wrong if they include highly unlikely timelines but not more likely timelines. The more likely timelines should be made available by adding to or revising the model and—because our simulation does not provide probabilities—the unlikely outcomes should not be removed. If a sequence of events is overly general, then vertices should be subdivided. This approach does not address fundamental errors when adding vertices and edges to the model—if a facility in the model does not exist, then the results are, in fact, wrong. This possibility should be deliberately avoided when establishing the CEG and DN, but this limitation is not inconsistent with model risk in alternative modeling and simulation approaches.

After revising the CEG and DN, the multiverse simulation is repeated, providing a new result to analyze. Because output validation is broadly defined—there are always more possible outcomes—this iterative process can continue indefinitely. However, each repetition provides a more detailed model of the real-world system's interactions and interdependencies. At any point, the modeler and user may choose to end the iterative process.

TABLE I. MODEL REVISION OPTIONS

Modification	CEG/DN Impact	Required Details
Add outcome event	$v_i: v_i \in V_O$	Impact on states, new outgoing edges
Divide outcome event	$V_i(v_j): v_j \in V_i \subset V_O$	Assign state updates and outgoing edges
Add prompt event	$v_i: v_i \in V_P$	Identify incoming edges, add at least one outcome
Divide prompt event	$V_i(v_j): v_j \in V_i \subset V_P$	Assign state updates, incoming edges, and outcomes
Add scheduling edge	$e_i: e_i \in E_{OP}$	Identify origin outcome and destination prompt
Add element	$n_i: n_i \in N$	Add states, events, and incoming and outgoing edges for events
Divide element	$N_i(n_j): n_j \in N_i \subset N$	Assign states and events

B. *Performance Metrics and Targeted Revision*

Comparing a multiverse simulation output to a specific historical sequence of events is relatively straightforward—even with many simulated timelines, the results can be reviewed using tabulation and visualization techniques like filters. However, validation becomes more difficult when probing stakeholders for non-obvious, theoretical possibilities. Yet this is an essential for revising the underlying model. We seek to systematically focus the modeler and user's focus on CEG and DN components that are the most likely candidates for revision.

First, the modeler and user must determine the performance metrics, θ, upon which the system can be evaluated. Recall that during the multiverse simulation, this is updated by the *watcher* after an outcome event's state updates, so performance metrics are not limited to local information. Potential resiliency performance metrics include state of the network, energy loss, and outage frequency. The system's performance metrics can also be a set (e.g., the energy loss at each facility). Our approach is deliberately agnostic, allowing the modeler to implement the metric that best fits the user's requirements and goals. For many performance metrics, recovery curves illustrate the function of a system over time and can be used to illustrate the sequence of a particular timeline [25].

We denote the final performance metric of timeline L_i as $\hat{\theta}(L_i)$. Because our simulation does not include probabilities, many standard statistical approaches are not appropriate. However, because we aren't seeking an actual estimator for the system's performance, we can abuse convention to focus the modeler and user's attentions. We extend our notation to the mean performance metrics for a set of timelines:

$$\hat{\theta}(\mathcal{L}) = \frac{1}{|\mathcal{L}|}\sum_{k=1}^{|\mathcal{L}|} \hat{\theta}(L_k) \tag{22}$$

Next, we define notation for a subset of timelines. Subsets of timelines can be established based on the inclusion of a specific outcome event, prompting event, or infrastructure element. These are established as the following, respectively:

$$\mathcal{L}_{v_i} \subset \mathcal{L} \mid v_i \in V_O, v_i \in \varepsilon(L)\ \forall\ L \in \mathcal{L}_{v_i} \tag{23}$$

$$\mathcal{L}_{v_i} \subset \mathcal{L} \mid v_i \in V_P, \{v_i, v_j\} \in E_{PO}, v_j \in \varepsilon(L) \forall\ L \in \mathcal{L}_{v_i} \tag{24}$$

$$\mathcal{L}_{n_i} \subset \mathcal{L} \mid n_i \in N, \mathcal{V}(n_i) \in \varepsilon(L)\ \forall\ L \in \mathcal{L}_{n_i} \tag{25}$$

TABLE II. EXAMPLE SIMULATION OUTPUT REVIEW OPTIONS

Description	Function	User Query
Events that correlate with high absolute performance	$\underset{v_i}{argmin}\ -\hat{\theta}(\mathcal{L}_{v_i})$	Event v_i correlates with high performance—when is that not the case?
Events that correlate with low absolute performance	$\underset{v_i}{argmin}\ \hat{\theta}(\mathcal{L}_{v_i})$	Event v_i correlates with low performance—when is that not the case?
Prompting events that correlate with high relative performance	$\underset{v_i}{argmin}\ \hat{\theta}(\mathcal{L}_{v_i}) - \hat{\theta}(\mathcal{L} \setminus \mathcal{L}_{v_i})$	Event v_i correlates with high relative performance—when is that not the case?
Prompting events that correlate low relative performance	$\underset{v_i}{argmin}\ \hat{\theta}(\mathcal{L} \setminus \mathcal{L}_{v_i}) - \hat{\theta}(\mathcal{L}_{v_i})$	Event v_i correlates with low relative performance—when is that not the case?
Prompting events with a single outcome event	$v_i \in V_P \mid \lvert\{(v_i, v_j)\} \subset E\rvert = 1$	Prompting event v_i only has a single possible outcome—what else could happen?
Outcome events that do not schedule any prompting events	$v_i \in V_O \mid \{(v_i, v_j)\} \subset E = \emptyset$	Outcome v_i does not schedule any future events—is anything missing?
Timelines in which the performance measure varies the least	$\underset{L_i}{argmin} \frac{1}{\lvert L_i\rvert}\left(\sum_{k=1}^{\lvert L_i\rvert} \theta_k(L_i) - \hat{\theta}(L_i)\right)^2$	The performance measure for timeline L_i does not vary much—is anything missing?
Any timeline	$L_i \in \mathcal{L}$	Here is one simulated timeline—is this a realistic sequence of events?

From this, we determine the mean performance of a subset of timelines containing event v_i as $\hat{\theta}(\mathcal{L}_{v_i})$ and, using the complement set, we determine the mean performance of all timelines not including event v_i as $\hat{\theta}(\mathcal{L} \setminus \mathcal{L}_{v_i})$.

Our goal is to incorporate stakeholder knowledge into our model. Another way to describe this goal is to identify situations that would provide fundamental surprise to our model but situational surprise to our stakeholders. Pursuing this goal provides multiple options for analyzing multiverse simulation results, $\mathcal{L}$. Many of these options can be described with an objective function; examples are listed in Table II. Each option highlights a component of the model for consideration and subsequent discussion; revision is not limited to, nor required to include, the specific component.

Each of these options provides the opportunity for revision, but not a guaranteed improvement—the user's response could simply be "nothing seems to be missing". With this in mind, user queries should be varied across possible output review options and the analysis step should consist of many user prompts before running the revised simulation. A model revision can have two impacts on the next iteration: increased accuracy for existing timelines and additional simulated timelines.

Faced with the prompt, the user and modeler work together to provide additional context to the model. Queries should be left open-ended to prompt discussion with the user—from this discussion, the modeler works to incorporate revisions to the model. For a given infrastructure system, there are many stakeholders, each with their own perspective. User queries should be targeted—each stakeholder's knowledge can be described as a subset of infrastructure elements ($K_i \subset N$). Validation options can be modified to consider only this subset and its neighborhood.

Validation queries should be tracked between simulation iterations. One, validation queries with a negative response (i.e., no changes necessary), should be prevented from occurring in the next validation iteration. Two, when analyzing simulation results, it is possible that the user cannot provide definitive model revisions. In these cases, the validation query should be flagged for further consideration with different stakeholders or after research by the current user.

V. DISCUSSION AND FUTURE WORK

We developed this approach as a powerful and flexible framework for identifying interdependencies in real-world infrastructure systems. The results of modeling and simulating a specific infrastructure system can be used to directly provide insights to stakeholders, and the revised system model can be used for more advanced analytical techniques. To apply this framework, we are developing a MATLAB-based software implementation which can be used for focused case studies. During software development, we have two key goals: managing computational complexity and providing usability.

Computational complexity is driven largely by the threat of combinatorial explosion—unchecked, each alternative outcome doubles the number of simulated timelines. In our implementation, we consider multiple methods to address this reality. One, the watcher routine will be developed to identify patterns and convergence of simulated timelines; timelines may be merged or pruned when appropriate. Two, we will develop and implement terminating conditions and simulation interruption when appropriate; certain queries may be obvious without running the entire simulation to completion. Three, simulation iterations may be limited to a subset of possibilities by enabling or disabling sets of outcomes; similar to focusing queries based on a specific stakeholder's knowledge. Finally, software development will explore both parallel computing opportunities and tradeoffs between breadth- and depth-first timeline simulation.

For software usability, we seek to enable the iterative simulation process in a real-time setting. Computational complexity should be managed so simulation can occur in minutes to tens of minutes. The analysis stage should provide recommended user queries and visualization of results. User responses should be incorporate quickly and without significant custom coding. To this end, during software development and subsequent case studies, we will develop a library of standard CEG and DN components. This library should cover expected and observed relationships and should be deliberately expansive—it should, for example, span the physical, information, cognitive, and social domains [27].

Once developed, the software will be implemented in case studies. We are targeting military installations and college campuses, as both provide geographically clustered facilities, a

diversity of stakeholders, a wide range of energy requirements, and unique resilience goals. For future case studies, we are exploring robust element-level performance metrics. For example, energy requirements could be grouped by their criticality or subdivided into element capacity, essential function, and full function [28]. Additionally, through case studies, we anticipate developing the methods in which results are provided to stakeholders; the modeling process should improve decision support tools and inform planning and investment. One specific objective is to identify cascading failures that lead to unrecoverable system states. When unrecoverable system states are identified, the revised CEG system model could be analyzed to identify disruptions or key events that would lead to such states. Ultimately, these disruptions or sequences of events should be mitigated through deliberate, real-world contingency planning.

Finally, one challenge with case study implementation is determining the appropriate level of model resolution. Our proposed iterative modeling framework is never-ending without clear conditions for model sufficiency. These criteria should be developed to meet the stakeholders' goals and requirements; further work is needed to describe, coordinate, and implement these conditions.

VI. Conclusion

To identify and model key interdependencies in infrastructure systems, we propose a novel simulation methodology and iterative modeling approach. Expanding upon discrete event simulation and simulation graph models, we establish dependency networks, counterfactual event graphs, and multiverse simulation. Multiverse simulation considers each event outcome by duplicating and diverging the simulation into multiple timelines. Users validate the simulation results by analyzing these sequences of events. Focused queries can be generated to prompt the user to refine the model for the next simulation iteration. From this framework, we seek to develop a software implementation for use in real-world case studies at campus-sized infrastructure systems.

References

[1] D. E. Nye, *When the Lights Went Out: A History of Blackouts in America*. MIT Press, 2010.

[2] B. Obama, "Presidential Policy Directive 21 - Critical Infrastructure Security and Resilience," The White House, Presidential Policy Directive 21, Feb. 2013.

[3] Homeland Security Council, "National Strategy for Homeland Security," 2007.

[4] D. L. Alderson, G. G. Brown, and W. M. Carlyle, "Operational Models of Infrastructure Resilience," *Risk Anal.*, vol. 35, no. 4, pp. 562–586, Apr. 2015.

[5] Federal Energy Regulatory Commission, "The Con Edison Power Failure of July 13 and 14, 1977," Jun. 1978.

[6] S. M. Rinaldi, J. P. Peerenboom, and T. K. Kelly, "Identifying, understanding, and analyzing critical infrastructure interdependencies," *IEEE Control Syst. Mag.*, vol. 21, no. 6, pp. 11–25, Dec. 2001.

[7] M. Ouyang, "Review on modeling and simulation of interdependent critical infrastructure systems," *Reliab. Eng. Syst. Saf.*, vol. 121, pp. 43–60, Jan. 2014.

[8] J. J. Cochran, *INFORMS Analytics Body of Knowledge*. John Wiley & Sons, 2018.

[9] D. Eisenberg, T. Seager, and D. L. Alderson, "Rethinking Resilience Analytics," *Risk Anal.*, May 2019.

[10] V. Grimm *et al.*, "Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology," *Science*, vol. 310, no. 5750, pp. 987–991, Nov. 2005.

[11] S. F. Railsback and V. Grimm, *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

[12] D. A. Eisenberg, I. Linkov, J. Park, M. E. Bates, C. Fox-Lent, and T. P. Seager, "Resilience Metrics: Lessons from Military Doctrines," *Solut. J.*, vol. 5, no. 5, pp. 76–85, 2014.

[13] R. G. Ingalls, D. J. Morrice, E. Yücesan, and A. B. Whinston, "Execution Conditions: A Formalization of Event Cancellation in Simulation Graphs," *Inf. J. Comput.*, vol. 15, no. 4, pp. 397–411, Nov. 2003.

[14] G. S. Fishman, *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer Science & Business Media, 2013.

[15] B. L. Nelson, *Foundations and Methods of Stochastic Simulation: A First Course*. New York: Springer, 2013.

[16] A. H. Buss, "Modeling with event graphs," in *Proceedings of the 1996 Winter Simulation Conference*, Coronado, California, United States, 1996, pp. 153–160.

[17] E. L. Savage, L. W. Schruben, and E. Yücesan, "On the Generality of Event-Graph Models," *Inf. J. Comput.*, vol. 17, no. 1, pp. 3–9, Feb. 2005.

[18] T. Maynard, G. Woo, and J. Seria, "Reimagining History: Counterfactual Risk Analysis," Lloyd's of London, London, UK, Lloyd's Emerging Risk Report, Oct. 2017.

[19] J. Bärgman, V. Lisovskaja, T. Victor, C. Flannagan, and M. Dozza, "How does glance behavior influence crash and injury risk? A 'what-if' counterfactual simulation using crashes and near-crashes from SHRP2," *Transp. Res. Part F Traffic Psychol. Behav.*, vol. 35, pp. 152–169, Nov. 2015.

[20] P. Schulam and S. Saria, "Reliable Decision Support using Counterfactual Models," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1697–1708.

[21] E. J. Oughton *et al.*, "Stochastic Counterfactual Risk Analysis for the Vulnerability Assessment of Cyber-Physical Attacks on Electricity Distribution Infrastructure Networks," *Risk Anal.*, vol. 39, no. 9, pp. 2012–2031, Sep. 2019.

[22] J. He, Y. Liu, Y. Yu, W. Tang, W. Xiang, and D. Liu, "A counterfactual scenario simulation approach for assessing the impact of farmland preservation policies on urban sprawl and food security in a major grain-producing area of China," *Appl. Geogr.*, vol. 37, pp. 127–138, Feb. 2013.

[23] P. Bollen, "BPMN: A Meta Model for the Happy Path," 2010.

[24] Roy Thomas, "What if Spider-Man joined the Fantastic Four?," *What If?*, vol. 1, no. 1, Feb-1977.

[25] I. Linkov *et al.*, "Changing the resilience paradigm," *Nat. Clim. Change*, vol. 4, no. 6, pp. 407–409, Jun. 2014.

[26] M. Myers, "Fort Bragg apologizes for faux power outage," *Army Times*, 29-Apr-2019. [Online]. Available: https://www.armytimes.com/news/your-army/2019/04/29/fort-bragg-apologizes-for-faux-power-outage/. [Accessed: 27-Jun-2019].

[27] I. Linkov *et al.*, "Measurable Resilience for Actionable Policy," *Environ. Sci. Technol.*, vol. 47, no. 18, pp. 10108–10110, Sep. 2013.

[28] S. E. Flynn, "Bolstering Critical Infrastructure Resilience After Superstorm Sandy: Lessons for New York and the Nation," Northeastern University, Apr. 2015.