NDBAS: A New Distributed Building Automation System

Why yet another BAS?

There are any number of commercial products and open source building control systems in the world, why does it need another one? The problem grows out of the proliferation of smart items within the building that people then want to be tied together into a coordinated system. Some of these are fancy commercial products and some are home brew widgets based on inexpensive microcontrollers and sensors.

The problem is that building automation system come from a history of monolithic control systems build with expensive control hardware and expensive to develop and maintain software. The reason for this approach is quite simple. The reason for this is clear to those who build the systems: synchronization and coordination of complex application logic is a Hard Problem [tm]. Any number of approaches promise to make this simple, but they almost always do this by making assumptions about either the control logic capability, the development and runtime environment or the ability of the user to understand the system as designed.

NDBAS attempts to address this by embracing the fundamental issues that the multitude of smart devices (often referred to as the Internet of Things (IoT)) brings and creating an architecture and framework in which diversity and flexibility are core concepts.

At the end of each problem section are numbered, proposed architectural decisions for the system. These are the starting point for a discussion on how a fundamentally distributed building automation system should be designed.

Motivation

I have particular interests in the field of building automation. I am particularly interesting in how to better control heating and cooling systems by exploiting extracted knowledge about the building and the larger environment. I also have a house that is much more complex than the average one and need something that can deal with that complexity.

I am also interested in trying to build a thermostat that is capable and flexible. I want it to be able to layer building wide control and local control and I want it to have a modular approach to the sensors that it can include.

I have looked for a system that has the modularity and flexibility to allow me to focus on the areas of interest while avoiding having to develop yet another ground up system. None of those I have discovered have come close to meeting these needs. They have very specific data models, operation models and programming choices. I am looking for the opposite, something that has a great deal of choice in how any part of the system is implemented, including data representation, programming language, middleware and the like.

Problem statement

There are a series of problems that exist in the current generation of automation systems when presented with the quantity and diversity of smart devices. Many of these stem from the monolithic nature of various parts of the architecture and design of the system. These will need to be addresses in a new system

Here and Now

The sensor readings and the actions to be done by a monolithic system are done in the context of here (this machine, this code) and now. Distributed systems violate both of these fundamental assumptions. There are many places in a distributed system that can do various parts of the system's work. In fact, that is actually the goal of a system of smart components. So not only do various parts of the work happen in different places, these may also change over time. A simple example is a smart device with local control logic that is turned off when it starts communicating with a building wide control program.

What is also true is that as these systems communicate, it becomes more and more difficult to say with certainty what now is. If one looks at the trivial case of a distributed sensor, processor and control point, this quickly shows up. A sensor took a reading at some time and sent it to some processing software, which then sent an action to a control point. How much time was spent in the sending is unknown and undefined. How many messages are either dropped or delayed is simply ignored.

1) So a new system must embrace the ideas that processing can happen anywhere in the overall system, that the process of deciding where something happens is dynamic and that time must be explicitly noted rather than assumed to be now.
2) There is a messaging system that is the common integration point of all the components of the system. This messaging system must be scalable, flexible and have implementations available in a wide range of systems and languages. The messaging system should put little additional requirement on the systems, so that it is applicable to constrained capability systems as well as general purpose computers.
3) There is no center of the NDBAS system. There is no single source of all information. Everything is represented in the dynamic flow of information in the system. This allows scaling to be done over many orders of magnitude.
4) All messages should contain the relevant time for the message. Time synchronization is not part of the general system but can be added as a function in cases where the platforms do not provide a more general mechanism for maintaining synchronized time.

Modularity is often ignored

Another part of the problem for many systems is that there is not strict attention to modularity. This partly goes back to the Hard Problem of cooperating software components. It may also come from the method that these systems are started from. The typical form is to start with a small set of items, build the functionality on those pieces and then continue to expand the capabilities and scale the system.

This becomes a problem when you add different components that each think they hold a task within the system. By avoiding the Hard Problem of coordination and synchronization in the small, they make it almost impossible in the large.

If one looks at the humble mechanical wall thermostat, it turns out that it is actually doing at least functions within the small round housing. First, it is an environmental sensor, measuring the air temperature. Second, it is a control point, switching something on and off. Third, it is a control algorithm that determines when to turn on and off with hysteresis. Fourth, it is a UI that both allows someone to set the control parameter and also displays the current value for the environmental parameter. All with some mercury, glass and a couple metal coils.

It may seem pedantic to break things down so carefully, but when making things that can cooperate, this level of granularity is critical. Without the granularity of function, the goal of allowing any function to happen anywhere in the networked system.

5) The NDBAS system is built of components called agents. Agents are software components that talk to the messaging system and provide interfaces to the capabilities of the system. Agents communicate with each other with specific protocols that are written down and reviewed for interoperability.
6) The agents should be limited in function such that they can easily be moved and replaced in the normal operation of the system. There should be clear contracts as to what agents communicate about and the models of communication they provide.
7) The system must be designed to handle differing operations requests for the same function from different agents. Rather than having the control software try to coordinate, the an agent should be given the information to locally select the highest priority action and implement that. The control software must recognize that its requests may not be realized and maintain a consistent state across all controlled elements when some are not available to act.

Failure is always an option

With monolithic systems, if there was a failure, the system or large parts of it became unusable. This led to great deal of effort making the hardware and software constrained and reliable. This also made the systems expensive to build and challenging to implement. This worked well until people started wanting to do more than the system was designed to do. The ability to scale and upgrade was greatly hampered by the investment in reliability.

The Internet of Things takes a very different approach. There are many devices connected by various types of networking. This values flexibility and feature development over rock solid reliability. We now have many more devices that are each somewhat less reliable than the monolithic systems.

The trouble is that the users expect the same high level of reliability from the system. Nobody wants to be in a building when the heat isn't working or the lights won't turn on and off. We have to take a different approach to the system design to meet these divergent requirements.

The system and all its components must embrace the possibility of failure and design for partial operational states. It is not enough to detect any problems; the system must continue to provide whatever functions it can within the system design. This may look like it only happens in extraordinary situations, but it is a common occurrence. Any time a new device is installed or a dynamic system change occurs, there are times when parts of the system may appear to be unavailable. So the same thing that addresses failures also addresses the bootstrap problem.

We will look at the smart version of the humble thermostat in this light. If the thermostat been installed so that the local control point manages something that effects the environment of the thermostat, then the thermostat should be able to run in a degraded mode with minimal local control when the rest of the control network is unavailable.

8) All agents must be able to deal with unavailability of other parts of the system. The agent should be designed to provide the most functionality possible given the current system state it knows about. This is similar to the situation with priority and preemption.
9) Agents must invalidate any stored data about another agent when that agent becomes unavailable. When availability returns, the data can either be reacquired or rebuilt during normal operations.
10) Agents should continuously attempt to reestablish full functionality by watching for messages from other agents that were missing.

Configuration is too difficult

Whether the system uses free software or the most expensive commercial platform, the difficulty of configuring the system is always a big part of a user's problem. One of the main selling points of the Nest thermostat is that you just wire up the contacts, give it power and it works.

One major problem is when the language that the users choose to describe something is different from what the system designers and developers would use. This is sometimes a question of focus and sometimes an issue of precision. User terms are often far less precise, which can lead to conflicting information.

There is a major challenge in providing a system that is both flexible in terms of capability and easy for a user to get running is a great challenge. It is also one that does not have a single solution that will fit all cases. Rather than giving up, we should embrace this the same way that mobility, diversity and reliability problems have been embraced.

The proposed solution to this is to have a system where the users work with a meta-model of the building, and that is then translated into the actual configuration to be given to agents. There is no one meta-model that the system uses, these can be changed to fit the class of user involved. So a simple house can have a completely different meta-model to fill in that and moderate size commercial building. These meta-models are hierarchical, to avoid making the user repeat common information for a group of elements.

As the information for the meta-model is filled in, these become attributes that pass down the hierarchy to the lower levels. A node in the hierarchy can have more than one parent. Then there are agent generators who walk the meta-model and create the agent specific configuration information. The agent generators also look for missing information and conflicts, working with the user to resolve them in the meta-model.

11) The configuration system is based on the concept of meta-models and agent generators. The meta-model is what the user interacts with to define the system. The agent generators examine the meta-model and extract the information needed and create the configuration commands to set up the agent instances.
12) The choice of meta-model is up to the user and installer. The agent generators are based on the agents that are to be deployed throughout the system.
13) On startup, an agent instance contacts a configuration server for its configuration, then watches for configuration change messages.
14) Agents may operate locally, but cannot be active on the network in terms of sending or receiving messages until a current configuration is received.

Supporting and encouraging openness

One thing about almost all building control systems is that they assume they are the only control system active. With an IoT environment, this is no longer an appropriate view. Many devices will have custom configuration and control functions that are either not available or appropriate to include in the larger building automation system. For most systems, this requires that the systems are not integrated in or they are integrated in with a very low level of capability.

This is where the real power of the agent model comes into play. The agents can provide as much or as little of the actual capability as desired. Other agents can either use or ignore the element capabilities, as they are wish. There can be agents that bridge entire management systems together.

One of the express goals of this system is to encourage open use of the software and system architecture. One part of this includes using widely available protocols and definitions that are

appropriate to this work. One example of this is the use of ZeroMQ as the messaging base of the system. Another is the use of Project Haystack for the initial meta-model design and terminology.

The other part of this is that there is no one set of agreements for the communication between agents. We strongly encourage generality and reusability in these, but there is no cabal that says yes or no to whether this framing of those messages are allowed. It is expected that these communications use as much of the common communications system as possible. This is important for the growth of the platform, as it allows experimentation and discussion that keeps the system vital and growing.

15) There is no limiting list of permitted messages and framing to be used between agents.
16) Agent developers are expected to use the existing defined messages and framing unless it is not possible for the purpose of the agent.
17) All new agent messages are to be documented and available to all. This allows for the exchange of ideas that makes the platform stronger. This also encourages consistency in the messaging decisions, making things easier for all developers. It also enables message capture systems to be able to understand the messages that are on the network.

A reference system

For many building automation systems, the code is the reference model. In other situations, people define complex protocols without a reference implementation to compare against.

The goal of the NDBAS is a widely deployed system into which many things can be incorporated. To facilitate this, there will always be a software reference system available to developers to build from and to test against. In some cases, this will be a set of virtual machines that can simulate the components and configuration of a given building model. In other cases, this can also include specific hardware that can be included into a building model.

To this end, and to demonstrate the functioning of real devices in a NDBAS environment, we will design and publish a working thermostat that can work as a test case and as a hardware platform to explore integrating new hardware into NDBAS. We will encourage others to publish reference hardware and agents to be used in simulation and interoperability testing.