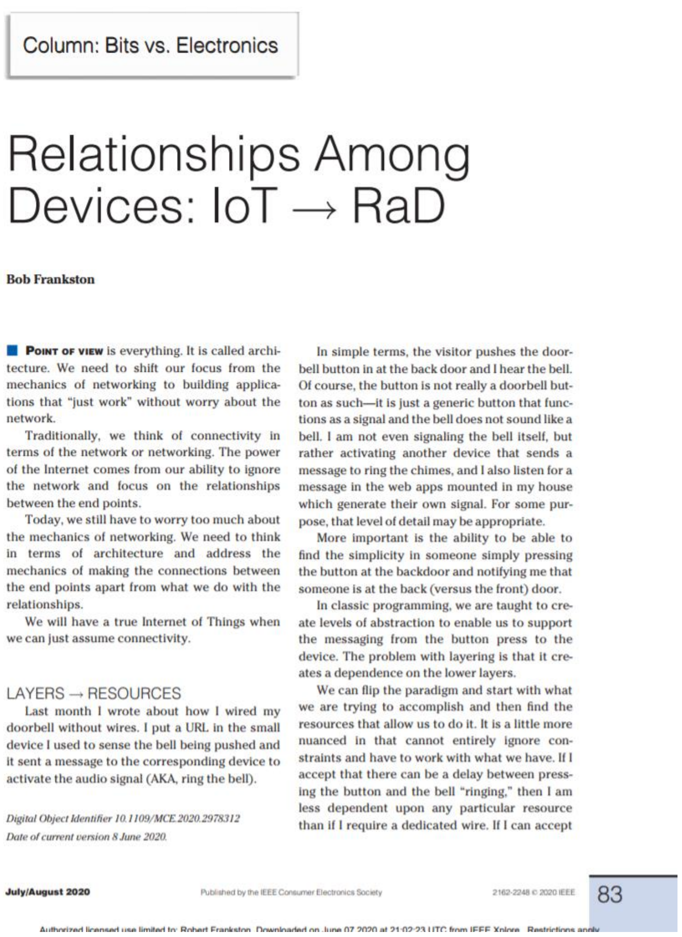
Relationships Among Devices: IoT □ RaD


Column: Bits vs. Electronics

Relationships Among Devices: IoT → RaD

Bob Frankston

Point of view is everything. It is called architecture. We need to shift our focus from the mechanics of networking to building applications that "just work" without worry about the network.

Traditionally, we think of connectivity in terms of the network or networking. The power of the Internet comes from our ability to ignore the network and focus on the relationships between the end points.

Today, we still have to worry too much about the mechanics of networking. We need to think in terms of architecture and address the mechanics of making the connections between the end points apart from what we do with the relationships.

We will have a true Internet of Things when we can just assume connectivity.

LAYERS → RESOURCES

Last month I wrote about how I wired my doorbell without wires. I put a URL in the small device I used to sense the bell being pushed and it sent a message to the corresponding device to activate the audio signal (AKA, ring the bell).

Digital Object Identifier 10.1109/MCE.2020.2978312
Date of current version 8 June 2020.

In simple terms, the visitor pushes the doorbell button in at the back door and I hear the bell. Of course, the button is not really a doorbell button as such—it is just a generic button that functions as a signal and the bell does not sound like a bell. I am not even signaling the bell itself, but rather activating another device that sends a message to ring the chimes, and I also listen for a message in the web apps mounted in my house which generate their own signal. For some purpose, that level of detail may be appropriate.

More important is the ability to be able to find the simplicity in someone simply pressing the button at the backdoor and notifying me that someone is at the back (versus the front) door.

In classic programming, we are taught to create levels of abstraction to enable us to support the messaging from the button press to the device. The problem with layering is that it creates a dependence on the lower layers.

We can flip the paradigm and start with what we are trying to accomplish and then find the resources that allow us to do it. It is a little more nuanced in that cannot entirely ignore constraints and have to work with what we have. If I accept that there can be a delay between pressing the button and the bell "ringing," then I am less dependent upon any particular resource than if I require a dedicated wire. If I can accept

Point of view is everything. It is called architecture. We need to shift our focus from the mechanics of networking to building applications that "just work" without worry about the network.

Traditionally, we think of connectivity in terms of the network or networking. The power of the Internet comes from our ability to ignore the network and focus on the relationships between the end points.

Today, we still have to worry too much about the mechanics of networking. We need to think in terms of architecture and address the mechanics of making the connections between the end points apart from what we do with the relationships.

We will have a true Internet of Things when we can just assume connectivity.

Layers □Resources

Last month I wrote about how I wired my doorbell without wires. I put a URL in the small device I used to sense the bell being pushed and it sent a message to the corresponding device to activate the audio signal (AKA, ring the bell).

In simple terms, the visitor pushes the doorbell button in at the back door and I hear the bell. Of course, the button is not really a doorbell button as such – it is just a generic button that functions as a signal and the bell does not sound like a bell. I am not even signaling the bell itself, but rather activating another device that sends a message to ring the chimes, and I also listen for a message in the web apps mounted in my house which generate their own signal. For some purpose,v that level of detail may be appropriate.

More important is the ability to be able to find the simplicity in someone simply pressing the button at the backdoor and notifying me that someone is at the back (versus the front) door.

In classic programming, we are taught to create levels of abstraction to enable us to support the messaging from the button press to the device. The problem with layering is that it creates a dependence on the lower layers.

We can flip the paradigm and start with what we are trying to accomplish and then find the resources that allow us to do it. It is a little more nuanced in that cannot entirely ignore constraints and have to work with what we have. If I accept that there can be a delay between pressing the button and the bell "ringing," then I am less dependent upon any particular resource than if I require a dedicated wire. If I can accept the idea that sometimes I cannot get the message through, then I have even more freedom.

There is an additional benefit to accepting limitations and failure – it reflects the real world. It may seem like a dedicated wire is far more reliable than depending on message, but the wire or button can fail due to mechanical and other problems. The more we assume there cannot be failure, the more brittle the system is.

Conversely there is the concept of opportunistic messaging. An interesting example is a system that allows a truck to send a message by bouncing it off the ionosphere when conditions are favorable. Most messages may not get through but enough get through to allow the truck to be tracked. Creativity creates opportunities.

Today, the cell phone is an alternative if there is a problem with the bell. More to the point, unexpected visitors are the exception because one typically calls ahead. Admittedly, failing to sign for a package can be a problem but, conversely, you now have the ability to have an alert if you are not at home—a feature unavailable if one is dependent upon a fixed wire. Conversely, the cell phone is another way to extend the reach for visitors.

We tend to assume the current practices are necessary but when we step back, we discover there are alternatives and opportunities to do “better”. The reason why this may not be obvious is that we have to think about what we mean by “better”. OK, getting into a deep philosophical dive.

Binding

Instead of thinking in terms of sending messages, I think in terms of binding the back-door button to the device in my office that announces someone is pressing the button.

Conceptually, I name the signal “BackDoorBell” and then use that name in the office device and bind it to the back-door alert. That’s it.

This is how we use services such as Alexa. When you buy a smart bulb and call it “Kitchen” and it is discovered by Alexa, it is binding to that identifier. If you get a new bulb you have to reestablish the binding.

In the rewiring example, I used a URL to send a message when the button was pressed to send a message to “ring the bell”. But, that is not the way I think of it. For me it is just one way to implement the relationship. It does use the network to send a message, but I think of it as binding the two endpoints. I can then manage the bindings rather than worrying about networks.

This shift in perspective is key to effective architecture. Both views are valid. Sometimes we need to think about the mechanics of getting message from A to B. When I am building an application in which I respond to the doorbell, I want to totally decouple the application perspective from the mechanics of the messaging. Doing so means giving up control.

If I don not hear the doorbell (signal) it may be because one of the modules is not working but it also may be because there is a general system failure such as a network problem. The latter affects all interactions and the fix is not specific to the doorbell.

I can extend this relationship so I can get the doorbell signal while traveling – something I cannot do if I am limited to a wire or local network. Today’s networking protocols make the implementation complicated because of the barriers in the path such as firewalls and paywalls (subscriptions). These impediments are treated as features because they seem to provide a security perimeter. But it is an ineffective approach that prevents lifesaving applications.

My medical monitor should work everywhere. Of course, it must use a secure (encrypted) connection using best practices. More important, the path must not be blocked by a myriad of second-guessers, be they firewalls or companies demanding a payment.

For some devices, such as my doorbell, having someone misusing it may be annoying and I can choose to limit access if it is a problem. The real security problem is not a prankster ringing the doorbell but rather other devices that depend on the firewall. If someone attacks the doorbell app and repurposes it to attack other devices there is a real vulnerability. Rather than relying on a physical security perimeter each device needs to take responsibility for its own safety, and we need an approach that is not associated with the physical network as I wrote in my column *Communities of Things*[i].

An “Internet” of Whatever

Which brings us to the Internet of Things – a term often used without thinking about how an IoT differs from traditional connected devices.

In reading papers, it seems as if the term is applied to any device that happens to be connected, especially using radios. The projects are often purpose-built embedded systems projects that happen to use some connectivity. The problem is that many of these are carefully constructed purpose-designed radio configurations and policies.

What is missing is the architectural magic that comes from separating the relationships (the bindings) from the messaging between the endpoints. Thus, the projects fail to take advantage of the synergy of generic connectivity and the ability to mix and match different approaches. Even when they use Wi-Fi there is a tendency to carefully map out the system in a static configuration.

If we are to realize the promise of an Internet of Things, we need to think outside the network, so we focus on architecture without being tied to the accidental properties of the network between the end points.

That would not only simplify design, but it allows us to provide generic connectivity that can be shared by all applications.

The Public Packet Infrastructure

This column focused on the relationships among the devices. We can now turn our attention to show to support connectivity and how to pay for the common infrastructure. But that is a full column in its own right.

[i] https://rmf.vc/IEEECommunitiesOfThings .