Putting it all Together?

My First Half Century of Learning

Bits Versus Electrons

Putting It All Together?

By Bob Frankston

I've been programming for more than half a century, yet it seems as if little has changed in that time as I attempt to build systems out of elements. The lessons I learned long ago are just as timely today as we see a new generation of devices being connected to each other, what we call the Internet of Things (IoT). I was reminded of this recently when I was adding a new capability to my system and ran into bugs lurking in my code. Very simply, the more code we have in all the devices, the more strange interactions we'll run into.

For years, two of the lights in my house refused to stay on. I'd press on and soon they would go off. I discovered that if I only turned the light on part of the way, it would stay on. The lights happened to be in rooms where I had a presence sensor, so I figured the problem was with the sensors. But disabling them didn't solve the problem.

In the 1990s, I was experimenting with my own home control software using X10 devices. (X10 was developed in the 1970s in conjunction with British Sound Recording to turn record changers, or phonographs, on and off.) X10 survived for many years because it simply turned things on and off—it wasn't specific to any particular applications. X10 is a peer protocol without a central controller. This made it easy to deploy.

My software used an X10 serial interface that allowed me to not only send commands to the devices but also

FIGURE 1. A screenshot of the home control software developed by the author for his personal use.

observe what was happening. This is important because the user isn't in the same room as the lights, so we need to provide feedback to the user (i.e., people living in the house). This is even more important for scripts.

Over the years, I've continued to evolve the program, and Figure 1 shows the current version that still acts as a test bed for my ideas. It is also the back end for the HTML5 control panel I wrote about in my April 2014 column (http://rmf.vc/IEEEHTML5).

Yet, after all these years, there are still bugs lurking in the code. When upgrading the code for new devices, I discovered that the problem with the lights was actually due to a minor bug in my handling of timeouts and had nothing to do with the center. I thought I had explored that possibility in the past but obviously I had missed it. These kinds of bugs hiding bugs and complex interactions are common in production code, especially when we have systems interacting and evolving.

Supporting X10 presented interesting challenges because it isn't very reliable and doesn't scale well, so you can't trust the observed state. Using my family as a beta site provides me with ample feedback on the impact of this unreliability. Smarthome and others worked to improve the performance of X10, including making devices that could be queried to report their status.

I also implemented a sophisticated scripting capability but found surprisingly

Digital Object Identifier 10.1109/MCE.2014.2361400
Date of publication: 17 December 2014

104 IEEE CONSUMER ELECTRONICS MAGAZINE ^ JANUARY 2015

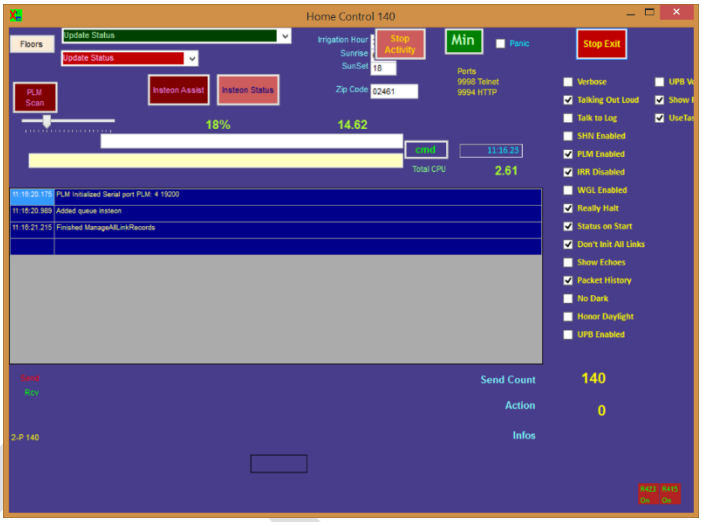
This is now my third year of writing this column. I thank the IEEE for giving me this opportunity.

The columns are available online through the IEEE. I've also posted pointers on my site. To make them easy to find you can use a URL like `http://rmf.vc/IEEECEyyyymm` year/month. This column is IEEECE201501. Columns have appeared in January (01), April (04), July (07) and October (10).

You can also reach email me at IEEEColumn@bob.ma.

I've been programming for more than half a century yet it seems as if little has changed in that time as I attempt to build systems out of elements.

The lessons I learned long ago are just as timely today as we see a new generation of devices being connected to each other – what we call the Internet-of-Things or IoT.



I was reminded of this recently when I was adding a new capability to my system and ran into bugs lurking in my code. Very simply the more code we have in all the devices the more strange interactions we'll run into.

For years two of the lights in my house refused to stay on. I'd press on and soon they would go off. I discovered that if I only turned the light on part way it would stay on. The lights happened to be in rooms where I had a presence sensor so I figured the problem was with the sensors. But disabling them didn't solve the problem.

In the 1990's I was experimenting with my own home control software using X10 devices. X10 was developed in the 1970's in conjunction with BSR (British Sound Recording to turn record changers - phonographs - on and off). X10 survived for many years because it simply turned things on and off – it wasn't specific to any particular applications. X10 is a peer protocol without a central controller. This made it easy to deploy.

My software used an X10 serial interface that allowed me to not only send commands to the devices but also allowed me to observe what was happening. This is important because the user isn't in the same room as the lights so we need to provide feedback to the user (AKA, people living in the house). This is even more important for scripts.

Over the years I've continued to evolve the program and the screen picture shows the current version that still acts as a test bed for my ideas. It is also the back end for the HTML5 control panel I wrote about in my April 2014 column (http://rmf.vc/IEEEHTML5).[i]

Yet after all these years there are still bugs lurking in the code. When upgrading the code for new devices I discovered that the problem with the lights was actually due to a minor bug in my handling of timeouts and had nothing to do with the center. I thought I had explored that possibility in the past but obviously I had missed it.

These kind of bugs hiding bugs and complex interactions are common in production code, especially when we have systems interacting and evolving.

Supporting X10 presented interesting challenges because it isn't very reliable and doesn't scale well so you can't trust the observed state. Using my family as a beta site provides me with ample feedback on the impact of this unreliability.

Smarthome and others worked to improve the performance of X10 including making devices that could be queried to report their status.

I also implemented a sophisticated scripting capability but found surprisingly few real uses. The main ones I still use are a script to turn the lights off in the middle of the night and another one to keep the porch lights on at night. Both of these are options. I purposely didn't want to create a central point of failure. My house has to function even if my PC is turned off or a hub device fails.

Programming around all X10 issues was educational but added complexity to my code. I was reminded of some of these lessons when I came across a cache of memos (Word files) I wrote twenty (!) years ago about home control.

I was working at Microsoft at the time and was also experimenting with how to network my home computers and connect them to the rest of the Internet. It was a time when home networking and home control were used interchangeably so I had to work to distinguish between the two concepts even as I found myself working in both areas by default including working on an effort called "Home Plug and Play".

And learn I did though mainly by what did not work. Perhaps it was the limitations of X10 that tempted my visions of automation but more likely it was the decades of working with the best engineered commercial systems and learning how quickly complexity overwhelmed even the best design as Fred Brooks so reminded us in *The Mythical Man Month*.

Protocol Efforts

CEBus attempted to bring automation to the masses by enabling the smart devices to just work together without complex setup. A thermostat would automagically connect to the temperature sensors and heating system. The problem with magic is that you don't know what is happening behind the curtain. Needless to say it was a failure because it was too damn smart.

We also saw **IEEE-1394 (Firewire)** which was another networking protocol with application requirements built into the networking protocols limiting it to a 6 meter distance.

Even as people tried to develop more sophisticated systems X10 rolled on because it was like the zeros and ones of computing – it just turned things on and off without having to second-guess what was being turned on and off.

With the Internet we exchange ones and zeros (packets) apart from their meaning. In both cases the separation of technology from what we do with it puts the user in control and enables rapid innovation as people discover new ways to meet their own needs.

Smarthome developed its **Insteon** protocol building on the experience with X10:

- Send messages directly between devices
- Stable addresses
- Modest expectations (light scenes)
- Inexpensive
- Leverage power line signaling

As an observer I watched as Insteon evolved. The initial market seemed to be OEMs who would use the core modules to build their own devices but that didn't work out and Smarthome picked up the slack and shipped a range of products that allowed me to replace all of the light switches and a number of other devices in my house with Insteon devices. They worked far better than X10 to the point that it functioned as reliably as traditional light switches but is far more flexible.

The devices were shipped without a program to manage them. Instead you'd press setup on one device and then press a button on the device that it was supposed to control. This approach has always seemed awkward especially when I would want to change the behavior of inaccessible devices. So I learned the protocol in a fair amount of detail and wrote my own program to manage the devices.

(This pairwise setup is also one of the annoying aspects of Bluetooth and much of Wi-Fi security.)

Eventually Smarthome implemented its own management software (Houselinc). My program is still important in that it complements the capabilities of Houselinc and allows support a wider variety of devices.

As the number of devices increased (well over 100 but who's counting?) the limitations of the protocol become a serious problem. Insteon messages are limited to three hops (wired or wireless) and must be acknowledged immediately by echoing the messages with a few bytes changed.

Lessons Learned

The lessons I learned in the process of living with home control technologies and both getting into the details of the implementations and living with them have been invaluable. I also need to thank my family for being a beta site, even if not always enthusiastically. If the lights don't work then the family is not happy.

You can read about what I've learned in writing the columns for this magazine. It's no coincidence that the columns are as much about the Internet as home control for both are coming to terms with the consequences of the increased computing power in our devices.

I would go even further and say that the two worlds of the Internet and home control are very much the same!

With home control (and consumer electronics in general) it's easy to think in term of "use cases" and assure that the protocols and devices work for those cases. A company or consortium of companies will design systems around these use cases. This is fine until we try to extend the system beyond the original examples.

Unlike protocols that are limited to their "use cases" the Internet protocols are the result of the inability to find any commonality among disparate uses other than the best efforts exchange of packets. It isn't owned by a single entity so it doesn't have to return value to an exclusive owner. It is a way we use existing facilities be they wires and radios or, for that matter, the existing telecom infrastructure, as a resource for exchanging packets. Instead of using up this resource it becomes more valuable with each application and participant.

Users with their computers are then free to find out what they could do with this new resource without having to justify the value to a third party. Thus a simple application like the web can be made available to all for even the most mundane application. After all who would've guessed that a modest application for sharing scientific papers would become today's Web?

Home Control: Another Planet

Home control has its origins in the world of consumer electronics in which we have manufacturers trying to make a profit on the work they do in developing chips and building devices. These manufacturers also design protocols to support their products and use cases.

When the protocols are extended outside their use cases they fail as with IEEE-1394.

Today we see a plethora of protocols that are really applications carefully engineered against requirements. We've seen this before when we spent many billions if not trillions of dollars building a phone system designed to assure that voice calls worked and then we built another parallel infrastructure for video and then again for cellular. It's amazing that even with the Internet demonstrating the value of fungible connectivity proof we keep seeing protocols and radios limited to their use cases.

As I've run into the limitations of Insteon I've become interested in these alternatives. I'm not going to manufacturer my own light bulbs or wall switches so I have to use software to work around the limitations of what I can buy.

And, indeed there are now a large number of devices to choose from but the market is hobbled by the need of each participant to add value rather than simply provide commodity components with little or even negative margin.

The protocols are driven by the familiar use cases of lighting scenes, automating the morning cup of coffee, opening the garage door and controlling the temperature in the house. The stories tend to assume Stepford (a reference to the book *Stepford Wives* in which real wives were replaced by robots). These are automatons who live in automated homes. They are very different from my real family with children and pets that think they are more important than the devices. They find themselves in a role of acting as a beta site or what I sometimes call my beta family.

Our smart devices and our ability to measure and quantify has given us many solutions in search of problems. And in search of profits.

The very complexity of the Z protocols makes interoperability hit or miss and much of the market these days is driven by companies selling hubs through which all devices must communicate. In order to add value to the hubs the companies try to provide added value smarts in various forms.

More frustrating is that so much of the value-added is in the form of applications on smart phones without the underlying API being exposed! To the extent that the API is exposed it shows the limitations of its design point.

I'm currently using SmartThings because of the API but I'm very aware that it's a shim and fragile at best. The idea that my home is dependent upon "the cloud" is troubling in itself. There is also a naiveté in expecting lots of procedural code to composite into a coherent and maintainable whole.

The experience I had with not knowing why my lights were turning off happened despite it being a very simple piece of code in isolation written by someone with decades of experience. Imagine what happens when you have hundreds of modules trying to work together. It can't and it won't. At least not without architectural thinking.

The successes we've seen are basically in embedded systems built with commodity technology. Electric Imp has been a leading supplier for the guts of such systems including Nest.

But whatever successes Nest's products have had they've been dwarfed by winning the lottery and having been bought out by Google for billions of dollars. SmartThings too won when they were purchased by Samsung. These big payouts have distorted the market and shifted the emphasis to winning the lottery rather than creating sustainable opportunity.

I'm writing this column months before CES and fear the many smart devices that will win innovation awards because, as one tag line says "she takes care of what matters most to you today". And how does this device know what matters to me today?

Humpty Dumpty

> Humpty Dumpty sat on a wall,
> Humpty Dumpty had a great fall.
> All the king's horses and all the king's men
> Couldn't put Humpty together again.

You can think of the challenge we face as the Humpty Dumpty problem – we have an idea about how the pieces can work together just like today's light switches do the right thing.

As the king's men found out – assembling a whole out of pieces isn't easy. Even more so when, as we do assemble the pieces, we find that we can't really articulate what we want because we're confronted by so many new choices and inherent ambiguity.

Looking Ahead

At this point I could sigh and lament the kids these days but I am optimistic. There are a lot of building blocks lying around and a lot of experimenting.

And there is a catalyst in those Internet protocols that continue to serve as the glue connecting all the disparate elements. Those Internet protocols are also limited by the use cases from the mainframe days but they are a starting point.

The lessons from my early days in working on systems in the 1960's still resonate today – the need to think beyond solutions to well-defined problems. Today our devices are really computer systems that interact in ways we can't anticipate. We are not fully in control. We need to learn how to design devices and craft businesses in this new environment.

We can't put Humpty together again but we can discover new paradigms into which the pieces fit.

[i] Too bad the IEEE doesn't have a similar simple way to provide links to articles using short keys.