BuildingDepot 2.0: An Integrated Management System for Building Analysis and Control

Thomas Weng[†], Anthony Nwokafor[†], Yuvraj Agarwal[‡]

[†]University of California, San Diego [‡]Carnegie Mellon University

[†]{tweng, aanwokafor}@cs.ucsd.edu, [‡]yuvraj.agarwal@cs.cmu.edu

Abstract

Improving energy efficiency in buildings is a key objective for sensor researchers and promises significant reductions in energy usage across the world. The key technological driver for these gains are the novel sensor network deployments and the large amounts of data that they generate. The challenge however is making sense of this data, and using it effectively to design smarter building control schemes.

Several recent research efforts have sought to address the challenge of data access and building control. However, while these systems have made progress in specific areas, many unanswered questions still revolve around data management and what exactly it means to develop building applications. Critically, how would such a solution work in a real building setting and how can applications be written such that they can be re-used in other settings? To resolve these issues we have developed BuildingDepot 2.0, a building management control platform that significantly updates on our first iteration of the system for data analysis and high level supervisory control.

Categories and Subject Descriptors

J.7 [**Computers in Other Systems**]: [Industrial control]

General Terms

Design, Management, Human Factors

Keywords

Building Energy Management

1 Introduction

Modern commercial buildings have become prime targets for energy efficiency research. Commercial building energy consumption already represents 35% of the total US electricity consumption and is projected to rise even more[6]. Given the costly nature of generating this electricity, finding ways to reduce energy usage is an important societal goal. Better understanding of the building processes (which include HVAC, occupancy behavior, and energy usage), and designing smarter building control techniques that can both maintain occupant comfort while minimizing energy consumption can lead to large improvements in building operation.

Building research has examined using wireless sensor networks to monitor these building processes [8]. Combined with existing building control systems (such as those that run the HVAC), a significant amount of data is being generated. Unfortunately, each of these systems are effectively closed off from each other, and thus the potential for truly interesting data analysis or control applications is lost. In fact, for many industrial systems, the data is not only inaccessible, many times it is simply thrown away.

Therefore, developing a platform to allow for applications that can span across the multiple systems that monitor and control the building environment can potentially lead to a much deeper understanding of building operations. Several recent efforts have indeed sought to improve data management and control in buildings. Our own effort on BuildingDepot[1] was a sensor data storage system that allowed flexibility and sharing. Such systems help push forward the state of the art in building data management, but there are still missing pieces in terms of how they can be used to develop applications. Specifically, there were several unaddressed issues about what it means to develop building applications. What exactly are applications, and more importantly, who writes them and who uses them? How does any proposed system fit within the existing infrastructure of buildings? There are also important secondary considerations such as performance and scalability.

Based on the current state of research, as well as our own experiences deploying building sensing networks and our original BuildingDepot data storage system, we have developed BuildingDepot 2.0 (BD2), a significant update to our original system that goes beyond a datastore and improves on all aspects relating to developing building applications. Chiefly, we introduce a template system that defines a common language for describing both sensors and buildings. We also clarify how such a system can be deployed and used in a real world setting (as opposed to just an experimental research scenario). This paper describes our vision and the overall architecture of BD2, and in particular we highlight the architectural changes over the first iteration.

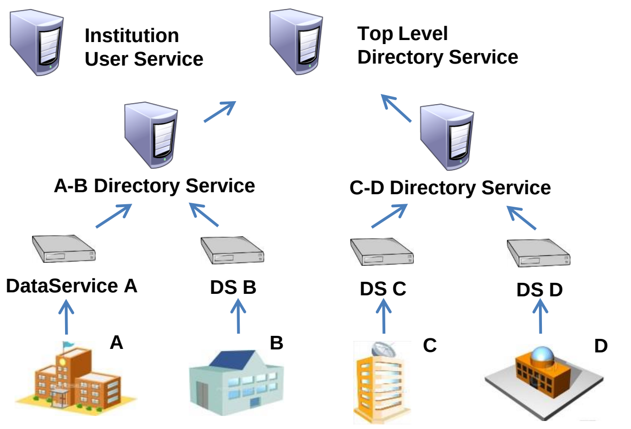

Figure 1. The original BuildingDepot system. An institution could instantiate as many Directory Services and Data Services as they wanted - there were no guidelines on how to organize the system. In this case, a campus with 4 buildings (A-D) each have their own DS, each connected to a intermediate DirS. Confusion often arose in how to deploy the services.

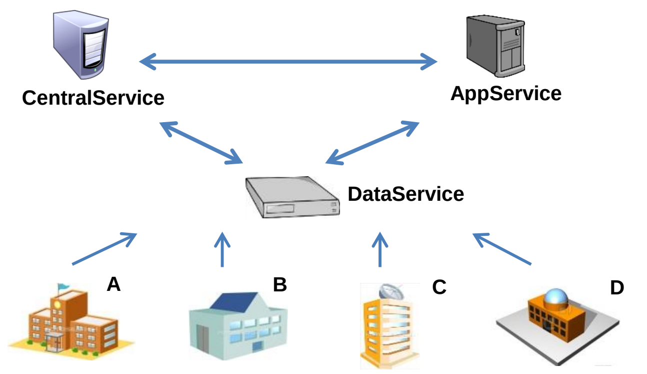

Figure 2. High-level view of the new BuildingDepot 2 system. We have collapsed the Directory Service and User Service into a single Central Service that acts as the gateway for the institution's BD deployment. Also we have clarified the role of a Data Service, and recommend one unless administrative groups require their own DS, in which case as many DSes as needed can be deployed.

2 Background and Related Work

Buildings have attracted a significant amount of attention among researchers across many different domains. The sensor networking community in particular has developed an increasingly large array of interesting networked sensing and actuation systems. Combined with the existing SCADA systems that modern commercial buildings employ, a significant amount of data is generated.

Current existing building management systems however greatly limit analysis and innovation potential. A commercial building will likely have a BACNet network consisting of equipment controlled by commercial building software from companies such as Johnson Controls. These systems do not usually prioritize storing this data, and thus most of it is simply lost. These systems also tend to be independent from one another. This limits the amount of innovation that can be applied since each system only has information from its own network, and thus control algorithms tend to be simplistic. This can be seen with the common static HVAC schedules, which are inherently wasteful since they do not observe if the buildings are even occupied.

Several recent research efforts have sought to address this issue. HomeOS[7] attempted to abstract an operating system for residential homes as drivers on a computer, and built up layers to capture common functionality. Another system is BOSS[5], which describes a distributed control system for buildings using different layers of services. A low layer of sMAP systems[4] provide an interface to underlying sensors, while higher layers provide an abstraction of sorts for applications. However, BOSS assumes complete freedom and access to the existing building systems and makes implicit assumptions on who is using the system - specifically researchers who understand the system and have the technical skills to utilize it. This might be at odds with how buildings are actually managed, and one that would possibly meet resistance from building managers.

Other efforts have examined the data management side. SensorAct[2] utilizes multiple systems in order to maintain privacy and controls over sensor data. Our own system targeted data storage for buildings with an emphasis on sharing. BuildingDepot (BD) was a system that consisted of three separate components - a User Service that hosted all of an institution's users, a Data Service that stored the data values from existing sensor networks, and a Directory Service that essentially exposed what sensors existed in subordinate data services. An institution could have multiple Data Services, along with Directory Services, thus distributing both across a campus. Figure 1 highlights our original system.

Our experiences with the initial deployment of BD led us to realize that there were weaknesses in the design. First of all, the only communication that the three services supported was through a REST API over HTTPS. For external clients, RESTful APIs are sufficient, but for internal communications between the services themselves, the REST API had a significant overhead. The Directory Services were redundant too, as we could not see a need for having more than one in an institution. While the Data Services supported actuation, in reality there were issues with contention, latency, and conflicts. Additionally, the Data Connectors required manual mapping between the actual data source and the Data Service, which caused maintenance issues.

Ultimately, the very general nature of the system meant that each deployer was left to figure out how to design the abstract sensor entities and how to connect the datastore with their current infrastructure. It was not clear how to design apps around the system since the entire context of the sensors (how sensors are described in the environment) were completely arbitrary. These lessons spurred us to develop a better system, one that would better allow for reusable applications, and one that would clarify how our system should be deployed. While the original BD worked as a data storage for building data, BD2.0 is a more complete system that addresses not just storage but application-development, control, and improved functionality.

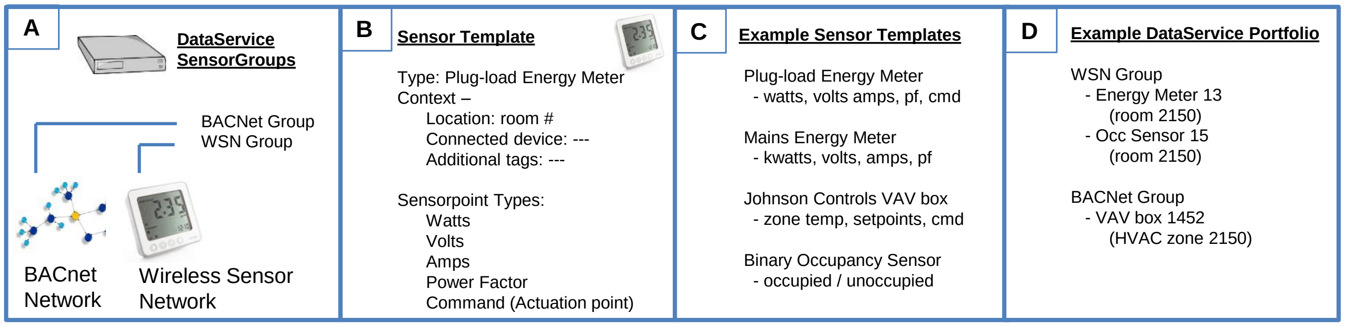

Figure 3. Resources in the BD2 DataService. (A) highlights how the Sensor Group resource maps to a physical sensor network. (B) shows an example of a Sensor Template (for a plug-load energy meter). (C) lists some of the Sensor Templates that are included in the current BD2 ST library. (D) is an example of a DataService with some representative sensor devices.

3 Vision and Architecture

In this section we describe the vision and overall architecture of BuildingDepot 2.0 (BD2). BD2 is meant to serve all parties that have a role or stake to play in building management. This not only includes researchers (who would likely have an engineering or computer programming background) but also non-technical building managers, energy auditors, and analysts. The central entity that installs and uses BD2 is the institution. The institution is an organization that controls a campus, which itself is a collection of related buildings. A typical example would be a campus or a commercial park. Alternatively, it might only control a single building.

An institution is an abstract concept, so the actual person in charge of a commercial building or campus is likely a building or facility manager. A large campus would typically have several people working in a facilities group, while a smaller one might just have one person. The main responsibilities often include overseeing normal building operations. Building/facilities managers usually have some sort of technician training and utilize commercial software packages. They do not have the expertise to write their own applications, and rather will use whatever is installed by their infrastructure provider. They in effect treat any such building management system as an appliance. In addition, they often are very conservative and protective of their equipment, and are loathe to give up complete access.

Analysis researchers and auditors form another class of users who are interested in the building data. These users might be part of an energy management group (for the case of a large institution and campus) or the building manager. Their main goals revolve around monitoring energy performance and ensuring that the overall building and campus processes are being run efficiently. For these users, access to the data is critical (especially archival and historical data). The ability to run analysis and auditing applications that can detect anomalies and do comparisons between buildings is also extremely valuable.

Researchers and application writers form the third class of users. These are building researchers who are interested in developing innovative applications to control and analyze buildings. They have a programming background and can write applications provided that they have access to the necessary data and actuation points that comprise the building. These programs tend to be experimental. Application writers can also design their programs for broader release and usage. In this scenario the applications will have polished user interfaces so that non-programmers can use them.

Ensuring a data analysis and management plane that could be useful to all these classes of users have driven the design of BD2. To that end we have designed BD2 to act as a management and supervisory control integration system over existing campus building management systems (BMS)and sensor network systems. In the following subsections we describe the overall architecture of BD2 and the changes we have made over the original BD system, and how the design benefits the different types of interested users.

3.1 Overall System Architecture

BD2 consists of two core software services - the BuildingDepot DataService (DS) and the CentralService (CS) and an optional third service - the ApplicationService (AS). An institution will have exactly one CS and one or more DSes and ASes. The DS stores the data from underlying sensor networks, and exposes it through several different interfaces. The CS acts as both the directory for all of the institution's buildings and users. The AS is a server with BD libraries installed to facilitate the development of applications. The DS and CS are both independent systems, but are bound to each other and communicate with each other to fulfill the BD mission. The original BD system had three systems (the Directory Service, User Service, and Data Service), with institutions having the option of deploying multiple Directory Services and Data Services as needed. We realized that deploying more than one DirectoryService wasn't necessary, and have simplified the system by allowing only a single CS.

Also, in the original BD work, we stated that Data Services could be deployed according to what the institution desired. Unfortunately, there was some confusion on what would merit a new Data Service over using an existing one. Did every building require a Data Service? What about if multiple groups in the same building wanted one? For BD2, we have clarified that a Data Service should belong to any single administrative group that requires sole control over who can access the underlying data and actuation points. In many cases, an institution would only have one such group, and thus only require one Data Service. Alternatively, a facilities group might require their own Data Service for all of the HVAC related sensors while a research group might require one for all of their WSN data points. Or different buildings on a campus might desire their own DS. Thus it is up to an institution to determine how many Data Services are needed depending on the specific groups that exist and their needs. Figure 2 shows the architecture of BD2.

3.2 Building DataService

The BuildingDepot DataService (DS) is the core service of BD2 and manages the sensor data points that are assigned to it. The DS consists of server software and databases that retrieve and store the data from underlying sensor networks. It also can send commands to actuate control points. The original DS stored and exposed data solely through a REST interface over HTTPS. The primary resource was a sensor, which was arbitrarily defined by the DS admins to represent whatever sensors were being stored there. Each sensor held sensorpoints, which represented the actual datapoints for that sensor. Sensor groups were defined as arbitrary collections of sensors. In addition there were internal users that could directly administer the site.

Our experiences with this setup revealed some issues in the commissioning process, specifically with how arbitrary it was. For example, an energy meter could be defined by two different groups completely differently - this naturally limits any sort of reusable app when a common language cannot be agreed upon. This also causes problems in the connection with the underlying sensor fabric - the original Data Connectors had to be manually mapped between the actual data sources and the sensor/sensorpoint resources on the DS. These inherent weaknesses led us to completely redevelop the meanings of the DS resources. Rather than allow for arbitrary specifications, we have redefined the Sensor, Sensorpoint, SensorGroup, and have introduced a new resource called the Sensor Template. We explain the resources and specifics of the DS as follows.

Sensor Group - Originally, a Sensor Group (SG) was an arbitrary collection of sensors. In BD2, a SG actually now maps directly to an underlying network. For example, a building might have an energy meter WSN and a campus HVAC BACNet network. Each of these are separate network systems, and controlled by different groups. When a new DS is being commissioned, the admins will create a SG for each of the networks that will connect to the DS. SGs contain the sensor resource that map to the physical sensor device under the actual host sensor network. Figure 3A highlights how SGs are directly associated with an underlying network.

Sensor Template - Sensor Templates (ST) are new to BD2 and describe specifically what a sensor is. As a SG represents the host network, a ST defines the sensor that exist in the underlying networks. A ST consists of the sensor template type and the list of sensorpoint types that exist for that sensor. These sensor templates are meant to reflect the physical underlying sensor. For example, an energy meter typically has data for watts, volts, and amps; therefore an energy meter sensor template would contain the sensorpoint types that an energy meter sensor would have (e.g. watts, volts, amps, etc). Because these STs are standardized, applications can be written knowing how a sensor is defined (in terms of what its datapoints are). One of the goals of BD2 is to develop and release a large library of such STs. So far we have cataloged sensors that exist in our own campus, including plug load energy meters, occupancy sensors, and VAV boxes. As more existing sensors become cataloged, the library becomes increasingly useful for application development. In addition to the standard library that we are maintaining, users can also design their own custom sensor templates as well by specifying the type and what sensorpoints belong to the ST. Figure 3B and C give an example of a Sensor Template and list some STs in the ST library.

Sensors - Sensors are the core resource of the DS. As the SGs map to a physical sensor network, sensors represent the actual physical sensors that exist in the sensor network. Sensors consist of a sensor type that specifies what sensor templates it is, and a list of sensorpoints that reflect what sensor type it is. Sensors also contain metadata which describe its context, such as its location - these play a role in binding with the Building Templates (which will be explained in the next subsection). Figure 3D shows some representative sensors that exist on a BD2 DS. Custom context tags can also be added to a sensor if the institution so desires.

Virtual Sensors - This is a new sensor type introduced in BD2, and allows for a virtualized sensor that exists as an aggregation or calculation of other physical sensors. These represent data sources that do not physically exist, but can be calculated from existing sources. For example, a room might not have a circuit meter measuring total room energy consumption, but if each of the plug loads are measured, then the total room energy can be calculated by summing the individual plug load meters. Another example might be a virtual occupancy sensor that is based on indirect data, such as through Wi-Fi tracking. A separate program can calculate occupancy as a virtual sensor from this data, and act as the data source (as opposed to a physical sensor network).

Sensor Points - Sensors consist of sensor points (SP), which correspond to the actual data streams and actuation points that comprise a physical device. The actual time-series data (time-value pair) are associated with the SPs. For example, a plug load energy meter would have SPs for watts, volts, and amps. Unlike the first version, all SPs have a predefined type that describes what exactly the SensorPoint is and what data type it is. These SP types are used as part of the Sensor Templates to exactly define a type of sensor.

Administrative Users - Admins are responsible for the DS, and there are administrative functionalities that give the DS admins fine-grained control over who has access to what. There are three tiers of administrators, with the highest tier having complete control over the DS, and the lowest tier being restricted to only instantiating new sensors in specified SGs. This allows the main admins to give access to other users for their own SGs - for example, an institution's facility manager can give the groups that manage the various sensor networks their own lower-tier admin accounts to handle their own sensorgroups. This design is meant to allow flexibility in how the DSes get deployed in an institution. If different groups (who control different networks) can exist on one DS, then only one DS is needed for the institution. If however different groups require complete control over their own DS, then each can have their own DS within the institution.

This design choice came from conversations we had with the facilities group on our campus. Having ultimate control over the HVAC control and HVAC data for the buildings was a crucial feature that they required.

Normal Users and Permissions - Normal user accounts are stored in the CentralService, and thus the DS does not need to maintain the actual users who might use the system on a normal basis. Instead the DS simply queries the CS to authenticate any user access. The actual permissions to read or write from the sensors are stored at the DS.

Actuation - Actuation is a critical function, and one that the original BD allowed. The implementation however was too simplistic and we ran into issues with contention over the actuation resources. Thus we have completely revamped it and learned from BACNet and other more recent research efforts in how they handle this critical feature. In BD2, each actuation point is a special sensorpoint that resides under a sensor. These actuation points can be written into the DS, which will then notify the Data Connector to actuate the underlying control.

BD2 is geared more towards high level supervisory control. The reasoning behind this is that a great deal of work has been done on low level direct control, and seeking to supplant that (e.g. the PID control loop for cooling a room to a targeted temperature) could cause issues if not done properly. For this reason (among many others), facilities managers are loathe to give up unrestrained control to potentially delicate building control systems. To make this acceptable to the people in charge of building operations, BD2.0 allows fine-grained control over the authorization over actuation, ensuring that only certain actuation points are allowed (such as turning off or on the HVAC in a specific zone).

Actuation points have a special ACL that is separate from the permissions list of the parent sensor. By default, the only users who can access an actuation point are admins. The admins can let normal users access these if desired (for example, the HVAC controls for their rooms). Potentially, multiple users might have access to a single actuator. For example, a VAV unit could be controllable by the occupant as well as the building manager. Because of this, we incorporate a priority array similar to BACNet that determines which setting ends up being sent as the command. Two defaults must be set by the admin for every actuation point. The first is a special low-priority default value in case no other actuation request comes in. This can be any valid value for that actuation point, or a release command (which essentially relinquishes all control from the DS to the underlying network) for systems that support it (e.g. BACNet). The second is a conflict-default value that will be set if two or more users with the same priority level (which might be the case if two users share an office for example and have access to the VAV controls) conflict. This will be set as the command in that situation, and usually will be the more conservative of the options (for example, leaving the HVAC system on).

Data Connectors - As BD2 operates as a management fabric on top of an existing sensing infrastructure, transferring the data from the underlying networks to the DSes is a key operation. The original BD forced a manual mapping between the DS sensors/sensorpoints and the actual datapoint residing on the sensor network. BD2 attempts to automate this process. Many different types of networks exist, including BACNet, Lonworks, ION, and custom proprietary networks. Many such networks have a central server that stores all of the sensor metadata, and might even store the actual sensor data. Others might have a series of nodes that individually can be queried for data. A Data Connector (DC)is the software that handles the connection between these disparate networks and the BD2 DS.

Fundamentally, DCs have two roles to play - they need to determine what sensors and sensorpoints actually exist in the underlying network, and how to retrieve the respective data and transmit it to the matching BD2 sensorpoint. We have developed a framework for writing and reusing data connector drivers to allow this. Physically speaking, they can exist in one of two places - at the DS level, where they are essentially pulling data from the underlying network, or at the sensor network level, where they push out the data to DS. The former generally can be installed when the sensor network allows for retrieval, and has advantages in terms of speed since the DC can insert the data directly into the system. The latter is used when there is no way of easily accessing the network from the outside, and instead the DC must be installed within the sensor network itself (e.g. a protected BACNet network). In this case, the DC can POST the data with proper authentication to the DS using the REST API.

In either case, the DCs first determine what sensors exist. The DC can be programmed to either retrieve this meta information from the underlying network (for example, by connecting directly to the sensor network) or through an existing collection of the sensor meta-information (perhaps through a database or a text file). The DC developer must be able to make sense of this, so some amount of customization needs to take place for a custom sensor network. Once this data is parsed, the DC will create the matching set of sensors/sensorpoints for each of the underlying physical sensors and sensorpoints. When created, the sensors and sensorpoints maintain information on the underlying identifier and this information allows the DC to transmit the sensor data using the host identifier directly. The DS will be able to determine from the group ID and the identifier what sensorpoint's datastream needs to be updated. This allows the DC to automatically maintain the mapping and for the data to be transmitted correctly to the proper Dataservice sensorpoint. The DC can even periodically examine if there have been any new additions or changes to the metadata, and automatically make the appropriate adjustments (for example, by adding a new sensor to the dataservice).

Seeing how certain networks have standardized interfaces, what this also affords us is a means to publish shareable DCs. For example, our BACNet DC should work with any existing network (although some customization might need to be done given the use of proprietary tags which contain the metadata). For custom networks that rely on a similar infrastructure (such as data stored in a RDMS), we have provided frameworks that can simplify the process. For example, our RDMS DC will query the underlying database, parse it to determine the relevant sensors and sensorpoints (this must be customized based on the actual schema), and

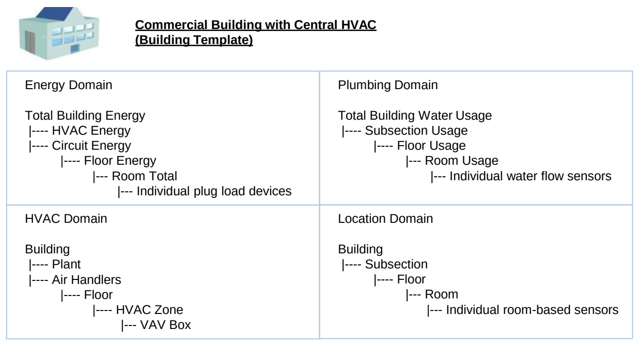

Figure 4. The Building Template for the type Commercial Building with Central HVAC (v1). This template specifies four domains for the building, each with its own hierarchy. Special predefined data points exist in some of the entities that map to commonly requested data values.

retrieve or transmit the sensorpoint data. Our goal is to have a library of DCs available for others to use.

3.3 Central Service

The Central Service (CS) acts as a gateway to the BD deployment for an institution. It serves two functions - it stores the building templates and models, acting as a directory for all of the sensors that the BD system contains throughout an institution, and it stores all of the user accounts across the institution. The user function is similar to the original BD User Service. The directory function is a significant improvement over the original Directory Service, which merely listed the sensors that existed in the Data Services. The BD2 CS now hosts the building models for the institution, each of which are based on a building template. These building templates define the structure of the building and provide a common language for applications and users to query.

Once the building model is defined, it can be connected to the institution's DSes through binding. Binding connects the CS to the DSes, and is the key to supporting reusable applications. As each building is defined by a building template, applications can query for specific known values. Binding is what maps the sensors that reside on the DS to the building model for the target building. The CS admins are effectively in charge of the entire BD deployment in an institution. They however can work with building managers and other facilities personnel to set the actual configuration of the system. We detail the components of the CS as follows.

Building Template - The Building Template (BT) defines the structure of a specific type of building. These are similar to the Sensor Templates that we introduced for the DS. Many commercial buildings are similar in terms of how they are laid out (multi-floored, divided into zones and rooms) and how they are provisioned for HVAC. For example, in our own CSE building, we have a typical HVAC deployment with air handlers feeding VAV boxes that provide air to the end-spaces (rooms). Many buildings in our own campus share a similar structure. Buildings that are similar can fall under a specific BT type. We have developed BTs for buildings that have a centrally managed HVAC system feeding to VAV boxes, and are looking to develop additional templates for other types of buildings (including residential homes which typically have direct exchange A/C units). A library of such templates can then be created, and applications that want to target a specific type of building can base their code structure against the targeted BT.

Figure 4 outlines the structure of our above-mentioned building template. We take cues from the Industry Foundation Classes (IFC), an industry effort to develop a standard way of describing buildings during construction and commissioning, and use their concept of domains. The BT for the commercial building specifies four domains that produce sensing data - energy, HVAC, plumbing, and location. Each of these domains maps an hierarchy from the top level (which represents the total building measurements) down to the end-spaces (which are generally rooms). The energy domain maps the overall energy consumption of the building, from the total amount, down to the circuits and then to the end-spaces (rooms), and would contain energy meter sensors (both industrial and plug-load). The plumbing domain maps the water usage of the building, from total water consumption to per end-space (typically bathrooms and kitchens). Water usage sensors are slotted in this domain. The HVAC domain contains the HVAC related sensors, and maps an hierarchy from the building (where the air-handlers are slotted) down to the HVAC zones (which are where the VAV boxes would reside).The location domain maps the location hierarchy for the building, from the building subspaces to floors down to the end-spaces (rooms). Room-based sensors can be slotted in the rooms under the location domain, and thus this forms a container for all other sensors that are physically located there (such as occupancy sensors). These domains are connected to each other - for example, the end-spaces under the location domain would be tied to a specific HVAC zone; a HVAC zone typically contains one or more rooms. Thus to access a room's HVAC values, you would query the parent HVAC zone. From there, you could see what other end-spaces that this HVAC zone is responsible for.

Each of the entries in the hierarchies for the domains contain special predefined datapoints that describe commonly sought-after attributes. For example, each end-space/room contains datapoints that describe binary occupancy, quantitative occupancy, and total energy consumption. If these sensors or values exist in the DS, they can be bounded to these points (see below for Data Service binding). These predefined points simplify application development and usage as they define specific points that applications can use to check if the data exists, and if so, access the data directly. We note that many buildings will likely not have sensors for many of these defined points. In that case, they would simply be empty, and any application that attempts to access the datapoint would simply see that there is no data there (and should handle that case accordingly). Also, because the sensor deployments of buildings continues to evolve, we version the templates so that any future increases in the library of sensors can result in a new version of that building template. Thus apps should target a specific version of the template that they are compatible with.

Building Model - The building model is the actual instantiation of a BT (similar to how a sensor is an actual instantiation of a sensor template), and describes the actual structure of the building, such as how many floors there are and what rooms exist. Once the building model has been created based on its template, context information can be added, such as the occupants of each room. The user accounts (which are described next) can be mapped to the rooms that they are occupants of.

User Accounts - User accounts are hosted on the CS, and these represent all of the occupants of the targeted buildings, as well as other interested users who would like access to the data and system. One aspect that we carried over from the original BD was the concept of using emails as user accounts - this means that all of an institution's users (who would have email addresses with the institution's domain) can automatically be enrolled and specified as occupants for their rooms. Users can be listed as occupant for a room/end-space, and any sensors that are set as occupant-accessible that belong to that location can be automatically accessed by the user. This allows for the building managers in charge of the sensor groups at the Data Service to pre-preemptively set permissions over their managed sensors without having to actively manage the ACLs. Otherwise, admins can set the permissions directly with these user accounts.

DataService Binding - Binding is what connects the DS to the CS. Each sensor represents data from a sensor network, but without context to how it relates to the building, application development can be difficult. Binding is the process in which the sensors are attached to the end-points in the building template models. As each sensor has location context data specified in the DS, the CS can retrieve that and use it to place the sensors under the domain entries where they belong in the building model. For example, plug-load meters hosted at the DS with a location to a specific end-space would automatically be added under the energy domain. VAV sensors would be added to the HVAC zone. This auto-population fills the building model with the sensors under the appropriate places from the DSes.

Next, the predefined datapoints can be manually bound if those sensors exist. If there is a sensor that measures overall energy consumption of a room (this most likely would be a virtual sensor that aggregates all of the individual plug load sensors), then this can be manually specified as that room's total energy consumption. Likewise for occupancy and the other predefined datapoints. A well-sensed building can potentially have many of these datapoints filled, giving application writers a very rich target to run interesting applications on. Figure 5 demonstrates a databinding example.

3.4 Application Service

The Application Service (AS) is a new and optional third system that allows for a tighter connection with the BD network. Previously, users who wanted to write an application against BD had to write against the REST API on another server. We realized however that many applications could benefit from accessing a faster interface. The AS provides a server system with libraries that allow for more direct applications to be designed and programmed, and uses the same internal RPC and messaging platforms that the DS, CS, and Data Connectors communicate on.

Currently, the application development environment for the AS utilizes Python libraries (and thus applications are written in Python). The presence of the AS helps satisfy the needs for the application writers and researchers. Combined with the template model, application writers can design their programs knowing how to access the relevant data that they desire. We note that it is possible to develop applications entirely through the REST API, and thus an app can be written in any language running on any computer, but the AS does provide a convenience and a central place where an institution's apps can be hosted.

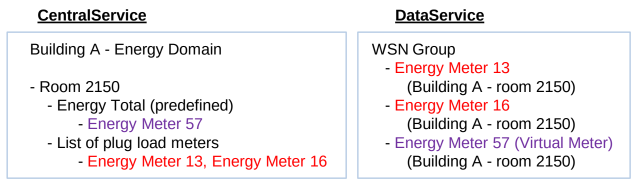

Figure 5. An example for Databinding. The DS lists three sensors (all with attached location metainfo). The CS automatically can add the 2 energy meter sensors as part of that location's plug-load energy meters. The third virtual meter, which represents an aggregate of that room's plug-load energy consumption, can be manually binded with the special predefined value of total-energy consumption for that room in the CS.

4 Implementation

Designing and implementing a building data analysis and control system must address several key challenges. As the system is designed to store and expose a large amount of data, the choice of underlying system infrastructure and datastore is important. Scalability is one issue - as the data grows, being able to add additional units to compensate is necessary. Another is latency and performance - users have certain expectations on data access speeds, and taking a significant amount of time to process data requests hinders the user experience. These requirements guided the technologies we chose to use for BD2. The most critical component of BD2 is the dataservice, as this system is what houses the data and responds to requests.

The DataService utilizes the Cassandra database to store the timeseries values, a MySQL relational database to store the metadata for the resources, and a Redis datastore to cache all of the values for faster retrievals. We average the timeseries values at 1 minute and 1 hour intervals to provide aggregate data when requested. The core software is a Python application that provides an API for interacting with the core DS resources. Interfaces can be built around this API that ultimately are accessed by client users. The web API is provided by a Python Flask application, and this includes both the webpage interface as well as the REST API interface. The fast API utilizes RPCs and the RabbitMQ messaging platform for faster access (relative to HTTPS). The deployment of the DS depends on the needs and IT infrastructure of the institution - Cassandra is a distributed datastore and thus a Cassandra cluster can be deployed if needed. The two other databases and the core software can also exist on their own VM/server. This type of separation is common in scalable web application development, and we take advantage of the work that has been done in that space to improve scalability for BD2. Alternatively, for a small deployment, all of the components can reside on the same VM/server. The

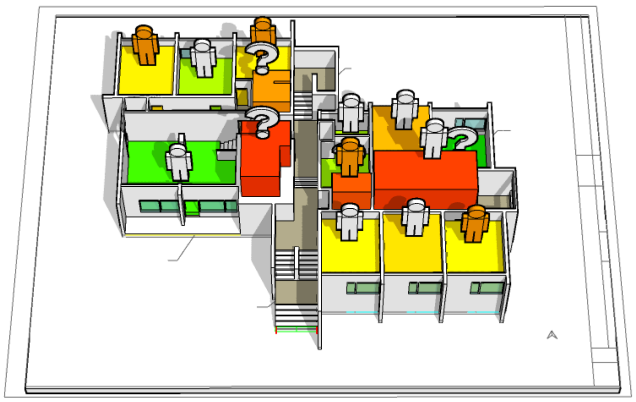

Figure 6. Screen shot of BuildingViz, an application that can display building data in a portable manner.

CentralService is similarly implemented, except without the Cassandra datastore. The main software exposes the core API for interacting with the resources, which are stored in the MySQL database and cached in Redis. Scalability in the constituent technologies is a core reason we opted to pursue a more centralized approach, as there exists clear paths for horizontally increasing the server assets to handle larger loads. Providing a central point of control also allows both building managers and researchers to better understand how the system connects with the underlying infrastructure.

5 Application Development

The new template system that BD 2.0 provides is how we approach the challenge of reusable application development. In this section we briefly describe how such applications can be written, and provide an example of some of the applications we have been developing for our own BD 2.0 deployment. An application can be written either on the AppServer (if available), or through the REST API. In either case, the steps are the same. As a simple example, if the application writer wanted to write an app that would actuate all the end-spaces/rooms based on occupancy, the steps would be as follows. The app would first ask for the user credentials that it would pass along to the CS and DSes. Then it would obtain a list of end-spaces, and look for the predefined occupancy datapoint. For the end-spaces that exist, it would examine the returned value from that resource. The app will go to the HVAC zone that is the parent of this end-space and attempt to actuate the VAV box (the actuation point is also a predefined point), assuming that the HVAC zone contains only one end-space. Similar applications can be designed for data analysis - an app can examine the energy usage of a building on a per room basis by examining all the sensors that match the energy meter sensor template.

Thus for researchers and app writers, BD2 provides a platform in which writing these types of applications is greatly simplified. Ultimately, the applications themselves can form a library that any BD2 deployments can use, increasing the value for such a system greatly. Our own lab has used BD to write some interesting applications - for instance we have written a WiFi occupancy application that utilizes WiFi traces to proxy for occupancy, and use that to command the VAV boxes in the HVAC zones in our building[3].

Another example is our BuildingViz application, which leverages the building templates and sensor templates in order to be portable across different buildings. The BV app is loaded with a map of the building and parses through the building to discover all of the rooms. The app uses the defined end-spaces (rooms) as the container for the sensors. As there are defined sensor types, the app can display relevant sensor data (such as energy meters or occupancy). These sensor points are defined with known template types and thus the BV app can determine how to populate the data points for each room based on the existence of data. It can then query the Data Service for the data points that are associated with the sensors. This app can be used across any building that conforms to a building template and uses sensors that map to public sensor templates, and demonstrates how portable applications can be developed. Figure 6 shows a screen shot of the app.

6 Conclusions and Future Work

In this paper we presented BuildingDepot 2.0, a significant upgrade over our first iteration of the system, which was targeted as a data storage system for building sensor data. BD 2.0 changes the structure of the system and takes a step towards allowing reusable applications. The key novel insight was the use of defined templates for both the sensors and the buildings in order to define a common language that applications (and users) can understand. We allow for expressiveness in both the sensor descriptions as well as the the building model by binding the two together.

Our current and future work revolve around increasing the number of standard templates in both the sensor template library and the building template library. We also hope to develop more automatic data connectors and useful standard applications. Like our first version of BD, BD 2.0is open source and available at buildingdepot.org, along with more detailed documentation.

7 References

[1] Y. Agarwal, R. Gupta, D. Komaki, and T. Weng. Buildingdepot: an extensible and distributed architecture for building data storage, access and sharing. In *BuildSys '12*.

[2] P. Arjunan, N. Batra, H. Choi, A. Singh, P. Singh, and M. B. Srivastava. Sensoract: a privacy and security aware federated middleware for building management. In *BuildSys '12*.

[3] B. Balaji, R. Gupta, and Y. Agarwal. Sentinel: An Occupancy Based HVAC Actuation System using WiFi Infrastructure in Commercial Buildings. In *Accepted at SenSys 2013*.

[4] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP: A Simple Measurement and Actuation Profile for Physical Information. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[5] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler. BOSS: Building Operating System Services. In *NSDI '13*.

[6] Department of Energy (DOE). Buildings Energy Data Book, March 2009. `http://buildingsdatabook.eren.doe.gov/`.

[7] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl. An operating system for the home. In *NSDI '12*.

[8] X. Jiang, M. V. Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a High-Fidelity Wireless Building Energy Auditing Network. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.