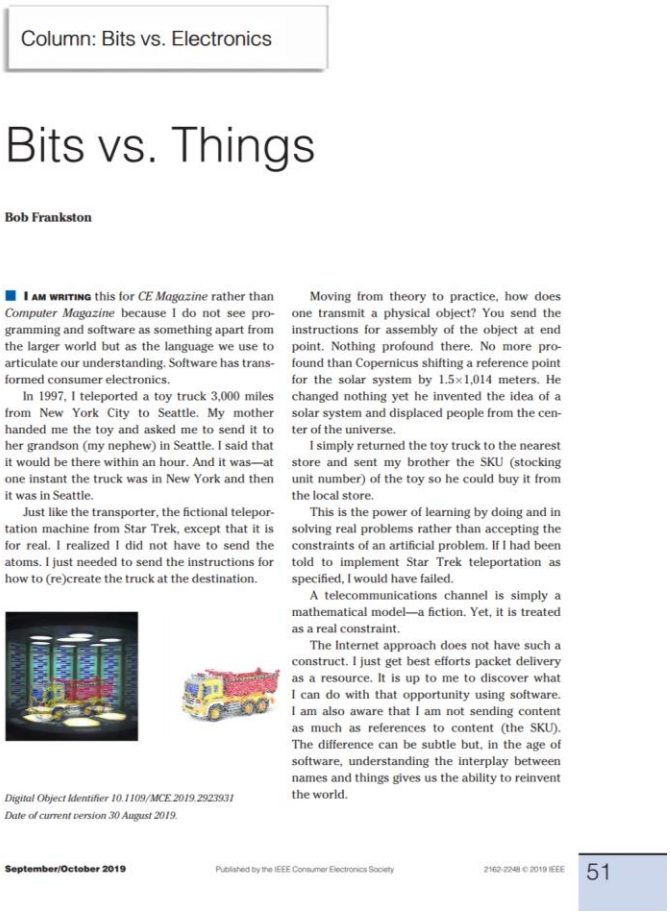
Bits vs. Things



Column: Bits vs. Electronics

Bits vs. Things

Bob Frankston

I AM WRITING this for *CE Magazine* rather than *Computer Magazine* because I do not see programming and software as something apart from the larger world but as the language we use to articulate our understanding. Software has transformed consumer electronics.

In 1997, I teleported a toy truck 3,000 miles from New York City to Seattle. My mother handed me the toy and asked me to send it to her grandson (my nephew) in Seattle. I said that it would be there within an hour. And it was—at one instant the truck was in New York and then it was in Seattle.

Just like the transporter, the fictional teleportation machine from Star Trek, except that it is for real. I realized I did not have to send the atoms. I just needed to send the instructions for how to (re)create the truck at the destination.

Moving from theory to practice, how does one transmit a physical object? You send the instructions for assembly of the object at end point. Nothing profound there. No more profound than Copernicus shifting a reference point for the solar system by 1.5×1,014 meters. He changed nothing yet he invented the idea of a solar system and displaced people from the center of the universe.

I simply returned the toy truck to the nearest store and sent my brother the SKU (stocking unit number) of the toy so he could buy it from the local store.

This is the power of learning by doing and in solving real problems rather than accepting the constraints of an artificial problem. If I had been told to implement Star Trek teleportation as specified, I would have failed.

A telecommunications channel is simply a mathematical model—a fiction. Yet, it is treated as a real constraint.

The Internet approach does not have such a construct. I just get best efforts packet delivery as a resource. It is up to me to discover what I can do with that opportunity using software. I am also aware that I am not sending content as much as references to content (the SKU). The difference can be subtle but, in the age of software, understanding the interplay between names and things gives us the ability to reinvent the world.

Digital Object Identifier 10.1109/MCE.2019.2923931
Date of current version 30 August 2019.

I'm writing this for *CE Magazine* rather than *Computer Magazine* because I don't see programming and software as something apart from the larger world but as the language, we use to articulate our understanding. Software has transformed consumer electronics from simply choosing what is available to having the ability to create and share one's solutions.

In 1997 I teleported a toy truck 3000 miles from New York City to Seattle. My mother handed me the toy and asked me to send it to her grandson (my nephew) in Seattle. I said that it would be there within an hour. And it was – at one instant the truck was in New York and then it was in Seattle.

Just like the transporter, the fictional teleportation machine from Star Trek, except that it's for real. The writers of Star Trek could just handwave the technical details. I had to engineer a solution that actually worked. That constraint was empowering. I realized I didn't have to send the atoms. I just needed to send the instructions for how to (re)create the truck at the destination.

The writers of the TV show didn't have to think about the distinction between sending a physical object and sending information about the object. They could mix the metaphor of radio transmissions of information with the metaphor of physical movement.

Moving from theory to practice, how does one transmit a physical object? The answer, when presented with the task, is fairly trivial. You send the instructions for assembly of the object at the endpoint. Nothing profound there. No more profound than Copernicus shifting a reference point for the solar system by 1.5×1014 meters. He changed nothing yet he invented the idea of a solar system and displaced people from the center of the universe.

I didn't actually send the instructions for building a truck out of parts but simply returned the toy truck to the nearest store and sent my brother the SKU (stocking unit number) of the toy so he could buy it from the local store. That is a valid instruction. If I sent step by setup instructions for building a toy truck out of Lego blocks, I would still be sending out a series of SKUs plus the instructions for how to assemble them into the specific whole that is the truck. I just chose a SKU that made the assembly trivial.

This is the power of learning by doing and in solving real problems rather than accepting the constraints of an artificial problem. If I had been told to implement Star Trek teleportation as specified, I would have failed.

This is like the challenge of debugging programs. We can prove that there are no general solutions, yet we still debug programs in the real world.

This kind of artificiality has bedeviled telecommunications policy and systems design for the last century. A communications channel is a mathematical construct that doesn't have a counterpart in the physical world. The Internet approach is to do more by doing less. Instead of providing me with a channel, I just get best efforts packet delivery and then discover what I can do with that opportunity using software. I'm also aware that I'm not sending content as much as references to content. The difference can be subtle but, in the age of software, understanding the

interplay between names and things gives us the ability to reinvent the world.

It allows me to transport (the information about) the truck and, in effect, send the truck through a thin wire.

We ignore the distinction in everyday language which creates a path-dependent understanding. If we ask the theoretic question about channels, we accept the constraints as if they were real. It doesn't help that the very words we use make this easy – "communicate", "information", and "channel" all sound like their day-to-day counterparts but are very different. In fact, we go so far as to create channels out of nothing when we allocate radio frequencies to serve as dedicated channels.

In engineering it is useful to build up abstractions so we don't have to solve all problems from first principles, but we mustn't forget they are just constructs. These constructs are understood in an implicit context.

The idea of sending a part number is not new. A hundred years ago you could've called me on the phone (or, more likely, sent a telegram) and told me that you admired the shirt I was wearing and then ask me to go to the store (say, a local Sears) and buy one just like it and send it to you. I may have told you that you could go to the Sears near you and buy the same shirt. Or that you could order it from Sears, and they would ship it from the nearest warehouse.

Yet, for communications engineers, if I pose the question a little differently and ask how many bits it would take to ship you a copy of an encyclopedia that takes a terabyte on disk you may accept that I could compress it to a zip file so that I only had to ship a few gigabytes. When I say I can compress it to a few bytes (less than a 64-bit word) I am told that, according to information theory, it's impossible. Using an ISBN number as my dictionary is not cheating any more than using English words. Yet communications engineers use this very technique when sending data – a shared clock is an out-of-band signal.

In the same way, stored programmed computing has evolved from high speed calculating to something far more nuanced. Copernicus's heliocentric solar system gave the impetus to physics by showing a regularity that Newton could build on.

There is value in thinking about names of things vs. the thing itself. Naming is not a simple concept. The distinctions we make between identity and addresses are operational and not fixed. One big idea is our ability to manipulating naming or binding.

Which brings us to the idea that the truck in New York and the truck in Seattle were the very same truck. You may not have thought about it that way because it wasn't a big issue at the point because you know my nephew wouldn't care. But it is a very big deal since so much policy and philosophy is based on a sharp distinction between direct and indirect actions. With software there isn't such a sharp distinction and we've operationalized abstractions.

This gets confusing when we use existing vocabulary in this new context. That's the way language works – economy of mechanism. Once we did depend on the telecommunications industry in giving us the (only) way to instantly communicate over a distance. Hence, we call it telecommunications. Now communications as technology and communications as speech are no longer aligned. Yet, in the US, we're left with a Federal Communications Commission that is betwixt and between. Other countries have their counterparts such as Ofcom in the UK.

I've written about how we discovered "best efforts" by programming around an unreliable middle using techniques such as retransmitting packets from the end points. We also discovered the power of better never than late and the idea that we can fill in gaps rather than assuming the missing value is zero.

When we transmit a SKU number it has no meaning to the facilities between the two end points. That has another deep philosophical concept – the (meaning of the) data not existing in the middle. The meaning only comes from context at the endpoints. This is even more interesting when we think of sending the toy truck across the country. It doesn't exist in the between. And the same process works for a full-scale truck too!

This concept of binding is central to programming. In 1958 John McCarthy came up with the idea of a programming language based on Lambda Calculus -- LISP. Unlike Fortran, which was used to perform calculations, LISP was used to operate on data schemas rather than the data itself. This is why it was an early favorite for AI. Schematic evaluation is also why relational databases are so powerful. And those concepts are now central to JavaScript.

Together these concepts are part of what I call a new literacy. It gives us a vocabulary for thinking about how systems work and gives us a way to understand how meaning derives from context rather than being intrinsic.

Alas, such an understanding is the basis for a surveillance economy that is not about privacy in the traditional way of protecting individual pieces of data but rather about how we deal with the data vapor all around us that doesn't affect us so much as individuals as much as part of populations. These issues are at the forefront of social policy as well as technical policy, yet we barely grasp these concepts and do not have a vocabulary for talking or even thinking about them.

These are interesting topics in the abstract, but I see them as coming to the fore in a consumer electronics industry that is increasingly defined by software rather than hardware. Or, should I say, the consumer technology industry as the focus shifts from the electrons and atoms to the new literacy and paradigms.

What will happen as true digital natives, the ones who casually write software not just consume content, come into their own? They won't be simply consumers of technology nor content creators; they will be citizens of a new reality.