Constraint Hypergraphs as a Unifying Framework for Digital Twins

John Morris[1*], Douglas L. Van Bossuyt[2], Edward Louis[1], Gregory Mocko[1], John Wagner[1]

[1]Department of Mechanical Engineering, Clemson University, Clemson, SC 29631, USA.

[2]Systems Engineering Department, Naval Postgraduate School, Monterey, CA 93943, USA.

[*]Corresponding author. Email: jhmrrs@clemson.edu

July 7, 2025

Summary

Digital twins, used to represent physical systems, have been lauded as tools for understanding reality. Complex system behavior is typically captured in domain-specific models crafted by subject experts. Contemporary methods for employing models in a digital twin require prescriptive interfaces, resulting in twins that are difficult to connect, redeploy, and modify. The limited interoperability of these twins has prompted calls for a universal framework enabling observability across model aggregations. Here we show how a new mathematical formalism called a constraint hypergraph serves as such a framework by representing system behavior as the composition of set-based functions. A digital twin is shown to be the second of two coupled systems where both adhere to the same constraint hypergraph, permitting the properties of the first to be observable from the second. Interoperability is given by deconstructing models into a structure enabling autonomous, white-box simulation of system properties. The resulting digital twins can interact immediately with both human and autonomous agents. This is demonstrated in a case study of a microgrid, showing how both measured and simulated data from the aggregated twins can be provided regardless of the operating environment. By connecting models, constraint hypergraphs supply scientists and modelers robust means to capture, communicate, and combine digital twins across all fields of study. We expect this framework to expand the use of digital twins, enriching scientific insights and collaborations by providing a structure for characterizing complex systems.

Introduction

Every thing in the world is a system, and every scientific endeavor is fundamentally concerned with describing those systems and their behaviors. A system is a collection of things that affect the world in unique ways when arranged together. Whether seed or rainforest, child or civilization, transistor or satellite, a scientist works to understand how individual parts combine to produce the actions of the whole (*1*). Explanations of system behavior are expressed in modeling languages such as the algebraic models used to represent systems of numbers, hierarchical charts for organizations, and diagrams of electric circuits. By default, a model represents only a virtual system, meaning the system it describes exists solely as information. For example, a model of child development describes not a specific child, but a virtual child representing all children. But additional, specialized information can be revealed by building models that describe a single, physical system. Such specific representations are termed digital twins (DTs), and are employed to expose the complex interactions of real systems, allowing previously indiscernible relationships to be described. DTs leveraging modern data processing capabilities have been proposed for solving problems in both scientific and industrial communities such as medical diagnostics (*2*), global weather forecasting (*3*), and machine maintenance (*4*).

Because of the complexity of the systems they represent, high fidelity DTs end up being excruciatingly intricate. Their convolutions result in fragile representations that are difficult to maintain (*5*), resistant to reuse and redeployment (*6*), and prone to failure (*7*). More disconcerting is the difficulty of connecting manually configured DTs with external agents, including other DTs (*8*). As the scope of studied systems increases to include more domains, DTs must aggregate into systems of systems, sharing information and models with other DTs (*9*). The challenge of forming interoperable DTs has been described as one the most significant facing modelers (*10, 11*), prompting calls for a universal representation schema for DTs (*12–14*) that ‘‘is reusable across multiple domains, supports multiple diverse activities, and serves the needs of multiple users’’ (*15*).

Though many frameworks have been proposed (*11, 13, 14, 16–24*), the authors observe that each is inhibited by the dualism of expressiveness at the cost of interpretability. Knowledge graphs (*11, 14, 22, 23*) represent the former end of the spectrum. With their connections defined arbitrarily, knowledge graphs can represent any set of objects and their relationships (*25*). However, to interpret the meaning of a relationship requires an extensive ontology, restricting their ability to externally integrate (*18, 26*). Alternatively, more explicit frameworks may fully convey meaning, but only apply to systems within a singular domain (*16, 18*), or with limited types of relationships such as Bayesian probability models (*24, 27*).

The purpose of this paper is to propose a new framework

for DTs that solves the paradox of generalized interpretation by deconstructing system behaviors into the composition of set-based functions. The resulting schema is mathematically rigorous, enables deterministic simulation of systems from and across any domain, and provides mechanisms for ready integration of models with external agents. This framework is demonstrated through building a DT for a microgrid, following the validation methodology of Pedersen et al. (*28*). Although the application and formulations of this framework are new, its methods will be familiar to mathematicians and computer scientists versed in the principles of the lambda calculi (*29*) and functional programming (*30*). It is the hope of the authors that applying these principles to DTs will empower scientists and engineers in every field to solve the complex systems framing society's most critical challenges.

Nature of Digital Twins

To maximize the application of the proposed framework, the authors have elected to consider DTs as generally as possible, focusing on the essential functionalities that a DT must provide. The definition of a DT is commonly given as a virtual representation of a physical system (*31–33*), used to inform an agent (whether human or automaton) about the state of some system of interest (SOI) in lieu of direct measurement.[1] Their usefulness stems from the enhanced observability of a digital system, where information about the system is obtained with greater ease from the DT than the SOI (*15*). Exposing system states allows agents to make decisions within the context of the SOI (*34*), converting virtualizations into actions influencing the real world. A state is a property of a system that is distinguishable from something else (*35*), such as the status of a lightbulb or the fuel level of a generator, and which can vary (or evolve) over a range of values (*36*). In this way an agent can distinguish between the lightbulb being *on* or *off*, or fuel level being *full* versus *empty*. The goal of an agent is to observe a set of state values exhibited from the SOI. If all states can be directly measured, then no DT is necessary. This is, however, not predominantly the case, necessitating the construction of a DT that informs the agent where direct observation of the SOI is untenable. The DT performs this through one of two mechanisms: direct coupling, where the DT updates in accordance with corresponding properties of the SOI; or simulation, where the DT evolves under a prescribed manipulation from some initial conditions until it approximates the unmeasured facts (*37*). The combination of these allow an agent to observe the SOI's state through indirect measurements of the DT. Both mechanisms require a connection from the SOI to the DT enabling the latter to reflect the states of the former (*9*). The authors posit that a digital system providing these mechanisms, along with the necessary intersystem connection, constitutes a DT.

A mock example of a DT is introduced in Figure 1 to motivate this limited definition. In the figure is an electrical circuit consisting of two lightbulbs connected by a bimodal switch such that only one lightbulb is illuminated at any instance. The disjointed bulbs form the SOI for some agent who desires to know which light is illuminated. To this end, the agent assigns two values to each lightbulb corresponding to whether the lightbulb is recognized as illuminated (*on*) or not (*off*). If the SOI is fully observable then discovering the current status of each lightbulb is trivial. A more significant case to consider is if the lightbulbs are obscured, preventing direct measurements. A solution involves the construction of the DT shown on the right of Figure 1, consisting of a box containing a pair of counters each capable of showing boolean integers zero or one. This DT is coupled to the SOI (perhaps via photoresistive sensors) so that the counters correspond to the various states of each lightbulb: one for *on* and zero for *off*. In this context, coupling refers to connecting the signals from one system to another, so that part of the state of one system is shared by the other. The degree of coupling determines whether the prescribed states of the SOI can be identified through observations of the DT.

It is, however, more likely that a SOI cannot be fully coupled to a DT, preventing some facts of the SOI from being exposed to the DT. To exhibit these data, the DT must be configured to have the same behavior as the SOI. Then the SOI and the DT will evolve commensurately, and the facts of the SOI can still be exposed. In the mock example, the behavior of the SOI is shown at the top of Figure 1, which describes the mutual exclusion of the states for the first and second lightbulbs. This behavior can be mimicked by connecting the boolean counters of the DT so that the left counter only displays the opposite value of the right. After implementing this behavior, the DT needs only to observe a single light to manifest the states of the entire system. Note that DT and SOI do not share any common physical properties: the SOI is electrical; the DT is mechanical, with digital counters. Yet the interpretation of the two systems results in the same virtual specification of a set of states (represented by two boolean variables) and the behavior relating these states.

Representing System Behaviors

This example shows that a DT is a physical, digital system configured in such a way that, when observed, facts of another system can be understood. By necessity, this digital system is programmed to behave similarly to the SOI, so that experimenting on the DT produces the same output as corresponding experiments of the SOI (*38*). Any framework proposed for constructing a DT must explicitly reproduce this shared behavior, motivating consideration of how system behavior is defined for simulation.

Simulation is only possible if the agent can discover a relationship between known and unknown facts. These relationships describe a system, and the agent's description of these relationships is a system model (*37*). Each relationship describes the effect of one variable on another. Willems (*36*) showed that the specification of all such relationships consti-

[1] An expanded overview of DTs, along with definitions for many of these terms as employed by the authors, is given in Supplementary Information 1.

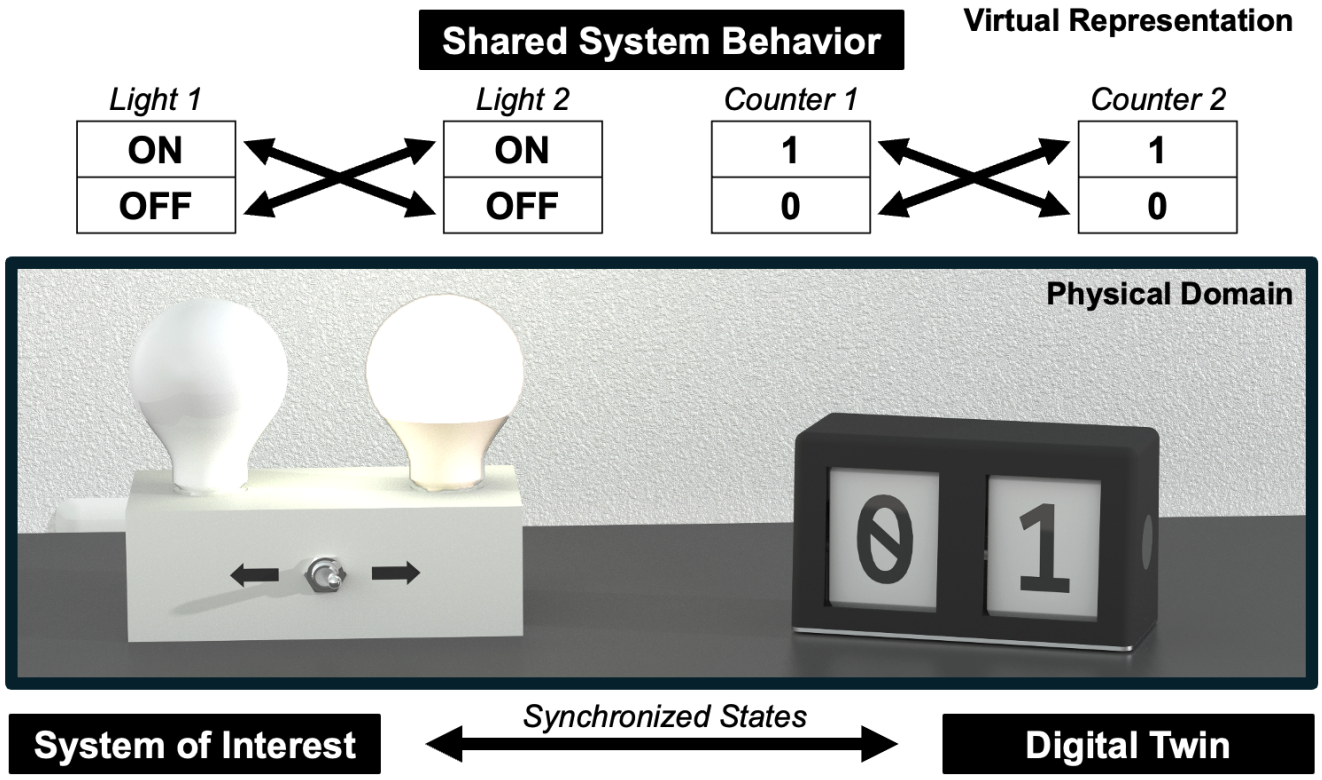

Figure 1: **A visual description of a DT,** with the system of interest and digital twin shown on the left and right respectively in the bottom (physical domain), and a virtual representation of shared behavior of the two systems at the top (virtual domain). For this example, the system of interest consists of two lightbulbs connected by a bimodal toggle switch such that only one bulb illuminates in either position of the switch. The digital analog similarly possesses two digital counters capable of displaying only two boolean numbers, 0 or 1.

tutes a system's behavior, and further that the effect of any behavior can be described as the restriction of the affected variable's possible values. For example, each light bulb in Figure 1 is constrained to only manifest the alternate state of its neighbor. Collecting restrictive relationships together results in an underdetermined model of a virtual system.

Reality is generally much more deterministic (*39*). Merely reducing a variable's exhibitable values is not indicative of a reified system, which must be temporally consistent, meaning that each variable exhibits only a single value within any frame of consideration (*39*). To simulate an unknown fact, an agent must discover relationships that reduce the set of possible values for a variable to a single datum. A function is the algebraic mechanism for mapping a value of one set to a value of another, consequently, a simulatable system model is one composed of functions. This can be illustrated by assigning variables to the bimodal lightbulb system in Figure 1, namely `light1` and `light2`, which each can exhibit the states *on* or *off*. The behavior of the first bulb is given by the constraint between `light2` and `light1`, described by a function f in Equation 1:

$$\texttt{light2} \xrightarrow{f} \texttt{light1} \coloneqq \begin{bmatrix} \texttt{on} \\ \texttt{off} \end{bmatrix} \rightleftarrows \begin{bmatrix} \texttt{on} \\ \texttt{off} \end{bmatrix} \tag{1}$$

As long as the SOI and DT in Figure 1 share states and behavior, equivalent state variables and functions can be applied to the digital display, as in Equation 2, where g is equivalent to f.

$$\texttt{counter2} \xrightarrow{g} \texttt{counter1} \coloneqq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightleftarrows \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2}$$

The collection of all known prescriptive functions can be arranged as edges in a graph, with each edge connecting nodes that represent the system variables they relate. In multiplearity relationships the function maps each combination of the domain variables to a unique value in the codomain. The inclusion of multiple variables (or nodes) in the domain makes the relationship a hyperedge, and the resulting holistic structure is termed a constraint hypergraph (CHG).[2] A technical definition is provided in (*40*), which establishes that any explicit system behavior can be represented in a CHG. This is important for a potential DT framework, which must reconcile any models used to stipulate the behavior shared between the DT and SOI into a unified representation. An example of a CHG is shown in Figure 2, where the arrows (edges) show

[2]CHGs are described most succinctly as a partial subcategory of **Set**. More information is given in Supplementary Information 2.

how the state variable comprising the edge's codomain can be constrained given values for each node in the edge's domain. Simulation is conducted by traversing a path from a set of nodes with known values to a node whose value is desired to be simulated.

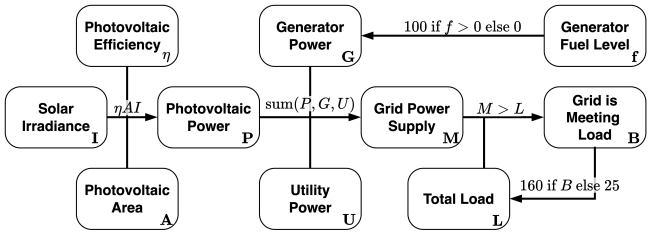

Figure 2: **A simple CHG of a microgrid,** with nodes for three grid actors (a photovoltaic array P, diesel generator G, and utility connection U) along with associated nodes. The behavioral functions that relate them are given by the arrows (pointing to the target) labeled with the algebraic rule governing the mapping.

Implementation of Digital Twins

Redefining a DT as a physical system sharing the behavior of another system clarifies what essential actions a framework must provide to implement a DT. In a system whose scope is not changing, the authors posit that there are two primary actions that a DT framework must support: (a) coupling the SOI to the DT, so that the DT has a shared-state with the SOI; and (b) configuring the DT to share the behavior of the SOI. Additionally, if the scope of the twinned system is changing, a framework must additionally (c) provide mechanisms ensuring the twinned representation is valid in the redefined context.

The first action is perhaps the most commonly provided functionality, generally implemented by sensors observing the SOI to the DT, such as an Internet of Things (IoT) network. Such a platform allows measurements of the SOI's state to be reflected in some repository, establishing a fully serviceable DT only if all information of the SOI is collected in the database–a rarity for representations of complex systems. To manifest information that cannot be measured, a DT must also twin the SOI's behavior. This occurs through the provision of models to the DT, which describe how observed inputs may be transformed into unobserved state values. CHGs adeptly perform these two tasks, storing all considered facts as nodes in the systems, and deconstructing models into the edges relating them. A serviceable DT further provides mechanisms for simulating the behavioral models. In addition to fulfilling the first two actions, the primary advantage of using a CHG to represent a DT is this ability to autonomously compose simulation processes. The combination of coupled states and simulatable models allows all desired information of the SOI to be observed via the DT, whether that data is measured or simulated, fulfilling the purpose of a DT.

While this alone merits consideration of a CHG as a formal framework for DTs, additional advantages derive from their ability to be redefined in different contexts. This last action required for representing a DT–proving a valid representation following arbitrary changes of the SOI's scope–is often characterized as interoperability (*10*) or extensibility (*41*), with the challenge of providing it being described the most significant barrier to widespread adoption of DTs (*42*). Adapting a definition typically employed for software development to a DT, extensibility is the twin's ability to validly manifest a SOI's facts despite changes to the SOI's scope (*43*). A model referencing the weight of a vehicle, for instance, might be extensible for changes of the vehicle's simulated location only as long as the location is on the Earth.

Providing extensibility for DTs adds additional complexity due to the expected use of a DT in forming system of systems representations (*14, 44*). Combining DTs into arbitrary aggregate systems is akin to modifying the scope of the original SOI; for example, a DT of a tire and of a car might be combined, expanding the scope of the SOI to the tire + car system. An effective framework ensures the representation of the individual systems remains valid across such scope changes, or else highlights when the representation is potentially invalid. This is especially important given the notion that systems are continuously changing, necessitating DTs that can evolve their virtual representations alongside the physical SOI (*45, 46*).

Decomposing a system into facts and behavioral constraints reveals the two vectors along which system scopes can change: first, adding or removing system data; and second, updating the relationships between that data. The two form the opposing sides of a modeling dilemma referred to as the ''Expression Problem'' (*47, 48*), which describes the difficulty of guaranteeing consistency after arbitrary updates to a model. Famously, a framework can generally be configured to be extensible for changes to either behaviors (relationships) or data, but not both (*49*). Such configured frameworks are respectively described as either functional or procedural (*50*). CHGs are intrinsically functional. This means that, while adding or removing facts in the system can indeed affect a CHG's consistency, modifications of a CHG's edges do not affect the validity of its representation, a characteristic referred to as having no side effects (*30*). This is especially useful for DTs, which generally concern an explicitly defined system. Behavioral consistency also allows users to update the models of a DT without needing *a priori* knowledge of the global system, greatly reducing the sensitivity of maintaining a convoluted DT architecture. Examples and further discussion on these points are provided in the Methods section.

Methods

Overview of Microgrid Demonstration

The claim of CHGs being used as a framework for deploying DTs was validated following the methodology of Pedersen et

al. (*28*) by specifically demonstrating a CHG-based DT made for microgrid system. A microgrid is an energy grid that operates semiautonomously from a standard utility network, provide additional flexibility to organizations whose needs go beyond the capabilities of established grids, such as renewable energy producers (*51*). Microgrids are composed primarily of energy sources and sinks, here referred to as actors. The system in question, overviewed in Figure 4, consists of an arrangement of grid actors including two diesel generators, a photovoltaic array, a battery energy storage system (BESS), and several energy loads scaled at three possible levels. The system also accounts for connections to the utility grid. Determining the behavior of the microgrid, including the power produced or consumed by its various actors, requires knowledge of loads on the system, the viability of all intermediary connections, and live information on energy fluctuations, such as the availability of solar-generated power. A DT might be used for monitoring purposes as well as testing, with specific importance placed on describing the resilience of the grid to cases of random failure or sabotage (*52, 53*).

The microgrid DT was individually validated by amending it to represent a test-scale microgrid that was operated over a three-day span in Monterey, California in the United States. This bench-top microgrid included a photovoltaic array, a BESS, one diesel generator, and an applied resistive load of approximately 0.7 kW. The DT represented these grid actors by using data inputs from the solar cells and load to simulate the states of the BESS and generator. Comparisons between the test run (*54*) and simulated values are shown in Figure 6, visually demonstrating its fidelity to the real system.

Development

The broader CHG was derived from standalone, generic models originally developed by researchers associated with the U.S. Navy (*55–57*). These models were developed in largely imperative formats, such as Microsoft Excel workbooks, MATLAB scripts, and Python modules. By deconstructing these into a CHG, the models could be simulated declaratively, rendering an effective DT. The process of conversion from a procedural model to a CHG involved four steps:

1. **Identify facts from the system.** Facts are any piece of information identified in the modeling process, including variables, parameters, or outputted data. These are listed as the nodes of the CHG in-preparation. Facts that were measured from the real system, such as solar irradiance or building usage rates, were set as initial values of their corresponding nodes, establishing the important connection between the virtual model and the physical SOI (*45*).
2. **Identify relationships between these facts.** Each relationship is described as an edge in the CHG. The partial hypergraph is visualized in Figure 4, with edges representing static database queries, discrete-event simulation, linear system solving, stochastic relationships, and file input/output.
3. **Organize the model.** The described nodes and edges are parsed and stored into a persistent collection.
4. **Interrogate the model.** An autonomous agent capable of searching the graph provides observations of the underlying SOI by connecting known inputs with desired outputs along discovered paths in the graph.

594 variables were identified for Step 1, corresponding to various parameters and states of the microgrid system, as well as settings for the simulation itself. These are tabulated in the Table A. The collection of all variables forms a state vector, and encodes every dimension along which the system (as described) may evolve. These states blend common interpretations of properties, attributes, and class variables. For instance, the states of a BESS include its current charge level, capacity, but also properties such as its maximum output. If any of these are observable to the modeler–such as from vendor specification sheets–they can be provided as inputs to the CHG, in which case the simulation of that node is the trivial provision of the input node's value (where input equals output). Other properties in the system include system facts conveying simulation settings, data directories, and timing variables. The latter is an interesting aspect of CHGs, where time is considered an observed state of a system, rather than as a universal property (such as in system formalisms given by Wymore (*58*) or Ziegler (*59*)). The benefit of this is that time can be considered uniquely for composing subsystems. Reconciling two models is contingent upon reconciling time steps and counters, a process captured in reconciling the CHGs.

The edges for each system are based on imperative models written by Peterson (*57*) that were converted to a functional structure. The relationships are of a varied nature: some are algebraic, such as the relationship between the number of seconds in a minute and the number of seconds in a year, others are more procedural, such as the functions calculating whether the BESS should be charging based on its charge levels and usage requirements. In every case, functions represent the lowest level of abstraction in the system–a black box within which the solver does not consider system complexity. These black boxes can be incredibly complex if needed; the function relating which grid members should be utilized over a given time step, for example, is calculated via a series of Python methods. By abstracting these steps into a single function, all the sub-variables and relationships in the methods are abstracted away so that they cannot be connected to other nodes in the graph. This is the unfortunate necessity of any system model, which must include a scope beyond which entities are not considered. Functions give this primitive level of abstraction.

Use of the Microgrid Digital Twin

Every path in the hypergraph is a potential simulation pairing a set of known inputs to a unique output. Deploying a CHG following its construction requires an engine capable of storing a CHG, parsing its members, and searching for a path mapping a set of inputs to an output. Such an engine may accomplish these tasks through any means, with its spe-

cific implementation affecting its efficiency of computation. Note that the performance of an engine is only tangentially related to the theoretical proposals here, and is consequently not explored in this study. The engine used by the authors for this article was a custom solver written in the Python programming language titled *ConstraintHg* (*60*), which employs a breadth-first search algorithm. Measured values (system information that is provided as an input) are given by discovering the trivial loop leading from a node to itself. In this case the engine acts as a database-management system: locating the fact's corresponding node and returning the stored value. Non-trivial paths supply simulated information, such as the projected cost of fuel for running the generator. In this case the agent seeds possible paths starting from each input, extending each path (a tree with inputs as its leaves) until one reaches the node corresponding to the desired output. The edges in the path represent the series of functions that, when composed, map the input set of the path's leaves to the path's root. The result of calculating each of these functions is the desired output.

After building the CHG and integrating it with the solving engine, the microgrid model was used to demonstrate several possible use cases of a DT:

1. **Data collection:** Relational database querying is perhaps the most similar form of system modeling, where each table represents the functional mapping between sets of values. In this case, the inputs for solar irradiance were taken from data sets compiled by the National Renewable Energy Laboratory (NREL) for Monterey, California, USA over the years 2000 to 2010 (*61, 62*). All aspects of data querying including file locations, column names, and primary keys were included as nodes in the hypergraph, allowing for full data retrieval and visualization, as shown with the dashed lines in Figure 6.
2. **Control:** Although the DT was implemented post-data collection as a mock example, feedback control was demonstrated by providing a controller strategy for the generator and BESS systems that prioritized lower-cost energy sources (such as the photovoltaic cells). Figure 6 shows the BESS switching from receiving to producing power, as well as the generator simulated as turning on to provide power to the grid when needed. Coupling the digital system to the microgrid actors would enable such real-time feedback to be implemented in the SOI. Methods for implementing such coupling have been widely shown, including programmable logic controllers (PLCs), MQTT messaging services, or built-in microcontrollers.
3. **System Interrogation:** Interfacing with the DT is the act of observing a state of the SOI. The microgrid DT enables universal interrogation of the microgrid's states by autonomously preparing simulation processes. This is programmatically conducted by calling the `solve(<state>)` function in the *ConstraintHg* package, passing it the name of the node representing the state to be queried. The takeaway from this is that an agent needs only to know the name of a state and have access to the `solve` method to be able to make full use of the DT. Serialized data, such as the time series of an actor's power output, can be provided by returning each value solved for along a path. Plots of serialized data are given in Figure 5 and 6.

Limitations

Many of the limitations of using CHGs to frame DTs are born from the nascent status of the supporting technologies. The custom solver used in this study, *ConstraintHg*, is limited in its solution speed and usability. In particular, the breadth-first search conducted within *ConstraintHg* to compose simulation paths is exhaustive and consequently computationally expensive. However, because these practical limitations are not tied to the theoretical foundations of CHGs, the authors suppose that future developments can address these early issues. Real time monitoring and control with DTs will require a more optimized approach, possibly leveraging search heuristics that allow quicker convergence to the optimal path.

Another reason limiting the scalability of systems is that pathfinding must be performed for every new input-output pairing. This is because a CHG is only a partial category, such that two functions that share a codomain and domain are not guaranteed to compose. Due to this partiality, a path in the hypergraph is only guaranteed to be solvable for its given set of inputs. For instance, a model within the microgrid for calculating the power received from the utility grid is to check if the utility network is connected to the microgrid. If it is not, then the power received is zero. This model can be explicitly written as an edge e between two nodes A and B, the first representing the connection status of the utility grid, and the second one relating the power it is providing. Note that e is only valid if the A is set to be `False`; there is no mapping provided from A to B by e if A is `True`. This extends to any path containing e. A pathfinding algorithm that encounters e cannot then know whether e can be composed into a valid path until it is supplied with a value for A, provided either by simulating an edge connected to A or passed as an input to the search sequence, precluding *a priori* traversal optimization by common methods such as Djikstra's algorithm (*63*). The authors address this in *ConstraintHg* by simulating every encountered edge in the search, so that the domain for every edge is always known to the solver. While this is guaranteed to be convergent, it is undeniably an untenable solution for scaled systems. It is the hope of the authors that this, as well as the problem of scope changes described in greater detail in Supplementary Information 4, can be addressed through continuing studies into the mathematics of CHGs.

Conclusion

The behavior of a system is a virtual concept, and can be represented as a set of explicit constraint functions acting on the state variables of a system. These functions and variables can be composed into a mathematical structure called a CHG, a

declarative model of a system that can be used in constructing analogously-behaving DTs. CHGs do not replace other modeling systems; rather they integrate them into a cohesive, global model. This model can be used to interrogate an SOI, expressing values that were either measured or else calculated within the CHG itself. These cross-cutting measures ensure that DTs built upon CHGs are universally accessible, and generally far more extensible, scalable, and redeployable than traditional DTs.

This work introduces both theoretical and practical measures for developing DTs for fields as diverse as engineering, organization management, medicine, and ecology. The resulting DTs capture meaning across all domains, providing a mathematically robust framework for representing the complex and integrated systems constituting the world in which we live.

Acknowledgements

The authors thank Richard Alves for configuring the Spanagel microgrid (*54*) used to validate the microgrid DT.

Author Statements

Supplementary Information is available for this paper.

Correspondence and requests for materials should be addressed to John Morris.

The authors declare no competing interests.

The views presented are those of the authors and do not necessarily represent the views of the U.S. Navy or any other organization. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. Approved for Public Release; distribution is unlimited.

Author Contributions

The primary drafter was J.M., with contributions from E.L. (uncertainty) and D.V.B. (definition of a DT). Reviewing was conducted by all authors, supervised by G.M. and J.W. The microgrid model was developed by J.M. and D.V.B., with the solver code written by J.M. Technical guidance was provided by D.V.B., G.M., and J.W.

Data Availability

The validation data (*54*) is available in Zenodo with the following identifiers: doi:10.5281/zenodo.15675037

Code Availability

All computer code used to generate these results has been made available under an open source license, with the latest versions available for use on GitHub. The microgrid CHG model (*64*) is available under the MIT license with the identifier doi:10.5281/zenodo.15278018, while the software for solving the CHGs (*60*) is available under the Apache 2.0 license with the identifier doi:10.5281/zenodo.15447063.

Extended Data Table A: List of Nodes in the Microgrid CHG

Node	Units	Description
Constants		
days in year	day	Number of days in a year
days in leap year	day	Number of days in a leap year
hours in day	hr	Number of hours in a day
hours in year	hr	Number of hours in a year
hours in leap year	hr	Number of hours in a leap year
seconds in minute	s	Number of seconds in a minute
minutes in hour	min	Number of minutes in an hour
seconds in hour	s	Number seconds in an hour
tolerance		Float tolerance
General Settings		
use random date	*bool*	True if the start date is set randomly
has random failure	*bool*	True if random failure is allowed
island mode	*bool*	true if all utility grids are disconnected
min year	year	Minimum year of simulation data
max year	year	Maximum year of simulation data
Timing Variables		
time	s	Total seconds passed during simulation
time step	s	seconds since last time calculation
hour	hr	Current hour (0-23)
day	day	Current day of the year (1-366)
year	year	Current year
start year	year	Starting year for simulation
start day	day	Starting day for simulation (1-366)
start hour	hr	Starting hour for simulation (1-24)
elapsed minutes	min	Number of minutes that have passed
elapsed hours	hr	Number of hours that have passed
num leap years	year	Number of leap years encountered
is leap year	*bool*	True if the current year is a leap year
hour index	hr	Current hour of the year (1-8784)
General Simulation		
state vector	*list*	State from each actor on the grid
names	*list*	Ordered names of actors on the grid
failing actors	*list*	List of failing components
is islanded	*bool*	True if the utiliy grid is disconnected
connectivity matrix	*list*	Cell $ij \rightarrow \text{actor}_i$ receives from actor_j
States for a General Grid Actor (X)		
name X	*string*	Label for X
state X	kW	Power receiving (-) or providing (+)
is connected X	*bool*	True if X is connected to grid
X receives from Y	*bool*	True if X receives power from Y
X receiving from Y	*bool*	True if X is actively receiving from Y
is failing X	*bool*	True if X is failing
prob failing X		Probability of X failing
prob fixed X		Probability of failing X getting fixed
supply X	kW	Current energy that the X can supply
cost X	\$/kWh	Cost of generating X's supply
is cost per unit X	*bool*	True if supply cost is per unit vs. lump
supply tuple X	*tuple*	Values for calculating X supply
req demand X	kW	Minimum power required for X to operate
max demand X	kW	Maximum power X can receive
benefit X	\$/kWh	Benefit of meeting X's demand
demand tuple X	*tuple*	Values for calculating X's demand

Node	Units	Description
States Associated with Photovoltaic Arrays (P)		
sunlight directory	*string*	Directory for solar data
sunlight filename	*string*	Filename for sunlight CSV file
sunlight data	*list*	Yearly sunlight values by hour
sunlight data label	*string*	Name of column for sunlight data
sunlight	Wh/m^2	Energy from sun for a specific hour
area P	m^2	Area of P
efficiency P		Efficiency of P
States Associated with Batteries (B)		
charge level B	kWh	Amount of charge in B
charge capacity B	kWh	Max charge B can hold
charge efficiency B		Efficiency of converting power to SOC
max output B	kW	Maximum power B can output
max charge rate B	kW	Maximum power B can receive
scarcity factor B		Cost gain of using depleted B
trickle prop B		SOC to reduce to trickle charge
is charging B	*bool*	True if B is charging
soc B		State of charge for the B
States Associated with Load Actors (L)		
normal load L	kW	Normal load (demand) of L
critical load L	kW	Critical load (demand) of L
States Associated with Buildings (B)		
type B	*enum*	Type of B
lights load B	kW	Lights load of B
equipment load B	kW	equipment load of B
load data B	*list*	List of dictionaries for B load data
building filename B	*string*	Filename for B load CSV data
load directory	*string*	Directory for B load CSV data
normal key B	*string*	Normal load column in CSV data
lights key B	*string*	Lights load column in CSV data
equipment key B	*string*	Equipment load column in CSV data
States Associated with Generators (G)		
next refuel hour G	hr	Next hour G will be refueled
prob refueling G		Probability of refueling G during the day
fuel level G	Liters	Amount of fuel in G
out of fuel G	*bool*	True if G is out of fuel
starting fuel level G	Liters	Starting fuel level in G
fuel capacity G	Liters	Max amount of fuel in G
max output G	kW	Max power G can output
consumption G	Liters	Fuel consumption for G
max consumption G	Liters	Max fuel consumption for G

Note: X indicates that the node is given for each actor $X \in \mathbf{X}$, where $\mathbf{X}$ is the set of all actors of a certain type (such as Generators, or all actors in general).

Extended Data Figure 3: Illustration of Concepts Pertaining to CHGs

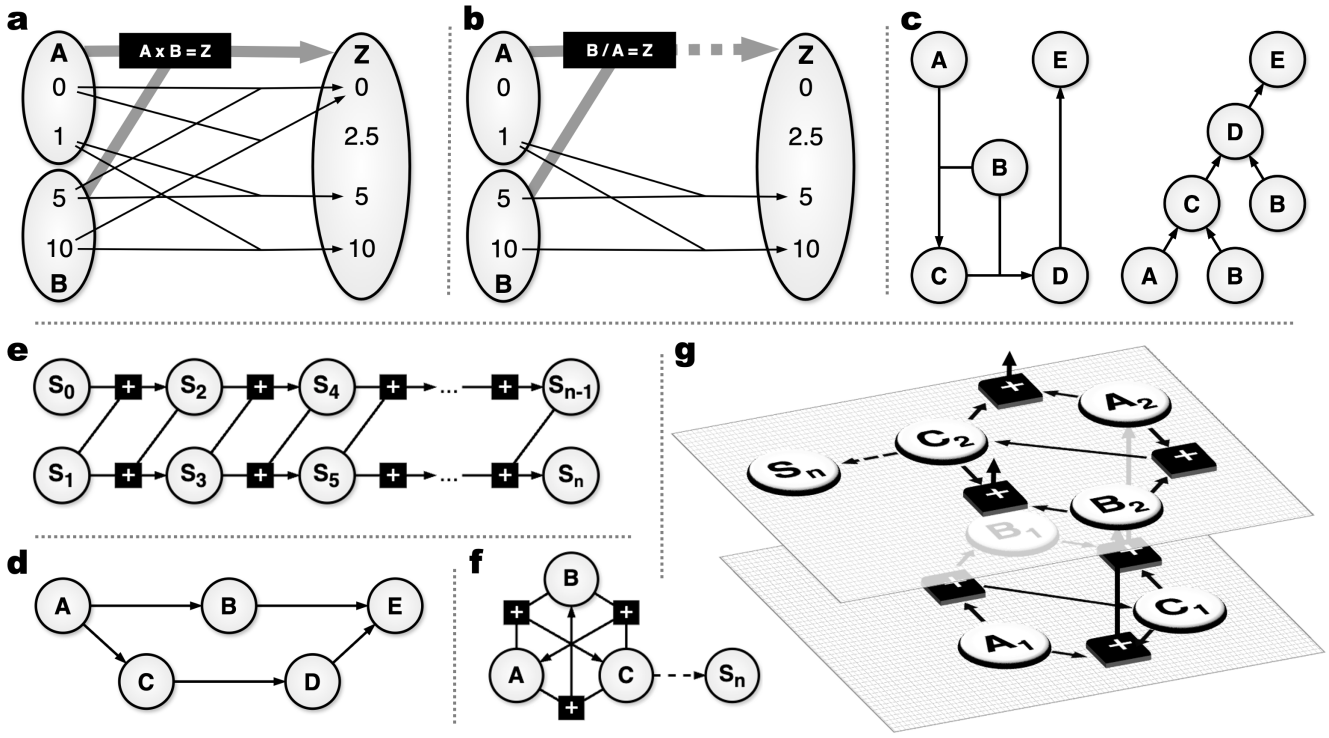

Figure 3: **Concepts pertaining to CHGs:** (**a**) deconstruction of an edge showing how each combination of set values is mapped to a value in the codomain; (**b**) similar to **a** but showing conditional edge mapping from a subset of domain values; (**c**) a path through a hypergraph (on left) described as a tree (on right); (**d**) a hypergraph with alternative routes from A to E demonstrating competing models; (**e, f, g**) demonstration of cycles using a CHG for computing the n-th value of the Fibonacci sequence S_n, with an acyclical, distributed CHG in **e**, and a version using a cycle (and a conditional edge, shown dotted) in **f**. **g** shows two iterations of the cyclical CHG in **f**, with each iteration separated on different planes of connectivity.

Extended Data Figure 4: Diagram of Microgrid CHG

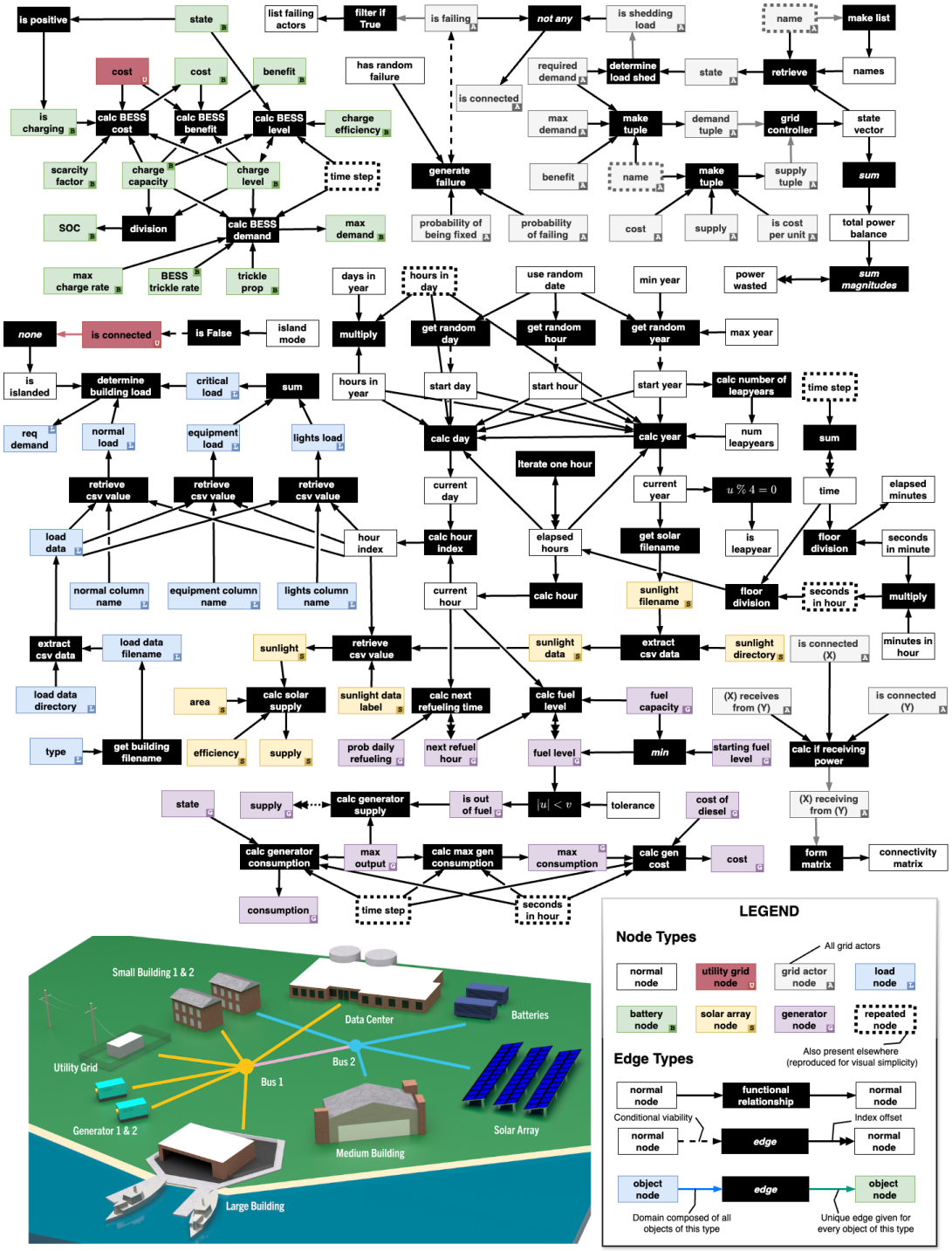

Figure 4: **Diagram of microgrid CHG,** showing 5 different types of connected actors either supplying or receiving energy. Each actor class adheres to a similar pattern, and for simplicity is only shown once. This means that all nodes that are not white (typical nodes) or black (edge descriptions) are actually represented once for each object; e.g. there are actually two nodes for charge capacity, one for each battery on the grid (green), and each connected in the same way. Note that the diagram is provided for illustrative purposes only and is not a formal description.

Extended Data Figure 5: Demonstration of Microgrid CHG

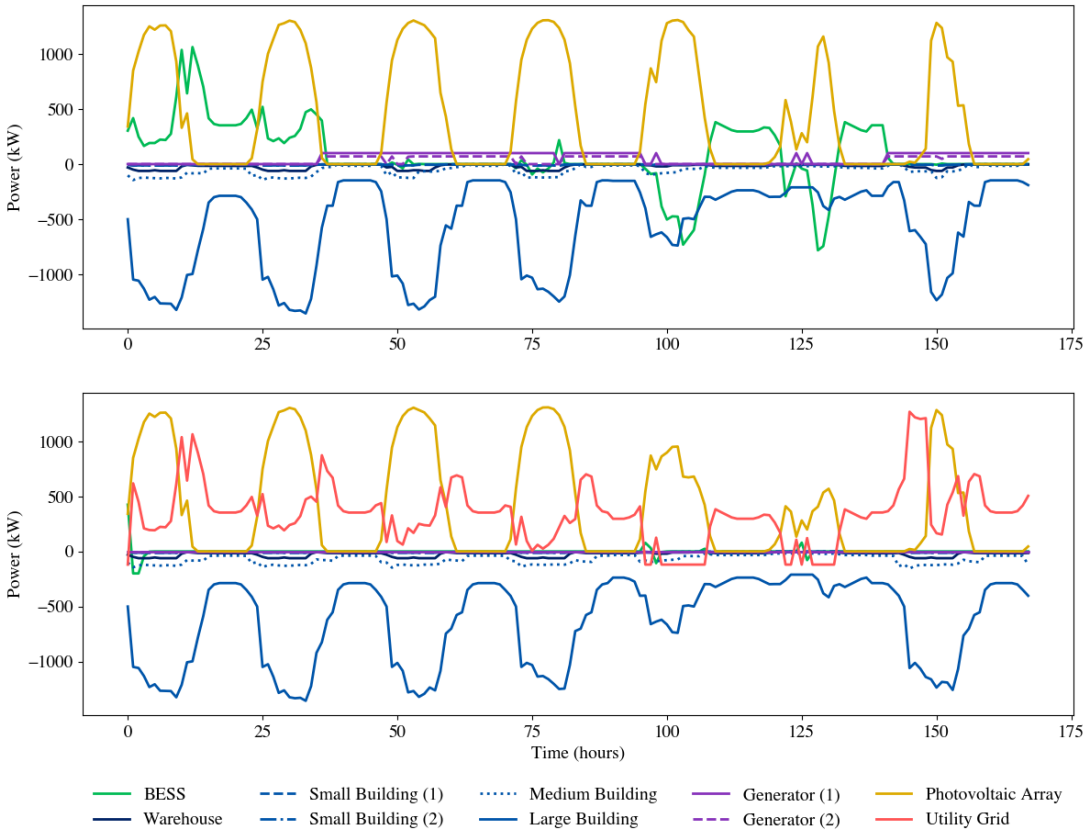

Figure 5: **Demonstration of Microgrid CHG** showing two simulations of the states of all grid actors over the same week-long span, with the top plot showing states of actors with the utility grid not connected (islanded) and the bottom plot showing the opposite.

Extended Data Figure 6: Validation of Microgrid DT Model

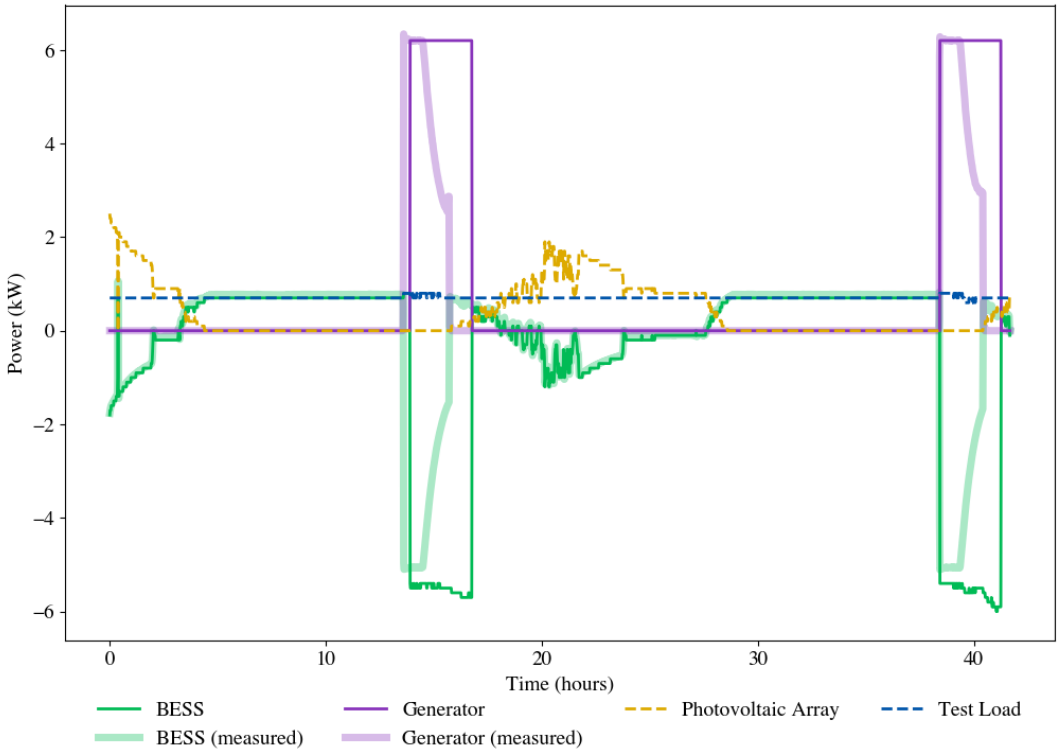

Figure 6: **Validation of microgrid DT Model** comparing data of the states of the four grid actors both simulated and compared to values observed during a 43-hour test run of a limited scale microgrid in Monterrey, California in April 2025 (*54*). The dashed lines indicate observed values (directly coupled to the DT), while the solid lines indicate simulated data. Simulated data is overlaid on top of translucent lines indicating the data observed from the validation run.

Extended Data Figure 7: Illustration of Modeling Uncertainty

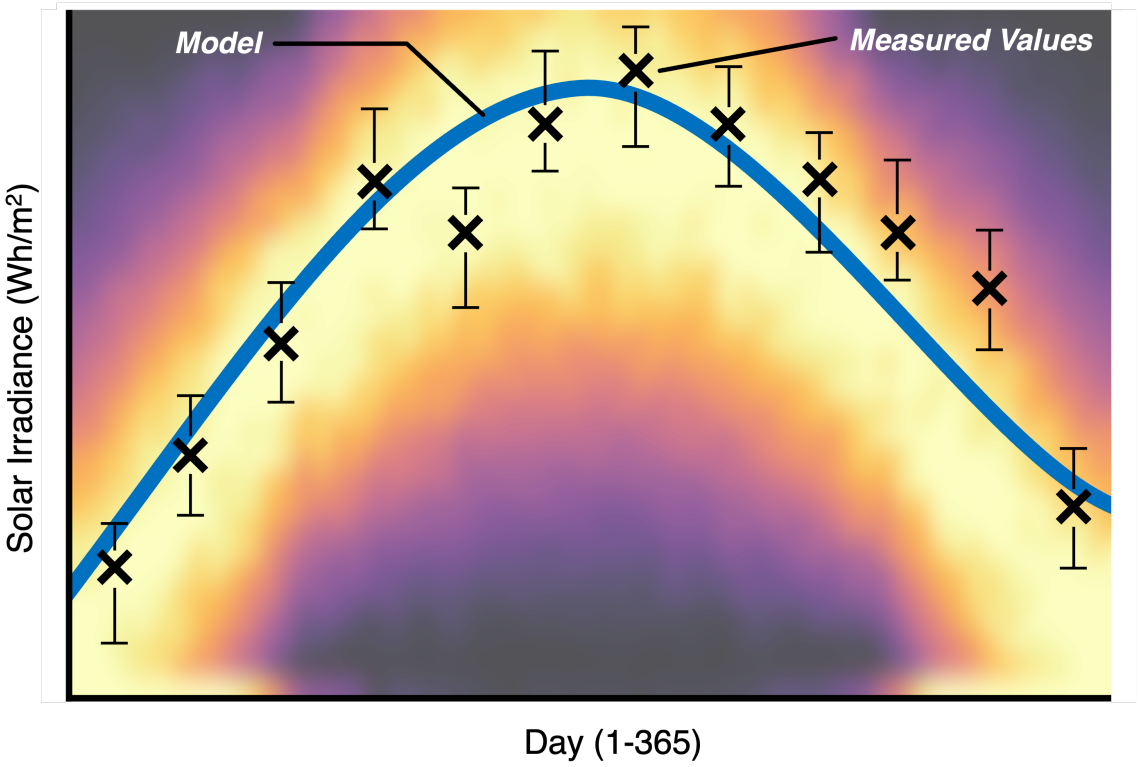

Figure 7: **Illustration of modeling uncertainty** for a relationship between day of the year and the average solar irradiance. The true value for irradiance is given probabilistically as the color map with brighter pixels indicating a higher likelihood of measuring a value in the given range. Twelve measured values are given by the black X's, each of which have some associated error that is often described using a measurement tolerance. A model, given by the blue line, is fit to the measured values with visible discrepancy between the modeled and measured values. Note that all data in the figure is synthetic and given for illustrative purposes only.

Supplementary Information 1: Overview of Digital Twins

The concepts of systems modeling are found in many fields, often resulting in competing interpretations of common words. The purpose of this section is to precisely define some of these terms as employed by the authors in the main paper. These interpretations are specific to this paper and intended to be supplementary in nature; they are not meant to serve as a formal standard or supersede existing definitions provided by works better suited for this purpose (*9, 15, 31, 33, 45, 65–67*). The traditional definition of a DT is of a virtual representation of a physical system (*31, 32, 65*). However, this definition conflicts with the worldview where actions in the physical domain can only be carried out by physical entities. Modern interpretations of DTs ascribe a verbose range of services for which a DT might be used for: decision-making (*66*), receiving (*68*) and recording (*69*) system data, validating (*70*) and updating (*45, 71*) system models, generating (*34*) and predicting (*9*) system states, controlling physical systems (*44*), and communicating signals (*72, 73*). To perform these physical actions, a DT must have some physical aspect to it beyond being solely defined as a virtual entity. Carefully exploring the ontological nuances of a DT allows greater consideration of how they can best represented.

The goal of any agent is to affect the world to achieve some specific outcome. This is true whether the agent is a swallow landing on a delicate branch, a synthetic automaton milling a workpiece, or a scientist trying to prevent the rise of global temperatures. To properly influence the physical world requires understanding of its complex network of interacting elements. Framing and characterizing the real world creates structures that are virtual, not physical; composed not of matter, but of information. Although entities in these two domains do not interact with each other, systems in the physical world are better influenced by agents who understood them virtually.

Definition 1 (Virtual Domain) *The domain whose quanta is information (always taken to be discrete), the associations of which describe phenomena observed in the physical domain, and whose bounds are finite.*

Definition 2 (Physical Domain) *The domain existing in space and time, the phenomena of which are described virtually. Used synonymously with ''real world.''*

The physical domain is taken as including elements characterized as *cyber*, including electromagnetic signals, transistor networks, and visual displays.

Definition 3 (Phenomenon) *Some distinguishable portion or aspect of the physical domain.*

Note that the physical domain includes both the tangible and intangible, such as phenomena that exist only in the future. This is important when considering DTs that represent these phenomena despite their immateriality. A key aspect of virtually representing elements of the real world is distinguishing the changes that result from interaction. This necessitates a definition of *state*:

Definition 4 (State) *The description of a phenomenon by assigning a unique datum to each distinctive arrangement, such that the phenomenon is characterized by only one value within any single frame of consideration. Used synonymously with ''state variable''.*

Note how the authors' use of *state* differs from the collective sense of the word as used in phrases such as the ''the total state of a system.'' The authors refer to this as the *state vector*.

Definition 5 (Observation) *The provision of a datum for a system state belonging to a single frame of consideration.*

The values of a state are termed *data*. Because the primary aspect of a datum is that it can be distinguished from another datum, the authors equate these data with the *information* given as the quanta of the virtual domain (*35*). The arrangement of states forms a system, defined in part here although a more comprehensive and philosophical treatment of the topic was given by Cellier (*37*).

Definition 6 (System) *A general description of some phenomena, consequently a virtual entity that represents the physical domain as a collection of states. The term ''physical system'' refers to the phenomena being described, while the term ''model'' refers to the virtual depiction.*

Definition 7 (System Behavior) *The restriction of the state values a system can exhibit within a unique frame of consideration (36).*

With this terminology, the goal of an agent interacting with the physical domain can be expressed as bringing some system to a set of desired states. A DT should assist with this task, providing some advantage in either understanding or manipulating the complexities of a physical system. There are three processes by which a DT can do this: measurement, control, and simulation.

Definition 8 (Measurement) *An observation made of the physical domain following specific interaction with it by an agent.*

Definition 9 (Control) *The manipulation of the physical domain so that it exhibits a specific set of states.*

Definition 10 (Simulation) *An observation of a physical system by an agent that is not coupled along the state to be observed.*

Each of these tasks are physical, and consequently must be performed in the physical domain by a second physical system, called a twin, that is coupled with the SOI.

Definition 11 (Coupling) *The synchronization of two phenomena such that changes in one engender changes in the other. These changes occur across relationships between synchronized states.*

The need for bidirectional coupling is well recognized as a requirement for a DT (*31, 45*), as each process is enabled by a distinctive form of synchronization. Measurement is conducted by coupling the states of the SOI to the twin, so that changes in the first can be measured by observing the latter. The reverse of this process is control, where arrangements of the twin cause the SOI to form a desired configuration. Simulation is based on the independent evaluation of the twin, yet still requires coupling with the SOI across the initial states from which the twin evolves. Additionally, simulation is only effective if the two systems behave similarly, so that observing of the twin informs the agent about the unmeasured state of the SOI.

The advantages in evaluating a twin versus the SOI come only if the twin has greater observability. Twins should also be highly configurable, so that the cost of matching the behavior of the SOI is minimized. Digital systems, whose states are described by Boolean variables, optimally provide these traits. Because of this, the twin is nearly always implemented on a digital computer, and is consequently termed a DT. DTs under this interpretation are physical rather than virtual, generally composed of transistors characterized by Boolean states. The example in Figure 1 provides an intuitive demonstration a DT, where the digital counter allows the states of the lightbulb system to be understood either through measurement (where the lightbulb system informs the digital one) or simulation (where the DT infers the state of the SOI). This motivates the following definition:

Definition 12 (Digital Twin) *A digital system coupled to another system so that each can affect the states of the other, configured so that measurement of the digital system engenders observation of all desired states of the other system.*

Because these processes are performed via the DT, it is always possible to enact them by manipulating and observing the DT's states. Both measured and simulated values are relayed to an agent through observation, while control sequences are triggered by certain states and verified by others. From this, the authors posit that for a framework to represent a DT it is sufficient to provide mechanisms for setting and observing the state of a DT.

Given this minimal interpretation, it is necessary to clarify the usage of DTs in practical deployments. Although control is not the primary purpose of a DT, it is often used to represent the control aspects of a real system (*74*). For each of the functionalities of a DT listed at the beginning of the section there is some aspect of the physical domain that must be influenced, whether transistors in persistent memory, actuation of a manufacturing machine, or illuminating pixels on a display. A virtual entity does not perform these physical actions, rather it describes the system behaviors that lead to these actions occurring. An agent that understands the virtual description can then configure ''the physical system [to] emulate the model'' (*39*). For instance, the microgrid DT in Figure 4 could be used to turn on and off the diesel generators (as shown in Figure 5) when the balance of the grid is overloaded. While the actual automation would occur along some coupling between the digital system and diesel generators, the representation of this action would be given by relating the balance of the grid (a state) with the status of the diesel generators (another state), the first serving as the trigger and the second one verifying that the action was performed correctly. Observing these values and enforcing their relationship is equivalent to performing the control task, assuming that the twin and SOI are coupled bidirectionally.

Semantic clarity is obtained by realizing that services provided by a DT (such as automation or visualization) are simply phenomena of the real world. Including these phenomena in the SOI allows them to be measured and influenced by the DT. The authors have demonstrated this separately by performing automating solid geometry modeling by expressing the functions of Computer-Aided Design (CAD) software within a constraint hypergraph (*75*). It is consequently their belief that the minimalistic interpretation of a DT as a tool for measurement, control, and simulation includes all possible services that can be configured on a digital platform; a generalization rather than an exclusion that will hopefully allow DTs to find use in a variety of fields and applications.

Supplemental Information 2: Overview of Constraint Hypergraphs

Understanding the structure of a CHG is essential to learning how they fundamentally represent behavior, and consequently to how they can be employed as a universal framework for representing a DT. To this end, this section introduces the high-level information about a CHG, with additional technical information provided in (*40*). Although similar to the graphs of constraint programming (*76*), CHGs differ in that each edge expresses a complete reduction of a variable's degrees of freedom. The theory of CHGs were originally formalized by (*77, 78*) as a foundation for analyzing systems. Though they have been employed under various names in systems engineering–usually as algebraic links between state variables (*79, 80*)–the authors assert their usage in representing general system behavior to be far more expansive than previously considered. In addition to the frameworks laid out previously, additional discoveries are presented in this section concerning how CHGs should be configured to represent DTs, specifically: (a) an interpretation of pathfinding; (b) the calculation of cycles in the graph; and (c) how validity frames should be expressed.

Pathfinding

The structure of a CHG consists of a set of nodes and a set of hyperedges. Each node is a variable and each hyperedge connects a group of nodes (called the source set) to another node

(called the target). If the values for each node in the source set are known, the value of the target can be calculated by executing the function encoded in the edge. The target node can then be joined into the source set of another edge, and subsequently used to solve for the value of another variable, in a manner explicitly similar to the lambda calculus foundational to functional languages such as LISP and Haskell (*50*). A path is a chain of edges connecting nodes. The structure of a path through a hypergraph is given by a tree, rooted in a set of source nodes and terminating at a single target (*81*), as shown in Figure 3c. Traversing a path is equivalent to simulating the target node and is performed through composing all the functions in the path, as in $(A, B) \xrightarrow{f_1} C; (B, C) \xrightarrow{f_2} D \xrightarrow{f_3} E$ giving $f_3(f_2(B, f_1(A, B))) = E$. The result is a type of universal simulation: the set of variables that can be constrained (and consequently observed) from a set of known values is given by all the nodes reachable by paths rooted in the representative source nodes.

It is possible that multiple paths to a target node exist, as in Figure 3d. Each path can be interpreted as a competing model capable of solving for the target node. In order to select a preferred model, each path must be compared using some discrimination factor, such as weights given for each edge. Weights can represent criteria such as uncertainty or computational expense, and when summed determine a preferred path (*82*). The identification and selection of paths is known as pathfinding, and can be performed by a variety of well-documented search algorithms.

Each constraint represents an assumption made concerning the relationship between various phenomena, it can consequently be assumed that each constraint is inaccurate to some degree. Establishing edges is done by an independent modeler, and may be performed most conveniently in a domain-specific modeling framework, such as a diagrammatic modeling language. Such frameworks restrict the expressible constraint functions to better capture the domain-specific behavior; geometric kernels, for instance, permit only geometric and dimensional constraints to define a geometric model. The generality of a CHG allows any identified constraint to be recorded in the hypergraph, organizing formerly disparate models into a single, unified structure. Edges encode either a mapping scheme ($a_1 \rightarrow b_1$, $a_2 \rightarrow b_2$, etc., where $a_1, a_2, \ldots$ are the values of a variable a) or some calculable rule (e.g. $b = a^2$). A common case is providing a function created by some statistical process such as a neural net or regression algorithm. Functions can even be executable processes, such as calling the Application Programmer Interface (API) of an external client to perform the mapping. This allows a CHG to connect information across software platforms. In all cases, edges must map each combination of values from a set of variables to a unique datum of another variable.

Conditional Viability

It is not always true that a behavioral constraint relates all possible states of a system phenomenon. For instance, a model of laminar fluid flow is only valid if the Reynolds number is below a given threshold. Incorporating this model into an edge can be interpreted as the target node being constrained over a subset of the source nodes' values. Instead of representing the subset of the variables over which the constraint is valid as a unique node, a conditional function is supplied for each edge that determines whether the constraint is viable for the known source values (Figure 3b).

Conditional viability exacerbates the cost of pathfinding, as paths are no longer independent of inputs to the system. Valid sequences for simulating a given node must be discovered for each new set of inputs, unless it can be shown that the all values solved during the process of traversing the path are within the validity frame of the next edge. Generally, this makes presolving a CHG nonviable, resulting in increased runtime costs during simulation.

Cycles

A cycle is a path in a hypergraph whose target node is also a member of its source set (*83*). Theoretically, cycles in a CHG should not exist, as they indicate that a system phenomenon is constrained by its own state, violating causality. Since the real world is causal, such behaviors cannot arise in a physical system. However, it is often the case that a modeler wishes to simulate the evolution of a state variable (especially through time). In such cases it is important to note that each evolution constitutes a unique variable and consequently should be represented by a separate node with separate edges.

However, because each of these nodes represent the same phenomenon, they often repeat the same behavioral constraints, such as shown in Figure 3e. If it can be shown that each instance of a variable is constrained by the same edges, including a dependence upon the previous state of the variable, then the repeated iterations can be summarized into a cycle in the graph (Figure 3f).

Each cycle exists within its own frame of reference, so that all contained nodes are connected only within a single iteration of the cycle. When the cycle returns to the start, the frame of the system is concurrently increased, a pattern visualized in Figure 3g by a spiral. In order to exit a cycle, a solver must identify an exiting edge that is only viable for the last frame of the cycle.

Simulation

Simulations are defined by a sequence of tasks that, when executed by a digital computer, forms a digital system transforming inputs to outputs (sometimes described as the behavioral trace of a dynamic system (*84*)). In a procedural modeling paradigm, such as block diagrams, each transformation between a set of inputs and an output is encoded in an explicit sequence that can be repeatedly simulated for a range of input values (*85*). However, a unique sequence must be manually defined for each pairing of input to output variables. For a system with n variables, the maximum number of edges that can relate these variables is given by $\sum_{i=1}^{n-1} (n-i)\binom{n}{i}$ (*75*), with each permutation of these edges resulting in a possible simulation.

Explicitly defining all simulation transformations becomes infeasible even for systems of modest complexity. This is especially critical for DTs, which, to provide appropriate modeling fidelity, typically involve highly-complex system representations extending across multiple scales and domains (*15*). However, if all processes are not provided, the resulting DT becomes fragmented, failing to manifest the relationships between properties defined by isolated models. For instance, a finite-element simulation might calculate when a column will buckle, while failing to expose the duration of the buckling event required to execute a discrete-event simulation. Object-oriented frameworks, such as the Functional Mockup-Interface (FMI) (*86*), resolve this by providing an Application-Programmer Interface (API) that specifies the exact information that should be shared between models (*42*). However, such rigid interfaces result in brittle systems that, while functioning for the systems they are immediately defined for, often fail when the behavior of the SOI changes (*48*).

In contrast to imperative frameworks, a CHG enables simulation sequences to be autonomously derived from the graphical structure by connecting nodes representing input variables with those representing output variables. The resulting sequence of hyperedges forms a rooted tree (*83*) composing a function for calculating the desired outputs (*87*). One can think of a CHG as encoding all possible ways a simulation could be composed as paths through the graph.

Supplemental Information 3: Handling Uncertainty with Constraint Hypergraphs

Uncertainty is inherent and omnipresent in representing reality (*88*), making how to reconcile with uncertainty an essential question for any modeling endeavor (*89*). As a modeling framework, a CHG neither adds nor removes information to models adhering to its formalisms. However, deconstructing model elements into the nodes and edges of a CHG can provide significant insight into the uncertainty pertaining to a system representation. The tools provided in understanding uncertainty are best understood when framed against the three types of uncertainty described by Isukapalli et al. (*90*): natural variability, model uncertainty, and data uncertainty. This section describes each of these as well as how they can be expressed and understood within the syntax of a CHG.

Background

Natural variability is the uncertainty in a model due to the imperfect approximation of physical phenomenon. As described in Supplementary Information 1, information is a bounded, finite representation of the physical domain. Such discrete descriptions are inevitably insufficient for capturing the complex interactions of reality. For instance, while the connection between Brazilian butterflies and tornados in Texas could be identified (*91*), expressing the interaction as a direct relationship fails to identify the other features salient to the system's behavior. While the demonstrative relationship between butterflies and tornadoes is highly variable, similar variability is inherent in any considered model (though ideally on a reduced scale). The incongruence between the physical phenomena and its virtual description is intractable, meaning that it cannot be reduced by the efforts of a modeler (*92*); no matter how careful modeler measures the butterfly wing beats, the resulting model will fail to perfectly explain tornado formation. This is described in Figure 7, which shows an illustrative plot of a model predicting solar irradiance based on the time of year. The relationship between these two phenomena is stochastic rather than deterministic, and is described by the probability density function. This is shown as the color map of the plot where brighter colors are more likely to be measured as real values. No model will perfectly capture this stochastic distribution.

Distinct from natural variation is data uncertainty, which is associated with the fidelity between the inputs provided to the model and the state of the physical system they represent. Each input is a datum that represents the specific measurement of some real world phenomenon. Measurement error, resulting for instance from the resolution of a measuring device or process, forms a key aspect of data uncertainty. An example is given by the measurements of solar irradiance shown by the black Xs in Figure 7, which cannot be guaranteed to align in the exact position provided. The data uncertainty is described by the error bars describing the tolerance of each measurement.

The final form of uncertainty is model uncertainty, describing unknown information associated with the structure of the model. The formation of every model is based on assumptions. For instance, the interpolation of the measured data points in Figure 7 assumes that the solar irradiance follows a polynomial curve. Assumptions for calculating the energy production of a diesel generator might include the fuel being available and that the combustion cycle performs normatively. Each assumption is an aspect where the model might disagree with the real world, resulting in error. In addition to assumptions in model relationships, the modeler makes assumptions as to how data should be represented. While data uncertainty is the consideration of whether the diesel generator's fuel level is *full* or *empty*, model uncertainty is manifest as the assumption that fuel levels can only be *full* or *empty*. These two forms of uncertainty can be reduced and even eliminated through the efforts of the modeler (*93*), such as considering states with a higher degree of resolution or making more refined assumptions.

Expression in a Constraint Hypergraph

The structure of a CHG–its nodes and edges–provides an excellent framework for considering uncertainty in a simulation, particularly that attributed to data and model uncertainty. Nodal elements qualify a system's scope, while their values describe its resolution. Edges represent the assumptions inherent in the model that might need to be accepted to simulate specific information. More specific details can be found by describing how each form of simulation uncertainty is ex-

pressed by these two elements.

The fact that natural variability is not reducible relates to its inability to be represented in a model. It is instead the variation inherent in the form of representation selected by the modeler. As such it is not captured within the CHG, but it can be framed through use of the CHG. For instance, although the variability in the relationship shown in Figure 7 between the time of year and solar irradiance cannot be shown in the model, it can be estimated by comparing measured values against repeated simulations run over a sampling space (such as via Monte Carlo methods). These efforts benefit from a CHG, whose structure defines the explicit space for all simulations as well as their construction via the pathfinding measures described in the Methods section.

When an agent prescribes a value for a node they introduce information to the system from outside the model. This represents data uncertainty–the unknown accuracy of an assigned input. Like natural variability, data uncertainty is generally not captured explicitly in a model, as the information is derived external to the model's scope. However, the flexibility of a CHG allows quantified values of uncertainty, such as the tolerance stack of a measurement process or the confidence interval for a simulation, to be included as a node in the graph representing the specific datum. Treating uncertainty parameters as states in the system reveals the relationships between a model's uncertainty and its simulated outcomes–an important aspect of using models for decision-making and risk estimation.

Contrasted with the first two forms of simulation uncertainty, model uncertainty is intrinsically captured in the structure of the CHG. Nodes encode the set of all distinguishable ways some phenomena can be characterized. As such, each node provides the resolution of the model corresponding to the specific phenomena considered. Furthermore, the scope of the system is explicitly given by the collection of nodes pertaining to the CHG. Assumptions in the model are always made in reference to this scope. The microgrid model, for instance, contains no nodes relating to the states of transformers or inverters on the grid. This consequently limits the model's fidelity, or ability to capture the effects of these actors on the rest of the system. If the exclusion of such phenomena affects the output node's value significantly, then a method of recourse is to include a node in the graph that aggregates unconsidered phenomena into a single value that accounts for the noise between simulated and measured values (*45*).

Edges represent the relationships prescribed between phenomena. Descriptions of relationships are virtual arrangements rather than physical constructs, and are consequently based entirely on virtual assumptions of the behavior of the real world. These assumptions are implicitly given by each edge, such that conducting a simulation implies acceptance of the underlying assumptions for each edge in the simulation path. For instance, Figure 6 shows an obvious discrepancy between the simulated and observed outputs of the diesel generators in the validation study. This is due to the assumption present in the DT model that the discharge profile of the diesel generators was uniform, while the controller in the real-world test applied a non-linear reduction at the end of each discharge cycle. Like with data uncertainty, specific quantified values of uncertainty (such as comparisons to measured data) can be included as nodes in the graph, showing the relationships of uncertainty parameters to the simulated outputs.

Assumptions of the real world relate to data uncertainty as well, in that the modeler assumes that an input is appropriate within the validity frame of the model. This is distinct from verifying whether an inputted value lies in the domain of an edge in the simulation path–instead it is the assumption that the provided input accurately represents the corresponding phenomena. As with other assumptions, claiming an input to be appropriate can be expressed by an edge in the hypergraph. However, rather than relating two distinct nodes, assumptions on inputs are captured in loops: edges that have the same node as their source and target. This is the identity element in the category of a CHG, and its existence permits the expression of all modeling assumptions to be expressed solely by the edges in the graph.

Assumptions are the fundamental construct of any virtual representation, but are uniquely framed by a CHG. By representing system behavior as functions, relationships can be decomposed into chains of smaller edges. This allows each assumption to be isolated to a singular modeling construct. In addition to the increased precision in identifying assumptions, CHG models can be uniquely arranged to target or avoid specific assumptions depending on the needs of a modeler. Situations that require greater accuracy or execution time might prefer accepting different models. A CHG allows every simulation, with its inherent assumptions, to be compared by the modeler.

Supplemental Information 4: Digital Twin Composition with Constraint Hypergraphs

Figure 4 holistically describes the microgrid system. However, each collection of nodes within the greater CHG–along with the edges that relate them–form a sub-CHG, equally valid if not more limited in scope. These sub-CHGs often relate to specific components of the microgrid system, such as grid actors or transmission infrastructure, and can be consequently perceived as forming a DT of that identified subsystem. Integrating these subsystem DTs into an aggregate DT illustrates one of the greatest challenges with providing DTs: interoperability (*42*). Interoperability is an agent's ability to exchange information with another agent in a manner that preserves the meaning of the information (*94*). The case for why DTs need to be interoperable to usefully represent a system is based on two reasons: the first is that all systems are composed of systems and are themselves subsets of other systems. An adequate representation of such systems of systems requires a DT that can interface with other DTs (*14, 44*); e.g. a DT of a car with a DT of a tire, a tire DT with one of a road, etc. The second reason–known at least as early as

Heraclitus–is that all systems change. As a consequence, all DTs must be able to adapt when the behaviors or scope of the SOI evolves (*45, 46*).

Maintaining meaning in a DT is not as simple as prescribing a defined interface by which all prospective DT connections must abide. The microgrid is an excellent illustrative example, since the behavior of each grid actor is affected by the other actors to which it is connected. A rigid interface for a DT of a battery would need to define *a priori* how connecting to a diesel generator or utility grid would affect its output. This is only possible if the behavior of the DT is independent of any connected system. An independent system is typically encapsulated; because its behavior does not change, the only thing required for simulation is serialized messages concerning the state of connected objects, often passed along ports (*95*). Such *a priori* independence is an untenable requirement for DTs representing the reactive systems typical of reality. Key to this conundrum is understanding that connecting DTs does not form a system of independent systems, but rather a new aggregate system that consumes the interfacing models. Because of this, the problem of interoperability does not concern interface connections, but rather changes to the scope of the system representation. For two DTs to interoperate, their interactions must be captured in the aggregate system.

Forming an aggregate system is the result of either adding or removing information, the last of the three operations performed on persistent data. Because a CHG fully captures a system description, the types of changes typical of DT interoperability can be described by the possible modifications that can be made to a CHG. In this sense the term interoperability is equivalent to extensibility (*47*), as employed in the development of programming languages.

There are two entities in a CHG: edges and nodes. Adding or removing an edge in a CHG amounts to redefining the behavior of the represented system, e.g. modifying the relationship between solar irradiation and power supply, $f \coloneqq (\eta, I, a) \rightarrow P$ from $P = \eta I a$ to $P = \eta I a(1 - \eta)$ defines a new interaction between the domain (composed of the cartesian product of η, I and a) and codomain P. Because CHGs are function-based in the same sense as languages such as Haskell and pure LISP, behavioral modifications of a CHG's edges do not affect the consistency of the graph, a characteristic referred to as having no side effects (*30*). In other words, observations of a fact in the system are not affected by adding edges to the graph. The same is true for removing edges, as long as the observation is not exclusively dependent upon a simulated path along the removed edge. Additionally, a CHG's functional nature allows it to reveal a type of emergent behavior resulting from non-localized interactions. If the cost of running the microgrid is influenced by the power output of the photovoltaic array, and a separate model describes how that power output is affected by the day's weather, then combining the two graphs along the shared variable allows the interaction between weather on operational cost to emerge.

Behavioral interoperability–changes along a system's relationships–is the first type of interoperability. The second involves adding or removing system facts corresponding to the nodes in a CHG. This type of interoperability is less well-defined in functional models which struggle with adding new variants of information. This result is studied under the label the ''expression problem'' (*47,48*), with Wadler famously describing it as a table, where extending the rows requires fixing the columns and vice versa. Allowing both rows (behaviors) and columns (facts) to be extended without *a posteriori* modifications is famously difficult (*96*).

With CHGs, there are two potential issues with adding or removing nodes from the system. The first is highly tractable, in that adding or removing nodes disrupts the definition of any connected edge, which must be redefined to accommodate the new nodes. This problem is highly localized; because edges can be modified without side effects, the disruption of modifying a node is limited to its connected edges, and does not propagate further in the graph. The second issue requires additional research to allow CHGs to be ideally employed. It is that changing the considered scope of the model potentially invalidates the assumptions undergirding each model relationship. For example, the relationship calculating the maximum power generation capacity of the microgrid might be defined as summing the individual capacities of each grid actor. Such a relationship might be based on the assumption that all actors capable of generating power form part of the domain of the edge. This assumption is valid in the scope of the system as defined. However, if the scope changes to include, for example, a wind turbine on the microgrid, the assumption is invalidated, and the demonstrative edge would need to be modified by someone *a posteriori* in order to accommodate the proposed scope change. There are several potential solutions to this problem (reviewed in (*47*)), although additional research needs to be directed towards this topic to identify one that maximizes a CHG's utility as a DT representation framework.

References

1. International Organization for Standardization, Systems and Software Engineering - System Life Cycle Processes (2023), https://www.iso.org/standard/81702.html.
2. R. Sahal, S. H. Alsamhi, K. N. Brown, Personal Digital Twin: A Close Look into the Present and a Step Towards the Future of Personalised Healthcare Industry. *Sensors* **22** (15), 5918 (2022), doi:10.3390/s22155918.
3. J. Hoffmann, *et al.*, Destination Earth – A Digital Twin in Support of Climate Services. *Climate Services* **30**, 100394 (2023), doi:10.1016/j.cliser.2023.100394.
4. Y. Fu, G. Zhu, M. Zhu, F. Xuan, Digital Twin for Integration of Design-Manufacturing-Maintenance: An Overview. *Chinese Journal of Mechanical Engineering* **35** (1), 80 (2022), doi:10.1186/s10033-022-00760-x.
5. J. Koskinen, H. Lahtonen, T. Tilus, *Software Maintenance Cost Estimation and Modernization Support*, Tech. rep., Information Technology Research Institute (2003), https://static.aminer.org/pdf/PDF/000/364/724/estimating_the_costs_of_software_maintenance_tasks.pdf.
6. V. Zambrano, *et al.*, Industrial Digitalization in the Industry 4.0 Era: Classification, Reuse and Authoring of Digital Models on Digital Twin Platforms. *Array* **14**, 100176 (2022), doi:10.1016/j.array.2022.100176.

7. P. Pileggi, A. Bujari, O. Barrowclough, J. Haenisch, R. Woitsch, *Overcoming Digital Twin Barriers for Manufacturing SMEs*, Tech. rep., Change2Twin (2021), https://www.change2twin.eu/about/download/.
8. X. Wang, *et al.*, How a Vast Digital Twin of the Yangtze River Could Prevent Flooding in China. *Nature* **639** (8054), 303–305 (2025), doi:10.1038/d41586-025-00720-0.
9. D. L. Van Bossuyt, *et al.*, The Future of Digital Twin Research and Development. *Journal of Computing and Information Science in Engineering* **25** (8), 080801 (2025), doi:10.1115/1.4068082.
10. A. Budiardjo, D. Migliori, *Digital Twin System Interoperability Framework*, Tech. rep., Digital Twin Consortium (2021), https://www.digitaltwinconsortium.org/wp-content/uploads/sites/3/2022/06/Digital-Twin-System-Interoperability-Framework-12072021.pdf.
11. X. Wang, X. Hu, Z. Ren, T. Tian, J. Wan, Knowledge-Graph-Based Multi-Domain Model Integration Method for Digital-Twin Workshops. *The International Journal of Advanced Manufacturing Technology* **128** (1-2), 405–421 (2023), doi:10.1007/s00170-023-11874-4.
12. S. Boschert, C. Heinrich, R. Rosen, Next Generation Digital Twin, in *Proceedings of TMCE 2018* (University of Technology, Delft, Las Palmas de Gran Canaria, Spain) (2018), https://www.researchgate.net/publication/325119950_Next_Generation_Digital_Twin.
13. X. Zheng, J. Lu, D. Kiritsis, The Emergence of Cognitive Digital Twin: Vision, Challenges and Opportunities. *International Journal of Production Research* **60** (24), 7610–7632 (2022), doi:10.1080/00207543.2021.2014591.
14. A. Ricci, A. Croatti, S. Mariani, S. Montagna, M. Picone, Web of Digital Twins. *ACM Transactions on Internet Technology* **22** (4), 101:1–101:30 (2022), doi:10.1145/3507909.
15. National Academies of Sciences, Engineering, and Medicine, *Foundational Research Gaps and Future Directions for Digital Twins* (The National Academies Press, Washington, D.C.) (2024), doi:10.17226/26894.
16. International Organization for Standardization, Automation Systems and Integration — Digital Twin Framework for Manufacturing (2021), https://www.iso.org/obp/ui/en/#iso:std:iso:23247:-1:ed-1:v1:en.
17. J. Osho, *et al.*, Four Rs Framework for the Development of a Digital Twin: The Implementation of Representation with a FDM Manufacturing Machine. *Journal of Manufacturing Systems* **63**, 370–380 (2022), doi:10.1016/j.jmsy.2022.04.014.
18. M. Drobnjakovic, *et al.*, Towards Ontologizing a Digital Twin Framework for Manufacturing, in *Advances in Production Management Systems. Production Management Systems for Responsible Manufacturing, Service, and Logistics Futures* (Springer, Cham, Trondheim, NO), vol. 689 of *IFIPAICT* (2023), pp. 317–329, doi:10.1007/978-3-031-43666-6_22.
19. R. Minerva, N. Crespi, Digital Twins: Properties, Software Frameworks, and Application Scenarios. *IT Professional* **23** (1), 51–55 (2021), doi:10.1109/MITP.2020.2982896.
20. ITU-T, Digital Twin Network - Requirements and Architecture (2022), https://www.itu.int/rec/T-REC-Y.3090-202202-I.
21. F. Longo, *et al.*, Distributed Simulation for Digital Twins: An Application to Support the Autonomous Robotics for the Extended Ship, in *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2022), pp. 179–186, doi:10.1109/DS-RT55542.2022.9932057.
22. J. Akroyd, S. Mosbach, A. Bhave, M. Kraft, Universal Digital Twin - A Dynamic Knowledge Graph. *Data-Centric Engineering* **2**, e14 (2021), doi:10.1017/dce.2021.10.
23. M. Waszak, *et al.*, Let the Asset Decide: Digital Twins with Knowledge Graphs, in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)* (2022), pp. 35–39, doi:10.1109/ICSA-C54293.2022.00014.
24. M. G. Kapteyn, J. V. R. Pretorius, K. E. Willcox, A Probabilistic Graphical Model Foundation for Enabling Predictive Digital Twins at Scale. *Nature Computational Science* **1** (5), 337–347 (2021), doi:10.1038/s43588-021-00069-0.
25. A. Chatterjee, C. Helbig, R. Malak, A. Layton, A Comparison of Graph-Theoretic Approaches for Resilient System of Systems Design. *Journal of Computing and Information Science in Engineering* **23** (030906) (2023), doi:10.1115/1.4062231.
26. A. Hogan, M. Cochez, G. de Melo, *Knowledge Graphs*, no. 22 in Synthesis Lectures on Data, Semantics and Knowledge (Springer, Cham) (2022).
27. Michael. G. Kapteyn, D. J. Knezevic, D. B. P. Huynh, M. Tran, Karen. E. Willcox, Data-Driven Physics-Based Digital Twins via a Library of Component-Based Reduced-Order Models. *International Journal for Numerical Methods in Engineering* **123** (13), 2986–3003 (2022), doi:10.1002/nme.6423.
28. K. Pedersen, J. Emblemsvåg, R. Bailey, J. Allen, F. Mistree, Validating Design Methods & Research: The Validation Square, in *DETC 2000 ASME Design Engineering Technical Conferences* (ASME, Baltimore, MD) (2000).
29. H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, no. 103 in Studies in Logic and the Foundations of Mathematics (North-Holland, Amsterdam New York Oxford) (1981).
30. B. J. MacLennan, *Functional Programming: Practice and Theory* (Addison-Wesley, Reading, Mass) (1990).
31. Digital Twin Consortium, Definition of a Digital Twin (2020), https://www.digitaltwinconsortium.org/initiatives/the-definition-of-a-digital-twin.htm.
32. M. Grieves, *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*, Tech. rep., Dassault Systèmes (2014).
33. M. Shafto, *et al.*, *Draft Modeling, Simulation, Information Technology & Processing Roadmap*, Tech. Rep. TA11-27, National Aeronautics and Space Administration, Washington DC (2010), https://www.nasa.gov/pdf/501321main_TA11-MSITP-DRAFT-Nov2010-A1.pdf.
34. A. Villalonga, *et al.*, A Decision-Making Framework for Dynamic Scheduling of Cyber-Physical Production Systems Based on Digital Twins. *Annual Reviews in Control* **51**, 357–373 (2021), doi:10.1016/j.arcontrol.2021.04.008.
35. L. Floridi, *Information: A Very Short Introduction*, Very Short Introductions (Oxford University Press, Oxford) (2010).
36. J. C. Willems, The Behavioral Approach to Open and Interconnected Systems. *IEEE Control Systems Magazine* **27** (6), 46–99 (2007), doi:10.1109/MCS.2007.906923.
37. F. E. Cellier, *Continuous System Modeling* (Springer, New York, NY) (1991), doi:10.1007/978-1-4757-3922-0.
38. F. E. Cellier, E. Kofman, *Continuous System Simulation* (Springer, New York) (2006), doi:10.1007/0-387-30260-3.
39. E. A. Lee, Determinism. *ACM Trans. Embed. Comput. Syst.* **20** (5), 38:1–38:34 (2021), doi:10.1145/3453652.
40. J. Morris, G. Mocko, J. Wagner, Unified System Modeling and Simulation via Constraint Hypergraphs. *Journal of Computing and Information Science in Engineering* **25** (6), 061005 (2025), doi:10.1115/1.4068375.
41. C. H. Duarte, *Proof-Theoretic Foundations for the Design of Extensible Software Systems*, Ph.D. thesis, University of London, London (1998), doi:10.13140/RG.2.2.29584.46080.
42. V. Piroumian, Digital Twins: Universal Interoperability for the Digital Age. *Computer* **54** (1), 61–69 (2021), doi:10.1109/MC.2020.3032148.
43. M. Zenger, *Programming Language Abstractions for Extensible Software Components*, Ph.D. thesis, EPFL, Lausanne, Switzerland (2004), doi:10.5075/epfl-thesis-2930.
44. J. Michael, J. Pfeiffer, B. Rumpe, A. Wortmann, Integration Challenges for Digital Twin Systems-of-Systems, in *Proceedings of the 10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, SESoS '22 (Association for Computing Machinery, New York, NY, USA) (2022), pp. 9–12, doi:10.1145/3528229.3529384.

45. D. J. Wagg, K. Worden, R. J. Barthorpe, P. Gardner, Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications [Special Section]. *ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg* **6** (3), 030901:1–030901:18 (2020), doi:10.1115/1.4046739.

46. P. G. Carlock, R. E. Fenton, System of Systems (SoS) Enterprise Systems Engineering for Information-Intensive Organizations. *Systems Engineering* **4** (4), 242–261 (2001), doi:10.1002/sys.1021.

47. M. Zenger, M. Odersky, Independently Extensible Solutions to the Expression Problem, in *Foundations of Object-Oriented Languages (FOOL 2005)* (École Polytechnique Fédérale de Lausanne, Long Beach, CA, USA) (2005), https://homepages.inf.ed.ac.uk/wadler/fool/program/10.html.

48. S. Krishnamurthi, M. Felleisen, D. P. Friedman, Synthesizing Object-Oriented and Functional Design to Promote Re-Use, in *Proceedings of the 12th European Conference on Object-Oriented Programming* (Springer-Verlag, Berlin, Heidelberg), vol. 1445 of *ECCOP '98* (1998), pp. 91–113, doi:10.1007/BFb0054088.

49. J. C. Reynolds, User-Defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction, in *Programming Methodology: A Collection of Articles by Members of IFIP WG2.3*, D. Gries, Ed. (Springer, New York, NY), pp. 309–317 (1978), doi:10.1007/978-1-4612-6315-9_22.

50. P. Hudak, Conception, Evolution, and Application of Functional Programming Languages. *ACM Comput. Surv.* **21** (3), 359–411 (1989), doi:10.1145/72551.72554.

51. R. Lasseter, *et al.*, *Integration of Distributed Energy Resources: The CERTS Microgrid Concept*, Consultant Report LBNL–50829, 799644, Consortium for Electric Reliability Technology Solutions, Berkeley, CA, USA (2002), doi:10.2172/799644.

52. A. Chatterjee, A. Bushagour, A. Layton, Resilient Microgrid Design Using Ecological Network Analysis, in *The Proceedings of the 2023 Conference on Systems Engineering Research*, D. Verma, A. M. Madni, S. Hoffenson, L. Xiao, Eds. (Springer Nature Switzerland, Cham), pp. 603–617 (2024), doi:10.1007/978-3-031-49179-5_41.

53. W. W. Anderson Jr, D. L. Van Bossuyt, Foundations of Microgrid Resilience, in *Microgrids* (John Wiley & Sons, Ltd), chap. 28, pp. 655–679 (2024), doi:10.1002/9781119890881.ch27.

54. R. Alves, D. Van Bossuyt, Spanagel Microgrid Experimental Run (2025), doi:10.5281/zenodo.15675037.

55. O. G. A. Oliveros, *Test Model for Power Distribution on U.S. Naval Installations*, Ph.D. thesis, Naval Postgraduate School, Monterey, CA (2024), https://hdl.handle.net/10945/73198.

56. D. Reich, L. Frye, Microgrid Planner: An Open-Source Software Platform. *INFORMS Journal on Computing* (2024), doi:10.1287/ijoc.2023.0336.

57. C. J. Peterson, *Systems Architecture Design and Validation Methods for Microgrid Systems*, Master's thesis, Naval Postgraduate School, Monterey, CA (2019), https://hdl.handle.net/10945/63493.

58. A. W. Wymore, *Model-Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, Systems Engineering Series (CRC Press, Boca Raton, Fla.) (1993).

59. B. P. Zeigler, A. Muzy, E. Kofman, *Theory of Modeling and Simulation* (Elsevier), third ed. (1976), doi:10.1016/B978-0-12-813370-5.00002-X.

60. J. Morris, ConstraintHg (2024), doi:10.5281/zenodo.15278018.

61. National Renewable Energy Laboratory, National Solar Radiation Database (NSRDB) (2012), https://nsrdb.nrel.gov/data-viewer.

62. S. Wilcox, *National Solar Radiation Database 1991–2010 Update: User's Manual*, Technical Report NREL/TP-5500-54824, National Renewable Energy Laboratory, Golden, CO, USA (2012), https://docs.nrel.gov/docs/fy12osti/54824.pdf.

63. G. Ausiello, R. Giaccio, G. Italiano, U. Nanni, *Optimal Traversal of Directed Hypergraphs*, ICSI Technical Report ICSI TR-92-073, International Computer Science Institute, Berkeley, CA (1992), https://www.icsi.berkeley.edu/icsi/publication_details?n=778.

64. J. Morris, MicrogridHg (2025), https://github.com/jmorris335/MicrogridHg.

65. International Organization for Standaridization, Digital Twin: Concepts and Terminology (2023), https://www.iso.org/standard/81442.html.

66. AIAA Digital Engineering Integration Committee, *Digital Twin: Definition and Value*, Tech. rep., American Institute of Aeronautics and Astronautics and Aerospace Industries Association (2020), https://www.aiaa.org/docs/default-source/uploadedfiles/issues-and-advocacy/policy-papers/digital-twin-institute-position-paper-(december-2020).pdf.

67. M. W. Grieves, Virtually Intelligent Product Systems: Digital and Physical Twins, in *Complex Systems Engineering: Theory and Practice* (American Institute of Aeronautics and Astronautics, Inc.), vol. 256 of *Progress in Astronautics and Aeronautics*, pp. 175–200 (2019), doi:10.2514/5.9781624105654.0175.0200.

68. R. D. D'Amico, S. Addepalli, J. A. Erkoyuncu, Industrial Insights on Digital Twins in Manufacturing: Application Landscape, Current Practices, and Future Needs. *Big Data and Cognitive Computing* **7** (3), 126 (2023), doi:10.3390/bdcc7030126.

69. S. P. A. Datta, Emergence of Digital Twins - Is This the March of Reason? *Journal of Innovation Management* **5** (3), 14–33 (2017), doi:10.24840/2183-0606_005.003_0003.

70. G. Shao, J. Hightower, W. Schindel, Credibility Consideration for Digital Twins in Manufacturing. *Manufacturing Letters* **35**, 24–28 (2023), doi:10.1016/j.mfglet.2022.11.009.

71. A. Hakiri, A. Gokhale, S. B. Yahia, N. Mellouli, A Comprehensive Survey on Digital Twin for Future Networks and Emerging Internet of Things Industry. *Computer Networks* **244**, 110350 (2024), doi:10.1016/j.comnet.2024.110350.

72. K. Y. H. Lim, P. Zheng, C.-H. Chen, A State-of-the-Art Survey of Digital Twin: Techniques, Engineering Product Lifecycle Management and Business Innovation Perspectives. *Journal of Intelligent Manufacturing* **31** (6), 1313–1337 (2020), doi:10.1007/s10845-019-01512-w.

73. C. Human, A. H. Basson, K. Kruger, A Design Framework for a System of Digital Twins and Services. *Computers in Industry* **144**, 103796 (2023), doi:10.1016/j.compind.2022.103796.

74. J. Lussier, *et al.*, Differentiating Between Digital Twins and Control Systems for Complex Systems. *[Manuscript submitted for review in the Journal of Computing and Information Science in Engineering]* (2025).

75. J. Morris, G. Mocko, J. Wagner, S. Ramnath, Declarative Integration of CAD Software into Multi-Physics Simulation via Constraint Hypergraphs, in *Proceedings of the ASME 2025 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (ASME, Anaheim, CA) (2025).

76. F. Rossi, P. van Beek, T. Walsh, Constraint Programming, in *Foundations of Artificial Intelligence*, F. van Harmelen, V. Lifschitz, B. Porter, Eds. (Elsevier), vol. 3 of *Handbook of Knowledge Representation*, pp. 181–211 (2008), doi:10.1016/S1574-6526(07)03004-0.

77. G. J. Friedman, C. T. Leondes, Constraint Theory, Part II: Model Graphs and Regular Relations. *IEEE Transactions on Systems Science and Cybernetics* **5** (2), 132–140 (1969), doi:10.1109/TSSC.1969.300204.

78. G. J. Friedman, P. Phan, *Constraint Theory*, vol. 23 of *IFSR International Series on Systems Science and Engineering* (Springer International Publishing, Cham) (2017), doi:10.1007/978-3-319-54792-3.

79. R. Peak, C. Paredis, D. Tamburini, S. Waterbury, *The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs)*, Technical Report, National Aeronautics and Space Administration (2005), https://www.eislab.gatech.edu/projects/nasa-ngcobs/COB_Requirements_v1.0.pdf.

80. S. Friedenthal, A. Moore, R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language* (Elsevier, Waltham), 3 ed. (2015).

81. C. Berge, *Graphs and Hypergraphs*, no. 6 in North-Holland Mathematical Library (North-Holland Pub. Co., Amsterdam, New York) (1973).

82. G. Ausiello, L. Laura, Directed Hypergraphs: Introduction and Fundamental Algorithms—A Survey. *Theoretical Computer Science* **658** (Part B), 293–306 (2017), doi:10.1016/j.tcs.2016.03.016.

83. A. Bretto, *Hypergraph Theory: An Introduction*, Mathematical Engineering (Springer International Publishing, Heidelberg) (2013), doi:10.1007/978-3-319-00080-0.

84. C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-Simulation: A Survey. *ACM Computing Surveys* **51** (3), 49:1–49:33 (2018), doi:10.1145/3179993.

85. C. Paredis, A. Diaz-Calderon, R. Sinha, P. Khosla, Composable Models for Simulation-Based Design. *Engineering with Computers* **17** (2), 112–128 (2001), doi:10.1007/PL00007197.

86. M. Wiens, T. Meyer, P. Thomas, The Potential of FMI for the Development of Digital Twins for Large Modular Multi-Domain Systems, in *Proceedings of the 14th International Modelica Conference* (Linköping, Sweden) (2021), pp. 235–240, doi:10.3384/ecp21181235.

87. J. Morris, G. Mocko, J. Wagner, Effects of Functional and Declarative Modeling Frameworks on System Simulation. *[Manuscript submitted for review to the ASME Journal of Dynamic Systems, Measurement and Control]* (2025).

88. G. E. P. Box, N. R. Draper, *Empirical Model-Building and Response Surfaces*, Wiley Series in Probability and Mathematical Statistics (Wiley, New York) (1987).

89. G. J. Klir, *Uncertainty and Information* (John Wiley & Sons, Inc, Hoboken, NJ, USA) (2006).

90. S. S. Isukapalli, A. Roy, P. G. Georgopoulos, Stochastic Response Surface Methods (SRSMs) for Uncertainty Propagation: Application to Environmental and Biological Systems. *Risk Analysis* **18** (3), 351–363 (1998), doi:10.1111/j.1539-6924.1998.tb01301.x.

91. E. N. Lorenz, *The Essence of Chaos*, The Jessie and John Danz Lectures (University of Washington Press, Seattle) (1993).

92. H.-J. Choi, D. L. Mcdowell, J. K. Allen, F. Mistree, An Inductive Design Exploration Method for Hierarchical Systems Design under Uncertainty. *Engineering Optimization* **40** (4), 287–307 (2008), doi:10.1080/03052150701742201.

93. A. Sinha, N. Bera, J. K. Allen, J. H. Panchal, F. Mistree, Uncertainty Management in the Design of Multiscale Systems. *Journal of Mechanical Design* **135** (1), 011008 (2012), doi:10.1115/1.4006186.

94. International Organization for Standardization, Information Technology-Cloud Computing-Interoperability and Portability (2017), doi:10.3403/30313036U.

95. P. Wegner, Concepts and Paradigms of Object-Oriented Programming. *SIGPLAN OOPS Mess.* **1** (1), 7–87 (1990), doi:10.1145/382192.383004.

96. P. Wadler, The Expression Problem (1998), https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt.