Open Web Services-based Indoor Climate Control System

Marek Podgorny[1], Luke Beca[1], Suresh Santanam[2], Gregg Lewandowski[1], Roman Markowski[1], Greg Michalak[3], Paul Roman[3], Paul Gelling[4], Edward Lipson[5] and Edward Bogucz[2]

[1]Electrical Engineering and Computer Science, Syracuse University, and CollabWorx, Inc
[2]Syracuse Center of Excellence in Environmental and Energy Systems, and Syracuse University
[3]CollabWorx, Inc; [4]SenSyr, LLC; [5]Dept. of Physics, Syracuse University and SenSyr, LLC

Corresponding email: marek.podgorny@collabworx.com

SUMMARY

The paper describes an open, extensible software and hardware framework supporting all aspects of control and monitoring of a Smart Building. We designed an 'intelligent', real-time control system capable of both autonomous process control and interaction with human operators. The system can easily accommodate both functional extensions and a broad array of sensors and control devices.

The system described below represents a clear bias towards pervasive, open-source, Internet and Web technologies and away from the proprietary or vertically specialized networks, protocols, and application frameworks typical for the current industrial automation systems. The basic goal of this research was to find out if and at what level of effort one can build a functional prototype of a Building Automation System (BAS) assembled exclusively from open-source elements. The answer we found is affirmative and the results of the experiment have the potential to affect the building-automation industry in a way similar to the impact of Linux in the domain of operating systems and of the Asterix server on the VoIP industry.

INTRODUCTION

When first analyzed by a technologist raised in the Internet cocoon, the current BAS's seem to have originated in a different technological civilization. The only commonality of the two technology stacks is the notion of a packet network; from this point on, there does not seem to be one concept or idea that is shared. This technological abyss only gets wider as we move up the protocol stack. This seems to translate into very different business models for software companies involved in both fields. We present a more detailed discussion of this "digital divergence" phenomenon in another paper submitted to this conference [1].

This paper focuses on a practical aspect of subverting the current technological status quo in building automation by creating a nucleus of a disruptive-technology framework, a time-honored mechanism of change and progress in the Internet-centric universe. The goal of this paper is to stimulate the change or, at least, to contribute to the metamorphosis that seems to be germinating in more than one research location. We have no doubt that the current BAS architectures will disappear within a decade, and that building control will completely converge with IP networks. The really interesting question is, "How do we get there?" To gain such an insight we have designed, implemented, integrated, and deployed a complete control system using only open-source or free standard software technologies and protocols, and off-the-shelf electronic components, assembly of which requires only basic technical

skills. We hope that the findings described here will enable faster, cheaper, and more efficient implementation of automation systems. The application we particularly cherish is to provide research facilities with a complete, open software testbed for building automation, so that the researchers can focus on solving real problems in fields such as indoor-climate control, rather than having to struggle with cobbling together instrumentation necessary to perform experiments and measurements. We also hope to contribute to the discussion related to emerging standards, security issues, and "open vs. proprietary" software and business models.

METHODS

The overall system design is Internet-centric, following the critical requirement of providing an efficient mechanism to ensure an easier and faster process of bringing building automation systems into the fold of the global digitally converged communications infrastructure. The system uses only pervasive Internet network and open-source subsystems, applications, and application-development tools. These tools rely in turn on a number of standard protocols and/or software development methodologies.

All system functions are implemented as Web Services and all real-time signaling uses pervasive messaging technologies proven effective in mainstream communication systems, such as JMS. Sensors and control devices used with the system are required to be Internet Protocol-based, but the XML-coded device data structures are based on the BACnet networking standard [2].

Although our goal was to build a system with a broad functionality, potentially covering all important aspects of building automation, we have deliberately approached the project as a focused exercise in system design for rather specific functionality. The issue we address is climate control in a space where users can express their preferences for several factors and realize them using personal environmental modules, while the system is responsible for global optimization and resolution of potential conflicts.

Further, the research reported here differs from the work of several industrial consortia on unification of BAS protocols using XML [3-5] and standardize BAS related web services. Our focus is on the system architecture. We believe that the system described in this paper could be readily adapted to provide a reference implementation of protocols being designed in the arduous industrial standard setting process practiced by the ASHRAE BACnet/XML Working Group [3], oBIX [4], and CABA [5].

RESULTS

The requirements for the system can logically be grouped into three categories:

1. *Broad system goals:* a) The system should support interaction of the building automation elements with the human occupants of the space, and b) it should have the ability to make 'smart'[1] decisions in response to ever changing conditions in the building.
2. *Specific system functionality, including*: a) Adjusting environment variables to increase occupants' productivity; b) Tracking space occupants; c) Maintaining knowledge of the physical space including information about zones; d) Monitoring environmental factors in zones; e) Allowing individuals to communicate personal environmental preferences

[1] The notion of 'smart' or 'intelligent' system should be interpreted conservatively. The vocabulary merely implies that the system integrates an inference engine.

(temperature, noise, humidity, etc); f) Providing a visual interface (similar to a weather map) to monitor the space/zones and their parameters; and g) Possessing the ability to self-adjust based on user preferences and output from the inference engine.

3. *Additional considerations*: We aimed at building a solution that is vendor-independent, and uses only open-source tools and components, unless none were available, in which case we resorted to public standards.

BASIC CONCEPTS

We have used the following basic concepts in designing the system:

- *Controlled space* – a physical space where the system controls environmental factors such as temperature and humidity.
- *User presence* – Identity of the user (or users) present in the environmental zone.
- *Environmental zone* – a unit of space controlled by the system, which assumes that all environmental factors have uniform values within the environmental zone.
- *Environmental profile* – a set of key/value pairs that describe environmental parameters of the environmental zone. The system strives to maintain environmental parameters of the zone within a given range from the values specified by the environmental profile.*Environmental preference* – a set of key/value pairs that describe environmental parameters selected by the space occupant as the most desirable. Environmental preference of the user present in the environmental zone affects the environmental profile associated with this zone.*Device* – any hardware or software entity that can generate or accept data. The system receives information about environmental parameters of the environmental zone, and influences these parameters through devices installed in the zone.

TECHNOLOGIES AND STANDARDS

The following technologies have been used to assemble and integrate the system:

- Web Services [6] – communication among system components
- BACNet [2] – base for internal device description
- CLIPS [7] – open-source inference-engine platform
- Java 2 Enterprise Edition [8] – set of APIs for building enterprise applications
- Open Source technology stack:
  - JBoss [9] – J2EE Application Server
  - Struts [10] – Web Application framework
  - Axis [11] – Web Services framework
  - Hibernate [12] – Persistence framework
  - Hypersonic [13] – Database Management System
  - JMS [14] – Java Messaging System
  - AJAX [15] – Web GUI implementation methodology

SYSTEM ARCHITECTURE AND COMPONENTS

Figure 1 below shows all major components of the system. For each component, the communication interfaces and dependencies are depicted. The description of each interface contains information about the technology used for its implementation.

The communication pattern between system modules is based entirely on the Web Services model. This applies to both "read" and "write" operations, as described below. In addition, since devices and other modules can generate events that require handling, we implement two mechanisms to propagate event information: for time-critical events, we use JMS[2] [14]; for event supporting services, such as archiving, we use a pooling mechanism under Web Services.

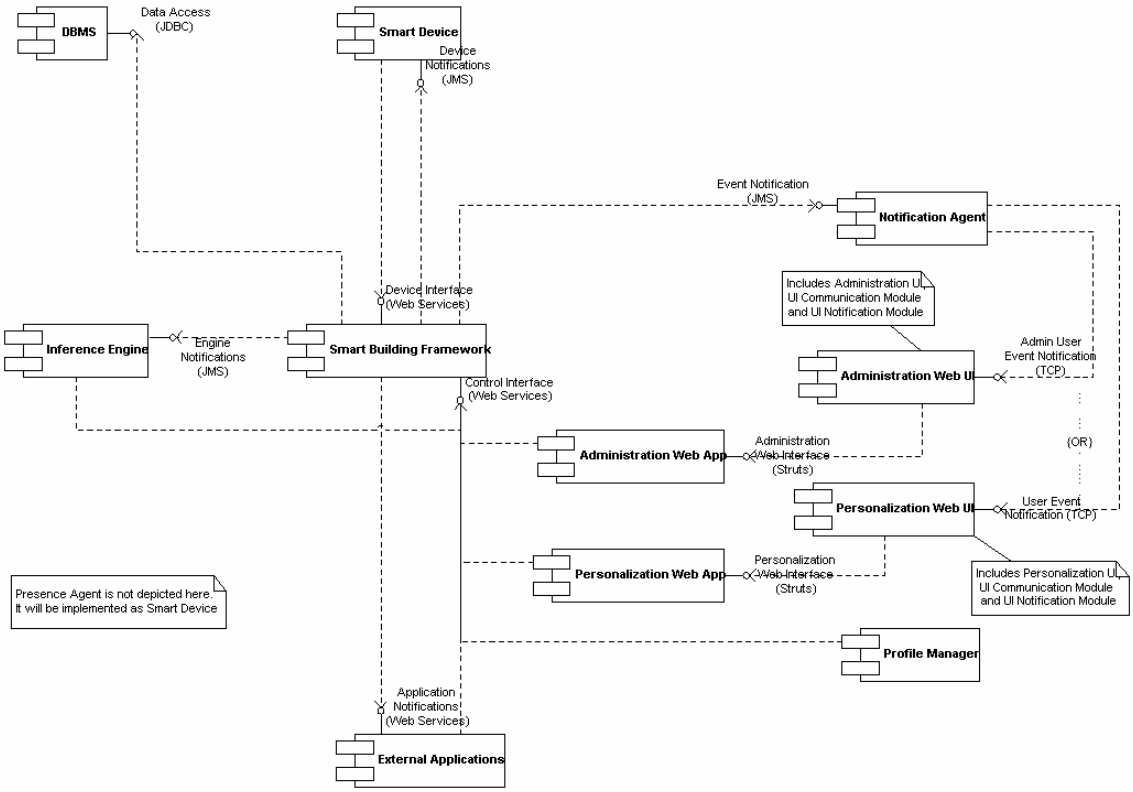

Figure 1. UML Diagram of System Components

Most of the modules on the diagram are self-explanatory. Software modules provide services roughly corresponding to the system requirements listed earlier in this section. Three modules of special interest are a) the Smart Building framework, b) the Inference Engine and its integration with the system, and c) the Smart Device. These three elements will be described below in more detail. The system is extensible, as it can accommodate External Applications of any kind. A good example might be an energy optimization engine, a computational fluid dynamics application, or a set of web services reaching out to external data sources, such as a weather forecast.

Smart Building Framework

The Smart Building Framework is the core of the Smart Building system to which all other system components connect. It implements the following services:

- *Device status monitoring and data collection*
- *Device data archiving and retrieval*
- *Access to and manipulation of an object-oriented representation of the system state*
- *Event-based communication among system entities*
- *System configuration*

[2] JMS engine is an integral part of the JBoss Application Server [9]

Web Service APIs

Most of the services described above are exposed through a set of ***web service APIs***:

1. *Device Web Service:* allows external devices to connect to the system. External devices use this service to register their presence and to report changes in their state.
2. *Object Access Web Service:* provides access to object-oriented representation of the system state maintained by the framework. The Object Access interface can be used to both read and write properties of the objects maintained by the system. Write operations on the system state may translate into commands issued to the components of the system controlled by the Smart Building Framework. The service is also used by system components that need to access the information about the system entities (for example, Inference Engine or Administration Interface).
3. *Event Access Web Service:* implements a messaging system for sending prioritized, point-to-point or system-wide alerts related to the state of the system. For example, the Inference Engine might send an alert to the Administration Interface component, if the readings from the external devices exceed a certain threshold. Both sending and retrieving operations are supported. The service is used by all system components that need to send and/or retrieve alerts.
4. *Archive Web Service:* allows starting and stopping archiving of the data submitted by the external devices connected to the system. It also allows retrieval of the archived information. The Archive Web Service is used by the system entities that rely on the archived information about devices.

The web services implemented by the Smart Building Framework can be accessed not only by the components of the Smart Building system but also by other external (remote) entities that need to access the information about the status of the building infrastructure.

Communication between modules

All components of the Smart Building system can learn about the system-state changes through polling functionality embedded in all implemented web services. This approach is simple and works well with standard web infrastructure. However, it also has significant drawbacks such as embedded latency in the system responses due to a set polling interval. To handle propagation of time-critical events we designed a server-side component implementing JMS-based notifications. Each component interested in receiving notifications from the Smart Building framework can embed JMS client functionality. This also applies to our wrapper for devices and to the Inference Engine, as well as to the Administration and Personalization Interfaces. The JMS messages may carry either actual module commands, or a request for a module to poll the message sender for information.

Smart Building Inference Engine

The inference engine provides the ‘intelligence’ driving the Smart Building Framework. While the devices connected to the framework might be autonomous enough to make their own limited decisions, there exists a range of decisions that are beyond the capabilities of any single device. The inference engine fills that gap by gathering information from the entire framework, making decisions based on the gathered data and then controlling the framework to execute the decisions.

Communication: The inference engine communicates with the rest of the Smart Building Framework via a set of web-service interfaces. These allow it to extract information necessary

for making a decision, and then to control the framework to execute a decision taken. In particular, the framework interfaces allow:

- reading device input
- controlling (writing to) devices)
- reading and modifying current environmental data
- sending system alerts

Since the communication mechanism is documented, it is possible to replace the engine with another agent that would drive the framework.

Engine: The inference engine is implemented as a stand-alone Java application. Its main functionality, the decision making, is split into *decision beans*. The application schedules a bean to execute at a specified time after the application start. From then on, a bean is in charge of its own scheduling. In addition to making decisions, the beans can also be used for simpler tasks, for example to translate the information retrieved from the devices into current condition data.

It remains unspecified whether the entire decision logic should be implemented as one complex bean, or as a several (possibly many) simpler beans. The inference engine supports any approach. Typically, when a bean is executed, it will gather required data, analyze them and send the results (a decision) back to the framework. The bean can then ask the inference engine to be scheduled again at some later time.

Simple decision beans can be implemented as Java objects. In the case of large systems and complex decision beans, the capabilities of the Java language might prove to be insufficient. For these scenarios, we have integrated a rule engine to allow us to use a more sophisticated decision making mechanism. The rule engine we used, CLIPS [7], can improve over Java in the areas of a) processing more advanced condition statements, b) resolving the conflicts among multiple conditions and c) general performance of complex-condition processing. Both Java and CLIPS decision beans can be used simultaneously.

Example: Below we provide a very simple example of a CLIPS script. The reader should not infer that this nearly trivial example represents the norm. There is no limit to the complexity of the rules that can be incorporated into the entire framework, and some of the rules we have implemented are rather complex.

The data retrieved from the framework are supplied to the script as CLIPS facts. The output of the script is also facts that are asserted or retracted based on the input.

```
;;;======================================================================================
;;; This script establishes user presence in a zone based on a reading from a presence sensor
;;;  Input facts - presence-sensor; Output facts - zone-user
;;;======================================================================================
;;;  DEFTEMPLATES
(deftemplate presence-sensor
  (slot status)
  (slot user-id)
  (slot activity))
(deftemplate zone-user
  (slot user-id))
;;;  RULES
(defrule current-user ""
  (presence-sensor (status active) (user-id ?u) (activity ?a))
  =>
  (assert (zone-user (user-id ?u)))
  (printout t "user present is: " ?u crlf))
```

Smart Device

The Smart Device component of the system is a Java wrapper implementing a generalized API used to communicate with real devices. This module provides a two-way translation between device proprietary protocols (if any) and Web Services used by the Smart Building Framework and it may implement a JMS client (see *Communication between Modules* section above). A driver for each real device in the system is a specialization of the Smart Device. As mentioned in Methods, we decided to use an XML-coded BACnet [2,3] data model to describe device data structures. This decision is a compromise we accepted for several reasons, some of them discussed in [1]. We recognized the facts that a) BACnet is a public standard under a committee control, b) its device data structures represent decades of domain-specific engineering experience and, as such, it is unlikely we will be able to come up with a better data model, and c) the particular choice of the device data model is not really relevant to the goals of this research[3].

We have implemented several devices to interact with the system, including a Universal Device (as a template for future development of device drivers), a Presence Agent identifying personnel entering a controlled space, and a simulated A/C heater-thermometer device simulating temperature changes according to a simple physical model driven by input from the Inference Engine. The most interesting device though is the so-called PEM controller. PEM (Personal Environment System) is a commercial product designed to provide an individually controlled work environment. The module, installed on the user's desk (Fig. 2) regulates air temperature and air supply to the desktop; controls the radiant heat panel output; and provides background noise masking and task lighting level. In its basic configuration, the module is locally controlled and not networked. It can be connected to a data bus, but the cost of the controller and proprietary software is substantial. For the purpose of the project we implemented an IP-based system controller for the PEM device. The controller is inserted between the manual slider board and the PEM proper. When active, the IP controller overrides manual user preferences, and instead adjusts the environmental parameters to the user preferences stored by the system, subject to optimizations necessary to take into account preferences of users in other cubicles and energy usage constraints in the controlled space. The system's Inference Engine implements the optimization procedures. Manual controls are replaced by a small application running on the user's workstation, which however can be deactivated by a system operator.

Figure 2: System controlled Personal Environment Module

The PEM controller is activated by user login to the workstation, although it can also be started by a signal when the building security system reports that the user is entering the building security gate. This *per se* is not particularly innovative, but the system offers certain niceties, such as biometrics identifying the person and transferring its environmental

[3] Use of such data structures does not imply that the system is compatible with BACnet network protocol layers, although, since the device data structures are rigorously derived from the BACnet paradigm [2], the system could be relatively easily adapted to access and control most of the existing BACnet BAS infrastructures.

preferences to the proper cubicle, together with the person's VoIP phone number. Further, the global optimization functionality mentioned provides a relatively new and interesting element. The PEM controller implements a complete HTTP server on top of the IP stack, and is simply connected to the office LAN. The application running on the server communicates with controller's A/D and D/A converters, providing full control of PEM control elements, as well as providing additional temperature and humidity sensors plus several auxiliary industrial standard 0–10V inputs and outputs.

DISCUSSION

While the existing implementation and deployment focus on environmental controls and human efficiency issues, we believe that the system described here offers a nontrivial technological advantage over current proprietary industrial solutions, and provides a sustainable foundation for future collaborative development of Smart Building software by academic and industrial consortia and alliances. To facilitate such a process, the system has also been carefully designed to avoid intellectual property infringement of existing patented or otherwise protected frameworks while replacing many aspects of their functionality [1].

The modular design of the system permits replacement of any of is components by a proprietary or simply different implementation. Our focus on open-source software provides an affordable starting configuration. Device drivers, which are a very significant obstacle to interoperability, can be implemented in a matter of days or even hours by extending or replacing software modules provided in templates. The selection of Web Services implemented in the framework covers all typical uses we have identified in BAS systems, and implementation of an operational system of medium complexity provides a strong proof of the concept.

The research presented here is complementary to the work of ASHRAE and OASIS/oBIX; those groups focus on protocols and we have focused on systems research. We believe that our system can provide an almost instant reference implementation of the results of the work of these industrial consortia. This represents an important step in validation of ideas leading to future Building Automation Systems.

ACKNOWLEDGEMENT

This project was supported by funding from Empire State Development Corporation (ESDC) of New York State under Award No: ESDC- R 580, granted to Syracuse University and Syracuse Center of Excellence in Environmental and Energy Systems (SCoEEES). CollabWorx acknowledges financial help and intellectual leadership of the SCoEEES, without which this study would not have been possible.

REFERENCES

1. Podgorny, M, Markowski, R, Santanam, S, *et.al.*, Digital Convergence and Building Automation Systems, this proceedings, p. TBD
2. BACNet http://www.bacnet.org/
3. ASHRAE BACnet/XML working group http://groups.yahoo.com/group/BACnet-XML-WG/files/
4 OASIS/oBIX http://www.obix.org/
5 Continental Automated Buildings Association (CABA) http://www.caba.org/index.html
6. Web Services http://www.webservices.org/
7. CLIPS: A Tool for Building Expert Systems http://www.ghg.net/clips/CLIPS.html

8. J2EE: Java 2 Enterprise Edition http://java.sun.com/javaee/
9. JBoss Application Server http://www.jboss.org/products/jbossas
10. Struts: Apache Web Application framework for Java http://struts.apache.org/2.x/
11. AXIS: Apache Web Services SOAP framework http://ws.apache.org/axis/
12. Hibernate: Persistence framework for Java http://www.hibernate.org/
13. Hypersonic: Open source Java RDBMS http://hsqldb.org/
14. JMS: Java Messaging System http://java.sun.com/products/jms
15. AJAX: Asynchronous JavaScript and XML, a framework for building Web GUIs https://bpcatalog.dev.java.net/nonav/ajax/index.html