An operating system for sensor deployments in buildings

Jorge J Ortiz
Computer Science Division
Univeristy of California at Berkeley
jortiz@eecs.berkeley.edu

Abstract

Buildings consume over 70% of the electricity in the United States and much of it is wasted. The goal of my research is to reduce this figure and move towards more power proportional buildings. In order to achieve these goals we need a system that combines fine-grained monitoring, modeling, actuation, and visualization. For my Ph.D. thesis I am designing a systems that combines these facilities using operating system principles and interfaces based on open standards.

I propose a layered architecture that builds up from a sensor network deployment to a policy-expression interface that ultimately allows a user to express the dataflow from sensors through models, actuators, and visualization components. By giving the user the right set of primitives she can build arbitrarily complex scripts that express her desired management policy; a combination of observation and modeling gives the user better insight into what policies are most effective. My thesis will make various contributions in sensor network data management, stability and analysis of policies for actuation and control, and sensornet deployment management.

1 Biography

Jorge Ortiz is a student at the University of California, Berkeley. His research advisor is Professor David E. Culler. His expected graduation date is May 2012. This November he will be taking his Ph.D. qualifying exam on the topic presented in this paper.

2 Introduction

Buildings consume 72% of the energy produced in the United States amounting to over $300 billion in energy costs [7] . Furthermore, studies show that at least 20-30% is wasted [8]. In order to reduce building energy consumption we must observe its use and design ways to reduce the waste, schedule the load, and continuously repeat the process over time. This requires fine-grained monitoring, building models, actuation, and systematic management of the interaction between these components.

Many modern buildings already contain a large sensor network deployment as part of the building management system (i.e. about 1300 sense points in the computer science building at UC Berkeley). However, these deployments only monitor about 40% of the total energy of the building [11]. In order to capture the rest of the energy consumed, a richer monitoring infrastructure is necessary. The monitoring infrastructure produces additional data streams such as weather, activity, and measurements which can be used in concert with building and behavioral models to understand the energy consumption landscape. In order to have an impact on the profile for the building, we must close the loop through actuation, either by motivating the occupants or through direct actuation.

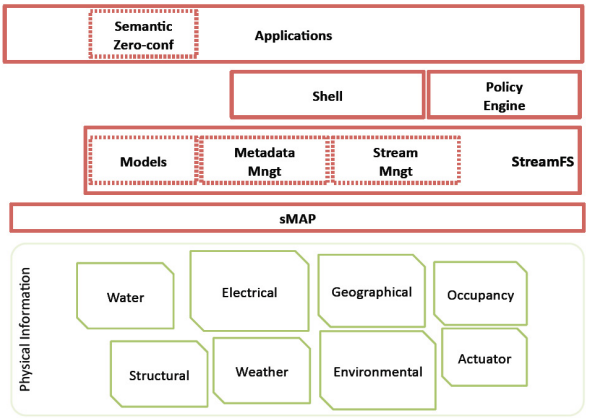

Figure 1. Building Energy-Management Operating System (BEMOS) architecture.

For my Ph.D. thesis, I am constructing a system that uses the recently proposed data abstraction layer for physical data – the Simple Monitoring and Actuation protocol (sMAP) [9] – and adds a data collection layer with a file-system interface and facilities called StreamFS [13]. Together these systems provide a platform for installing models and provide a uniform interface for referring to data streams, deployment metadata, models, and actuators that will be used by a policy expression interface. Furthermore, StreamFS implements

a standard interface for piping data streams from the data source to a given target. The concept of a target is open to either a modeling "file" within StreamFS or an external application URL; the latter provides a simple interface for exporting the data to visualization components.

A policy expression engine will be built on top of StreamFS. The engine will be used to interpret policy scripts expressed as declarative dataflow graphs; combining ideas from previous scripting languages such as Occam [15] and Click [10]. The policy expression engine will refer to nodes in StreamFS and all process management will work back to the file system, as process facilities are materialized in the *\proc* directory on Unix-based file systems.

I am also currently developing a standard resource hierarchy within StreamFS, also similar to the file-system hierarchy standard used in traditional Unix file systems [1]. The standard portion of the hierarchy is used to manage the policies installed, the StreamFS setup, and the representation of the deployment in the file system. To ease the integration of newly installed sensors to a deployment managed by StreamFS, I will be working on a zero-configuration protocol that uses the structure of the file system as semantic information about where sensors are in physical spaces and attempt to deduce where in the file system the newly installed node should be placed; its placement in StreamFS should reflect its deployment context.

3 Related Work

My work draw from various areas in computer science. StreamFS is largely inspired by work in the file system literature [12, 16, 17], streaming databases [5, 4, 2], and distributed operating systems [19]. StreamFS combines mounting and symbolic linking as a single operation, however, rather than simply ease application development on a distributed file system, StreamFS uses this functionality to enable horizontal scalability and to combine StreamFS deployments across multiple buildings. The model "files" in StreamFS take concepts from steaming databases, as operators, such as averaging, interpolation, and prediction, are performed on the streams of data coming into the system.

The policy engine draws from work on dataflow programming such as Ptolemy [3], Occam [15], and Click [10]. In order to combine the collected data streams and models, my system will allow users to express how the data flows between them. Click allows users to combine components in a dataflow graph that performs operations on data packets and routes them according to the policy expressed by the click script. Occam combines processing into its syntax by letting users write functions of operations to be performed on sequences of data and forwarding data between processing components. Ptolemy provides a way to express your communication model and system composition. My policy engine work is most closely related to Click and Ptolemy.

The system as a whole is also closely related to work in pervasive computing, smart environments [6, 14], and industrial building management systems. BREXBAS [18] is an expert system that manages the building by collecting space temperature to evaluate current conditions and determine if there are any problems. If any inefficiencies are detected the system provides advice to the end user. This work is different from BEMOS in scale and approach. BEMOS integrates at least two orders of magnitude more data streams that BREXBAS and approaches the problem based on effective stream data and metadata management.

4 BEMOS Architecture

This section describes the main components of the BEMOS architecture. The bottom two layers – sMAP and StreamFS – have already been developed. For my thesis I will design and implement the modeling component of StreamFS, the policy engine, a policy script interpreter, and a policy script runtime engine that uses StreamFS to manage the resources referred to by each submitted policy script. I am also currently involved in building various applications over StreamFS that include ways to ease deployment management through automatic registration of sensors in a given deployment through a semantic zero-configuration protocol.

4.1 Data abstraction layer

In order to integrate data streams from different sources we designed a data abstraction layer and an architecture to expose the data streams called the Simple Monitoring and Actuation Protocol (sMAP). sMAP defines a common data representation format using javascript object notation (JSON) and provides a RESTful web services architecture that can be implemented either directly on a measurement device or an associated proxy for the measurement device. sMAP decouples the physical sensors from the data they produce and enables applications to innovate based on the data rather than the particular set of devices that are producing the data.

4.2 Data management layer

StreamFS brings together the relationship between streaming data and their associated metadata as well as deployment management facilities through a file system interface. There are four types of nodes in StreamFS: r-nodes, s-nodes, m-nodes, and a-nodes. R-nodes are generic resource nodes that represent systems or spaces in your deployment. For example, *\Soda\spaces\floor4\room410* is the name of the resource node that represents room 410 on the 4th floor of the Soda building. Every r-node has properties that are updated by the user. StreamFS proposes a schema for *space* resource properties, as well as *HVAC system* components, and element in the *electrical load tree*. However, the design is generic and open to any hierarchical decomposition of a deployment. S-nodes are stream nodes; nodes in the system that represent streams of data. S-node resource properties are also updatable,

however, the main difference with r-nodes is that s-nodes data is stored in database. M-nodes represents models or processing. M-nodes accept incoming data and perform processing operations on them. They also output data to target URLs or other m-nodes. Finally, a-nodes represents actuators and are linked to the actuation channel for a given sensor. Figure 2 shows the StreamFS architecture.

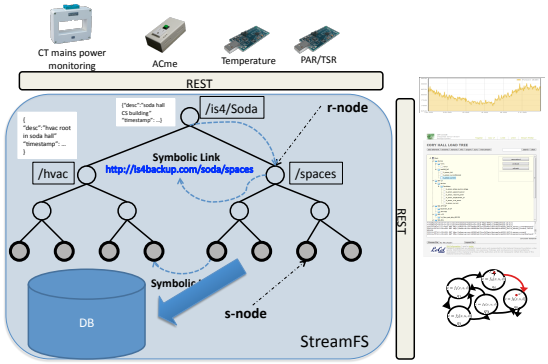

Figure 2. StreamFS architecture. StreamFS provides a combination of a file system interface over a RESTful HTTP layer for creating and deleting nodes in the hierarchy that represent the sensor deployment and allow us to connect streams, processing/models, and actuation.

StreamFS also exposes a RESTful interface, so all properties and data streams are queryable. Furthermore, StreamFS provides a hierarchy querying features that allows users to crawl the resource hierarchy, filter by node properties, and query timeseries data in a single call. StreamFS is designed to handle hundreds of thousands of data streams. If more data streams need to be added, the files system can be spread across a cluster of machines and symbolically linked. This allows you to spread the load horizontally and scale arbitrarily. The querying facilities are built to run on the entire file system, even if it is spread across many machines.

4.3 Policy engine

Policies will be written in a high-level language that uses both models and live sensor data in order to make actionable decisions – such as turning off a light switch or alerting the end user of an faulty sensor. I am designing a simple language similar to Click [10] that allows users to declaratively express the dataflow between s-nodes, m-nodes, a-nodes and visualization components. Since users are effectively performing operations and actions on streaming data, a dataflow scripting language can be used to express how data flows between such elements.

A simple actuation script to control room temperature could be written with three elements: a temperature sensor element, a threshold model element, and an air-conditioning actuator element. The threshold element consumes the temperature data streams, constantly checking temperature values. While the threshold element sees values below threshold, it produces a "0" output. When the threshold is exceeded, the model produces a "1". The AC element will interpret a "1" as an ON signal turn the AC on. Once the temperature falls below threshold, the model element will emit "0" again, causing the AC actuator to turn the air conditioner off. A similar script can be written to control the heater and arbitrarily complex scripts can be constructed with more sophisticated modeling and more complex dataflow configuration.

4.4 Policy and resource management

Policy scripts may conflict and this needs to be detected when the policy is "compiled". Policy script reference actuators and if the actuation conditions contradict each other, the actuator and system it is controlling can be put into an unstable state. This should be caught before the policy script is run. My work will statically evaluate policy scripts and determine where conflicts may occur. By representing each script as a state-transition graph and simulating the activity, I will perform stability analysis by observing the frequency with which the system reaches certain states. The main challenge is determining what the states are, when a transition has occurred, and what frequency constitutes instability. This is especially tricky in building systems because most systems naturally fluctuate between states as the building system works to maintain a particular environmental comfort. A much simpler approach is to construct a reservation system where actuators are reserved by specific policy scripts, or reserved for a certain period of time. This can at least limit the stability analysis to a single policy for a fixed period of time.

4.5 Shell and applications

I have built a shell over StreamFS that allows you to directly interact with any StreamFS deployment. With the shell users can manage their policy files, their model modules, and their streams. They may also view and control what processes are currently running, scale their deployment, and direct streams to external processes, such as visualization components.

5 Contributions to Sensor Networks

My thesis will make several contributions to sensor networks. StreamFS provides a novel way to manage streaming sensor data. Using a file-system structure over a RESTful interface, StreamFS provides a natural structure for organizing deployment metadata and streaming sensor data. Furthermore, file system facilities, such as symbolic linking and piping, allow users to scale StreamFS horizontally and share readings with external applications. StreamFS proposes type-specific resource schemas as well as a standard hierarchy when decomposing systems, spaces, and the electrical load tree.

This provides a structure for organizing and querying information about your building-wide sensor deployment.

StreamFS will also provide a way to manage model/processing components. When coupled with piping it will serve as a tool for managing dataflow between sensor streams, model/processing, and actuation. These StreamFS primitives will be used to build a high-level language for real sensor network deployment. This is another contribution to the sensor network community. It has been a challenge to streamline the deployment process with how it is managed, shared, and used by applications. Although my application is for building energy management, the techniques and tools that I am building generalize to a wide range of sensor network applications.

Using StreamFS as the main platform for my deployment, I will work to address the automatic configuration problem in sensor network deployments. With the standard hierarchy proposed by StreamFS, semantic information can be deduced about the deployment setup from the hierarchy itself. My goal is to get sensors to use this information to determine, with high probability, where in the deployment hierarchy they should be placed. Initially, my solution will involve a human-in-the-loop but online learning techniques might be able to improve the accuracy of the semantic zero-configuration scheme I formulate.

Finally, as the number of sensors and integrated sensor streams grow, I will surely encounter performance and management scalability issues. I expect important lessons to be learned that could be useful to the sensor networks community about how to manage large deployments.

6 Discussion and Future Work

My PhD thesis touches upon various areas in computer science, and more specifically, sensor networks. StreamFS deals with sensor data management issues in this space, as well as deployment management. The model processing component manages a set of external processing operations in a uniform fashion and allows users to test their models on real data. The policy engine will explores ways to express policies by abstracting away the sensors and logical entities as strictly resources that consume and produce data streams. There will also be contributions in the area of stability analysis and zero-configuration.

In the future, I believe that BEMOS will make contributions to the community that it does not make directly in its design. BEMOS will be a valuable tool to make progress on the problem of managing building energy consumption and, more generally, managing a deployment of sensors as resources that produce streaming data. BEMOS is agnostic to the internals of the models and how they are used, but it provides a way to test your models in a real setting and lets you test how effective those models are at optimizing performance on a set of metrics. It will also ease the coupling of end-user actions and how the sensor deployment is used to perform certain actions.

In addition, BEMOS decouples policies from a particular building. This allows a user to express various management policies and execute those policies across several buildings. This may prove to be a very effective way at reducing energy consumption quickly across many buildings and maintain their energy consumption low over time. Overall I believe the contribution of my thesis can potentially leave a lasting impact on the community and I intend to release and support all the code that I write.

7 References

[1] http://www.pathname.com/fhs/2.2/.

[2] D. J. Abadi, D. Carney, U. etintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management, 2003.

[3] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems, 1992.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertan world, 2003.

[5] J. Chen, D. J. Dewitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *In SIGMOD*, pages 379–390, 2000.

[6] M. H. Coen. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, number SS-98-92, pages 547–554, Madison, WI, 1998. AAAI, AAAI Press.

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

[8] N. Gershenfeld, S. Samouhos, and B. Nordman. Intelligent infrastructure for energy efficiency. *Science*, 327(5969):3, 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] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[11] G. Levermore. *Building Energy Management Systems: Applications to Low-Energy HVAC and Natural Ventilation Control*. E&FN Spon, 2000.

[12] M. K. Mckusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for unix. *ACM Transactions on Computer Systems*, 2:181–197, 1984.

[13] J. Ortiz. A system for managing physical data in buildings. Technical Report UCB/EECS-2010-128, EECS Department, University of California, Berkeley, Sep 2010.

[14] S. Peters and H. Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth, TX, USA, March 2003. IEEE.

[15] A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. *Theor. Comput. Sci.*, 60(2):177–229, 1988.

[16] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10:1–15, 1992.

[17] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation or the sun network filesystem, 1985.

[18] M. R. Shaw. Applying expert systems to environmental management and control problems. pages 141–151, 1988.

[19] A. S. Tanenbaum, G. J. Sharp, and D. B. A. The amoeba distributed operating system. 1992.