2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)

Decentralized Access Control for Smart Buildings Using Metadata and Smart Contracts

Leepakshi Bindra, Changyuan Lin, Eleni Stroulia, Omid Ardakanian
Department of Computing Science
University of Alberta
Edmonton, Canada
{leepaksh, changyua, stroulia, ardakanian}@ualberta.ca

***Abstract*—Managing the privileges of occupants and visitors of large commercial buildings to access different building areas, control systems and equipment therein is a challenging task. The best practice today involves giving long-term building occupants, for example employees working in the building, access privileges to their organization areas and requiring visitors to be escorted by them. This approach is conservative and inflexible. Ideally, an automated solution is needed to manage access delegations; however, traditional role-based access control models are unwieldy in that they require the specification of all roles and their relative authority, which is a challenge in large buildings home of multiple organizations and numerous visitors. In this paper, we present a methodology based on blockchain smart contracts to describe, grant, and revoke fine-grained permissions for building users in a decentralized fashion. This method supports access control using resource description framework (RDF) graphs and implements two APIs for client applications. Leveraging the metadata of a real building, we have applied the proposed method to manage privileges in some realistic use-cases and shown that it can greatly reduce the administration overhead while providing fine-grained access control.**

***Keywords*-Blockchain; access control; smart buildings**

I. Introduction

Buildings are increasingly being equipped with sensors and controllers, ranging from card readers for controlling doors, to thermostats and air quality sensors feeding into the Heating, Ventilation, and Air Conditioning (HVAC) system and controlling the ambient environment [1]. Many of these electronic and mechanical components are monitored and managed by the so-called Building Management System (BMS). While the components embedded in buildings are highly complex and heterogeneous, there is currently limited support for interoperability among them. BACnet is a communications protocol that provides mechanisms for computerized building automation devices to exchange information, but it is relatively unaware of the overall building structure and complex functional and spatial relationships among the building subsystems, making it impossible for software developers to identify, access, and manage the physical resources that exist in various building subsystems.

Today, two representation formats are emerging as the de-facto standards for specifying the structure of buildings and their assets. The first is the Building Information Model (BIM) and the corresponding IFC [2] syntax, which represents the building assets, their spatial relationships and geometry. The latter is Brick [3], a schema for a semantic representation of the building sensors and systems, such as electrical, plumbing, HVAC, and lighting, and their relationships in the context of the building space. Brick adheres to the Resource Description Framework (RDF) data model [4], expressed as a collection of triples, to represent knowledge. Queries can be written using SPARQL to access the RDF triples.

Managing access to the building's spaces and its monitoring and automation systems is currently an ad-hoc process, with the key facilities-management personnel accessing the building's major systems and occupants accessing sensor and control points in the spaces to which they have physical access. Visitors tend to have limited access, and are frequently required to be escorted by building occupants.

With this work, we seek to establish a simple software layer for fine-grained management of people's access privileges within a commercial building, where by fine-grained we mean (a) person-centric (instead of role-based), (b) tailored to different space/system granularities, and (c) spanning different timescales. The underlying rationale for this work is that *if a person is authorized to have physical access to a particular location in a building, they are also authorized to access the sensors and actuators in this space.* For example, facilities management personnel have access to the relatively restricted spaces where equipment is installed and also have access to the BMS that enables them to control the equipment settings. People working in a building have access to less sensitive spaces (e.g., their offices and common areas), and may control components in these spaces such as the thermostats and lights. Visitors, on the other hand, have access to the meeting rooms where their business is taking place and may control the components in these rooms, but only for the duration of their meeting. The proposed software layer implements this intuition in the form of smart contracts, through which space-access privileges are given to (and revoked from) individuals through an API that can be invoked by multiple different software applications. The underlying *access control layer* is responsible for accepting or rejecting individual access requests based on the currently valid smart contracts and their implications regarding access to sensing and control components.

The rest of this paper is organized as follows. Section II defines the terms and concepts used in this paper. Section III presents a review of related work on building information models and authorization and access control solutions developed for the built environment. Section IV describes the software layer and blockchain smart contracts. Section V examines three specific use cases in which the proposed solution can greatly enhance the authorization and access control procedures in a real building. The paper is concluded by providing avenues for future work in Section VI.

II. Background

A. Building Subsystems

A commercial building typically contains hundreds of programmable thermostats, connected lights, power meters, motion sensors, and surveillance cameras serving different purposes in various building subsystems, such as Heating Ventilation and Air Conditioning (HVAC), lighting, electrical, plumbing, fire protection and security. These subsystems are traditionally vertically integrated and closed; they are installed by multiple vendors throughout the life cycle of the building.

For example, the HVAC system consists of one or more Air Handling Units (AHUs), which supply cool air through ducts to Variable Air Volume (VAV) systems partitioning the building into independently controllable *thermal zones*. The heating or cooling action of a VAV unit is determined by nested control loops which keep the zone temperature around its setpoint, while maintaining the required minimum airflow. The VAV control system monitors the zone temperature and actuates the associated dampers and valves. The instantaneous values of the sensors and the states of the actuators are logged at regular intervals by the Building Management System (BMS)—a digital control and sensing system which retains historical data, provides visualization, and carries out automation and management tasks for the building. We refer to the historical data archived by the BMS as *trend data*, and a single sensor or actuator described in the BMS metadata as a *point*.

B. Brick Scheme and RDF Ontology

Brick defines a class hierarchy describing different entities in a building and provides a set of relationships for describing the associations and connections between these entities [3]. Examples of these entities are sensors, actuators and equipment in the building subsystems, which are captured in an extensible RDF ontology. Queries can be written using SPARQL to access the Brick representation of a building. For example, the set of sensors and actuators located in a specific floor of a building can be determined using a query. Another query can be written to return the set of sensors feeding a specific VAV control loop.

III. Related Work

IFC [2] is a building information model designed for building structures. It is designed to capture spatial relationships within building components, such as 3D architecture. However, it lacks many common metadata attributes in BMS, common querying mechanisms and flexible naming. Project Haystack [5] is a building metadata format to describe resources in buildings using tags. However, it just provides annotation of each entity and only defines one relationship. It also lacks relational query mechanisms and standardized categorization of vocabularies, which is vital for extensibility. Brick [3] builds upon these works and extends the tagging concept of Haystack. It provides a unified semantic building metadata schema and semantic querying mechanisms. Standardized categorization of vocabularies, class hierarchy and relationships of buildings are well defined in Brick. Brick has demonstrated effectiveness in three aspects, completeness, expressiveness and usability respectively, which means that Brick can represent nearly all entities and relationships in a building's BMS, express them in both human and machine readable ways, and provide efficient querying tools.

Using a semantic ontology to abstract systems and describe resources is common in many areas. Borgo et al. [6] propose a semantic ontology to provide a formalized representation of plan-based controllers in a smart manufacturing system. They define standardized vocabularies of entities and relationships in shop floors. Then, they use the ontology to describe the functions of each machine and relationships developed among the machines. When a machine malfunctions, SPARQL query can be made over the resource RDF graph to identify an alternative machine with same and available functions, and redirect the pipeline to ensure continuous production. Evesti et al. [7] design a metadata schema to describe security models in a smart space. They use a standardized vocabulary to abstract the security mechanism and its attributes as well as relationships among attributes. A secure access control loop can be implemented by querying information about security models using SPARQL.

Conventional methods used for authentication, authorization, and revocation rely on a trusted central authority. For example, existing authorization method LDAP [8] uses Role-Based Access Control (RBAC) with a single central authority. Kerberos [9] and Jabber [10] are similar. Systems developed to avoid the usage of central authority include CCN [11] and the Web of Trust [12], [13]. They work as a decentralized peer-to-peer trust model in which a principal, denoted by a public key, can publish a signature of another public key to denote trust. RBAC-SC [14] builds role-based access control using the blockchain and a challenge-response protocol for authentication. However, neither of the two incorporates how metadata can be linked with smart contracts to extend their usage to defining access control.

Using smart contracts for authorization and delegation of trust was first proposed in WAVE [15]. WAVE uses smart contracts and blockchain as a global ledger for all authorizations, Delegation of Trust (DoT), and revocations, guaranteeing that all participants know the current state of all permissions. The same transparency is achieved by our work. WAVE also supports out-of-order and non-interactive delegation, which is replicated in our smart contracts. Smart contracts are also used in BOSSWAVE [16] to provide democratized access to the physical resources in buildings. Our system is similar to

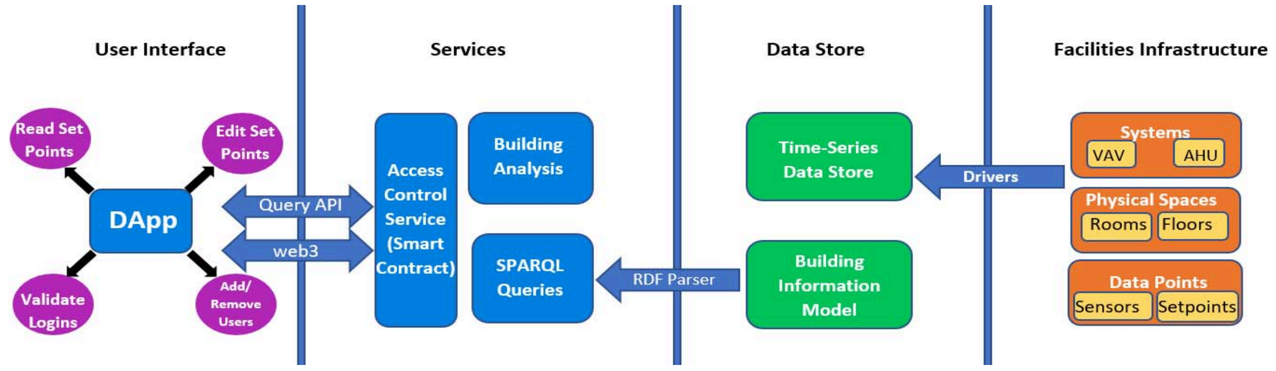

Fig. 1. Overview of the Architecture.

WAVE in that they both leverage blockchain technology. However, using APIs and decentralized applications to create the interaction between RDF schema and authorization through smart contracts is novel in our work.

Our work shows how using decentralized applications (dApp) it is possible to make an interactive user interface. The interactions of the dApp with the smart contracts and the resource RDF graph could open opportunities for further application development. Apart from authorization, authentication, and revocation, it is possible to build several other applications on top of the existing authorizing smart contracts and the building metadata schema, thanks to adaptability and flexibility of the proposed solution.

IV. Methodology

The proposed decentralized access control methodology is shown in Figure 1. This figure depicts how smart contracts and the resource RDF graph are utilized to design a decentralized authorization mechanism. The *building infrastructure* is described in the hardware abstraction layer. In particular, this layer describes the heterogeneous physical resources within the building and provides mechanisms (i.e., appropriate drivers) to interact with these resources. This information is mapped into the resource RDF graph following the Brick schema. The *data store* layer manages two types of data: (a) the building information model, and (b) time-series data collected from various subsystems in the building. The former establishes the context in which the latter is created and should be analyzed. The structural information is also stored in the RDF graph with relationships established to the sensing and control points of different systems in the building. A service layer uses SPARQL to read the resource RDF graph. In principle, there are some specific types of queries that are supported. One such query is to find various sensor and control points for some location which have a specific relation to the location. Section V describes several examples of the queries used by the proposed access control method. Linked to the SPARQL query engine is a smart-contract component that manages (adds, updates, and revokes) access-control privileges for individual occupants to building spaces and systems.

A. *Building Schema and Data Store*

We analyzed the BMS data and metadata of a commercial buildings in Edmonton. The building has one basement and three floors above ground and is equipped with a BMS that archives sensor data, and monitors and controls the HVAC system. There are 136 rooms, 7 HVAC zones, 92 VAV systems, and 1035 points in this building.

The HVAC system has one AHU, which feeds all VAV systems in this building and 12 subsystems, such as the glycol feed unit and chilled water system. We use Brick schema to map out all sensing and control points of AHU and its subsystems, a total of 198 points. Mapping these entities and relationships among them is done in a semi-automatic way. The relationships among the rooms, HVAC zones and VAV systems can be complex. Figure 2 illustrates some of these functional and spatial relationships between entities (equipment, locations, and sensing and control points) in our test commercial building.

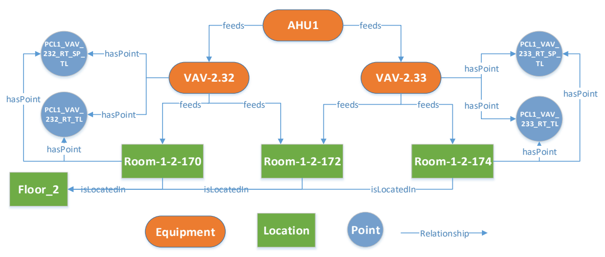

Fig. 2. An example of functional and spatial relationships captured by Brick in our test building.

As shown in this figure, the AHU system feeds the VAVs (linked by the "feeds" relationship) which are present in individual rooms controlling the indoor environment (linked by the "hasPoint" relationship). Each VAV system controls a *thermal zone* comprising one or several rooms and may even span across floors. It is also possible that a single room is divided into multiple zones . Thus, a room can be controlled by more than one VAV system, e.g., Room-1-2-170 and Room-

TABLE I
KEYWORDS EXTRACTED FROM VAV POINTS IN THE BMS DATA

Keywords	Definition	Type
DMP_TL	Damper sensor point	Damper_Sensor
DP_TL	Differential pressure sensor point	Pressure_Sensor
FLW_SP_TL	Flow speed setpoint	Setpoint
FLW_TL	Flow speed sensor point	VAV_Flow_Sensor
OperationMode	VAV operation mode	Command
RAD_TL	Radiator valve setpoint	Setpoint
RT_SP_TL	Room temperature setpoint	Setpoint
RT_TL	Room temperature sensor point	Temperature_Sensor

TABLE II
SUBSET OF KEYWORDS EXTRACTED FROM AHU POINTS IN THE BMS DATA

Keywords	Definition	Type
AFLW_SP_TL	Air flow setpoint	Setpoint
EAT_TL	Entering air temperature sensor point	Temperature_Sensor
SF_SPD_TL	Supply fan speed sensor point	Speed_Sensor
FIL_PRES_TL	Filter pressure sensor	Pressure_Sensor

1-2-174, and may be monitored by the sensors which are part of one of these VAVs. A large room (e.g., a corridor) can be controlled by as many as 10 VAVs, while it is monitored by some of the corresponding sensors. The VAV points can be accessed by making SPARQL query using the relationships "feeds" and "hasPoint". The Brick ontology captures such complex relationships and we can specify constraints and patterns of triples in SPARQL queries in a semantic way to retrieve essential information for the access control.

A total of 4220 triples are required to cover 136 rooms, 7 HVAC zones, 92 VAV systems, 1035 points and relationships that exist among them in our test building. We develop a parser to convert these entities and relationships into Brick triples in a semi-automatic way. This is done by parsing the names of VAV points in BMS and analyzing them. We extract 8 common keywords which stand for 8 types of VAV points that are converted to Brick triples as described in Table I.

Similarly, 45 keywords are extracted from the names of AHU points and these constitute the names of entities that are mapped into the RDF graph as Brick triples. Table II shows a subset of keywords extracted from AHU.

We also need spatial information about rooms and thermal zones, and information about relationships, such as "feeds" and "hasPoints", developed among locations, VAV systems, and the corresponding points. To this end, we manually extract these two types of information from the building's floor plans and store them in csv files. The parser can read and parse the csv file, extract keywords, and pull in information about locations and relationships from the files to convert the building metadata into Brick triples.

B. Interaction with Building Resource Graph

We develop two Flask-based APIs, namely the Query API and the Simulated BACnet API, for querying resource RDF graph and simulating the BACnet controller to store setpoints and return sensor readings respectively. These two APIs act as the bridge between the building metadata and the smart contract layer. The Query API takes a JSON file containing SPARQL [17] queries. It then parses the JSON file, traverses the resource RDF graph of building, and returns the match in JSON format. The matched entities are those that meet relationship constraints and patterns of triples specified in the queries.

The entities returned by SPARQL queries are the floors, rooms, VAVs, and various points in the resource RDF graph. A set of common queries used to read our resource RDF graph includes finding various rooms and HVAC zones on a floor, locating VAV points that feed a room or HVAC zone, listing all sensors or set points that are related to some location(s). The query in Listing 1 returns a list of all VAVs on the second floor of the building. These are nodes of type VAV in the resource RDF graph that "feed" an entity of type room, which is related to "Floor_2" with the relationship "LocatedIn". Similarly, the query in Listing 2 returns a list of points where each point is an entity related to a VAV entity with the relationship "hasPoint" and the VAV entity "feeds" a specific room, Room-1-2-172. This query is useful when the access control service wants to ensure that an occupant having access to the Room-1-2-172 can read sensor measurements or adjust setpoints that correspond to this room.

```
"""select ?vav where {
?room rdf:type brick:Room .
?room bf:isLocatedIn BLD:Floor_2 .
?vav rdf:type brick:VAV .
?vav bf:feeds+ ?room .
}"""
```

Listing 1. SPARQL Query to get all VAV systems on Floor 2

```
"""select ?points where {
?vav rdf:type brick:VAV .
?vav bf:feeds BLD:Room-1-2-172.
?vav bf:hasPoint ?points.
}"""
```

Listing 2. SPARQL Query to get all points in a certain room

The simulated BACnet API is responsible for communicating with the underlying resources using the BACnet protocol. A JSON file containing the point's name can be posted to the Simulated BACnet API and it will subsequently return the value of the point. If the posted JSON file contains a setpoint's name and a value, the value of the setpoint will be updated in the corresponding VAV control loop. The BACnet API can be altered to work with the BMS of a real commercial building to facilitate application development.

C. Access Control Service

Today, a typical medium-sized commercial building contains several vertically integrated systems such as HVAC, lighting, plumbing, electrical, security, and fire protection. To realize the notion of smart buildings, applications should run on top of all these systems and perform read and write operations on the underlying physical resources. Examples of these applications are maintaining the room temperature, viewing the camera recording for a specific time duration, estimating the number of occupants, monitoring the energy consumption of a particular set of appliances in a building. It is essential to decide who has access to what resources in a given time window as it is not always the building manager who runs these applications. This manifests the need to provide access to different levels of management, occupants, and sometimes visitors to run some of these applications and interact with the underlying resources.

Traditional ways of authorization require that the central authority is contacted and a new user with their credentials is added to the set of people who have access. For this, the central authority needs to be online and the process of approval usually takes a considerable amount of time. Apart from this, only this central authority would have all the credentials and be able to decide about the authorization policy. Should the central server be compromised, the loss of data and malicious attacks on the building systems would be possible.

Blockchain technology can be used to manage access control in a decentralized way. A smart contract is a snippet of code that validates each transaction based on a set of rules. The code written is immutable once it is on the blockchain. The authorization policy of the building can be programmed in these smart contracts that run on the Ethereum virtual machine. Ethereum is the platform for developing smart contracts and storing their data in a decentralized manner.

The rules described in the smart contracts can assign access to users. For example, a root user has access to all systems in the building. This could be the building manager who traditionally manages all the systems and subsystems. This user could provide access for a set of systems and assign a set of permissions to another user in the building. For instance, each floor can have a floor manager who has access to all the subsystems on the floor, such as the VAV systems, electrical panels, etc. Along with this, the floor manager can have the permission to access, alter and further provide more users with the access. This leads to distribution of the access in a decentralized way, because the credentials are not stored on any central server, but on all the nodes of the blockchain in the form of a global ledger. Any transaction of providing access or altering a system is stored and would be viewable on the global ledger of the blockchain. This makes it transparent and there is no need to wait for the central authority when some user needs to prove their authenticity.

In addition to assigning access, revoking access can be done using the same smart contract. Once the user no longer needs to have access, the user's credentials are invalidated. This transaction is also stored on the global ledger and is updated on all the nodes in the network. For example, a visitor may need to have access to a small set of resources, say a room and the setpoints in it, for a specific amount of time. After this time elapses, the access has to be revoked. This scenario points out how time-restricted access is needed in the building environment, which is also implemented in the smart contracts. To ensure security, providing restricted and controlled access to some users is also necessary. Specifically, users may have access to either view, alter, revoke, further delegate access or a combination of these permissions.

The smart contract designed in this work uses the public and private key pairs to authorize each user. As previously mentioned, the building manager has universal permission to all the systems and can further delegate access. This forms a graph of permissions as further delegations are done. Each node in the graph has the public key, the expiry of the access if there is any, and a hash of information describing the node. The branch from the building manager's node to the floor manager's node stores this metadata and the data about subset of resources that the floor manager has access to. Figure 3 shows an example of the authorization graph created with a small number of users.

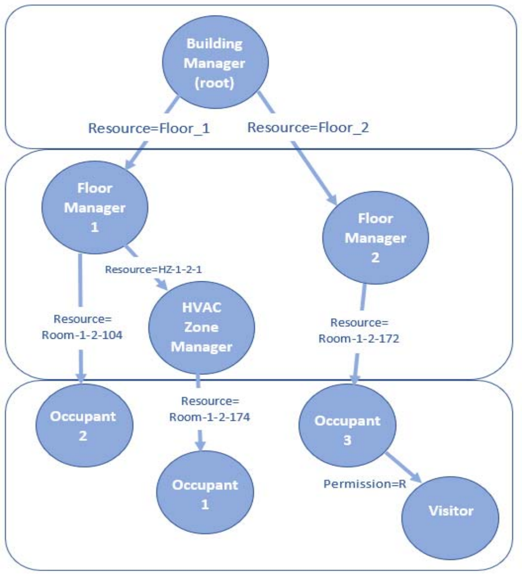

Fig. 3. An illustration of the authorization graph

The Delegation of Trust (DoT) [15] is the relationship between a pair of nodes in the authorization graph from the user granting access to the user receiving access. It stores data about the resources that the new user has been granted access to, the expiry time and other information. The resource is one of the nodes in the resource RDF graph. The new user will have access to every node (location, sensing or control point) in the resource RDF graph that is related to the resource in DoT by some relationships like "feeds" or "hasPoint", i.e., is in a subtree rooted at that node. The creation of DoT is a transaction and does not need to be approved by the principal authority. The granter also authenticates that he has provided the access by signing the transaction with his private key. The DoT containing the metadata and signature forms the proof of authorization whenever the user tries to access these subsets of resources.

The smart contracts also handle the withdrawal of access from users and invalidation of the DoTs. This helps remove temporary access in a scenario where time-restricted access has been granted. Also, if the user leaves the space, his access can be revoked. DoTs can also expire or be revoked.

V. Example Use Cases

In the following, we describe three specific use cases we worked upon to corroborate the efficacy of the proposed solution. These example use cases are prevalent in most commercial buildings.

A. Basic Authorization Tasks

Authorization tasks include adding or removing permissions for users. This requires interaction with the smart contracts to add or remove entities and DOTs initiating a state change in the contract. A front-end application provides a user interface to interact with smart contracts and implement state changes. It also allows for reading and interacting with the resource RDF graph. The user inputs the required fields asked by the front-end to perform the respective action. The required information will include the from and to entities (i.e., the user providing the access and the new user who will be granted access), the set of resources to which access is provided, and a set of permissions.

Suppose User A has access to zone HZ-1-2-1. This zone spans two rooms, namely Room-1-2-104 and Room-1-2-102. Hence, these rooms are linked to the zone with the "isLocatedIn" relationship in the resource RDF graph. Recall that User A could be provided a set of permissions including reading sensor measurements, writing a new value to a control point, providing a subset of these permissions to other users, and removing entities or DOTs. Therefore, if User A is permitted to change setpoints in Room-1-2-102 (i.e., having the write permission for that control point) and to delegate permissions, they can give this permission to User B for a subset of the setpoints in this room. A DOT will be created subsequently from User A to User B with the information about the corresponding permissions and resources. User A can restrict the permissions provided to User B, for example, by not granting them the permission to write the temperature setpoint in that room or to further delegate this permission to other users.

Each of the actions is instantiated with a callback function to the smart contract. If the user who is making the state change has the permission to perform the action, the transaction needs to be signed by the user and is then executed and updated on all nodes of the network. Otherwise, the transaction fails and reports that the user is not authorized to do the state change.

B. Monitoring and Actuation

Viewing or editing the controls associated with the systems and subsystems that each user has access to is another instance of our methodology. For example, a room occupant wishes to change the temperature setpoint of the VAV in the room, we must first fetch all the setpoints that they have access to using the Query API and then provide the interface to edit the value of the specified setpoint. For this, we retrieve the set of resources in the RDF graph that the user has access to using the information of the DoT in the smart contract. Upon receiving the root node of the subtree of resources that the user has access to, we call the Query API to retrieve the list of accessible setpoints within this subtree. The subtree is the set of all nodes of a specific type that are related to the root node by one or more relationships in the resource RDF graph. For example, to view all the rooms that are located on Floor_2, we use Floor_2 as the root node and all the nodes (i.e., rooms and equipment) that are related to this root with the relationship "isLocatedIn" are in the subtree rooted at this floor. Another case of finding all the temperature setpoints in the building related to a specific AHU system would require us to first find all the VAV points that the AHU has a relationship of "feeds" with and then find all the nodes with relationship of "isPointOf" with each identified VAV node. In particular, we search the points in the subtree rooted at the AHU system. The floor manager can select the VAV that they wish to update or view. To test this functionality, we call the Simulated BACnet API, which has synthetic data about some of the setpoints. To view the setpoints only, the values of the selected VAV received from the interaction with the Simulated BACnet API can be displayed. To update the values, this API is used to write back to the synthetic data. In future work, we plan to integrate the functionality of this system with the rudimentary BMS of the building.

C. Governing Temporary Accesses

Our methodology helps users control their surrounding environment, be it their workplace or a conference room booked for their meeting, as long as they have access to the sensing and control points in that place. For instance, a meeting room can be booked for a number of hours and some people may be invited to the meeting. In such a case, the floor manager can use this method to provide access of the room's thermostat to the meeting participants during the meeting. The level of access that must be provided varies for different participants. For example, people who have access to the floor on which the meeting room is located may have access to the meeting room's thermostat already, whereas people whose workstations are on another floor would need temporary access (during the meeting) to the thermostat of the meeting room.

Visitors from outside who are invited to the meeting would require certain permissions to be granted to their entities after new entities are created for them. This can be done by the floor manager or by any of the meeting participants hosting the meeting and having the required permission to create entities and grant permissions to them. One of the visitors who does not currently have access to the meeting room, but has permissions to authorize visitors, can also add new entities for other visitors at any point in time. Once that visitor receives access to the room, all other visitors that they added will also have the necessary access. This way, the smart contract makes access delegation more fine-grained, supporting what is known as out-of-order authorization [16]. Once new users receive the specified permissions, they can monitor sensor measurements or partake in controlling the spaces they have access to using another application.

VI. Conclusions and Future Work

This paper describes our initial work towards a methodology and a software system to support reasoning about and flexibly managing the access privileges of occupants and visitors in smart buildings. We use blockchain smart contracts to authorize users to access building spaces and systems, at specific times, and subject to specific constraints. The unification of RDF graphs to portray building metadata and smart contracts for authorization is done using the decentralized application with the employment of two APIs. Through some example use cases, we demonstrated the potential usefulness of the proposed access control as a layer that supports smart building applications.

The Query API and Simulated BACnet API developed in this work will serve as a reference point for numerous avenues of future work. With small modifications in the Simulated BACnet API to interact with the actual building's BMS, the system can be extended to enable real-world applications without changing the authorizing smart contracts and the dApp. We also intend to make the creation of resource RDF graphs from using raw building metadata a fully automated process. This can be done by takes advantage of unsupervised learning algorithms. Furthermore, incorporating the use of roles instead of individual users receiving access is another improvement that we plan to make in future work. Using role-based access control helps group similar types of users into one role and provide them with identical access. For example, all the occupants of an office room with multiple workstations should have access to the same set of physical resources. The cost (ether) of using smart contracts can be reduced by adopting various measures which we plan to investigate in future work.

References

[1] O. Ardakanian, A. Bhattacharya, and D. Culler, "Non-intrusive occupancy monitoring for energy conservation in commercial buildings," *Energy and Buildings*, vol. 179, pp. 311–323, 2018.

[2] V. Bazjanac and D. Crawley, "Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings," in *Proceedings of Building Simulation99*, vol. 2, 1999, pp. 661–667.

[3] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal *et al.*, "Brick: Towards a unified metadata schema for buildings," in *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*. ACM, 2016, pp. 41–50.

[4] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification," 1999.

[5] "Project haystack," http://project-haystack.org/.

[6] S. Borgo, A. Cesta, A. Orlandini, and A. Umbrico, "An ontology-based domain representation for plan-based controllers in a reconfigurable manufacturing system." in *FLAIRS Conference*, 2015, pp. 354–359.

[7] A. Evesti, J. Suomalainen, and E. Ovaska, "Architecture and knowledge-driven self-adaptive security in smart space," *Computers*, vol. 2, no. 1, pp. 34–66, 2013.

[8] K. Zeilenga, "Lightweight directory access protocol (ldap): Technical specification road map," Tech. Rep., 2006.

[9] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[10] P. Saint-Andre, "Streaming xml with jabber/xmpp," *IEEE internet computing*, vol. 9, no. 5, pp. 82–89, 2005.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[12] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "Openpgp message format," Tech. Rep., 2007.

[13] G. Caronni, "Walking the web of trust," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000). Proeedings. IEEE 9th International Workshops on*. IEEE, 2000, pp. 153–158.

[14] J. P. Cruz, Y. Kaji, and N. Yanai, "Rbac-sc: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12 240–12 251, 2018.

[15] M. P. Andersen, J. Kolb, K. Chen, G. Fierro, D. E. Culler, and R. A. Popa, "Wave: A decentralized authorization system for iot via blockchain smart contracts," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-234, Dec 2017. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-234.html

[16] M. P. Andersen, J. Kolb, K. Chen, G. Fierro, D. E. Culler, and R. Katz, "Democratizing authority in the built environment," *ACM Transactions on Sensor Networks (TOSN)*, vol. 14, no. 3-4, p. 17, 2018.

[17] "Sparql query language," http://www.w3.org/TR/rdf-sparql-query/.