A System for Managing Physical Data in Buildings

Jorge J. Ortiz and David E. Culler
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720
{jortiz,culler}@eecs.berkeley.edu

Abstract

In order to reduce building energy consumption, we need a global, accurate view of the building. Many modern buildings have a sensing infrastructure that can be used to do fine-grained accounting and understand the complex interactions between systems and spaces. However, through our experience with an active, campus-wide Building Management System (BMS) system, we observe that BMSs are not well suited for this task.

In buildings, there is a fundamental relationship between systems and spaces. The Integrated Sensor-Stream Storage System (IS4) captures this relationship by constructing a resource hierarchy that names measurement instruments relative to their context. This allows us to bring together systems and spaces through the instruments within them. Furthermore, IS4 keeps track of changes in the hierarchy over time. It is also designed to simplify data acquisition and exploits the naming structure for simpler, context-dependent queries. We have implemented IS4 and are currently using it to monitor Cory Hall at UC Berkeley as well as independent deployments at Samsung, Intel, and Lawrence Berkeley National Laboratory.

Categories and Subject Descriptors

E.2 [**Data**]: Data Storage Representations; D.4.7 [**Software**]: Operating Systems Organization and Design

General Terms

Design, Implementation, Storage, Metadata

Keywords

REST, JSON, Graphical queries, Time-series

1 Introduction

Buildings consume 72% of the electricity produced in the United States [5] and although most modern buildings are heavily instrumented with sensors, they perform poorly from an energy-efficiency perspective [6]. Building management systems (BMS) collect and use the data from building sensor deployments. However, BMS's are primarily used to allow building managers to remotely manage a building and quickly identify problems, such as a clogged vent or a broken fan. More sophisticated data analysis is done externally by exporting the data.

Close examination of the data only takes place when the building is re-commissioned *every few years*. Performance is examined and the suggested changes are implemented. This process is much too slow to react to changes in the building as they occur, allowing inefficiencies to develop and remain for extended periods. Furthermore, in using an actual campus-wide BMS system, we found the exported format difficult to decipher. The supervisory control and data acquisition system (SCADA) – a sub-system of the BMS that manages the sensor data – exports the data as hundreds of files, tagged with their metadata. Moreover, the files are not self-contained and must be used in conjunction with the graphical interface in order to interpret them correctly. This makes it very difficult to construct a global view of the building.

In buildings, there is a fundamental relationship between *spaces* and *systems* and this relationship is brought together through the measurement instruments within them. SCADA attempts to capture this relationship in the file tags that are used, but it fails at capturing all the information through the tags alone. In this paper, we introduce a data management system for physical data in buildings that addresses the issues in current SCADA architectures. Our system – the Integrated Sensor-Stream Storage System (IS4) – captures this fundamental relationship and provides mechanisms for querying and managing the data over time.

Our naming scheme naturally fits into a RESTful architecture [7]; where names become URIs and queries are constructed with HTTP methods. IS4 also provides a publish-subscribe facility to export the data. Our system captures the relationship between spaces and systems by combining naming and querying. One can query the resource hierarchy as well as the timeseries data. Through this combination you get the context and location from the name and the data from device streams.

IS4 is currently used to monitor Cory Hall – 7-story, 10,000 square meter building on the UC Berkeley campus. The current instance is collecting data from over 3700 live data streams produced by over 50 electrical meters. Our next deployment will include 150 more data streams from 50 ACme meters [10] at Lawrence Berkeley National Laboratory. It is also being used in external deployments at Intel and Samsung.

2 Related Work

The individual components of IS4 have all been addressed in previous work [2, 3]. IS4's combination of certain components is novel for understanding and managing building energy consumption. Microsoft SenseWeb provides an integrated sensing and querying infrastructure for sensor data [11]. Interaction with the system is through a .NET API where users create objects that represent sensors. SenseWeb also provides a staged stream-processing query facility. In contrast, IS4's RESTful architecture exposes sensors and their context through the URI structure; providing a simple query interface for the sensor data.

Pachube [1] is a web application that collects sensor data using HTTP and Javascript object notation (JSON). It also provide facilities that allow you to visualize the published data. IS4 decouples the visualization facility from IS4 itself. Allowing application developers to write applications that *use* IS4, rather than rely on the specific web interface.

Pubsubhubbub [8] is a pub-sub system for atom and RSS feeds. To make sensor feeds available they must be converted to atom or RSS. Pubsubhubbub defines a distributed architecture for publishers (pubs) and subscription hubs that feed to published data to subscribers. IS4 provides similar facilities and can be set up in a distributed fashion. IS4 explicitly manages the data/metadata relationship through naming and operational semantics; a resource structure that explicitly references components of the building.

3 Motivation

In order to analyze a data stream you need to understand its context. By context we mean the location of the measurement instrument that produces the stream and the type of measurement it is making. In a building deployment, it is also necessary to know how it relates to sub-systems in the building. Fundamentally, the relationship between *spaces* and *systems* is brought together by the instruments within them. For example, the environment is maintained by the HVAC system, which is powered by the electrical system and both may share a power meter at the location where they are connected. There is also a thermostat in a room that triggers the HVAC system. The data streams produced by the power meter and the thermostat allow us to infer the relationship between the systems and spaces.

In order to obtain the necessary readings, we interpret the metadata for the streams of interest, make associations between them, and extract only the streams that are related before we do any analysis. In SCADA the association task is unsystematic. In this section, we describe how the data and metadata are organized in SCADA and problems with its exportation. We present a systematic approach for making and tracking associations over time.

3.1 SCADA data management

SCADA systems tag exported data files with the location of the instrument that produced the data, the system to which it is connected, and the measurement type. Figure 1 shows how the tag is typically constructed. The prefix characters reference the building, the integer following refers to the system using its identifier, the next few characters reference the room, and the final characters reference the measurement type. To attain more information about the device, the user interacts with the graphical interface, locates the sensor through a series of clicks, and finds the associated metadata. The user may also view the data over a given time interval and have it displayed as a graph.

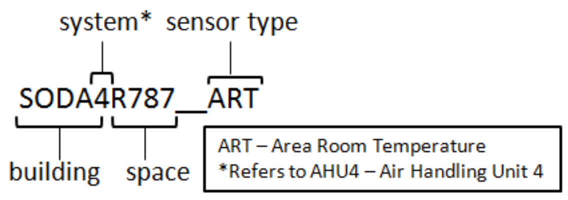

system*
sensor type
SODA4R787__ART
building
space
ART – Area Room Temperature
*Refers to AHU4 – Air Handling Unit 4

Figure 1. SCADA data tag example and data/metadata reference structure. The structure must be well known before the data can be properly interpreted.

Querying the data from outside the SCADA system requires that you 1) export the data, 2) learn the data/metadata reference structure, 3) design a schema based on the reference structure, and finally 4) import the data into a database. In these systems, there is a loose coupling between the data and the metadata. However, because of their relationship, we argue that they should be kept more systematically interdependent. Changes in the building structure (or deployment) must be reflected in the SCADA system to maintain the integrity of the interpretation of the data. However, since this is not done systematically, changes are difficult to keep track of and detailed data analysis loses integrity over time.

3.2 Tracking evolution

It is not uncommon for the structure of the building to deviate from its original design. Not only are there difference between design and implementation – due to unanticipated issues at design time – but over time, the building may serve different purposes. The electrical engineering building on our campus was built over 50 years ago. Since then, it has gone through multiple dramatic changes. Two extra floors have been added, one as mezzanine between floors, and one as a new floor at the top of the building. These changes forced the engineers to re-organize the electric load tree and a new HVAC system was installed along with the old one.

Besides structural changes, measurement instruments and system labels change as well. For example, if an electrical panel is moved from one location to another, the label on the panel is changed. This is often missed and it must be noted by the building manager. Measurement and actuation instruments may be removed, replaced, or added and must be re-associated with their new context. Again, this is generally noted by the building manager. In addition, small changes occur several times a week. Set-points are changed, schedules are changed, and all of it affects the behavior of the building and its energy consumption. If these changes are not properly tracked and analyzed, operational inefficiencies will be introduced.

3.3 SCADA design issues

Based on our assessment of the SCADA system used in the buildings on our campus we see various problems that need to be addressed in future system designs:

1. **Data and metadata are distributed across the file content, file names, and the user interface.** This makes it very difficult to interpret the data without detailed knowledge of the particular SCADA instance and building it was designed for.
2. **Changes in the building require updating the user interface and the reference structure between the data and the metadata.** Systematic tracking of data/metadata changes and association are necessary to ease overall management.
3. **Changes are not systematically recorded.** Without systematic record keeping, its is easy to lose track of the context of the data that was collected.

In the remaining sections we discuss our data/metadata naming scheme and describe several mechanisms for querying the collected data, querying across measurement points, finding interdependencies between systems in the building, and keeping track of changes over time.

4 IS4 System Design

IS4 has a web-services architecture and follows the principals behind RESTful services. The exchange format is JSON. IS4 provides references to resources that represent physical and logical entities and offers a historical and streaming query interface.

In this section we introduce the naming convention for the contextual information in the building. We show the hierarchical decomposition of spaces, the electrical load tree, and both sides of the Humidity Ventilation and Air Conditioning (HVAC) system (wet and dry). We also show how devices are named using this convention. We introduce *join-points* – devices named in multiple namespaces – and show how they are used to infer relationships between systems and spaces. Finally, we describe the query facility and show how the hierarchical organization of the metadata, join-points, and sensor data is used to understand complex processes in the building.

4.1 Namespaces

IS4 defines various resource types and rules for their hierarchical composition. The arrangement of resources in the hierarchy are named with a URI. Each resource hierarchy is organized with the building at the root. The children of the `/is4/<building>` resource are the divided into spaces, the load tree, and the HVAC system. [1] All the children of `/is4/<building>` refer to different things within the building. For example, `/is4/<building>/spaces`, `/is4/<building>/lt`, and `/is4/<building>/hvac` refer to the spaces, the electrical load tree, and the HVAC system, respectively. Each hierarchical namespace is treated as its own tree. We refer to them as trees and sub-trees hereafter. The HVAC namespace is divided into two sub-trees – *HVAC wet*, which is composed of references to the side of the HVAC system that moves water or steam around the building, and *HVAC dry* which moves air around the building.

At the leaves of each sub-tree are devices and their data streams. The same device may be referenced by multiple sub-trees, therefore the leaves become the point of intersection between sub-systems. Resource nodes that refer to space/system entities also contain a `/devices` child, whose children are the devices attached to that space/system. In IS4, we call a data stream a *publisher* and the children of a device are resource nodes that refer to a particular publisher of that device. The name of a publisher is constructed as `.../devices/<device>/<publisher>`.

Each namespace sub-tree can only contain a device-instance once. This ensures that each device has a unique name in that namespace. However, devices may be referred to across namespaces. This allows it to be named according to any context it belongs to and to draw links across namespaces. Given these links, we query and infer relationships between namespaces.

4.1.1 *Physical spaces*

The hierarchy of spaces in a building is partitioned by `/<floor>` or `/<suite>`. The children of either is `/<area>` or `/<room>`. Both contain the `devices` resources. The `/<room>` resource node may have `/<area>`-type children, but not vice-versa. Suites are like floors, except floors can contain more than one suite. Both suites and floors have one or more rooms. This hierarchy is derived from a web application that

[1] In the text we differentiate between explicit resource node names and names that are set by the user by include $<$ and $>$ on either end of the resource node that can be named by the user. The name in between refers to the resource node's type.

manages spaces called EveryBuilding [9]. Figure 2 demonstrates and example.

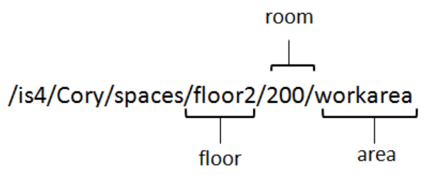


Figure 2. Spatial resource hierarchy.

4.1.2 The electrical load tree

The electrical load tree is the structure within a building that carries electricity to all electrical devices. It also powers the components that are part of the HVAC system and all plug loads throughout the building. Each resource node in the load tree namespace is called an element. In order to keep the resource hierarchy simple, we only have two types of resource nodes: a *panel* and a *load*. Panels have other panels as children. A panel may contain devices. A load may also contain devices, but a load has no other children. Figure 3 shows an example of the electrical load tree URI.

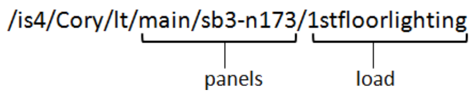


Figure 3. Eletric load tree resource hierarchy.

4.1.3 HVAC wet

The HVAC system controls the environment directly by moving air and controlling temperature throughout the building. Part of the system uses water (or steam) and contains various measurement/actuation instruments. We define the elements under `/hvac_wet`. The elements include a `CT` (cooling tower), `pump`, and a `Chiller`. Although it is not connected hierarchically, the HVAC system is set up in discernable stages that decompose naturally into a hierarchy when partitioned into two pieces. One piece represents the *source* side of an element and the other represents the *return* side of the element. Each physical element is represented by two resources. For example the cooling tower is represented by `CT_source` and `CT_return`.

Similar to the other namespace hierarchies, devices can be contained within each element. Figure 4 shows an example of the *hvac_wet* namespace.

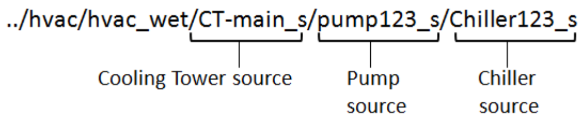


Figure 4. HVAC wet resource hierarchy.

4.1.4 HVAC dry

Another part of the HVAC system controls the movement, temperature, filtering, and mixing of air throughout the building. This sub-system is also instrumented with measurement/actuation devices. The setup is exactly like the *hvac_wet* side except with dry-side elements. The dry elements include `AHU` (air handling unit), `fan`, `heater`, and `AC`. Instead of *source* and *return*, we use the *supply* and *exhaust*. Figure 5 show an example of the `hvac_dry` resource URI.

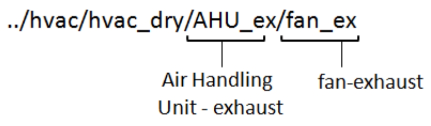


Figure 5. HVAC dry resource hierarchy.

4.1.5 Resource properties

Since IS4 has a RESTful architecture, the resources themselves respond to standard HTTP requests. As such, not only do you `POST` your data to a publisher resource or query it with a `GET` request that may include some parameters, you may also post the associated properties of the resource depending on its type. For example, a panel element in the electric load tree has contains voltage, ampage, and phase properties. You may query against these properties as well as the data streams collected from publishers. In the next section we describe some query features and how they can be used to discover interdependencies.

4.2 Join-points

Resources are virtual references to physical entities in the building. By managing each sub-tree as a standalone tree we acknowledge that namespaces are independent. However, we want to uncover their relationships. As mentioned in section 4.1, every device must be unique to its sub-tree. However, more than one subtree may refer to the same device. When two or more sub-trees reference the same device, we refer to that device as a *join-point*. Join-points are used to explicitly identify places where systems/spaces have a direct relationship. Figure 8 shows how join-points logically link our namespaces.

Join-points also allow us to reason the relationship between join-points and their neighboring measurement instruments. This is important for understanding complex inter-relationships through sensor data across devices in a similar contextual setting. Next we describe the query facility and show how the combination of join-points, temporal, and structural queries makes it easier to deduce inter-dependencies among physical entities.

4.3 Query interface

IS4 provides multiple query modalities for querying the attributes of resource nodes, querying the structure of the namespaces, and querying the data collected from the measurement devices. It also offers a query facility

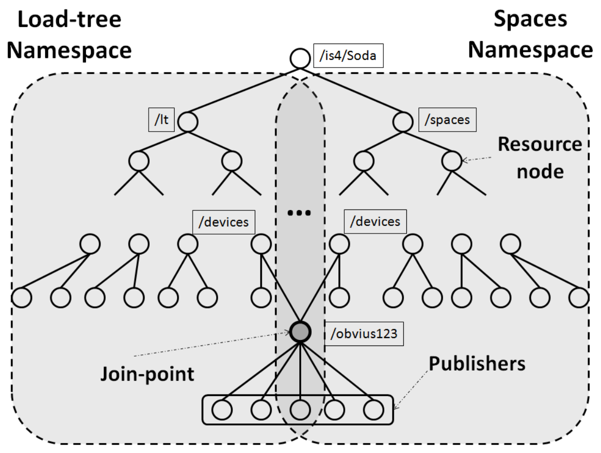
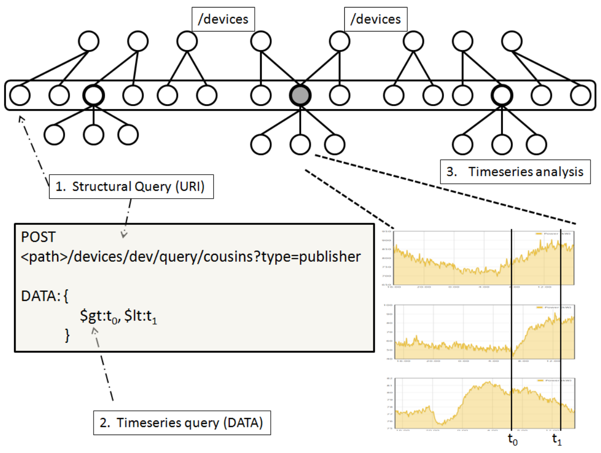


Figure 6. ***Join-points*** **are devices at the leaves of at least two sub-trees in our namespaces. These points of intersection are important for identifying the relationship between systems and/or spaces in the building.**

that lets your query the hierarchical structure over time similar to the work in [4, 12]. Figure 7 shows an example of this kind of query. Notice how the hierarchical structure is used to identify the devices of interest and their associated data streams. The ability to query the namespace structure is useful for drawing direct comparisons.

4.3.1 State discovery queries – an example



```
POST
<path>/devices/dev/query/cousins?type=publisher

DATA: {
        $gt:t0, $lt:t1
      }
```

Figure 7. Queries traverse the resource sub-trees associated with each building. This figure shows how the timeseries data for each device at the building-space room level are extracted for cross-correlation.

Not all spaces in the building serve the same purpose. Some house people while others contain machine or equipment. Usually this means their environments are maintained independently. Problems arise when this assumption does not hold and they are usually difficult to detect. For example, someone may leave a door open between independent spaces or a wall may have been removed without accounting for their independence. An example of independent spaces are a computer machine room and an office. The computer machine room is maintained at a lower temperature that the office.

Today, this is detected when someone complains about the temperature. The building manager checks he SCADA interface and walks around until he/she finds that the problem; perhaps a door left open. With IS4 you could write a script that queries the data streams of all the rooms in that area and correlates their readings over time. This is done by running a *cousins* query for all the cousin-nodes of the join-points between the spaces and the system. The cousin nodes in the hierarchy of the join-point might be relevant since they are either in the same location or on the same sub-system in the building. The data from devices in with similar or related context might reveal an important underlying relationship.

Figure 7 shows how the query would be constructed for a device called *dev*. The results returned from IS4 would contain all the data points for publishers associated with the cousin-devices of *dev*. The cross-correlation function is not a component currently in IS4, however data returned from the query could either be fed to an external processing component or a computation resource.

4.3.2 Snapshots

Like source control, we keep timestamped version of the resource hierarchy. Whenever changes are made to the resources, such as adding/removing a device or a publisher, creating/remove a resource node, etc, we timestamps the entire tree and save it. Their associate data is also serialized, timestamped, and saved. This is useful for running timeseries queries with respect to the right contextual information. This keeps the integrity of data interpretation intact.

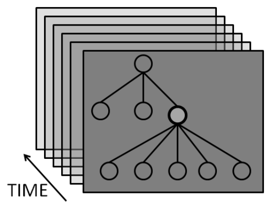


Figure 8. Snapshots are taken each time there are changes made to the resource hierarchy. Changes occur when devices are remove/added or the structure of building itself changes and that change is inputed into IS4.

5 Comparison with other systems

IS4 makes several contributions in the space of building data and metadata management:

1. Naming of measurement instruments, buildings, systems, and spaces and capturing their relationship by composition.

2. Type-semantics for the resource types and their construction.
3. Binding of measurement instruments and their context and tracking their relationship over time.

These contributions address the issues in current building management systems. The data is combined in a resource-oriented architecture, where the resources maintain resource-type specific metadata information and the resource name is constructed following systematic construction rules. This makes is easy to fetch and update the information associated with the data, instruments, functions, and context. Changes in the building only require an update to the name in IS4 that refers thing that changed and *snapshots* allow you to track these changes as they occur.

	O	MM	RS	S	Query		
					H	RT	G
Relational DB	+	+	-	-	+	-	-
Pachube	+	+	-	-	+	-	-
Pubsubhubbub	+	+	-	-	-	-	-
SCADA	-	+	+	-	-	-	-
IS4	+	+	+	+	+	+	+

Table 1. IS4 offers all the necessary features to address the issues in current systems that monitor building deployments: Open (O) architecture, metadata management (MM), resource semantics (RS), metadata and resource snapshots (S), historical (H), real time (RT), and graphical (G) query capabilities.

Table 1 compares IS4 with similar systems. We compared these system across various features, including an open architecture, metadata management, resource semantics, deployment snapshots, and querying facilities. Most of the other systems were not made specifically for managing building data, but could have been potentially tailored to serve that purpose. However, we found that a re-design was necessary, as the combination of features and capabilities contained in IS4 were necessary to address the data management issues in this space.

6 Lessons Learned

In buildings, understanding the relationship between systems and spaces is crucial for reducing energy consumption. Activities within spaces directly affects the energy consumed by the HVAC and electricity systems supporting them. Through our experience with a campus-wide BMS, we observed that their relationship can captured through the measurement instruments within them. We also learned that no current system captures this relationship effectively.

IS4 brings together systems and spaces through a combination of naming and querying. We name systems, spaces, and devices hierarchically. The name itself describes the relationship between building entities and the placement of devices. The element properties describes the entities themselves and the data shows its behavior. By querying the naming structure and the data, we naturally combine the physical structure and behavior of the system over time.

A RESTful architecture naturally falls from our naming and querying scheme. Resources fit into a URI structure and HTTP methods provide the calls to those resources. We are currently using our system in Cory Hall at UC Berkeley, as well as external deployments done by Samsung, Intel, Electricite de France, and Lawrence Berkeley National Laboratory. These will serve as useful instances for which to test the effectiveness of data/metadata management approach. For future work we will quantify the performance of IS4 and address the open security questions left unanswered in the current release.

7 Acknowledgements

This work is supported in part by the National Science Foundation under grants #CNS-0932209, #CNS-0931843, the FCRP MySyC Center, Intel Corp, and Samsung Corp.

8 References

[1] Pachube. `http://www.pachube.com/`, June 2010.

[2] J. Campbell, P. B. Gibbons, S. Nath, P. Pillai, S. Seshan, and R. Sukthankar. Irisnet: an internet-scale architecture for multimedia sensors. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 81–88, New York, NY, USA, 2005. ACM.

[3] R. Dickerson, J. Lu, J. Lu, and K. Whitehouse. Stream feeds- an abstraction for the world wide sensor web.

[4] Z. Ding, R. H. Gting, and P. I. Iv. Modeling temporally variable transportation networks. In *in Proc. of DASFAA*, pages 154–168, 2004.

[5] *Energy Outlook 2010.* Energy Information Administration, `http://www.eia.doe.gov/oiaf/ieo/index.html`, 2010.

[6] N. Gershenfeld, S. Samouhos, and B. Nordman. Intelligent infrastructure for energy efficiency. *Science*, 327(5969):3, 2010.

[7] N. Gershenfeld, S. Samouhos, and B. Nordman. Intelligent infrastructure for energy efficiency. *Science*, 327(5969):3, 2010.

[8] GoogleLabs. Pubsubhubbub. `http://code.google.com/p/pubsubhubbub/`, June 2010.

[9] S. D. Haggerty, J. Ortiz, X. Jiang, J. Hsu, and S. Shankar. Enabling green building applications. In *HotEmnets 2010 Workshop on Hot Topics in Embedded Networked Sensors*, 2010.

[10] X. Jiang, M. V. Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a high-fidelity wireless building energy auditing network.

[11] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 253–266, New York, NY, USA, 2008. ACM.

[12] J. Rasinmki. Modelling spatio-temporal environmental data. In *5th AGILE Conference on Geographic Information Science, Palma (Balearic Islands*, pages 18–877, 2002.