Smart Application Development for IoT Asset Management Using Graph Database Modeling and High-Availability Web Services

Holm Smidt
University of Hawai'i at Mānoa
hsmidt@hawaii.edu

Matsu Thornton
University of Hawai'i at Mānoa
matsut@hawaii.edu

Reza Ghorbani
University of Hawai'i at Mānoa
rezag@hawaii.edu

Abstract

The rapid transition from purely physical or purely virtual systems, as we know them, to increasingly interconnected cyber-physical systems with high integration of the Internet-of-Things demands a paradigm shift in the development of information systems–smart applications–for the planning and operation of these systems. To address the demand of managing the integration of connected devices and enabling new business models from the heavily interconnected systems, current architectural reference models were considered and components of each synthesized into a proposed software stack for smart application development. This work lends its implementation approach to the utility of graph theory in modeling complex systems, and implements a graph database for managing and maintaining connected components that emphasize each component's virtual and physical connectivity, technical functionalities, and state. The graph database microservice is then integrated with a highly available web framework and communication broker service in a multi-layered software framework to integrate Internet-of-Things devices and make services available over the web. The framework's–and respective components'–feasibility and utility is demonstrated through a use case for modeling, connecting, and controlling interconnected homes in a modern smart grid, and abstracting transactional device data for new business models, such as demand response ancillary services.

1. Introduction

Graph databases are grounded in graph theory, a proven tool for modeling complex, highly interconnected systems, e.g. computer systems, biological systems, and social network systems, that uses graph structures such as nodes, edges, and labels. Graph-theoretic approaches to system modeling allow emphasis on component interactions and interconnections rather than device level logic. One particularly interesting and fitting application for this approach is in power grid modeling, where physical power lines are viewed as the connections (edges) between power grid components (nodes). These applications range from pure topological modeling (see [1, 2, 3]) to extended topological methods that integrate power flow considerations to conventional network science modeling techniques (e.g. [4]) for grid robustness analysis (e.g. [5, 6, 7]) and system design (e.g. [8, 9, 10]).

Power grid systems are considered an integral part of modern society. Their robustness and efficiency not only impact our daily lives, but also influence economy, politics, and the global environment [11]. Power grids continually evolve over time to accommodate and promote societal developments. One of which being the transition of transmission grids from traditional centralized utility-based power generation to integrating distributed energy resources (DER) such as photovoltaic (PV) systems, distributed energy storage (DES) such as battery systems, and demand response (DR). With the additional push for the integration of smart Internet-of-Things (IoT) devices, the smart grid is continually evolving into an increasingly interconnected cyber-physical system (CPS) that requires a paradigm shift in both its planning and operation [12, 13]. The use of graph databases as a sophisticated tool for modeling the connection between system components–transformers, distribution lines, DER, DES, IoT devices, grid operators, electricity consumers, etc.–can aid in managing the rapidly evolving smart grid [14].

This shift is driven by the need to move from antiquated grid systems that are reliant upon conventional generators whose control systems are completely described by relatively simple closed feedback control–steam valves and other governors–compared to the developing smart grid which is characterized by a vast and distributed network

URI: http://hdl.handle.net/10125/50614
ISBN: 978-0-9981331-1-9

of intermittent renewable generation and control points. Advances in network coordinated computing, communication, and mathematical modeling techniques are enabled by far improved processor and software technology. New algorithms which can take advantage of these improvements have been developed for the optimization of combined edge and centralized IoT device networks and make upgrades to these distributed systems available.

The power grid system is only one of many systems that are integral to society and undergoing a fundamental paradigm shift driven by the development of smart applications with integration of IoT devices. Other domains include, but are not limited to e-healthcare, smart cities, and smart factories. Given the vast interest in such smart applications, groups of researchers have started developing architectural reference models for integrating and connecting new technologies to enable new business models. These models can be specific to their application fields such as the smart grid [15] and smart factories [16]. [17, 18] discuss applications of these models that can be applied generally and are overarching across several application fields. Proposed models aim to not only establish standards to ensure interoperability but also provide guidance for software developers and architects in the design of smart applications. A common theme in proposed architectural models is the goal of developing new business models that can leverage the interconnectivity of assets (e.g. physical components, IoT devices, stakeholders).

Given the challenge of managing a vast variety and high number of components in typical CPS, and the demand of economic pressures, our work makes use of graph databases to develop applications which will add value to existing and planned for assets by leveraging connectivity. The further integration of concurrency-oriented web frameworks addresses the challenge of developing a single system for near real-time machine-to-machine and human-computer interaction. Specifically, the integration of connected homes to the smart grid is considered as a use case, to address the demand for modeling and managing connected components in smart applications with high proportions of IoT devices, and thus test the overall feasibility of the proposed framework. The proposed implementation uses neo4j for its graph database [19], VerneMQ for its communications broker for providing a publish/subscribe communication infrastructure for IoT sensors and actuators [20], and the Phoenix framework for web service development including soft real-time IoT event processing, IoT device management (monitoring and control), data querying, and access control [21]. This work focuses on the framework's utility for challenges in smart grid applications, namely device management and control, and system state monitoring for residential DR ancillary services.

This paper is structured as follows: Section 2 introduces related work on smart grid simulation frameworks for IoT integrated systems, as well as reference model architectures. Section 3 outlines the proposed system components in our implementation and Section 4 illustrates its use for modeling connected homes in the smart grid and simulating DR event communications. Results and future work have then been detailed in Sections 5 and 6 respectively.

2. Related work

2.1. ICT simulation platform for the smart grid

Simulation platforms tailored to connected devices in power grids are presented in [22, 23, 24] showing that information and communication technology (ICT) solutions can leverage the IoT for energy management in smart cities. Simulations such as these are integral in providing grid operators and policy makers with the data they need to make informed decisions with confidence. Further described in [25] is a simulation testbed that leverages PSIM, an existing power modeling software [26], by connecting existing tools to network connected devices in order to test advanced optimization algorithms against simulated grid events with both real and simulated device nodes.

[22] proposed an ICT framework for smart homes that addressed requirements that other frameworks for embedded devices were lacking; that is, concurrent, multi-user support through the Web, uniform access to heterogeneous embedded devices, and acceptable performance to name a few, were implemented by using a web-oriented application framework. More specifically, a RESTful application framework consisting of a device layer for embedded device management and control, a control layer as the central processing unit of the system, and a presentation layer to dynamically generate representations of available services to the Web.

[23] similarly stresses the use of a multi-layered software infrastructure to ensure interoperability of heterogeneous devices. A publish/subscribe mechanism is employed where sensors can publish to their respective topics and then store data in a relational database system (RDBS), which is organized in ten tables for device data, list of deployed devices, network information, and more.

[24] focused on interrelating historical device data,

building information models, geographical information systems, and system information models to lower energy consumption at the district level in smart cities. Each of the four components is comprised of its own datastore and is made available through RESTful interfaces to other system services. The infrastructure further implements both request/response (REST) and publish/subscribe (Message Queue Telemetry Transport (MQTT) [27]) communication approaches to interface with heterogeneous sensors (the integration layer). Ultimately, a simulation engine service can leverage the described services and communication approaches to define new control policies and test these policies through simulations with various devices publishing and subscribing to topics in soft real-time. The services are made available to users through the application layer.

Existing simulation infrastructures showed the value of ICT systems in energy grids; yet, they were designed as islanded systems focused on specific optimization problems.

2.2. Reference architectures

Ongoing developments of architectural reference models are indicative of the many challenges faced when developing interoperable smart applications. Work by [15, 16, 17, 18] represent a selection of models that have been in recent development and are relevant to smart application architectures. These models are being developed by different entities in different countries but share the common goal of providing guidance in realizing the needed paradigm shift in smart application domains characterized by a high integration of connected IoT components.

The architectural framework in development by the IEEE P2413 working group [17] as well as the Industrial Internet Reference Architecture (IIRA) [18] have been proposed as frameworks abstracted from the IoT domain, whereas the Reference Architecture Model Industrie 4.0 (RAMI4.0) [16] and the Smart Grid Architecture Model (SGAM) framework [15] have been specifically developed for the smart factory and smart grid domains, respectively.

By first describing IoT domains, defining domain abstractions, and identifying cross-domain commonalities, the IEEE P2413 is developing a reference model that highlights the relationships between the IoT domains. [17] uses a similar approach to [18] in developing a framework using the ISO/IEC/IEEE 42010:2011 Systems and Software Engineering–Architecture Description'. This involves framing the architecture based on stakeholders (individual, team, organization), concerns (topics of interest), viewpoints (conventions for construction, interpretation and use of architecture views to frame specific system concerns), and model kinds (conventions for capturing type of model), and representing it with architectural views and models.

IIRA, as described by [18], has formulated four viewpoints as the basis for expressing system concerns in Industrial IoT (IIoT) applications. These are: the business viewpoint (stakeholder's business vision), the usage viewpoint (concerns of system usage), the functional viewpoint (functional components), and the implementation viewpoint (technologies to implement functional viewpoint). The functional viewpoint is further defined by five distinct functional domains, the control, operations, information, application, and business domains. The control domain is closely connected to the physical system as it is typically comprised of the sensors and actuators interacting with the physical system. Work done by the Industrial Internet Consortium [18], shows how information is constantly exchanged, processed, and transformed within and between layers, but generally information becomes richer as it moves from the control domain up to the business domain that entails functions for end-to-end operations of IIoT systems. Parallels between the functional viewpoint in the IIRA and the RAMI4.0 and SGAM layers can easily be drawn for interoperability comparisons as described in [28], despite their in part parallel and islanded development. RAMI4.0 is a three-dimensional model to represent the Industrie 4.0 (I4.0) space that, like the IIRA, splits complex projects into clusters of manageable parts using a vertically multi-layered approach (business, functional, information, communication, integration, and asset layers) [16]. As RAMI4.0 reflects features of the SGAM, one can also find these layers in the SGAM model (with the exception that integration and asset layers in RAMI4.0 are grouped into a component layer). Both RAMI4.0 and SGAM layers can be mapped in their meaning to viewpoint layers and functional domains in the IIRA and vice versa (see [28]), as such, for the remainder of this paper, when referring to layers from one model, the same reference may be made to an analogous layer in another framework.

RAMI4.0 introduces the concept of administration shells that turn objects into I4.0 components. These administration shells describe the virtual representation and technical functionality of objects that are needed for integrating, managing, and operating objects (e.g. a machine in a smart factory). It is worth noting that administration shells can be separated from the actual component and agglomerated in a repository of administration shells, and that I4.0 components can

contain several objects (e.g. complete electrical axis system with controller, motors, sensors, etc.). Among a rich set of I4.0 component characteristics (see [18]) is the state model characteristic that informs on the latest state the component is in at a stated time.

In contrast to RAMI4.0, the SGAM is designed specifically for addressing challenges in the smart grid and thus defines domains and hierarchical zones (i.e. smart grid planes) for each of the vertical layers. The five horizontal domains are generation, transmission, distribution, DER, and customer premise, which allows for example the mapping of hardware components in the component layer and their functions in the function layer to these five domains. A residential PV system could thus be mapped vertically in the component layer and horizontally as a DER. The reader may refer to Section B.2.4 SGAM Mapping Example in [15] for further examples.

These described models are said to undergo ongoing changes as theoretical considerations for these architectures are refined with implementation use cases and evolving technology. The scope of these architectures by far extend beyond the scope of the work presented here; yet, select concepts were used as the basis for design.

2.3. Graph databases and the IoT

[29] presented the use of neo4j for real-time processing of IoT events in combination with Ejabberd and Apache spark. D'silva et al.'s work emphasized the interconnectivity of things and users and focused on a scalable and pluggable solution for processing requests from IoT devices and showed that graph databases are viable solutions for modeling IoT networks. [29] further pointed out that many operations in IoT systems are performed at real-time and require tools that can show connections quickly, and that many queries in graph databases outperform similar queries that would otherwise require JOIN operators in relational database queries. [30] presented the utility of graph databases for modeling relations in smart distribution systems and big data analytics.

The proposed work extends existing approaches by focusing on how graph structures can be used to model and implement complex CPS. It is a method which can abstract away complexity from vast webs of interconnected features to create a useful structure that can help to bring clarity to system architects and provide the actual control and management mechanisms. Consider the implications of IoT in application. Systems are providing a vast amount of telemetry data from a multitude of distributed resources and banks of historical data, while the outputs of the system include both virtual interaction and actuation as well as physical interactions.

To give a physical context, consider a smart grid. Say that there are inputs to the system which include physical telemetry which might give localized power demand, solar and other renewable generation availability; less volatile variables such as fuel costs; and analytics from other historical data. Then there are the outputs of the system which can also include a wide range of effects including physical actuation like demand response control, or virtual interaction such as financial transactions or even pure virtual manipulation of state variables within constituent components' own processing systems. The result is a complicated web of interaction where we can imagine any number of relationships between not only inputs and outputs, but also the intermediary effects of processing logic in between.

Graph theory gives us a way to keep track of these relationships in a relatively systematic fashion. In considering overall goals for a system which is so vast and complicated in interaction through the web of relationships we see the prospect for producing a viable method for control logic a daunting task. Taking our smart-grid example, we could list several plausible system goals such as maximizing system stability, maximizing profit for a utility, or minimizing use of fossil fuels. Most likely, it is a balance between several of these types of goals. A full system deployment would include optimization schema for the above tasks but is beyond the scope of the work presented here. We oversimplify the model to produce demonstration data and a demonstration model which is used to show the foundations of a viable system based on graph theory. We imagine that these bases can be built upon to produce a full scale and deployable model for interaction of IoT devices for wide scale optimization.

3. Implementation

Following established and proven practices in [22, 23, 24], a multi-layered software infrastructure is proposed as shown in Figure 1. The integration layer is composed of various smart devices which provide telemetry and control for the smart grid. The service layer is composed of a graph database (neo4j), the MQTT message broker (VerneMQ), and the Phoenix server backend for routing and control. Lastly, the Phoenix presentation layer provides views and applications for user interaction. The following discussion is limited to the implementation of the service layer, as this layer provides the main

functionality in this framework. The service layer

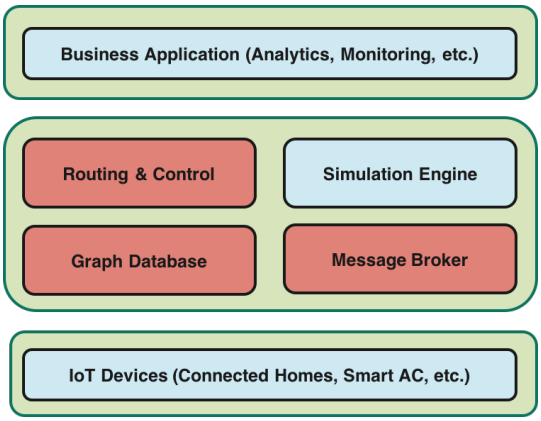

Figure 1. Layered software architecture with integration, service, and application layers.

in Figure 1 was further ideated by the functional domains of the functional viewpoint presented in the IIRA framework. The message broker enables the communication between assets as described in the control domain in [18]. The graph database service provides functionalities of the IIRA operations layer such as management, monitoring, and diagnostics of devices. The graph database further implements the administration shell repository described by the RAMI4.0 model. Functions of devices and their virtual connectivity can simply be queried from the graph. The RAMI4.0 state model and the horizontal domains from the SGAM are similarly represented and stored in the graph as described in Section 3.1. The 'routing & control block' in Figure 1 refers to the web server's ability to process IoT events and provide application interfaces for described functionalities; this block draws concepts from the information and application domains in the IIRA functional viewpoint.

3.1. Graph database modeling

The CPS and its assets (physical components, IoT components, etc.) are modeled as one graph using the neo4j graph database, which implements the property graph model [31]. As such, the graph consists of nodes and relationships. Nodes are basic entities that can exist in and of themselves. Relationships connect exactly two nodes, the source node and the target node. Tokens are nonempty strings of Unicode characters; nodes can have sets of labels (one or more tokens) and relationships have exactly one relationship type (exactly one token). Both, nodes and relationships can have properties, which are key-value pairs (one or more tokens). Graph traversal describes how the graph database is being traveled or in other words the navigation through a graph to find paths. Figure 2 depicted an illustration of three nodes connected by three relationships. Each node has a different label (i.e. house, person, thermostat) and different sets of properties (i.e. name, address, communication protocols, etc.). Unlike with relational databases, nodes (with the same or different labels) don't need to have the same set of properties; that is, one person may have information on the type of devices he/she is using whereas another person does not need to have a *dev* property. Properties can be defined as strings, lists, or numeric datatypes as indicated by the properties of the `:CONTR_TEMP` relationship. Graph traversal can be illustrated in Figure 2: if one wanted to know the owner of the building the thermostat is in, one would need to follow the `:CONTR_TEMP` relationship from the thermostat node and then the `:OWNED_BY` relationship from the returned node.

Cypher, the query language used in neo4j provides declarative ways of querying the database using asci-art syntax. The above described traversal, for example, could be implemented in Cypher as

```
MATCH (t:Thermostat) - [:CONTR_TEMP]
    -> () - [:OWNED_BY] -> (p)
RETURN p.name, p.fon
```

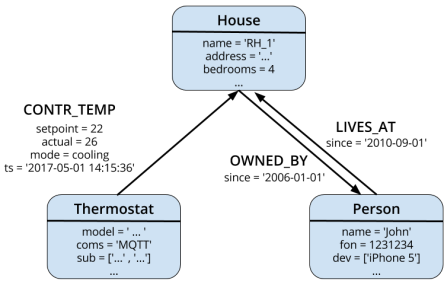

Figure 2. Sample graph illustrating concepts of property graph models in neo4j.

The syntax in the query allows the user to declare relationships between entities in the CPS model, without necessarily being interested in the node in between (i.e. the empty parentheses between the two relationships will match any node if they have the desired relationships). One thus builds the query natively based on the connections between nodes. This fundamentally different approach to leveraging interconnections between assets in smart applications could aid in addressing challenges in asset management. Using the Bolt protocol, a connection oriented network protocol over TCP connection integrated in neo4j, the graph database microservice can be made available securely and efficiently to other services.

To represent I4.0 administration shells, node properties are leveraged. These properties can be freely defined for any node, although maintaining conventions for using the same property keys for the same types of information is helpful, and used to express technical functions and virtual integration of components. Relationship properties are used for expressing the components' states by capturing the latest sensor updates and the timestamp of the last update. These states can either be sensor data published to specific topics, or commands sent by users to control actuators. States could also carry values computed periodically based on rules and conventions applicable for desired functionalities, e.g. periodically updated electricity price. Node labels and properties are further indicative of their domain in the overall smart grid as referenced in SGAM (e.g. nodes representing distribution lines, the distribution domain, can be easily differentiated from PV systems, the DER domain).

3.2. Message broker

The second microservice is the highly available MQTT message broker, that is built on the open source VerneMQ MQTT broker. Built on the Erlang/OTP (open telecom platform), a platform that has proven its concurrency model and fault tolerance in the operation of telecommunication networks, VerneMQ was specifically designed for soft real-time, distributed control and messaging applications while providing fault-isolation and fault-tolerance [20]. A VerneMQ plugin was developed to extend base functionality and use so-called VerneMQ hooks to send JSON API messages to the phoenix server when certain events occur (e.g. client_registration, on_publish, on_subscribe, etc.). These JSON API requests are then being processed by Phoenix to update the graph database.

For initial implementation, file-based authorization and authentication was implemented. Access control lists use patterns to define publish/subscribe access to topics based on user keys and topic structures. Usernames and passwords were added to the VerneMQ password file for authentication purposes. Both authentication and authorization can easily be implemented using databases in larger scale systems or production environments.

3.3. Web application

The web-application, developed using the Phoenix framework, was designed to implement the following subset of application functionalities:

- Connect to the graph database using the Bolt protocol;
- Process incoming requests from the VerneMQ plugin (REST API);
- Manage socket connections for subscribing and publishing to MQTT topics;
- Manage socket connections for querying the graph database.

The Phoenix framework follows a server-side MVC pattern and is known for bringing concurrency and functional programming to web application development [32]. Phoenix is built using the functional Elixir programming language and thus also runs on the Erlang VM [21]. The functional approach to web application development and use of concurrent lightweight Elixir processes (not operating-system processes) for handling real-time connections are especially well-suited for smart applications that require efficient socket connections for human-computer interactions (e.g. live updates for monitoring) as well as API-based machine-to-machine interactions (e.g. processing of IoT events). In the Phoenix framework, incoming requests are simply passed between layers and transformed at each step by groups of functions, so-called pipelines [32], making the handling of high numbers of API requests for IoT events possible.

Incoming VerneMQ API requests may be piped from the endpoint to the routing layer and from there sent to the controller, which parses and processes the JSON requests into Cypher queries. These queries can then be executed in neo4j using the Bolt protocol that is implemented using the Bolt.Sips Elixir driver [33]. This event-processing functionality can be mapped back to the concept of information processing in the information domain in the IIRA model. One may envision additional pipelines for requests besides updating the graph database, such as also storing sensor updates and control events in a database service. Other RESTful services currently implemented include HTML forms that can be used to test publishing of MQTT message to the message broker.

In addition to the request/response scheme, so-called channels are implemented for rich interactive functionalities through socket connection, e.g. visualization of graphs or continuous device control. In Phoenix channels, clients connect to specific channels and topics and can then send and receive messages, which could entail functions such as database querying or publishing of MQTT messages. The channel implementation thus addresses the demand for feature-rich asset management and simulations in smart applications.

4. Use case

4.1. Smart home modeling

The proposed framework's feasibility was tested by constructing a simplified model of several residential homes connected to several distribution lines. The different nodes in this CPS model are thus physical power grid components, buildings, gadgets, IoT devices, DERs, and human actors. Each residential home has a collection of gadgets (e.g. TV, air conditioner (AC), refrigerator), smart devices (e.g. smart AC), PV, battery storage, and a home energy management system (HEMS). The relationships between components describe physical or virtual connections between nodes, as illustrated in Figure 3. Properties of nodes and relationships are not shown in the model but select properties are summarized in Table 1.

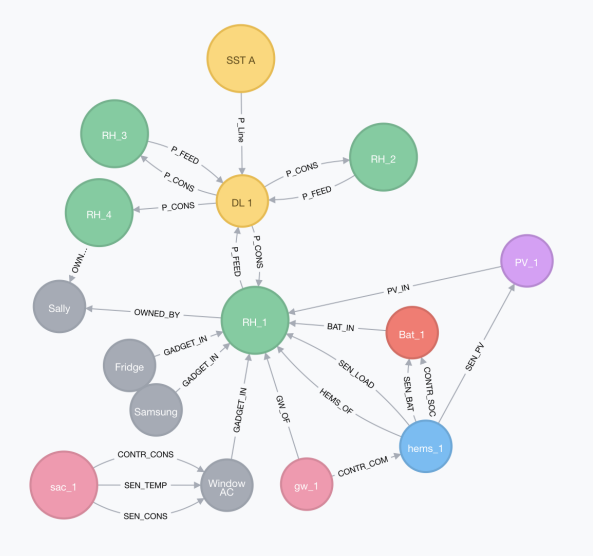

Figure 3. Sample model of a residential home with DER, DES, HEMS, and smart AC unit installed.

The shown sets of properties for nodes and relationships can be viewed as administration shells of each component as they inform on capabilities pertaining to asset management. The IoT device information, device configuration and network properties are stored in node properties, e.g. type, functions, coms, sub, and pub, which can always be updated, extended upon or streamlined using Cypher queries. The component statuses in this use case are further described by relationship properties, such as the temperature property of the `:SEN_TEMP` relationship. Data from an IoT device can be captured by and represented through the relationships between the device and its environment. This approach makes it efficient to query device data given that each relationship has exactly one relationship type, and enables the graph to be used for both asset management and capturing near real-time status of the CPS system.

Table 1. Select properties of nodes and relationships in the graph.

entity type	property key	sample property value
THING	name	sac_1
THING	type	smart_ac
THING	functions	['sense', 'control']
THING	coms	['MQTT']
THING	sub	['thing/HI/RH_1/smart_ac/827ebe0f622/control']
SEN_TEMP	temperature	24
SEN_TEMP	timestamp	'2017-05-01 13:00:05'
SEN_BAT	soc	75
SEN_BAT	timestamp	'2017-05-01 13:02:41'
P_FEED	wattage	2500
P_FEED	timestamp	'2017-05-01 13:05:41'

An illustrative example is the smart AC unit, `sac_1` that has the capabilities of monitoring temperature and power, and regulating the power consumption of the AC unit. It follows that there exist three relationships connecting the smart AC device to the actual AC gadget. Further, the `sac_1`'s node properties show that the node can communicate over MQTT and that the device publishes to the following MQTT topics:
thing/HI/RH_1/smart_ac/b827ebe0f622/sense/power
thing/HI/RH_1/smart_ac/b827ebe0f622/sense/temp

The properties of the two relationships `:SEN_CONS`, and `:SEN_TEMP` then reflect the latest published values, in other words the state, of the `sac_1` device with their timestamps (see Table 1). Similarly, the `:SEN_BAT`, `:SEN_PV`, and `:SEN_LOAD` relationships of the HEMS node carry the state of the battery storage, PV generation, and residential energy consumption. To abstract the state of agglomerated system components, e.g. the total DER feeding back to the grid connected to substations SST A, one may use the following expressive Cypher query:

```
MATCH (:Res_Home) - [p:P_FEED] ->
    () <- [:P_Line] -
    (:Substation {name: 'SST A'})
WHERE p.ts >= timestamp()-600000
RETURN SUM(p.wattage)
```

The query first finds all `:P_FEED` relationships

going from residential homes to distribution lines connected to substation A, assigns these relationships the variable p, then filters those relationships that were updated within the last ten minutes, and ultimately returns the total power consumption described by these relationships.

This illustrates the utility that originates from emphasizing system connectivity in the system model. Rather than having to JOIN information from different tables, relationships are used to link heterogeneous system entities. Soft real-time system monitoring and management, as referenced in the operation domain in the IIRA, is thus simplified and enables possibly new approaches to DR management. The following query could for example be used to extract information of controllable, variable AC load in a neighborhood:

```
MATCH (n:THING) - [c:CONTR_CONS]
    -> (g:Gadget) WHERE c.status = 0)
MATCH (n) - [s:SEN_CONS] -> (g) -
    [:GADGET_IN] -> () <- [:P_CONS]
    - (:Distr_Line {name: 'DL 1'})
RETURN SUM(s.wattage)
```

In two steps, one can first identify the loads that are not being controlled yet and that indirectly connect to distribution line DL 1', and then sum the individual load measurements representing the AC load. The returned value can then be used for DR decision making in simulations.

4.2. Data flow testing

To test the graph model with the message broker and Phoenix-powered services, the IoT device integrations layer was added to the software infrastructure. This layer consisted of virtual nodes (i.e. simulated MQTT clients), and an actual HEMS system. To add any node to the system, its credentials must be registered with the MQTT broker. Virtual nodes, nodes for the sole purpose of simulating communication flows in the system, were simulated using mqtt-client libraries and node-red. Figure 4 showed an example of a virtual node created on an Ubuntu virtual machine running node-red. Simulated inputs and outputs, e.g. the Raspberry Pi Sense HAT simulator node, were used for system testing. This approach showed great efficacy in productivity and in very rapidly simulating IoT nodes in the application as it provides all needed options to vary data input, data output, IoT device type, and node credentials.

The HEMS system was integrated as an actual IoT asset that contained a sensor suite with access to

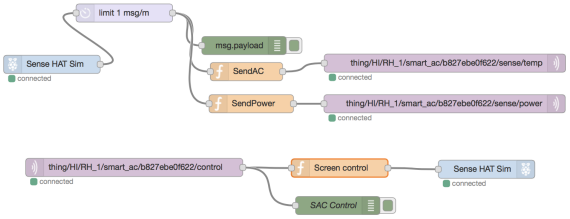

Figure 4. Node-red flow simulating IoT devices and testing the communication flow in the system infrastructure.

hundreds of points of telemetry data including solar power generation, battery state of charge, connected load data, frequency parameters, and grid feedthrough load. In addition to telemetry data, the HEMS controller allows access to control functions which include import/export of grid power, battery charge/discharge control, and controllable relay positions. Network connectivity was provided by a single board computer with network access to MODBUS registers. A Raspberry Pi gateway was used to read/write MODBUS registers and forward/receive messages over MQTT to the broker to update the graph model. Figure 5 showed the implementation of a node-red flow on the Raspberry Pi gateway. The gateway accessed the MODBUS on an internal wireless area network, read its registers, and then formatted and sent the data to the external message broker over MQTT.

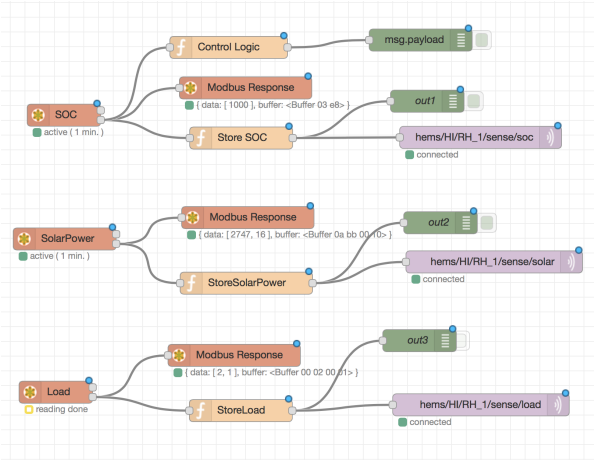

Figure 5. Node-red flow for reading HEMS data from the Modbus and sending it over MQTT to the broker.

To test the publishing of control events over MQTT from the server to the integrations layer, a front-end application view was integrated in the application layer of the Phoenix application, shown in Figure 6. Using

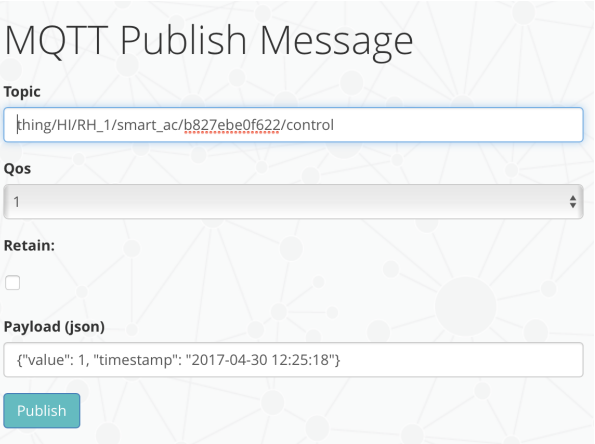

Figure 6. Front-end interface for publishing MQTT events from the Phoenix server.

forms, authorized users could publish control events to desired topics and thus start a DR event simulation. During event simulation, relevant publish/subscribe events were stored in a relational database.

5. Results

The use case design demonstrated the overall feasibility of the proposed framework by successfully integrating a graph database, a message broker, and a highly available server for smart grid network modeling and management. Implemented services integrated well together in updating graph models with device data, querying the database over the Bolt protocol, and sending control commands to the integration layer, thus providing desired functionalities for the given use case.

Small scale implementation of simulated and actual nodes showed high efficacy in polling the graph database for desired pub/sub topics of IoT devices, and listening/writing to these topics to update control commands. To rapidly scale the number of nodes in the system, one may either automate the node-red deployment with text updates for various flows (node-red flows are simply JSON objects), or utilize mqtt-client libraries for preferred scripting languages that pull simulated data and credentials from external files or databases and publish MQTT messages.

The Phoenix framework proved to be a productive and reliable tool in the development of web services for smart applications based on its code structure (ease of development) and use of concurrent lightweight Elixir processes for handling requests in a clean and functional approach. The native support for WebSocket connections in the form of channels with channel generators for increased productivity showed high utility for applications involving the monitoring of soft real-time system updates and the control of end-devices.

Combining the channel functionality with the use of the neo4j graph database allowed to perform relationship-based queries that would inform on agglomerated system states over many IoT devices. The further implementation of administrative shells for IoT components through node properties in the graph indicated high utility for systems with a wide range of components.

6. Conclusion and future work

This work introduced a software stack for modeling CPS using graph databases and using graph databases in conjunction with a highly available communication broker and web framework, and demonstrated its potential for addressing current challenges of IoT asset management and control that leverage and emphasize the interconnectivity in developing smart application domains. The framework's multi-layered approach was based on a variety of previously proposed frameworks, but extended these through the integration of modern concepts from architectural reference frameworks for smart application domains and concepts rooted in graph theory.

This paper detailed the implementation of the three main service components, graph database, communications broker, and web framework and illustrated their integration in a use case scenario. Results of this feasibility study showed promising utility in modeling CPS using graph databases as well as using the Phoenix framework for the development of web services for IoT integrated information systems. That is, the integration of the three main services in the proposed framework can facilitate human-to-machine interaction as well as machine-to-machine interaction, ease the administration of cyber and physical devices using administration shells, and thus ultimately enable new business strategies that leverage the interconnectivity of devices and actors in modern CPS.

To further show the systems utility, future work will focus on a performance test and the integration of different simulation engines that: a) can adequately model contingency events or edge cases which test robustness of a CPS control system using graph theoretical approaches, e.g. how are new business models, such as DR, effected by random node failure due to network intermittence, b) provide a WebSocket interface to integrating external internet hardware in the loop simulation systems, such as proposed in [25], and c) allow the simulation of system performance under various loads. For the scope of this paper,

discussion was also limited to the implementation of a MQTT message broker; yet, other important IoT communication protocols need to be considered for interoperability purposes. The authors anticipate the integration of CoAP [34], which will simply need additional routes, controllers, and channels for processing CoAP requests in the Phoenix server.

References

[1] A.-L. Barabasi, *Network science*. Cambridge, United Kingdom: Cambridge University Press, 2016.

[2] M. E. Newman, *Networks: An Introduction*. Oxford University Press, 2010.

[3] D. J. Watts and S. H. Strogatz, "Collective dynamics of'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[4] G. A. Pagani and M. Aiello, "The Power Grid as a complex network: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 11, pp. 2688–2700, 2013.

[5] R. Albert, I. Albert, and G. L. Nakarado, "Structural vulnerability of the North American power grid," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 69, no. 2 2, pp. 1–4, 2004.

[6] Z. Wang, A. Scaglione, and R. J. Thomas, "Electrical centrality measures for electric power grid vulnerability analysis," *Proceedings of the IEEE Conference on Decision and Control*, pp. 5792–5797, 2010.

[7] E. Bompard, R. Napoli, and F. Xue, "Analysis of structural vulnerabilities in power transmission grids," *International Journal of Critical Infrastructure Protection*, vol. 2, no. 1-2, pp. 5–12, 2009.

[8] I. Sayeekumar, K. Ahmed, P. Karthikeyan, K. Sah, and U. Raglend, "Graph Theory and its Applications in Power Systems -AReview," pp. 154–157, 2015.

[9] J. Quirós-tortós and S. Member, "A Graph Theory Based New Approach for Power System Restoration,"

[10] S. Dutta and T. Overbye, "A graph-theoretic approach for addressing trenching constraints in wind farm collector system design," *2013 IEEE Power and Energy Conference at Illinois, PECI 2013*, pp. 48–52, 2013.

[11] S. M. Amin, S. Massoud Amin, and S. M. Amin, "Smart Grid: Overview, Issues and Opportunities. Advances and Challenges in Sensing, Modeling, Simulation, Optimization and Control," *Eur. J. Control*, vol. 17, no. September, pp. 547–567, 2011.

[12] Z. Wang, A. Scaglione, and R. J. Thomas, "The node degree distribution in power grid and its topology robustness under random and selective node removals," *2010 IEEE International Conference on Communications Workshops, ICC 2010*, no. 1, 2010.

[13] S. Lehnhoff and A. Nieße, "Recent trends in energy informatics research," *it - Information Technology*, vol. 59, no. 1, pp. 1–3, 2017.

[14] A. Amato and S. Venticinque, "Big Data for Effective Management of Smart Grids," in *Data Science and Big Data: An Environment of Computational Intelligence* (C. S.-M. Witold Pedrycz, ed.), pp. 209–229, Springer.

[15] CEN/CENELEC/ETSI Joint Working Group on Standards for Smart Grids, "CEN-CENELEC-ETSI Smart Grid Coordination Group: Smart Grid Reference Architecture," no. November, pp. 1–107, 2012.

[16] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik, "Status Report;Reference Architecture Model Industrie 4.0 (RAMI4.0)," tech. rep., 2015.

[17] O. Logvinov, B. Kraemer, C. Adams, J. Heiles, G. Stuebing, M. L. Nielsen, and B. Mancuso, "Standard for an Architectural Framework for the Internet of Things (IoT) - IEEE P2413," Tech. Rep. September, 2016.

[18] Industrial Internet Consortium, "The Industrial Internet of Things Volume G1: Reference Architecture," tech. rep., 2017.

[19] "Neo4j, the world's leading graph database."

[20] "VerneMQ - A MQTT broker that is scalable, enterprise ready, and open source."

[21] "Phoenix Framework."

[22] A. Kamilaris, A. Pitsillides, and V. Trifa, "The Smart Home meets the Web of Things," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 7, no. 3, pp. 145–154, 2011.

[23] E. Patti, A. Acquaviva, M. Jahn, F. Pramudianto, R. Tomasi, D. Rabourdin, J. Virgone, and E. Macii, "Event-Driven User-Centric Middleware for Energy Efficient Buildings and Public Spaces," *IEEE Syst. J.*, vol. 10, no. 3, p. 11371146, 2016.

[24] F. G. Brundu, E. Patti, A. Osello, M. Del Giudice, N. Rapetti, A. Krylovskiy, M. Jahn, V. Verda, E. Guelpa, L. Rietto, and A. Acquaviva, "IoT Software Infrastructure for Energy Management and Simulation in Smart Cities," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 1–1, 2016.

[25] M. Thornton, H. Smidt, V. Schwarzer, M. Motalleb, and R. Ghorbani, "Internet-of-Things Hardware-in-the-Loop Simulation Testbed for Demand Response Ancillary Services," in *Materials for Energy, Efficiency and Sustainability, TechConnect Briefs 2017*, pp. 66–69, TechConnect, 2017.

[26] "Electronic Circuit Simulation Software — PSIM Products."

[27] "MQTT."

[28] M. I. Pai, "Interoperability between IIC Architecture & Industry 4.0 Reference Architecture for Industrial Assets," 2016.

[29] G. M. D'silva, S. Thakare, and V. A. Bharadi, "Real-Time processing of IoT events using a software as a service (SaaS) architecture with graph database," *Proceedings - 2nd International Conference on Computing, Communication, Control and Automation, ICCUBEA 2016*, 2017.

[30] T.-H. Dang-Ha, D. Roverso, and R. Olsson, "Graph of Virtual Actors (GOVA): a Big Data Analytics Architecture for IoT," 2012.

[31] "Property Graph Model."

[32] C. McCord, B. Tate, and J. Valim, *Programming Phoenix - Productive —¿ Reliable —¿ Fast*. Pragmatic Programmers, LLC, 2016.

[33] "Bolt.Sips."

[34] "CoAP - Constrained Application Protocol — Overview."