#APIFirst: How Things Speak to Each other

July 2015 Issue

Bits Versus Electrons

#APIFirst: How Things Speak to Each Other

By Bob Frankston

CONNECTED THINGS

If we are to understand the world of connected devices, we need to look behind the surface of the Web to the application program interfaces (APIs) that are the building blocks for the services we use and the ways devices "listen." The hash tag #APIFirst emphasizes the importance of this foundation.

The Internet is an outgrowth of the intelligence in our computing devices. We started interconnecting computers in the 1960s, and by the 1970s, we were extending local networks to form a worldwide Internet.

With the Web, we used those computers to enable people to work together whether they were in the same room or across the world. It is a testament to the success of that idea that we now think of the Internet as just the Web.

Today, our devices are increasingly becoming "intelligent" in the sense of having computing capabilities. We can now explore the opportunities created by our ability to interconnect or internetwork things.

I learn by doing and have been discovering the joys of Web-inspired APIs. An API is a way things speak to each other. The term comes from the way programs talk to each other, and now those conversations can happen between devices across the world.

Our homes (or at least my home) are places where we buy disparate devices and expect them to sort of work together. You expect that when you buy a coffee maker, you can just place it in your kitchen and plug it in without having to engineer it to carefully work with all the other devices.

We had similar flexibility in the days of the component audio system (aka Hi-Fi). In the 1990s, the consumer electronics industry tried to take advantage of the new computing technologies and created a new protocol, IEEE 1394, to interconnect the various audio and video devices. However, it was a step too far because it had too much application knowledge built into the protocol, including tight timing requirements, which limited its ability to act as a general transport.

Today, we still define point-to-point connections using physical cables such as HDMI cables, which work well in the simple cases. As I wrote in my last column, we now use software to define the relationships instead of running wires between each pair of end points. With devices such as Apple Air and Chromecast and protocols like the one that Microsoft uses for projects, I can connect any video source to any screen.

The key is simplicity—the kind of simplicity we get by normalizing everything to packets. We can then send the packets over any medium. Interconnecting all these various transports has given us today's Internet.

This thinking was the basis for the paper [1] I presented in 1997 at the IEEE Consumer Electronics Workshop in Singapore and led directly to my current involvement and this column.

At that time, I was at Microsoft trying to figure out how to implement home control systems with a simple constraint—preserving the ad-hoc connectivity that we had in our homes and simple consumer electronics systems.

The approach I took very much honored the spirit of the Internet in trying to provide building blocks rather than a single solution. I was not trying to be prescient, but rather, as a programmer, I wanted to be able to craft my own solutions. Again, this is the spirit that gave us the Internet. People with programmable devices did not want to petition telecommunication companies each time they wanted to implement a new capability or service.

The simple answer was to have an open interface for each device without trying to force it in a single framework beyond that. If one video device says that it displays the content of "Channel 508" and another identifies the source as "HBO," we have a mismatch. These inconsistencies crop up as we try to interconnect devices. People can deal with them, but if we want to type "Game of Thrones" and just see it, we need to have common practices. We use the flexibility of software to program around such mismatches. That's what Google TV does by having a database that it uses to match up the various ways of identifying the content.

This is not always simple because we cannot necessarily map a temperature, for example, into a comfort index. However, over time, we develop common practices and bring the various

Digital Object Identifier 10.1109/MCE.2015.2422853
Date of publication: 15 July 2015

This article is now available online. You see other articles listed in Further Reading. Also see the related article on OAuth.

Connecting Things

If we are to understand the world of connected devices we need to look behind the surface of the web to the APIs (Application Program Interfaces) that are the building blocks for the services we use and the ways devices "listen". The hashtag #APIFirst emphasizes the importance of this foundation.

The Internet is an outgrowth of the intelligence in our computing devices. We started interconnecting computers in the 1960's and by the 1970's we were extending local networks to form a worldwide Internet.

With the web we used those computers to enable people to work together whether they were in the same room or across the world. It is a testament to the success of that idea that we now think of the Internet as just the web.

Today our devices are increasingly becoming "intelligent" in the sense of having computing capabilities. We can now explore the opportunities created by our ability to interconnect or internetwork things.

I learn by doing and have been discovering the joys of web-inspired APIs. Application Program Interfaces are a way things speak to each other. The term comes from the way programs talk to each other and now those conversations can happen between devices across the world.

Our homes (or at least my home) are places where we buy disparate devices and expect them to sort of work together. You expect that when you buy a coffee maker you can just place it in your kitchen and simply plug it in without having to engineer it to carefully work with all the other devices.

We've had similar flexibility in the days of component audio system (AKA Hi-Fi).

In the 1990's the consumer electronics industry tried to take advantage of the new computing technologies and created a new protocol, IEEE-1394, to interconnect the various audio and video devices. But it was a step too far because it had too much application knowledge built into the protocol including tight timing requirements which limited its ability to act as a general transport.

Today we still define point to point connections using physical cables such as HDMI which work well in the simple cases. As I wrote in my last column, now we use software to define the relationships instead of running wires between each pair of end points. With devices like Apple Air, Chromecast and protocols like the one that Microsoft uses for projects I can connect any video source to any screen.

The key is simplicity – the kind of simplicity we get by normalizing everything to packets. We can then send the

packets over any medium. Interconnecting all these various transports has given us today's Internet.

This thinking was the basis for the paper (http://rmf.vc/IEEE1997) I presented in 1997 at the IEEE/CE workshop in Singapore and which led directly to my current involvement and this column.

At that time I was at Microsoft trying to figure out how to implement home control systems with a simple constraint – preserving the ad-hoc connectivity that we had in our homes and simple consumer electronics systems.

The approach I took very much honored the spirit of the Internet in trying to provide building blocks rather than a single solution. I wasn't trying to be prescient but rather, as a programmer I wanted to be able to craft my own solutions. Again, this is the spirit that gave us the Internet. People with programmable devices didn't want to petition telecommunication companies each time they wanted to implement a new capability or service.

The simple answer was to have an open interface for each device without trying to force them in a single framework beyond that. If one video device says that it displays the content of "Channel 508" and another identifies the source as "HBO" we have a mismatch. These inconsistencies crop up as we try to interconnect ~~each~~ devices. People can deal with them but if we want to type "Game of Thrones" and just see it we need to have common practices. We use the flexibility of software to program around such mismatches. That's what Google TV does by having a database it used to match up the various ways of identifying the content.

This isn't always simple because we can't necessarily map a temperature, for example, into a comfort index. But over time we develop common practices and bring the various description into alignment if there is sufficient interest just as we now use USB as a common power supply.

The good news is that there seems to be a convergence on the ingredients in this soup that are giving us the opportunity for an #APIFirst approach as an alternative to the idea that we need to build *and maintain* an app for each purpose on each platform.

RESTful APIs and the rise of JSON

As a software guy I want to be able to take advantage of existing devices rather than building new devices for each purpose. In 1997 my solution was open interfaces using XML. Today the world is discovering open interfaces using JSON instead in the form of RESTful APIs with OAuth.

These acronyms are inherited from common practices in exposing services for web set. We can use the same interfaces for machine-to-machine communications.

The end points are identified by URIs (Universal Resource Identifier). The URL also specifies the message format with HTTP ((Hyper-Text Transport Protocol) being the most common. It is similar to an email message and can carry content in any format.

A web page is an HTML message. For data we use XML which is very similar to HTML. Using JavaScript, programmers figured out that they could modify the HTML document dynamically to retrieve data without going to a new page. This technique became formalized as XMLHttp.

Over time the use of XMLHttp has evolved to become the standard way to provide APIs (Application Programming Interfaces) for web services and, increasingly, devices.

XMLHttp is simply a transport for any message format. Today it is increasingly common to use JSON (JavaScript Object Notation). JSON is simply the object notation from JavaScript:

```
{
  "First": "Joe",
  "Last: "Random",
  "Age": 34
}
```

It is simpler than XML and more readable. You might also recognize it as the notation used for CSS in HTML. It nicely maps into the class notation in common programming languages.

Both JSON and XML are represented as text strings. You don't need special tools to read them. Because we use HTTP as a generic transport the message is passed through directly to the device or service without depending on intermediaries. This is very different from protocols like Bluetooth which depend on every intermediary cooperating. In fact messages should be encrypted to protect the end-to-end relationship.

JSON notation can also be used to describe JSON objects. This is one reason for using TypeScript (a superset for JavaScript). One can declare:

```
interface Name {
 First: string;
 Last: string;
 Age: number:
}
```

IDEs (Integrated Development Environments) can offer hinting and help manage code as it evolves and is shared

with others. It's a useful way to exchange structured data over other transports such as WebSockets (a bidirectional streaming protocol for HTML5) and MQTT (a queued message protocol).

JSON is easy to use with a wide variety of programming languages. For example it's a nice match for anonymous types in C#. The site http://apiary.io provides examples in a number of programming languages. http://json-ld.org (where "ld" stands for Link Data) uses JSON to describe the semantics of the data.

Another part of this mix is OAuth (http://OAuth.net) which is a capability-like mechanism that can be used in conjunction with HTTPS (encrypted HTTP) for authorization. OAuth is part of an evolving set of practices that are loosely coupled and is becoming the norm for authorization for web services.

OAuth is a mechanism – it doesn't solve the problem of who you should trust but it does provide one way to give those you trust a way to present credentials.

This ability to evolve and share is what gives this community its energy and the ability to converge on a common set of good ideas. The simple protocols facilitate cooperation and learning much as the Internet packet protocols facilitate connectivity.

In addition to the sites publishing APIs, we have sites like http://GitHub.com with utilities and sample code that take advantage of these APIs.

We're still at an early stage. Many of the APIs go to web sites rather than directly to devices. This currently works well-enough because today's Internet is too reliable and we become overly dependent upon being connected all the time. As a result we can't turn on the lights in our homes if the Internet connection goes down. We need resilience rather than brittle dependencies.

These dependencies are also about business as we see companies trying to turn your door lock into a service with a monthly fee.

OAuth itself is a mechanism for trust relationships but it doesn't dictate our policies. Note that I am using the word "trust" rather than "security" to emphasize that we need a far more nuanced approach than secure/insecure. We can't just put a border around the stuff we want to protect.

Trust relationships don't necessarily correspond to physical boundaries. One well-publicized attack, Target, happened because the HVAC system inside the security perimeter was compromised. In terms of APIs a security perimeter represents an implicit policy determining which relationships are feasible and which aren't.

If I want to turn on a light it's not enough to say "that bulb". I need to be explicit and give the current network address of the bulb and which network it is on. You see the problem when you try to use an application and have to make sure you are on the same network.

Some of this is hidden by connecting via the cloud – to reach a light bulb on the "wrong" network it has to be registered with a distant provider and your device has to have its own rendezvous protocol. You could use a DNS name but then you have to make sure you renew the name of your house every year.

The need for simple peer connectivity and local discovery will force us to return to the design point of the Internet – local peer connectivity.

APIs vs. Apps

#APIFirst represents a shift in thinking for an industry ~~that is~~ focused on creating complete solutions for consumers. Today the APIs themselves are generally not seen as a source of value for consumer products (as opposed to commercial services).

This will change as companies recognize the importance of APIs in allowing their customers or third parties to add value to their products. A washing machine with an API would allow a customer to extend the capabilities and implement their own notification procedures.

We don't need perfect APIs – software can take advantage of the capabilities available as we develop common practices. While most users will choose to use well-honed applications built on tested APIs, there are an increasing number of people with the skills to take advantage of the APIs and discover new possibilities.

Recently my FiOS battery ran low and started beeping every 15 minutes. It took days to figure out the source of the annoying sound. If the battery (or other parts of the system) had an API (including notification capabilities) then my software and/or Verizon's could have given me a real message instead of speaking the language of R2D2.

In fact many such interfaces exist now but aren't accessible, that is, open to all. The challenge is to explain why we need open interfaces because the manufacturers can't anticipate all possibilities and with open interfaces their customers (and third parties) are able to add value to the product.

But it isn't a completely benign transition as the API might shift some of the added value from the device to external software.

As we've seen, the transition to APIs is happening in any case. Just as the web couldn't have existed if we didn't have open protocols to build on, the APIs are the building blocks that presage that next iteration of the industry.

They enable discovery and exploration of ideas. Of course they also have the potential to shift power and value away from incumbents. Yet without these building blocks the market tends to stall out. In earlier columns I've pointed out the limits of trying to interconnect products using traditional system design approaches aimed at solving particular problems.

Currently these APIs have primarily been used to provide access to online services and to hubs rather than a way to directly communicate with devices. That's likely to change as census forms.

We saw this happen with USB. After decades of wringing over electric power connectors the industry converged on USB as a power supply simply because it was available. It's only after micro-USB became a standard that efforts began to focus on a next generation USB-C connector.

We're starting to see such an evolution in home control with the Thread protocol moving in the direction of providing a pure packet protocol apart from any particular application. (Though, as of now, it still has a problematic security boundary in the architecture).

The API is just one mechanism. It allows us to focus separately on different aspects of connected things:

Trust relationships. The trivial example "who is allowed to turn on the light in the shared driveway". OAuth may be a good mechanism for implementation policy without dictating any particular policy. Which app has what rights on the device I'm holding?

Discovery. Search for things. How do I find *that* printer? If I'm traveling how, does the doorbell alert me that someone is asking permission to enter my house and how do I provide that permission?

Semantics. What do the messages mean? Even something as simple as turning on a light can be rethought. Do I turn on a bulb or open a shade? And what kind of lighting do I want? How do we assure that "do what I say" and "do what I mean" are in alignment?

We don't even know what the new architectures are. How do we deal with devices in shared environments? The list is long and we can only learn by doing. There is not one singular future user case we can aim at.

These issues are very different than the way we think of today's Internet. At the very least an API-based world of connected devices is incompatible with today's telecommunications policies that depend on charging for messages crossing network boundaries.

On the technical side connected things challenge a number of the engineering assumptions in today's Internet which has been tuned for ~~the~~ current web-oriented uses such as content delivery. The IP and DNS do not provide the stable relationships we need going forward and devices must be able to communicate with each other even if there is no connectivity with the rest of the Internet.

If you have two devices in the same room they should be able to communicate directly without having to depend on external services such as the DNS. If I'm living in an apartment house I should still be able to speak to my neighbors over the local network even if there is a storm and my Internet connection is out. Not just #APIFirst, but also #LocalFirst.

The Interconnecting of Devices

Today's Internet is an outgrowth of our effort to connect our computing devices. Initially connectivity was limited to local communities on LANs or proprietary networks. The Internet was, as the name implies, the internetworking of these local communities. The ability to reach distant computers was so exciting that it seemed to be the only purpose of the connectivity. As a result, to many the web *access* is the Internet.

#APIFirst shifts the emphasis back to the enabling technologies and peer connectivity where connecting to local devices is just if not more important than reaching distant sites. After all this concept of locality is fundamental to system design and nature itself. The light bulb you care about is the one next to you not the one a thousand kilometers away.

Such remote access may be the exception but if you're expecting a plumber while on vacation you want to have the option of letting the plumber in.

The web has brought the Internet to the forefront of society. We're now ready to build on this infrastructure and rediscover the power of peer connectivity.

Today's Internet policies have been centered on the idea of accessing web pages and content delivery. Connected devices will shift the emphasis to local connectivity and peer relationships.

Just as it took a decade to start to discover the potential of the web, we are at the beginning of understanding the world of connected devices – the promises and the challenges.