# Found Objects

## A View from a Room

This column is available online.

Bits Versus Electrons

Found Objects:
A View From a Room

By Bob Frankston

I was comfortable in my nice hospital room after elective surgery. Then I realized that I could turn the lights in my house on and off, but I couldn't control the window shade across the room from me. It made me think about the Internet, and I realized it wasn't simply that I was able to connect to the device in my house. I had cobbled together a path from my device, a smartphone, to the lights and other devices in my house.

Before I go further, I need to set some expectations. This isn't merely the story of how I can remotely turn my lights on or off. It's about the future of consumer electronics (CE) because software allows users to take advantage of today's complex technologies without having to wait for a manufacturer to create just the right product. The key is having an architecture that allows the developer or the *prosumer* (i.e., advanced user) to focus on the problem at hand and think only about the endpoints of each interaction without getting lost in the "between." That very idea has made the Internet what it is.

This is not (just) a story about home automation. It is almost the opposite, in the sense that it provides an understanding of how we can mix and match elements rather than having to carefully engineer systems as a single whole.

Digital Object Identifier 10.1109/MCE.2019.2892248
Date of publication: 8 April 2019

FIGURE 1. Advancements in technology led to the extinction of the manual window crank handle.

By limiting the dependencies between elements, we can improve agility and the ability to mix and match.

Finally, I am not making a sharp distinction between user and developer. Very few of the users will be developers, but the few who are can share what they create and build on the work of others as long as the architectural principles are respected. By limiting the dependencies between elements, we can improve agility and the ability to mix and match. Even something as simple as saying "that" switch turns on "that" light builds on these principles, which are increasingly important with the growing number of smart devices that need to interoperate.

IN CONTROL

To control the devices in my house, I had to get all the elements to line up—the communications paths and linkages, the software in each device, and the applications on each device and in my house. Not only that, but I've been evolving the software for the past 25 years. It's like building a house while living in it.

You might ask why I go through that trouble. Today, we open and close the windows in cars using a button that controls a motor—a feature that was once available only in luxury cars. The rest of us had to crank the windows open (Figure 1). It would've been hard to justify the cost of such a feature, but technology advanced, and it became simpler to install a motor in the door than to use and maintain mechanical linkages.

Once we have the electric motor, we can think about writing software that



I was comfortable in my nice hospital room after elective surgery and realized that I could turn the lights in my house on and off, but I couldn't control the window shade across the room from me. It made me think about the Internet and realized it wasn't simply that I was able to connect to the device in my house. I had cobbled together a path from my device, a smartphone, to the lights and other devices in my house.

Before I go further, I need to set the reader's expectations.

This isn't just the story of how I can remotely turn my lights on or off. It is the story of the future of consumer electronics because software allows users to take advantage of today's complex technologies without having to wait for a manufacturer to create just the right product. The key is to have an architecture that allows the developer or the prosumer (advanced user) to focus on the problem at hand and only think about the endpoints of each interaction without getting lost in the "between". That very idea that has made the Internet what it is.

This is not (just) a story of home automation. Almost the opposite in the sense of providing an understanding how we can mix and match elements rather than having to carefully engineer systems as a single whole.

Finally, I am not making a sharp distinction between user and developer. Very few of the users will be developers, but the few who are can share what they create and build on the work of others as long as the architectural principles are respected. By limiting the dependencies between elements, we can improve agility and the ability to mix and match. Even something as simple as saying "that" switch turns on "that" light builds on these principles. These principles are increasingly important with the growing number of "smart" devices that need to interoperate.

## In Control

To control the devices in my house I had to get all the elements to line up – the communications path and linkages, the software in each device as well as the applications on the device and in my house. Not only that, but I've been evolving the software over the past 25 years. It's like building a house while living in it.

You might ask why go through that trouble. Today we open and close the windows in cars using a button that controls a motor. That was a feature that was once available only in luxury cars. The rest of us had to crank the window open. It would've been hard to justify the cost of such a feature but technology advanced, and it becomes simpler to install a motor in the door than to use and maintain mechanical linkages.

*Figure 1Class car window crank*

Once we have the electric motor, we can think about writing software to allow the car to automatically close the windows when it is parked, and it detects rain. That is a benefit of, but not the primary reason for, the use of such motors.

In the same way, the Internet didn't require building a new infrastructure but instead used software to repurpose the existing telecommunications infrastructure. The software architecture separated the details of the transport (the "between") from what we did with this. Thus, the web could be implemented by one person since he need only do the software at the endpoints. The simple protocols then allowed others to contribute without waiting for new features in the network.

I took this to heart as I evolved the implementation of my home control capability. The biggest challenge has been taming the complexity and maintaining the simplicity of the end-to-end architecture. Each system element can focus on the task at hand. When I press "Turn on Office Light" I didn't have to be aware of the details of every step of the path. I purposely used the term "capability" rather than system because there isn't a single system. There isn't a central controller or hierarchy into which every device must fit.

## Finding and Connecting Objects

In the same way that the Internet repurposed existing technologies, and the car companies took advantage of available small motors, I too use existing devices beyond the initial purpose. My (smart) phone is an example. The very name is a reminder that it evolved from a device we used for spoken conversations to a portable computing device. The browser started as a document viewer but has become a portable computing platform that works across a variety of devices.

Traditionally smart devices have been built with use cases and required their own family of gear and their own networks. X10® was early and simply used power lines to turn things on and off. Later protocols (Zigbee, Z-Wave, Bluetooth required their own networks and families of devices. They had difficult playing with each other. From a business perspective, it made sense to focus on use cases. People want solutions and stories like "turn the light on at dusk" rather than raw technology.

I consider these to be *Found Objects* in the sense that I find them as-is and use software to repurpose them as components I can control with my own software.

Today devices are built on general purpose computing. After all, it's easier to use a general-purpose computer than hardwired logic for even something as simple as a light bulb (thanks to LEDs). The marginal cost of adding a software interface is relatively small and opens up new markets.

But the cost of doing a good interface and supporting developers can be high which is why so many companies focus using their own apps and gateways to Alexa, Apple, and a few others.

The good news is that this shift to using the Internet protocols to connect to these services is that the devices are increasingly accessible on my home network. Many of these have open protocols including LIFX, Nanoleaf, and Yeelight (Xiaomi).

I can also use hubs and services like Samsung's Smartthings even though some require a connection to the rest of the Internet to control devices in my home. This is

acceptable, for now, as long as I'm aware of the risks. It enables the evolution of a market for native devices that don't rely on such gateways.

Today's Internet is increasingly tuned for the web. The DNS (Domain Name System) and, for that matter, IP addresses depend on being connected to the larger world. Security certificates tend to be tied to the DNS. Connecting peer devices needs to work without that dependency so local systems can be autonomous. The goal is to connect devices to each other as peers so that we can focus on those two devices and nothing else. We describe this as "connecting to the Internet" because once two devices are "on the Internet" they can see each other. I don't have to worry about where a web page is hosted or which network, I'm on – I just connect my application to *that* device.

This isn't, yet, true for devices. A printer or light bulb on my home network is only visible to other devices on that network. In theory, IPv6 is supposed to fix this, but once you expose devices to the wider world you find that many devices and protocols aren't ready to be directly exposed to the larger world. They depend on having a barrier. This is one reason you need to go through a complex process of connecting to your local WiFi router. It's called on-boarding.

We see another form of this in the "agree" screens we see when we try to connect to a public Wi-Fi server. Devices using Bluetooth or other protocols require establishing a relation to a hub. Devices that use USB are only accessible with the right drivers and software in place.

In order to connect to the devices in my home from the outside, I need to set up forwarding ports to relay the connections. I use Ubiquiti's prosumer products to manage these connections and establish static IP addresses when needed such as setting up the port forwarding. I use a dynamic DNS and other software to make my home reachable from the outside. This allows me to connect directly with my house rather than depending on a cloud service like many webcams and other devices do.

I'm acutely aware of these issues because I can work around them sufficiently to get the benefits of connected devices despite these difficulties. Just as important, I am striving to do so in a way provides are a path to simpler interconnectivity.

Today's Internet – all devices having a global IP address is a good way to think about it even if today's implementation isn't quite there.

## Using Objects

In the 1990's I started by using X10 which is a simple peer protocol – pressing a button sends a short command over the power line. I wrote a program that used a PC⇔X10 interface so I could track the status of each device and run scripts to perform tasks such as turning off all my lights at night since the kids wouldn't do it. It could also operate the sprinklers. Since then it has evolved to support a wide variety of devices and protocols. One of the first was Insteon which was, essentially, a very improved version of X10. My program allowed me to use an Insteon button to turn on a Z-Wave lightbulb by using Smartthings' service as a gateway.

The architectural goal is to maintain the simple relationships despite the complexities behind the curtain. It also meant working around favors such as Smartthings' complex rule engine and cloud-dependency. When network operators try to improve performance by buffering, they break TCP by the real characteristics of the network.

By adding a web server to my program, I was able to write a browser app to control the devices and use inexpensive devices (such as the $30 Android tablets I can buy at Micrcenter) as control services mounted using 3M Command Strips. Because the application tracks events it can provide feedback to these screens showing the actual state of the devices.

I then reduced the dependence on the central program. It wasn't a problem when I was just using X10 and Insteon since the protocols didn't have a central point of dependency. But the browser app was dependent upon the server running on the PC. To reduce the dependence on a single server I created alternative servers to run on multiple machines including a Raspberry pi. They aren't as fully capable as the primary server, but they provide necessary resilience.

While my program does run scripts, it isn't so much a home control program as much as it complements the functionality of the devices and fills in some of the gaps. As part a shift to a more decentralized approach those scripts are now standalone programs that use the main program as a server, but which are capable of acting independently.

Today the sprinklers are run by their own computer more powerful than my original PC – a peer device rather than one that must be controlled.

## Caught in the Middle

Until we recognize we need transparent connectivity we're denying people the benefits of simple connectivity.

Working around many of the barriers along the path – the agree screens, the need for port forwarding, etc. has allowed me to achieve a degree of architectural simplicity.

I was just reminded of how far we have to go today in setting up a printer for a non-technical friend. It should be simple – tell the computer to print on *that* printer. But it won't work until I set up the WiFi SSID and then repeat the process if I move the printer somewhere else.

I've also been frustrated in doing everything right only to find a distant firewall setting block my path. Or an airline WiFi service second-guessing the connection.

This meddling in the path is done with the best of intentions, but it is second-guessing the relationships and applications.

## Writ Large

Much of the prosumer literature on home control focuses on all the wiring the details of how the system has been configured and setup.

My goal is different. I do want to be in control, but I know that homes are living and evolving. I should be able to manage my smart devices in the same way I manage furniture which I can rearrange as I see fit.

It is this dynamic interconnectivity that is how I interpret the "I" in IoT (Internet-of-Things). I write about the details of how I made my home system work as a cautionary tale. It shouldn't and needn't be this difficult.

I frame much of what I say in terms of "The Internet" but the Internet I'm thinking of is not limited to the details of the current protocols. For me it's the spirit of being able to "program around" and taking advantage of the opportunities to interconnect and the ability to enable to use the objects and capabilities.

The key is the architectural principle that contains complexity by separating the relationships among the end points (objects) from the complexity "between". Consumers (or at least prosumers) can then use software to redefine their world.

For now, though, I had to ask someone to close the shade in the hospital room.