BuildingDepot: An Extensible and Distributed Architecture for Building Data Storage, Access and Sharing

Yuvraj Agarwal, Rajesh Gupta, Daisuke Komaki, Thomas Weng
Department of Computer Science and Engineering, UCSD
{yuvraj, gupta, dkomaki, tweng}@cs.ucsd.edu

Abstract

Enabled by various sensing and data networking devices, modern buildings are beginning to generate extraordinary amounts of sensory data. The organization and availability of this data is currently a challenge, especially for researchers who seek to devise intelligent data-driven methods for energy efficient use of building systems. Most current solutions tend to be ad-hoc and proprietary, and thus do not support mechanisms for easy data acess and sharing.

In this paper we present BuildingDepot, an extensible and distributed system for building-related data with scalable data storage, ease of data access, fine-grained data sharing and access control as first class design principles. We focus on the overall architecture and highlight how our own experiences running multiple building deployments have shaped our design decisions. We have implemented a prototype of BuildingDepot, along with connectors to several standard energy management systems, showing how it enables enterprises to incrementally deploy the system as well as Get and Put data into BuildingDepot using a REST-style API. We have released it as open source software to the building research community.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Online Information Services; C.2 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Management

Keywords

Buildings, API, Sensors, Data Storage, Applications, Wireless Sensor Networks, Web Service

1 Introduction

Improving energy efficiency in buildings has emerged as an important societal issue and research area. In the US alone, commercial buildings consume almost 35% of the total electricity generated [8]. More importantly, their electricity usage is projected to continue to grow significantly [5]. The advent of battery-powered wireless sensors have enabled buildings to be retrofit for improved monitoring of building processes. This has led to several "green-building" applications such as occupancy detection [2], plug-load energy metering [9], load-disambiguation [10, 16], lighting control [7] and fine-grained HVAC control [2, 14].

These sensor-based applications however generate an immense amount of data. Generic sensor data storage and management solutions have been researched in several domains with a focus on attributes such as privacy [4]. Buildings have certain characteristics however that provide unique opportunities that can be leveraged as well as challenges that must be overcome, as compared to other types of deployments. They often have explicit notions of spatial hierarchy (building, floor, room) that form an important attribute of the collected data. Because building applications rely on control over systems, both sensing *and* actuation support is needed. This control however is often distributed among various individuals, groups, administrators, building occupants, and even building processes; thus a way of assigning actuation ability in a tightly controlled manner is important. Authentication and secure data access is critical.

Also, building sensor networks can generate a significant amount of data. This data can be very valuable in terms of analysis, and thus the ability to share data within an enterprise and with outside entities is useful. For example, access to the large amount of data that is being generated can lead to discoveries in how buildings operate and what correlations exists between the various building functions along with geographic, climate, and occupancy diversities. Thus a comprehensive data access system focused on buildings that can address the issues of data storage, access, actuation, and sharing will significantly benefit the building sensor community.

In particular, there are several key attributes we believe a system should support. It should be compatible with multiple WSN architectures. The underlying fabric should be orthogonal to the deployment of the data storage system, allowing the institution to choose their own network. Flexibil-

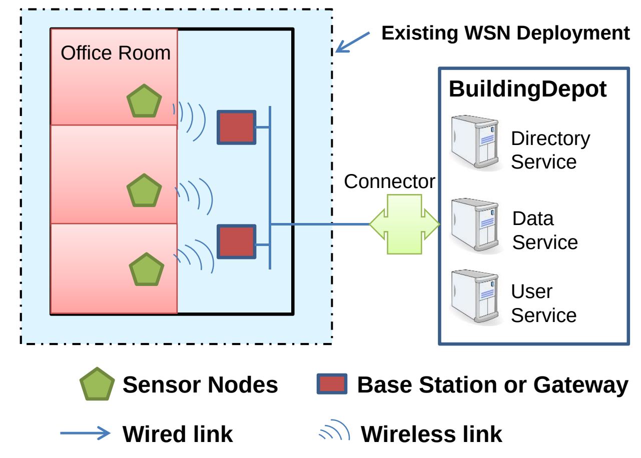

Figure 1. BuildingDepot connects to any existing sensor network deployment. Connectors bridge the gap between the wired infrastructure components (basestations, gateways, or routers).

ity and scalability are also important, ensuring that institutions of different sizes and needs can be supported. Institutions should have complete ownership, control, and access to their data. They make the decisions about who they want to share data with. To facilitate this, the system should provide rich support for users and groups that allow institutions the ability to set access policies for their own members as well as outsiders. It should also provide rich support for users and groups that allow institutions the ability to tightly set access policies for their own members as well as outsiders. Allowing for actuation ability (for example, an office room occupant who wants to control the temperature of their room through the building HVAC system) without giving the entity access to the actual network fabric is also an important feature. The system API needs to be well-defined, allowing applications to be implemented that can be re-used across deployments. Finally, there should be flexible mechanisms to search for and access different sensors and actuators.

Based on the above key attributes as well as our own experiences, we have developed **BuildingDepot** (BD), an extensible and distributed architecture for storage, access, and management of building sensor data. BD facilitates access to the data generated within buildings using user management tools that let institutions share their data to others in a controlled manner and allow standardized client applications to be built using a well defined API. Figure 1 illustrates the basic architecture of BD and where it fits inside a typical WSN deployment. A typical enterprise BD installation would, for example, consists of four components: core Data Server(s) combined with a web service that exposes access through a RESTful API; a Directory Service that maps the BD Data Server(s) in an institution; a User Service that provides access to outside users to the system; and Data Connectors that interface with the underlying network fabric to various BD data server(s).

This paper primarily focuses on the design and API of our system and seeks to explain the design choices that we made in light of our own data management efforts. We have released BD as open-source software (`www.buildingdepot.org`) and have provided detailed instructions and specifications for anyone wanting to set up their own installation. It is our hope that by releasing it to the research community, we can spur comments, feedback, and contributions to improve its architectural design and implementation.

2 Background and Related Work

Sensing within buildings has long been the domain of industrial control networks and processes. Many modern buildings, for example, have wired networks based on protocols such as BACNet or LonWorks that connect various sensors and actuators relating to maintaining environmental controls. These networks are only accessed by facility managers and thus the data that these networks generate (such as temperature) is not exposed. However, wired sensors are costly to deploy in buildings, and are usually only installed during initial construction. In contrast, wireless sensor networks (WSN) have emerged as a cost effective alternative for wider deployments, especially for building retrofits. Indeed, the research community has designed many interesting sensor networks in buildings, measuring such physical quantities as energy consumption [9, 12, 13, 17, 3], water usage [11], occupancy [2, 14, 1], and lighting levels [7].

A key challenge in these applications is storing and accessing the data that is being generated. Typically, data storage and aggregation architectures are different between deployments, and rely on a data server to store and later access the sensor data. Access to this server will likely be ad-hoc and require manual management of users. Some deployments might only store data for archival purposes, while others might expose some of the data in a web front-end, but any application that utilizes the data will need to be written specifically for each system. For one-shot sensor networks, BuildingDepot presents a uniform system that can act as a data store with a standardized interface. Rather than having to grant access to the internal elements of a network (such as the gateways or even the sensors), network managers can control user access through a single point, making user management much simpler. Networks then can remain "internalized" and out of reach from outsiders, while still providing the requisite access.

Recent efforts have attempted to provide unified interfaces for retrieving data from a sensor network. We highlight two of them as examples: sMAP [6] and Sensor Andrew [15]. sMAP [6] is a RESTful API for sensor nodes and other network elements that provides a unified interface to get data from a sensor network. For users with internal access to the sMAP-based sensor network, writing applications is simplified since regardless of the device, the interface is the same. However, sMAP does not address how outside access to the data producing physical devices should be handled. This is important because the implementation of sMAP assumes a resource constrained node - nodes that likely could not handle a significant amount of queries. In addition, the sMAP devices typically do not store much historical data, and users who want that information will have to go to a separate datastore. Sensor Andrew [15] is a campus-wide infrastructure

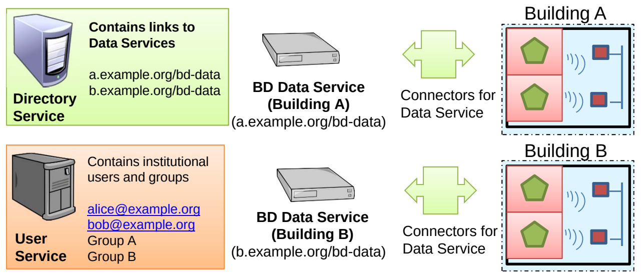
Contains links to Data Services
a.example.org/bd-data
b.example.org/bd-data
Directory Service
BD Data Service (Building A) (a.example.org/bd-data)
Connectors for Data Service
Building A
Contains institutional users and groups
alice@example.org
bob@example.org
Group A
Group B
User Service
BD Data Service (Building B) (b.example.org/bd-data)
Connectors for Data Service
Building B

Figure 2. High-level view of BuildingDepot. BuildingDepot consists of the three services. Each institution has one top-level Directory Service (and any number of lower-tier Directory Services), one User Service, and any number of Data Services.

that also seeks to provide uniform access across the various different devices in a large-scale sensor network deployment. Sensor Andrew includes a web application that helps manage the network and also store historical data. This component is similar to BD, but is only accessible to those who already have access to the underlying network (which does not solve the data sharing dilemma).

Both systems can form an ideal underlying backplane that can connect with BD for data storage and sharing, thus BD can be a complementary component attached to a sensor network deployment that is running a unified infrastructure. BD in these cases would allow a centrally managed data repository where people *outside* the network can access data using a standard interface, linking together several networks which an institution might have. Thus the core sensing fabric remains secure by limiting its direct access.

Commercial data storage systems like OSISoft PI exist, but are meant to retrieve SCADA data for analysis using proprietary tool kits. Data retrieval and sharing is restricted. Multiple cloud-based data stores have also been developed. Services such as Pachube and Nimbits present a REST API to input and retrieve data. Pachube is a hosted cloud service, while the latter has an installable open source implementation. Google PowerMeter was another REST based web service designed for storing smart meter data from home deployments, but is no longer active.

There are several limitations of the cloud services. First, the data resides with the services and institutions do not fully own it. Limited access control and sharing ability is also another issue. These cloud based services do not scale well since they essentially take a "one-size fits all" approach, which means that regardless of the institution, the interface is the same. The "flat" namespace approach may work for a small number of sensors from a single user, but has difficulty expanding to real deployments with many sensors.

In contrast to these systems, BuildingDepot is designed for building deployments and allows for a flexible hierarchical mapping of sensors, sharing, and actuation. This positions BD as a more effective system for storing building data that the cloud-based services.

3 Architecture and Specification

In this section we describe the architecture and API of BuildingDepot (BD). We wanted to satisfy several goals with the design of BD, based on our own experiences managing data. First, we wanted BD to be **scalable** and **incrementally deployable** so as to allow institutions of different sizes to grow their BD installations at their own pace. Second, we wanted it to be **flexible** so that different types of institutions with different data organization schemes can be accommodated. Third, we wanted the data in BD to be easily **searchable** so that users can find sensors and actuators easily. Fourth, we wanted it to be **extensible** with a standardized API so that a rich set of applications could be written against it. Finally, we wanted BD to allow institutions a rich set of **user management** features so that **data sharing** is efficient at multiple granularities.

BD is centered around three main services - a Data Service (DataS) that stores sensor data along with the context tags that describe each sensor, a Directory Service (DirS) that links to the DataS in an institution and stores context tags from child DataS to allow directory searching, and a User Service (UserS) that stores all of an institution's members and groups. Each of these services consists of REST web service software modules along with a connected web application that is installed on a physical server. In addition to the BD services, a fourth component is the BD Connectors – software that interfaces BD to the underlying sensor network fabric. Figure 2 illustrates the overall architecture.

An institution with many buildings can allow each building to control its own DataS and access privileges. The DirS binds together the DataS in an institution, ensuring that they are discoverable. Since the BD API is standardized across different devices, applications can be re-used across any BD deployment with only minor configuration changes. For ease of user management and sharing, we have defined three types of users for accessing building data - *internal users*, *institutional users*, and *external users*. Internal users have direct access to a DataS and are managed by the administrators for that DataS directly. Institutional users are those that are a part of an institution, but do not have an internal user account for a DataS. These types of accounts are managed at an institutional level, alleviating the need for DataS administrators to manage a large number of users. External users are those who do not belong to an institution, but wish to access data for analysis purposes. BD allows institutions to set which outside institutions and groups have access to its data.

3.1 BuildingDepot Data Service

The BuildingDepot Data Service (DataS) is the core component of BD and is responsible for handling the actual sensor data. The DataS exposes a RESTful API for retrieving data and inserting data into the system.

A DataS represents a single collection of sensors, where the actual meaning of the collection is defined by the host institution. For example, a small institution A with only two buildings and a limited number of sensors might choose to have a single DataS for both buildings. In this case, the DataS would be installed on a server `bd.example.org/ds`. Since there are two buildings being considered, each sen-

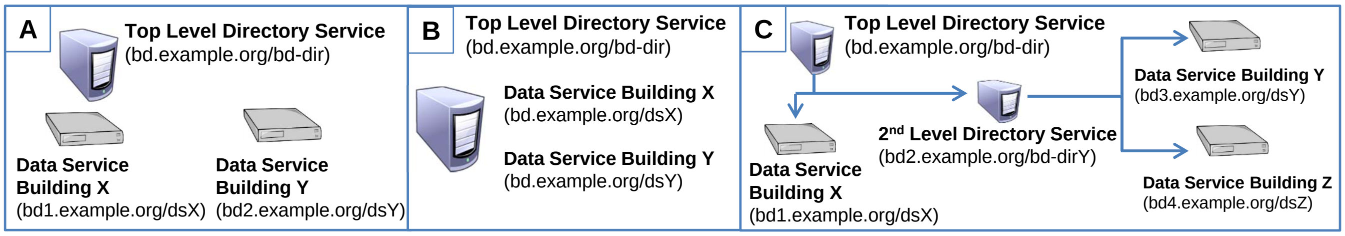

Figure 3. Example deployments for an institution with 3 buildings, X, Y (larger) and Z. In Fig (A), the institution chooses to have three servers, one for the Top Level Directory Service, and one each for the Data Services. In (B), all three services are co-located on the same server, handled at the institutional level. In Fig (C), another Building Z is added with a separate Data Service for each building. In addition, a 2nd level Directory Service is added to handle Building Y and Z's Data Services.

sor point (described below) would have a building context tag with the building it belongs to. Another institution B, also comprising of two buildings, might opt to have a DataS for each building, but with both services on the same physical server. In this case, the two DataS could be `bd.example.org/dsX` and `bd.example.org/dsY`. Both services are distinct, but reside on the same physical server. A third institution C, also with two buildings, might have a substantial number of sensors in each building. In this situation, the institution could opt to have a DataS representing each building on two separate physical servers. The two servers for this might be `bd1.example.org/dsX` and `bd2.example.org/dsY`.

This flexibility in hosting allows institutions to grow their BD installations as needed. This is in contrast to other solutions that assume a flat hierarchy and force users to adhere to those limitations. With BD, institutions can instantiate as many Data Services as needed based on their resources and actual sensor deployments. Figure 3 illustrates several example deployments for an institution with three buildings - X,Y,Z respectively. Another advantage of BD is that institutions can delegate data management authority to the actual building managers, rather than having to manage everything at an institutional level. Building managers can choose to control their own DataS along with access control for actuation and data retrieval. The DataS API consists of:

Sensor Devices - Each sensor device can have multiple sensor points. For example, an electricity meter device may have a sensor point for Watts, Volts, and Amps. `Tags` are used to mark context information on the sensor devices, such as sensor class, location and owner thereby allowing users to search for sensor devices. While sensor devices can map to a physical sensor (such as an energy meter), they can also represent conceptual sensors and thus do not need to directly map to one physical unit.

For example, an occupancy sensor could have an attached thermostat which could be represented as one sensor device with two sensor points (occupancy and temperature) or by two sensor devices, each with one sensor point. Each sensor device and sensor point has an access list that determines who can read the sensor data, the context data, and who can write into it. Figure 4 highlights the API for this resource. Sensor points can also be actuators, in which case `POSTing` a command will actuate the underlying device if the connector is in place to support it.

The data model for the sensor point's time series data relies on requests specifying a date range for a sensor point that the user wishes to retrieve. The response will contain both the time and value for the date range requested.

/sensordevice → lists out all sensor devices
/sensordevice/?{context_tag}={queryvalue}
→ applies a query to the list of sensor devices
/sensordevice/{sensordeviceID}
→ outputs context info for this device
/sensordevice/{sensordeviceID}/sensorpoints→ lists out all sensor points for this device
/sensordevice/{sensordeviceID}/sensorpoints/{sensorpointID} → info for this sensor point

Figure 4. Abbreviated API for the sensordevice resource.

Location Tags - These are tags that can be applied to a sensor device that correspond to its physical or conceptual location. Under each of the location tags that are defined, the actual location values can be set. The benefit of this scheme is that it allows the inherent hierarchy in building sensors to be explicitly defined, and allows users to find devices by location. Figure 5 highlights the location resource.

/locations → lists out all locations that are defined
/locations/{location_class} → info on that location class, an example would be /locations/rooms
/locations/{location_class}/{location}
→ specific location, e.g. /rooms/cs3130, this returns info on that location along with a link to /sensordevice/?{location_class}={location} which returns all sensors belonging to this location

Figure 5. Abbreviated API for the location resource.

Internal Users - This lists the users who have an account on that particular DataS. These users are internal users, that is, they only exist for a particular DataS and can have write access to the sensor points. These users must be manually created and approved by the administrators for that DataS.

Internal Usergroups - Lists the defined usergroups for that DataS. Internal usergroups can be defined by anyone with write privileges for this resource. Sensor devices can allow entire internal usergroups to have read access, making the process of defining who has access to the data more streamlined.

/users → lists internal users on system
/users/{user} → info on that user
/usergroups → lists internal user groups
/usergroups/{usergroup} → lists information on {usergroup}
/sensorgroups → lists sensor groups on system
/sensorgroups/{sensorgroup} → lists information on {sensorgroup}, lists the sensors and links

Figure 6. Abbreviated API for the users, usergroups, and sensor groups resource.

Sensor Groups - These are public sensor groups that can be defined by administrators or other internal users that have write access to this resource. This allows authorized users to publish a collection of otherwise disparate sensor devices that might have a common theme. Figure 6 highlights the API for Users, Usergroups, and Sn.

The design of the DataS API stems from our own issues with our prior data management solution. Previously we simply stored the data in a relational database, making access to the data inconsistent across our various network deployments. By defining location as a key component, finding relevant sensors becomes easier.

3.2 BuildingDepot Directory Service

The BD Directory Service (DirS) contains links to the various Data Services (DataS) for an institution. Like the DataS, the DirS is self-contained and can be installed on a server with other services or by itself, depending on the institutions preferences.

The DirS's main resource is a listing of DataS and DirS underneath it. These DirS can be used to form a hierarchical tree of all of the DataS and DirS that make up an institutions BD system. A DirS links to not only underlying DataS, but also other DirS that are conceptually beneath it in the hierarchy. This allows large institutions to design a hierarchy tree that works for the (many) buildings and sensor networks it might have. A DirS can also store context tags from children services. This feature allows users to be able to see any publically viewable sensor devices that are underneath the DirS, and provides a convenient way of searching by context tags (for example, if the user is only interested in occupancy sensors).

Every institution will have a top level DirS that acts as the entry point for that institutions BD deployment. For a small institution like the examples listed above, there will only be one DirS, and it will link to both of the Data Services. A large institution might have a hierarchy tree several levels deep. Figure 3 illustrates several example deployments of Data Services and Directory Services, including one with multiple (two) levels of hierarchy (Fig 3 C). Figure 7 highlights the API.

The DirS is an important component and helps makes the large amounts of sensors accessible to others. The design of this service was motivated by our experience in exposing our sensor data collection to others. Often the tags that specify a particular sensor would only reside as strings in a relational database, and thus anyone looking for data would need to parse through the list to find what they want. By presenting a standard directory service with context tags to identify systems and sensors of interest, users can more efficiently find sensors and data that are relevant to them across an entire enterprise.

/directory → lists out all subordinate services, including links and a description
/context -> contains context tags of subordinate services
/context/{context_tag} → contains information about that context tag as well as links to sensors that hold that context tag

Figure 7. Abbreviated API for the Directory Service.

3.3 BuildingDepot User Service

A key challenge of any shared data service is that of user management, specifically from the perspective of ease of data sharing and administration. The BD User Service (UserS) enables user management at an institutional level. Every institution that has an BD deployment will have one UserS for the entire institution.

The UserS has two resources - *users* and *usergroups*. Users correspond to institutional members that want to gain access to the Data Services within that institution. People who wish to create an account will do so using their email address – this becomes the primary identifier for the user account. User groups are collection of users that administrators can define, and are helpful for automatic sharing of data. Administrators can also allow others, such as building managers, to create usergroups by delegating administrative authority to them.

Individuals will register with the UserS using their email address. Once they are verified and have a user account, they can apply to join usergroups. Usergroups can be defined to have automatic acceptance based on email address wildcards – meaning that any user account can automatically join such user groups. Typically, an institutional group can automatically accept any account with the matching email address (e.g. "Institution-All" accepting any account with `@institution.org` email address). Automatic groups can be done with subdomains as well. Other groups can be maintained manually by the group owner or by other people that the group owner has given write privileges to.

The UserS is aware of all the Data Services within that institution. In order to request data from the UserS, each DataS has an access key that it uses to request data from the UserS. Once a user is registered and has an account, they will be able to generate single-site access keys (through the web application) to specific Data Services. These keys enable the user to authenticate with the DataS when requesting data.

Each sensor device stored in the DataS has an access control list that determines who has read access to it. An institution might not want certain data points to be publically accessible, but would expose it to members of its own institution. In that case, it can give read access to the group "Institution-All" and anyone in that group can view the data for that device. When a user in that group sends a request for that resource, the DataS will query the UserS with the user's single-site access key to A) verify that the user has the correct authentication key, and B) request all the groups that the user is a part of. The UserS will only respond with the information if A) the DataS matches that access key and B) the access key authenticates the user correctly.

If the user's authentication is correct, and the user belongs to a group that has access to the device, then the user will be able to retrieve the data. The DataS will record the user authentication information in a local database to speed up subsequent requests. This information will be valid up until a configurable time-out setting (we use one day as the default currently), after the expiry of which the DataS will query the UserS again. This ensures that user information does not become too stale.

Note, using email addresses as the user account name has one particular advantage. A building manager can preemptively mark devices as readable to people before they even register for an account with BuildingDepot. For example, during commissioning of a building occupancy sensor network, the managers can mark who has access to the data based on their email address. Later, if one of the occupants wishes to see the data, they can register for an account, and then access the data from the DataS immediately since their email address was already marked with read access.

In addition to generating access keys for each of the Data Services, the UserS will automatically generate a master access key for the user. This master access key will reveal not just the groups the user is apart of, but also all of the user's site-specific access keys. The user can use this key with trusted clients to access all of his user keys (for all of the data services) so that the client can access data on all of the Data Services. Figure 8 highlights the API for the User Service.

/authentication → Put/Post resource for authenticating a user with site-specific access key. Returns "invalid" if user and authtoken do not match, "valid" along with user's grouplist otherwise.
/inforequest → Put/Post resource for external User Services to query a user's group list. Must provide external access key for user.
/users/{username} → Returns info on username. Must be validated with credentials.
/users/{username}/sensors → Returns list of sensors for this user. Must be validated.

Figure 8. Abbreviated API for the User Service.

3.4 BuildingDepot Data Connectors

BD Data Connectors (DC) are how BD interfaces with the underlying sensor network. Each sensor network can differ in how the data is stored, where the data is generated, and what the relevant access points are. DCs are a framework for writing programs that can retrieve data from the sensor network and write data back (for actuation). Essentially, a DC retrieves data from the sensor network and sends it to the relevant DataS.

One key challenge with Data Connectors is managing the mapping between data items on two different systems. A DC is a piece of software that will retrieve data from the physical sources (which might be a collection server, base station, or the sensors themselves) and sends them to a DataS. The connector then has two parts - the retrieval part, which must be customized for the actual data source, and the sender, which is universally defined as a post request into the proper data point on a DataS. The DC is responsible for the mapping between the two as well. The benefit of DCs is that once written, they can be easily re-used in other deployments.

4 Usage

We give an example of how an institution consisting of several buildings can deploy and use BuildingDepot services. Our example is based on a hypothetical academic institution called Any University (AU) as shown in Figure 9. The AU campus comprises of two schools - the Engineering (SoE) and Arts (SoA). SoE has three buildings (CS, EE, and ME) while the SoA has two buildings (Arts and Social Studies). The three engineering buildings have deployed internal WSN, which happen to run on different protocols. The two buildings in SoA have no WSN.

The AU campus has a central facilities group that maintains a BACNet network for HVAC control. Each of the five buildings is connected to the BACNet network and has wired thermostats in every room along with a damper that controls the warm or cold air flow into the room. In addition, the facilities group has deployed industrial meters at the mains for every building as well as at the campus level (to measure total campus energy consumption) - these are connected through a wired ION network.

Figure 9 illustrates how AU might choose to organize their campus BD installation. Each of the SoE buildings has their own Data Service which stores sensor data (occupancy and plug-load meters) and is controlled by the building managers for each building. The two SoA buildings do not have a sensor network and therefore do not need their own DataS. However, the building managers in the SoA want control over the HVAC systems in their building, and thus have a shared DataS running on a server maintained by the campus facilities group. The facilities group maintains the BACNet and ION networks and has an OPC server with connector software that can retrieve the data values from the two networks and send them to the appropriate DataS.

The three SoE buildings have deployed additional client applications based on their sensor networks. Importantly, using BD building managers can still effectively run custom applications at the sensor fabric level (using Sensor Andrew or sMAP for example), without having to provide access to the underlying infrastructure except to trusted users. Thus BD is not attempting to replace such systems but rather augment them. In fact, in our own building we have our own sensor deployment which we trivially modified to use an BD connector and as a result we can use it for our sensing and actuation application. The tight controls on user access as well as the fact that server class machines offer much more in

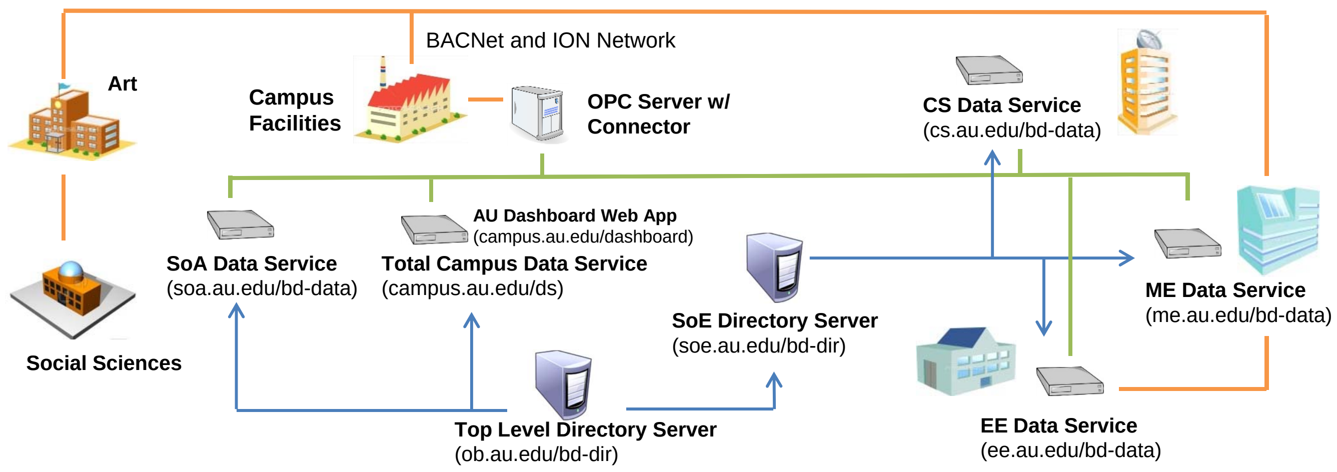

Figure 9. A hypothetical BuildingDepot installation at an Any University. BD enables data from different sensor networks deployed across multiple buildings to be accessed and stored as well as be searchable using the Directory Services. The administrators are flexible to tailor their deployment of Data Services and Directory Services as they please and can incrementally deploy BD services.

terms of capabilities that a sensor node makes BD a good solution for opening up data retrieval and access to the broader audience.

This is merely an example of how an institution can structure BD according to their own needs, highlighting its flexibility. An institution can start with just one DataS for one building, and incrementally add additional services as needed. Alternatively, an institution can plan for a larger deployment and prepare multiple servers for use in BuildingDepot. In any case, an institution with an existing deployment can quickly connect it to BuildingDepot, and determine what the required server infrastructure is based on the amount of sensors and data that will be expected.

5 Implementation

In this section we describe our implementation of the BuildingDepot architecture. While we plan to release our BD implementation as open source, it is important to point out that the goal of BD is to define the architecture, the API, and the specification while the specific implementations can change. In fact, part of open sourcing the software is to encourage different implementations.

We have implemented the services in BD using Python and the Flask micro-framework on top of Apache. Each of the services has a similar architecture. The main file utilizes the Flask framework to route web requests (to a specific URL) to the appropriate function. These functions contain the core logic that handles that request. We utilize models to abstract the database management systems that are used. Requests to a web application URL will return a templated web page, while requests to the REST web service URL will return a JSON response that contains the desired data. Configuration of the services (such as adding links or changing settings) is done through the web application interface and the settings are stored in a MySQL database.

Since security is vital, all services should be run over `HTTPS`. `HTTPS` utilizes the SSL/TLS protocols to create a secure channel over which all data is encrypted. In order to authenticate access, we use `HTTP basic authentication`. Because `HTTPS` is used, the access keys, passwords and data will be encrypted.

The Data Service is the most performance-sensitive of the services, and thus we sought to optimize for storage performance. We use the 2 DBMSes - MySQL (meta-data) and Cassandra (timeseries data). The use of Cassandra allows multiple back-end databases to be used for a given Data Service, as Cassandra is designed for distributing large amounts of data over multiple servers.

The Directory Service utilizes MySQL to store the links to its children. The schema for this is similar to the DataS. In order to grab the context tags, a crawler program must be run periodically to crawl through the children DataS and store all the context tags.

The data store for the User Service is a relational database (MySQL) and several tables store the user and groups information for the institution. Additionally, the UserS must be configured with all of the DataS in its institution to generate access keys for each DataS to use (the DataS must be configured with their unique key to be able to query the authentication resource). Three tables store the access keys - one for the main access key, one for the site-specific institutional access keys, and one for external access keys.

We have developed several Data Connectors to release as part of the BuildingDepot Connector library. An OPC Data Connector is one that will be useful in many deployments. This application is a Windows application that is built on top of ClientAce OPC library that will retrieve OPC variables and POST them to a specified Data Service. Similarly, we have written a BACNet connector that allows building managers to connect BACNet resources witih BuildingDepot. We have also developed a connector for our own sensor network deployment, which utilizes intermediate basestations. The connectors reside on the basestations, and will POST sensor data to the Data Service. We are also examining using internal Data Connectors that reside on the Data Services themselves and write the data directly into the data stores when a large number of data points must be inserted. We are continuing work on developing a large number of Connectors for use by the broader BuildingDepot community, such that a large library of these can be used by building managers and researchers.

6 Conclusion and Discussion

In this paper we describe BuildingDepot, a system for data storage and data access for sensor networks within buildings. We have discussed our design goals and how it differs from existing systems. We describe our architecture and how it addresses our original design principles of flexibility, scalability, data sharing, data access, and storage. We put particular emphasis on how the different services work together to allow ease of data management for host institutions. By giving full control to host institutions, we hope to spread its use as a means of sharing building-related data across the community.

We also present an implementation of BD that we have deployed on our own campus. As web services are a rapidly evolving field, we are also continuing to develop the architecture as new technologies become mature. We are releasing BD as open-source software, and we are in the process of communicating with other groups and institutions to deploy BD at their campuses. Based upon our experiences of using the BD system, we hope to continually improve and upgrade it while feedback from others will also drive new design decisions. We also hope to iterate over the design with the research community and present findings from an extended deployment of the system.

Due to the standardized way of accessing data in BuildingDepot, applications written on top of BD can be used on any BD deployment. This is an extremely useful aspect of BD, as a rich set of applications can be built that the entire community can use. To help spur this along, we plan on releasing useful applications to the community, including a dashboard that can visualize the data and a graphical building interface to enable easy actuation and control. This will allow BuildingDepot to be useful in most use scenarios, allowing end-users such as building managers to simply use the system as-is without having to worry about developing their own tools and systems.

The key focus of this paper is the design of the overall system and the API that is used to access it. We believe the distributed nature will help facilitate deployments that are able to grow as needed and that the API reflects our own struggles with coming up with a solution that is flexible enough that any institution can tailor it to their needs, but standardized so that applications can be shared.

BuildingDepot is available to download from `http://www.buildingdepot.org`. Both the source code and an installable package is available. Tutorials and user documentation is also provided at the website.

7 Acknowledgments

We wish to thank Bharathan Balaji, Panagiotis Vekris and Seemanta Dutta for their feedback on earlier versions of the system and their help in benchmarking and improving its performance.

This work is supported in part by NSF grants SHF-1018632, CCF-1029783 and a UCSD Facilities smart building deployment grant.

8 References

[1] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. Occupancy-Driven Energy Management for Smart Building Automation. In *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2010.

[2] Y. Agarwal, B. Balaji, R. Gupta, and T. Weng. Duty-Cycling Buildings Aggressively: The Next Frontier in HVAC Control. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[3] Y. Agarwal, T. Weng, and R. Gupta. The Energy Dashboard: Improving the Visibility of Energy Consumption at a Campus-Wide Scale. In *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2009.

[4] P. Arjunan, N. Batra, A. Singh, P. Singh, H. Choi, and M. B. Srivastava. SensorACT: A Privacy and Security Aware Federated Middleware for Building Management. *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2012.

[5] California Energy Commission. CEC End-Use Survey, CEC-400-2006-005, March 2006. `http://buildingsdatabook.eren.doe.gov/`.

[6] 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.

[7] D. T. Delaney, G. O'Hare, M. P. Gregory, and A. G. Ruzzelli. Evaluation of energy-efficiency in lighting systems using sensor networks. *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2009.

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

[9] 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.

[10] D. Jung and A. Savvides. Estimating Building Consumption Breakdowns using ON/OFF State Sensing and Incremental Sub-Meter Deployment. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[11] Y. Kim, T. Schmid, Z. Charbiwala, J. Friedman, and M. B. Srivastava. NAWMS: Nonintrusive Autonomous Water Monitoring System. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[12] Y. Kim, T. Schmid, Z. M. Charbiwala, and M. B. Srivastava. ViridiScope: Design and Implementation of a Fine Grained Power Monitoring System for Homes. In *ACM Conference on Ubiquitous Computing (Ubicomp)*, 2009.

[13] J. Lifton, M. Feldmeier, Y. Ono, C. Lewis, and J. A. Paradiso. A Platform for Ubiquitous Sensor Deployment in Occupational and Domestic Environments. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[14] J. Lu, T. Sookoor, V. Srinivasan, G. Ge, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[15] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, L. Soibelman, J. Garrett, and J. M. F. Moura. Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation, CMU Tech Report, ECE-TR-08-11, 2008.

[16] A. Rowe, M. Berges, and R. Rajkumar. Contactless Sensing of Appliance State Transitions Through Variations in Electromagnetic Fields. In *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2010.

[17] T. Weng, B. Balaji, S. Dutta, R. Gupta, and Y. Agarwal. Managing Plug-Loads for Demand Response within Buildings. In *ACM Workshop on Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, 2011.