

Digital Convergence and Building Automation Systems

Marek Podgorny¹, Luke Beca¹, Suresh Santanam², Gregg Lewandowski¹, Roman Markowski¹, Greg Michalak³, Paul Roman³, Paul Gelling⁴, Edward Lipson⁵, and Edward Bogucz²

¹Electrical Engineering and Computer Science, Syracuse University, and CollabWorx, Inc

²Syracuse Center of Excellence in Environmental and Energy Systems, and Syracuse University

³CollabWorx, Inc; ⁴SenSyr, LLC; ⁵Dept. of Physics, Syracuse University and SenSyr, LLC

Corresponding e-mail: marek.podgorny@collabworx.com

SUMMARY

In this paper we present our (possibly controversial) view, as Internet devotees, on the state of Building Automation Systems (BAS's). With one exception, the authors were not BAS practitioners until very recently, when we became engaged in an experiment of building an Internet Protocol (IP) based and open-source based BAS. This work is described in our other paper in these proceedings [1]. Being new to the field we started by studying the BAS state-of-the-art and by familiarizing ourselves with the technological, legal (mostly in the context of intellectual property protection), socio-technical, and business aspects of the BAS industry [2-7].

This paper summarizes our "lessons learned" in two parts. First, we discuss our view of the BAS industry from the point of view of outsiders who are expert in broad aspects of the Internet: IP networks and protocols, Internet-application implementation frameworks, and Internet-related business models, including open source. Second, we take a more critical view in our own work of implementing a BAS based entirely on an Internet technology stack. As a whole, our two papers lie within the genre of systems research.

INTRODUCTION

Digital convergence of communication networks has advanced very far in recent years. Voice and video communications are now well established on IP networks [8], and are quickly becoming as ubiquitous as basic information retrieval and e-commerce services. The global information infrastructure is essentially Internet Protocol based, with all higher layers of network stack, diverse as they are, utilizing the same IP transport layer [8,9].

In contrast, convergence in the field of building automation and control is much slower, to a point that BAS's seem not to be participating in the communication and media exchange convergence. The world of building and process automation is dominated by BACnet/LONTalk [2,3] networks, which, although packet-based, implement a design philosophy alien to IP, with application logic supported by low-level network layers. Web-based interfaces to some of the existing systems and BACnet migration to Ethernet transport are superficial at best. The industrial application development frameworks (ADF's) are mostly proprietary, and carry substantial royalties or even patent protection. We believe that this situation stifles innovation and makes true digital convergence of BAS's an arduous process, with a significant negative economical impact.

In this paper we present our observations of differences between the mainstream Internet technology stack and the approaches adopted by the BAS industry, and we point to weaknesses of the latter as we perceive them. Then, we discuss the architecture of a fully converged BAS system based entirely on open software solutions, with functionality matching typical current BAS ADF's. We believe that the system has the potential to liberate the industry from proprietary solutions and bring BAS's into the mainstream of the digital convergence process.

NOT ALL PACKET NETWORKS WERE MADE EQUAL

Pre-1995, BAS's were closed, isolated, inaccessible, and inward-looking. Faced with the Internet phenomenon, the BAS industry started supporting an interoperability process. BACnet¹ [2] and LONTalk/LONMark [3] were created. Both standards are packet networks. How similar are they to Internet Protocol?

We will focus here on the BACnet standard [4]. Intellectually, we consider BACnet an admirable piece of engineering work and we recognize its superiority over proprietary solutions. However, from the standpoint of the Internet community, the BACnet design is a never-ending source of surprises and conceptual difficulties. One of the differences is the approach to protocol layering: BAS networks define all seven layers of the OSI Reference Model. In the IP world, development moved forward in small, incremental steps. Each protocol was kept simple and designed to do one thing well. Each protocol could have been tested separately, and in fact has been tested until perfection is achieved. With a lower level protocol in place, the Internet Engineering Task Force (IETF) community [8] moved to a higher level, while low-level network engineers were designing carrier networks that could carry IP. This process has been repeated hundreds of times, leading to the current net of compatible, seamlessly interoperable protocols by which anything can be connected to anything, either directly or via a suitable adapter. The history of Internet protocols and of the IETF process is a history of stunning technological success.

BAS networks represent an orthogonal approach. BACnet specification [2,4] is a large library addressing every aspects of the networking system that the working group engineers and their corporate oversight could think of. The complexity of the design seems to reflect a trend to think about interoperability at every single protocol layer, as well as a desire to ensure that the definition will directly support existing products in all their proprietary diversity. The engineers did admirable work by unifying these requirements into a coherent design, but nevertheless the BACnet is an extremely complex design addressing issues that were best left outside of the specification. This, unfortunately, has grave economic consequences. Most devices connected via BACnet are relatively simple, sometimes as trivial as temperature or pressure sensors. The requirements for a BACnet device resulting from the complexity of the network are often far more complex than the sensor itself. We see this as a problem in the world in which wireless sensors can be made as small and ubiquitous as sand particles.

In [1], we describe a moderately complex Personal Environment Module with several sensors and actuators. This module can be purchased with an optional network interface. The cost of the network interface is approximately seven times that of the module itself and rivals the price of a small automobile. The interface designed and built by us following general methodologies perfected by the Internet technology stack can be manufactured for under US\$30. This is, of course, anecdotal evidence. However, it is rather obvious that a) manufacturing

¹ Work on BACnet started in 1987. In 1995 BACnet standardization process was taken over by ASHRAE.

BACnet compliant devices is expensive due to the complex network interface, and b) writing fully compliant device drivers is difficult and hence costly.

Other concepts of BACnet that are surprising for Internet community are: a) a fuzzy notion of device address or identifier, b) amount of the state that BACnet devices are required to handle, c) low-level support for sensor and service discovery, and d) a large but closed list of services that the devices are supposed to render on behalf of other network entities.

The first concept hatched and solved by the IP community was device addressing, defined in absolutely unambiguous terms. This was separated from hardware, by simply allowing for two lower layers — data and physical links — with the issue of identifying devices in these layers left outside of the IP concept. This allowed implementation of efficient routing protocols working uniformly across the entire domain, from LANs to the global Internet. Indigenous BACnet routing exists, but it now seems clear that BACnet domains are best connected using IP [7].

The amount of state that network nodes must carry is of paramount importance. It is well known that Internet devices and applications limit it to an absolute minimum. This concept permeates IP networks at all levels, from routers to HTTP-protocol web servers. BACnet devices carry a lot of state, and this *per se* suggests that large BACnet networks cannot ever reach IP robustness and stability. Why would an industry supporting enormous real-estate investments settle for anything but the best?

At the heart of the BACnet specification, there are discovery services, such as “Who-Is” and “Who Has” [10]. The presence of these services forces implementation of a bi-directional client-service model for network nodes (see our comment above regarding amount of state). While various data-link layers may implement discovery services, above the IP layer there are separate, high-level directory services such as Active Directory/LDAP [11,12], or UDDI [13,14] for web services, or NIS [15], to name a few. These services are completely separate from low-level network services and enjoy the reliability of the underlying transport protocols. They also typically use client-server rather than peer-to-peer (P2P) architecture. One could argue that the location of the service in the protocol stack does not matter. However, we have had negative experience with networks that rely on self-discovery, with Sytek/Microsoft NetBEUI/NetBIOS [16] being the most notorious example. NetBIOS is finally being phased out in Microsoft products such as Windows Server 2003, after having caused users grief for over a decade. This aside, we believe that self-discovery of devices is a useful, but not critical feature. Many simple applications do just fine with manual configuration. We are inclined therefore to see BACnet reliance on discovery services as a liability rather than strength.

Discovery services are just few (albeit important) examples of services defined by the BACnet standard. Overall, there are at least 32 services in five categories. In addition, the standard defines multiple types of objects that devices have to support. The overall impression for an outside reader is that BACnet designers were focused on a network that would easily support all possible existing sensors, devices, and applications in the protocol itself, instead of providing the basics and leaving the applications to developers. Again, one can argue that the approach taken is beneficial for interoperability. We remain skeptical. Interoperability is far harder to achieve for complex protocols than for simple ones. Further, any attempt to encapsulate functionality of applications available at a given time is futile, as it invites obsolescence once the standard is defined. BACnet proponents would probably respond that the protocol is extensible, but this is a self-defeating measure: the very mechanism of extensibility and the

possible extensions themselves increase the already significant complexity, and the extensions do not guarantee interoperability.

BAS APPLICATION DEVELOPMENT FRAMEWORKS

Let us consider now to the next layer of the technology stack: the application-development frameworks. This is a large and complex field, discussion of which exceeds the space available in this paper by orders of magnitude. The general trend is a division of the market space between a few major players. In our opinion, several vendors were able to attract a very large following for their products, precisely because interfacing to complex BAS networks is difficult. Two frameworks, supported by respective organizations, seem to dominate: a) Niagara Framework [17] and the industrial consortium behind it [18] and b) the OPC (OLE for Process Control) Foundation [19-21] with over 300 members supported single-handedly by Microsoft, as the technology is tied to the Microsoft DCOM. The Niagara Framework is used by many of the largest manufacturers of HVAC equipment. It is a complete software framework and development environment. Recent software releases clearly gravitate towards better integration with the Internet [22], but this is done by layering web services functionality on a rather thick stack of components already dealing with complex BAS networks. Interestingly, the framework description cites “device data unification” as one of core benefits. In other words, the complex networks are now being simplified to a level that applications really need. Of course, there is a price for any of the elements implementing this “standardize-complicate-simplify” seesaw cycle, and the end users eventually wind up paying the bill.

INTELLECTUAL PROPERTY PROTECTION

An additional interesting facet in this thread is that the Niagara Framework is patented in the United States (US Patent # 6,832,120 [23]). This patent is one of a multitude of US Patent and Trademark Office (USPTO) patents for software systems. In the simplest terms, the patent describes a methodology for integration of elements provided either by other vendors or by public standards using a pervasive, well known, and generic software infrastructure. The patent is an example of the recent trend followed by the USPTO for software patents: it is possible to obtain protection for arbitrarily complex systems built from a mixture of generic and widely used modules that do not need to contain any intellectual property of the patent author. US Patent #6,832,120 essentially claims that any software system used for building control that uses the notion of software objects to represent devices infringes on the intellectual property rights of the patent owners. This is a very broad claim that certainly discourages implementation of competitive frameworks.

OPEN SOURCE BASED BAS

The preceding sections should have made it clear that we are not particularly fond of the current state-of-the-art BAS technology. To be constructive, rather than merely critical, we have decided to implement a functional BAS completely from scratch using only pervasive Internet technologies and methodologies. Some aspects of this work have been described in our other paper in these proceedings [1]. Here we would like first to discuss certain recent developments that, in our opinion, justify such work. Next, we will discuss problems and difficulties we have encountered in the design and implementation process.

First, we are strong believers in the open-source phenomenon [24-33]. Before embarking on the implementation, we have searched for an “open” BAS system, or elements thereof [34]. Not only have we failed to find any significant work, but also we have also found it unusually

difficult to obtain information about commercial frameworks. In several cases, we were told that access to information is contingent upon a purchase of an expensive membership in a consortium and/or disclosing project goals and details. A study of the “open” projects usually revealed a lack of substance behind publicly stated claims. For a research group immersed in the Internet culture, this has been an unexpected and sobering experience, but also a stimulating challenge.

Our decision to implement a BAS the “Internet way” was reinforced by several technological developments:

Sensors are getting smaller: the “intelligent dust” is no longer a domain of science fiction. BACnet design philosophy does not appear suitable to handle this situation.

There has been profuse development of diverse types of wireless networks: WiFi promises blanket coverage in metropolitan areas [35]; satellite connectivity providing global reach is becoming both affordable and technically straightforward due to availability of Ka band [16] and low-orbit constellations, such as GlobalStar [36]; and mesh networks such as ZigBee [38] provide support for mini-sensors. All these technologies support IP by definition, and can serve as access media for BAS’s.

Construction of pure IP controllers for sensor and actuator devices has become very easy thanks to the availability of miniature, dense, highly reliable components. An HTTP server with processing power sufficient to implement web services is available in enclosures that include an RJ-45 Ethernet jack [39]. This makes the entire TCP/IP stack and a sophisticated application layer available off-the-shelf for any type of sensor assembly

There has been very substantial increase in processing power for embedded applications. Note that the entire concept of LONTalk was built around a Neuron processor [40-42]. Last year, development of ultra low-voltage CPUs such as Intel Celeron M enabled, for the first time, processing power comparable to that of a desktop PC in mobile and embedded platforms. This allows implementation of complex sensing devices. An example of such a complex sensor could be a high-resolution still-image camera capable of real-time image processing, change detection, and object/face recognition, or a video camera/GPS/IMU assembly (where IMU is short for Inertial Measurement Unit) supporting real-time georectification, object tracking, image stabilization, and feature extraction. Such complex sensor assemblies appear to be outside of what has been envisioned for BACnet domain. For our pure IP-based BAS handling of such devices is as natural as the handling of a temperature sensor.

There has been explosive growth of web services in form of Service-Oriented Architecture (SOA) [43], which at this point ensures that this particular technology will form the foundation of Internet application architecture for many years into the future.

SMART BUILDING SYSTEM

In this section we will briefly look at the way we approached design and implementation of our BAS, based on Internet Protocol (IP) and open-source technologies (for more details see [1]).

System Goals

While researching the feasibility of creating a BAS with IP-based, open-source technologies we focused on building a prototype of the system for Smart Buildings (hence its name: Smart Building System). The system should provide support for the following features:

Heterogeneous sensors and devices: It should be possible to connect arbitrary IP-based sensors and devices to the system with minimal implementation, configuration, deployment, and maintenance effort.

Personalized and dynamic environment: The system should be able to customize environmental properties of the building spaces to align them with the preferences declared by the individuals. Environmental preferences should form an environmental profile of an individual that would ‘travel’ with the individual across the building space.

Customizable goal-oriented control: The system should be able to apply ‘reasoning’ to maintain the environmental parameters (temperature, humidity, etc.) of the building within specified parameters. The reasoning logic should be customizable, so that various strategies can be implemented, analyzed, and compared.

Manual administrative control: The system should provide means for the authorized personnel to monitor the status of the system and set the global, overriding goals for the system., means for direct control of specific devices should also be provided.

Potential of integration with IT infrastructure: The system should provide interfaces for embedding it in the larger enterprise IT infrastructure.

Extensibility: It should be possible to add new or replace existing system components using common software engineering skills.

Technology

We decided to base our implementation on the following technologies:

Java 2 Enterprise Edition (J2EE [26]) is used as a general programming platform for most server-side components of the Smart Building System. This choice was not related to Java’s being an object-oriented language, but rather due to the following features and properties of J2EE:

a) support for transactional access to the relational database management systems (DBMS) [31]; b) well-defined communication mechanisms such as RMI, HTTP, Web-Services, and JMS [32]; c) mechanisms for accessing directory services such as LDAP [12]; d) J2EE is a set of specifications instead of products; the code is therefore easy to port to various implementations of J2EE; e) Inclusion or integration with technologies that provide scalability, security, and fault tolerance; f) J2EE is widely used as a platform for enterprise software systems; g) a selection of mature open-source implementations of J2EE platform is available.

Architecture

We adopted a modular approach, where the system components can reside on separate IP network nodes. The Smart Building Framework is at the core of the system. It constructs and maintains an up-to-date model, which represents the controlled environment. It allows components of the system to accurately interpret the data gathered from the devices and influence the environment as intended. The Smart Building Framework, which exposes its services

using web services, is implemented using the J2EE platform. It is deployed on a J2EE application server (JBoss [27]) and communicates with a DBMS (Hypersonic [31]). Third-party modules can connect to the Smart Building Framework over the network and access its services via a Web Services [24] application-programming interface (API).

All other system components interact with the Smart Building Framework (including devices) through its set of web services. We have implemented the following essential components:

The *Inference Engine* [25] implements the control logic of the Smart Building System. Refer to [1] for detailed description.

The *Smart Device* is a software wrapper for any real or virtual device that needs to be connected to the Smart Building System. It can be either deployed as a part of the real device software or it can run on a separate network node the real device connects to. Refer to [1] for more detailed description.

The *Administration Interface* provides a graphical user interface (GUI) for monitoring and controlling the Smart Building System. It has tools for defining target environmental parameters for the system (goal-oriented approach), as well as specific device settings (direct device control overriding settings calculated and enforced by the Inference Engine). It is implemented as a Web-based application (Java Servlet, HTML, and JavaScript) that interacts with the Smart Building Framework to access the information about the system state. It is deployed on a J2EE application server (Jboss [27]).

The *Personalization Interface* provides a GUI for regular users of the system. It allows users to specify their environmental preferences that the system tries to enforce. It is deployed on the Jboss [27] J2EE [26] application server.

DISCUSSION

During design and development of Smart Building System we identified the following benefits and drawbacks of our approach.

Benefits

Re-use of existing IT infrastructure: Most of the server-side components of the Smart Building System run on a standard J2EE-compliant application server commonly used in the enterprise. Any relational DBMS can be used. As a result, the server-side components can be deployed and integrated with an existing IT infrastructure. Standard TCP/IP-based networks can be used to connect devices.

Extensibility: Since all components of the system communicate through well-defined interfaces, they can be easily replaced or updated. New components can be created or existing ones modified using well know programming languages (Java) and communication technologies (web-services). A Java API is provided for creating new Smart Devices to simplify development even further. API's in other languages can be easily created.

Integration with enterprise systems: The systems based on the Smart Building platform can be controlled through a web-services interface. As a result, they become one more element in the SOA [43] palette in the organization.

Support for legacy devices: Adding legacy devices (BACNet, LONTalk) is possible through adapters. Representation of devices in the platform as well as the communication protocol between Smart Device and Smart Building platform are based on BACNet concepts; therefore effort required to develop of an adapter should be reduced.

Blending of real and virtual worlds: Devices connected to the Smart Building Platform can be both real and virtual. This creates the possibility of making other components of the enterprise software infrastructure behave as virtual devices. For example, a corporate calendar can behave as a device that prepares conference room for a meeting before the meeting participants arrive.

Scalability and reliability: Making Web-based and J2EE based applications scalable and fault tolerant is a relatively well-understood field.

Drawbacks

No support for real-time/priority processing: Real-time/priority requirements are not well supported by existing Web-based technologies.

Latency: The Smart Building Framework introduces an intermediate node in the communication between control logic and target devices. Careful capacity planning is required for mission-critical systems

Higher bandwidth/CPU requirement: Protocol overhead for simple communication is significant. HTTP and SOAP add a significant amount of data (impact on bandwidth and processing power). This is mitigated by increasing CPU power.

Security-related effort increased: We do not consider technologies used to build the Smart Building System inherently less secure than the ones used to build existing BAS deployments, but we believe that more tools for exploiting TCP/IP networks and HTTP-based services are available. Therefore security must be a top priority during deployment and maintenance.

Not feasible for small deployments: Smart Building System requires setup of software as well as hardware components. IT skills are required for setup and maintenance.

Complex handling of system state change notifications: This is more of an architectural issue that increases complexity of the code and bandwidth/CPU requirements. Handling notifications in HTTP-based systems is nontrivial. Several approaches are possible to provide this capability:

- *Polling:* The components of the system learn about system state changes by sending requests to the Smart Building Framework periodically. This approach is supported by our current system. It potentially suffers from high (albeit predictable) latency.
- *Web-services [24] (server)/JMS [32] (client):* Components of the system subscribe to the JMS service maintained by the Smart Building Framework and learn about state changes via JMS messages. This approach, which we consider optimal, has been designed, and is being integrated.
- *Web-services (server)/Web-services (client):* The components of the system expose the web service used for notifications, and register it with the Smart Building Framework. The Framework calls the notification web-service, when a specified system change occurs. This approach requires more work, and we plan to implement it in the next version of the system.

38. ZigBee mesh networks <http://www.zigbee.org>
39. XPort Embedded Ethernet Device Server <http://www.lantronix.com/device-networking/embedded-device-servers/xport.html>
40. LONTalk <http://en.wikipedia.org/wiki/LonTalk>
41. Echelon's i.LON <http://www.echelon.com/products/cis/>
42. LONWorx FAQ <http://www.echelon.com/products/lonworks/faq.htm>
43. Service Oriented Architecture http://en.wikipedia.org/wiki/Service-oriented_architecture