

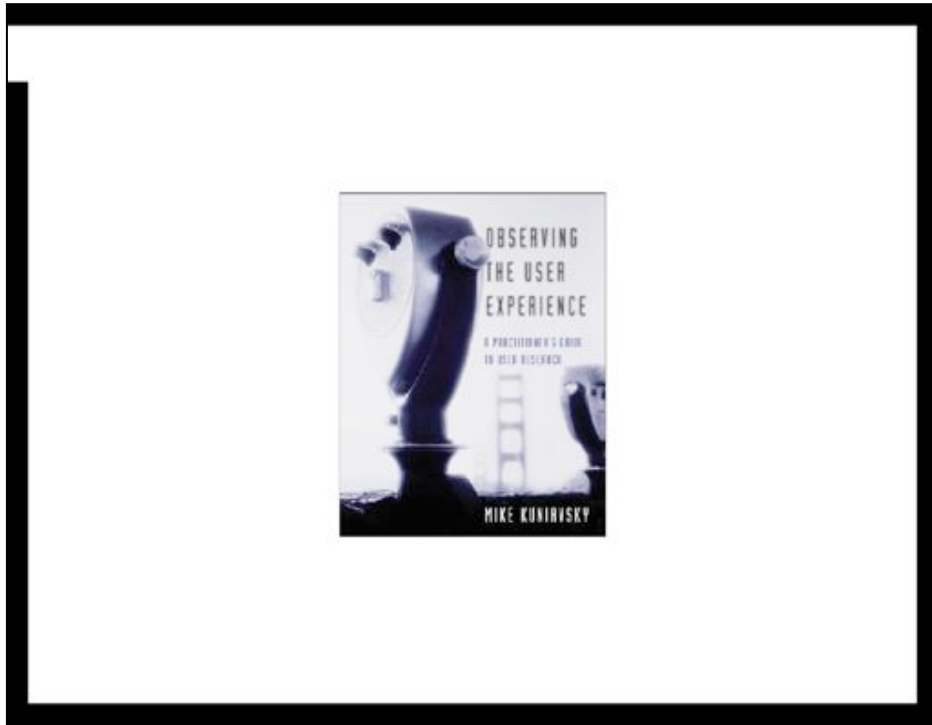
**SERVICE AVATARS**

**Mike Kuniavsky**  
SEE  
Amsterdam  
November 10, 2010

Good morning! Thank you for inviting me.



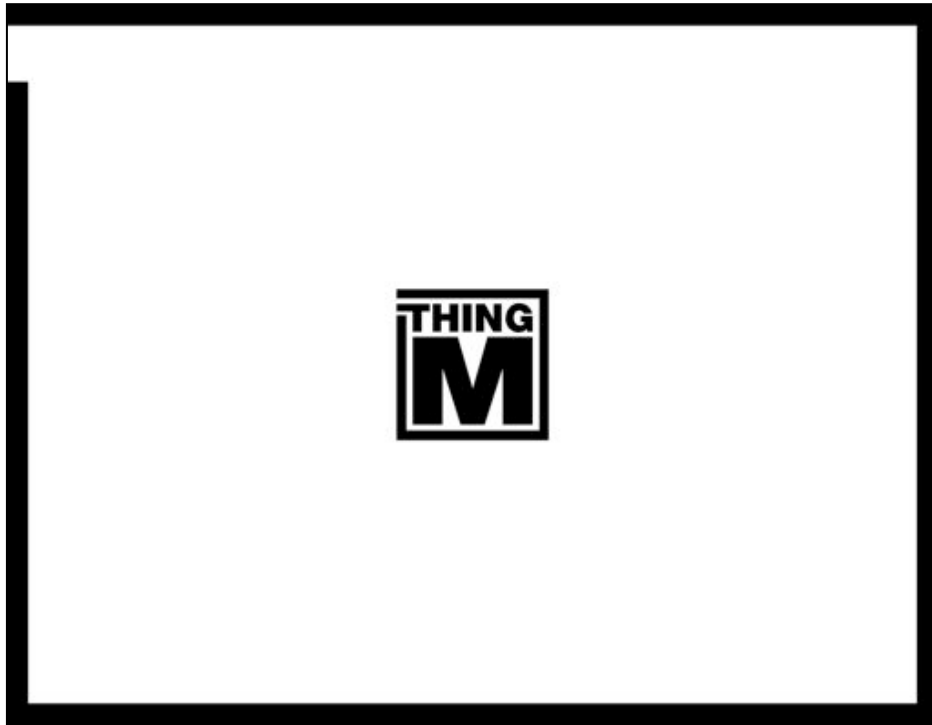
First, let me tell you a bit about myself. I'm a user experience designer and entrepreneur. I was one of the first professional Web designers in 1993. Since then I've worked on the user experience design of hundreds of web sites. I also consult on the design of digital consumer products, and I've helped a number of consumer electronics and appliance manufacturers create better user experiences and more user centered design cultures.



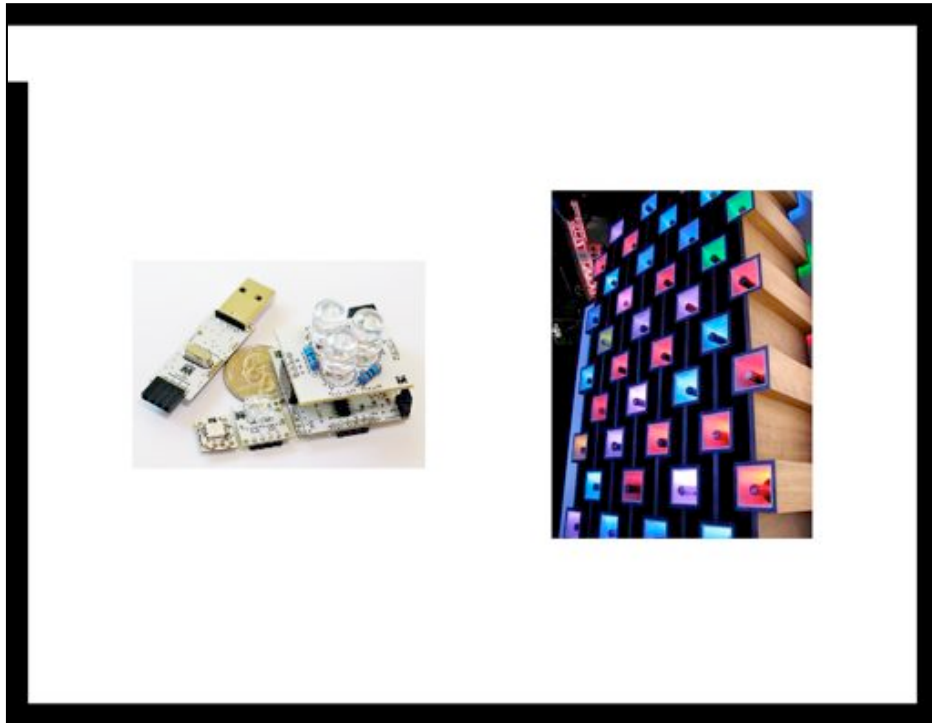
In 2003 I wrote a how-to book of user research methods for technology design. It has proven to be somewhat popular, as such books go.



Around the same time as I was writing that book, I co-founded a design and consulting company called Adaptive Path.



I wanted to get more hands-on with technology development, so I founded ThingM with Tod E. Kurt.



We're a micro-OEM. We design and manufactures a range of smart LEDs for architects, industrial designers and hackers. We're also spinning off a new company that's going to apply this technology to the consumer space. I have lots to say about that, but this talk is about something else, so talk to me offline if you'd like details.

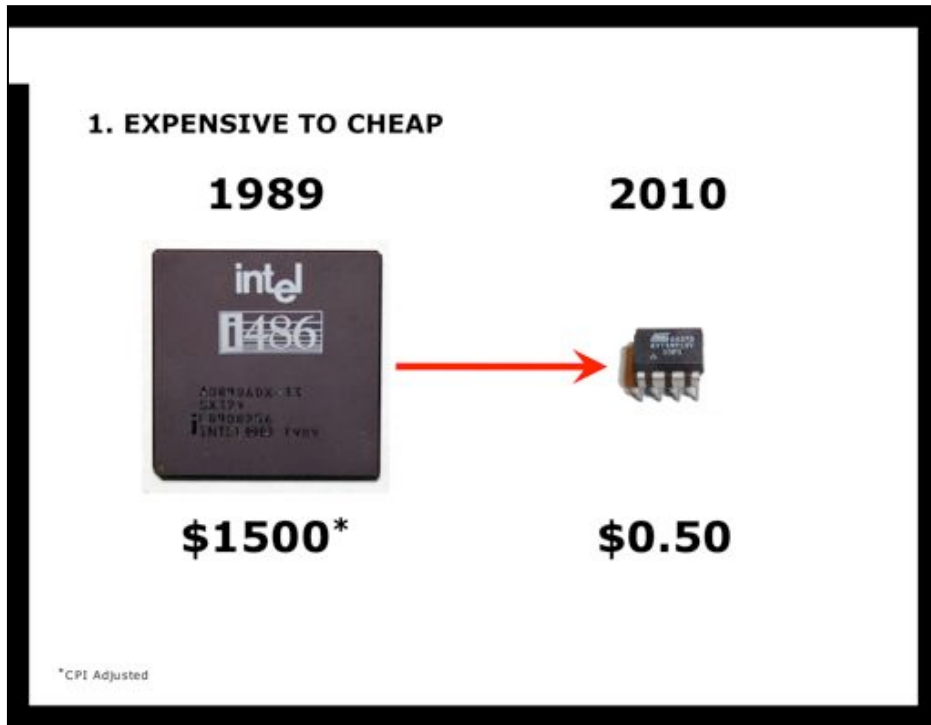


This talk is based on a chapter from my new book. It's called "Smart Things" and it came out a couple of weeks ago. In the book, I describe an approach for designing digital devices that combine software, hardware, physical and virtual components.

## **THREE TRENDS**

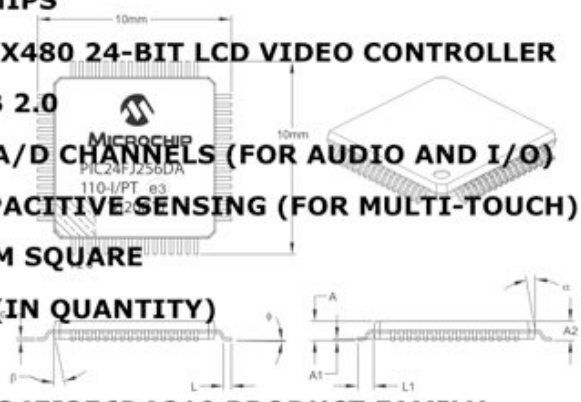
I want to start by talking about three trends that are combining to create a new class of digital products which are distributed through the environment and linked by cloud-based services. These devices not only create many opportunities for innovation, but they represent a new way of thinking about both products and services.





The first trend is a product of Moore's Law. Normally people think of Moore's Law in terms of processor speed, but the same technologies that makes the latest chips powerful push the price of older technology down. We have now reached a point where many powerful technologies are priced like basic commodities. For example, the Intel 486 was the processor that the Web was built for and with. It cost \$1500 in 1989. Today you can get as much processing power for about 50 cents.

- **16MIPS**
- **640X480 24-BIT LCD VIDEO CONTROLLER**
- **USB 2.0**
- **24 A/D CHANNELS (FOR AUDIO AND I/O)**
- **CAPACITIVE SENSING (FOR MULTI-TOUCH)**
- **1 CM SQUARE**
- **\$5 (IN QUANTITY)**
- **PIC24FJ256DA210 PRODUCT FAMILY**



The image contains technical diagrams of the PIC24FJ256DA210 chip. On the left is a top-down view of the square chip, labeled 'MICROCHIP PIC24FJ256DA 110-I/PT e3', with dimensions of 10mm by 10mm. To the right is a perspective view of the chip's package. Below these are two detailed cross-sectional diagrams of the package, showing various dimensions such as height (A), width (A2), and length (L1).

This new System on a Chip from Microchip has about as much processing power as that initial 486, but is also has an onboard video controller that can drive a VGA-class screen, a USB controller for peripherals, a 24-channel analog to digital converter for sensor, and a capacitive sensing driver that can drive a touch screen. It costs about \$5, uses less power than a keyring LED flashlight, and fits on a chip the size of your fingernail. It's also not unusual. Almost every semiconductor maker makes similar products.

This means that you can now include powerful processing and networking in almost anything, and start rethinking the design of everything in terms of embedded digital technology. The “how” problem of creating ubiquitous computing has almost been answered. Now the questions are what to create, and why.

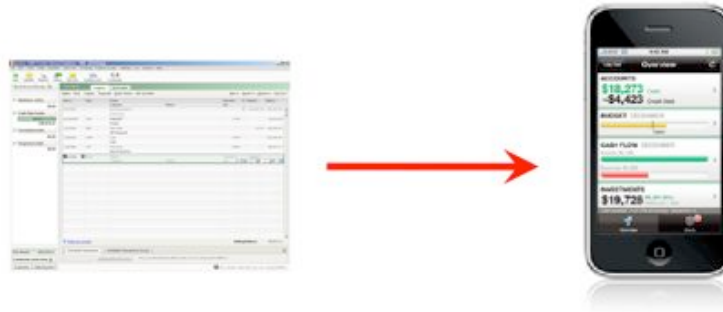


The answer to these is being driven by two other shifts.

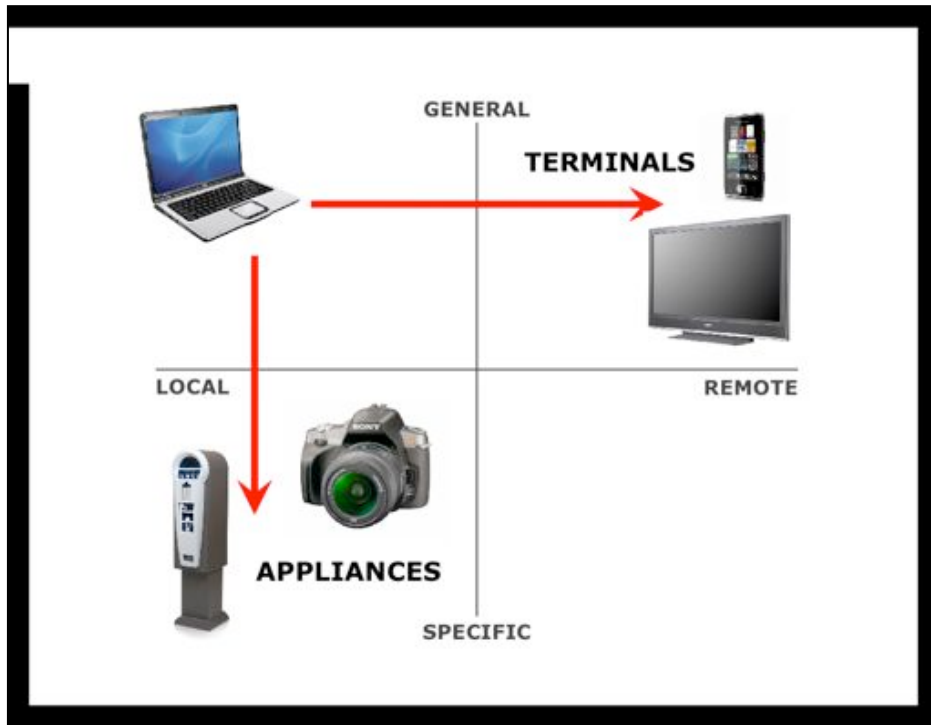
First, is a shift from generic devices and software to specialized devices and software. When computing was expensive, you had one or two general purpose devices that had deal with almost every situation. This necessitated design compromises that resulted in devices and software that could do almost everything, but did none of it well. It was then up to the user to take these generic tools and making them appropriate to the current situation.

Now that processing is so cheap, you can have a combination of 10, 20, or 30 computing devices and apps for the price of that one device, and you can acquire new functionality as needed. This means that every device and software package can have a narrower purpose.

### 3. LOCAL TO REMOTE



The third trend is that the lasting legacy of the Web has been a shift in the value digital technology from being primarily local to being primarily remote. The Web demonstrated that moving functionality online enables access to more compute power, continuous updates, real-time usage analytics, and (of course) social connections. It also created a shift in people's expectations. Today, most people understand that the experience you see on one device is often a part of something that's distributed throughout the world. There's no longer a need to pack everything into a single piece of software, and there's no expectation that everything will be there.

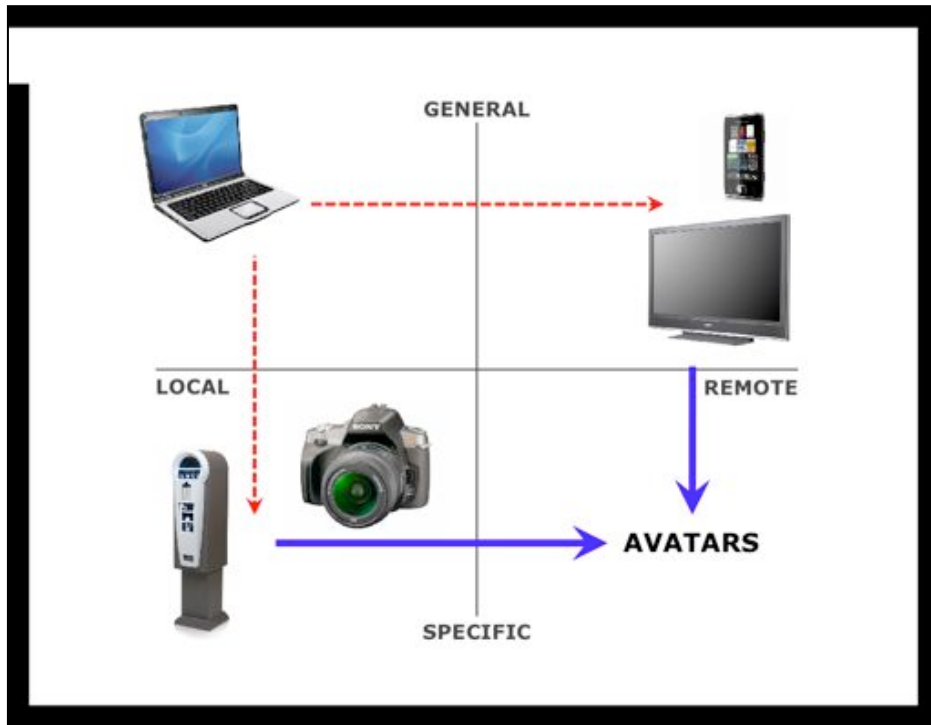


If we chart these last two trends, two broad classes of digital products emerge.

If we follow the general to specific axis, we see a shift is to more narrow-function devices that are designed to do a small, specific set of things really well. They primarily differ in what those specific things are. I call these devices appliances.

If we follow the local to remote axis, we find general-purpose devices that do roughly the same set of things, and differ primarily in size. They exist to provide access to online services, in a form factor that's appropriate to the context in which they're used.

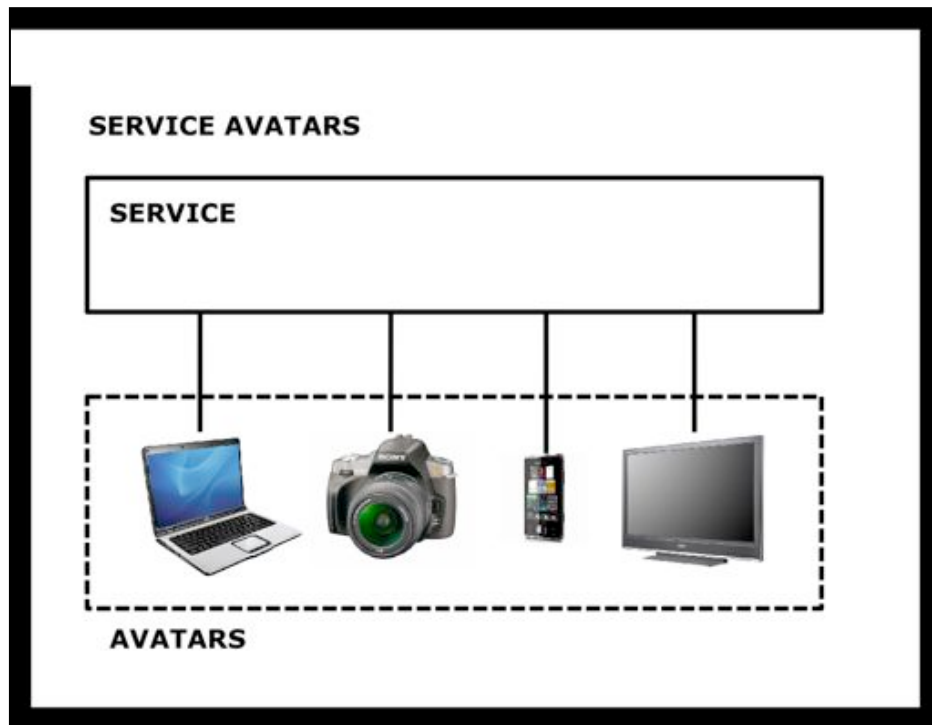
I call these devices terminals.



What I think is most interesting, however, is that these shifts appear to be the first part an even larger transition, one where devices are simultaneously specific AND deeply tied to online services. In this model, the service provides the majority of the value, and can be represented either as an inexpensive dedicated hardware device, an app running on a terminal, or anything in between.

It's an approach that combines the precision of appliances with the flexibility of terminals to create a fundamentally new class of products that can fill every possible niche where a service may be appropriate.

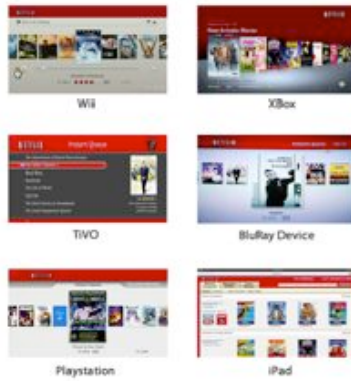
I call these devices service avatars.



As value shifts to services, the devices, software applications and websites used to access it—its avatars—become secondary. A camera becomes a really good appliance for taking photos for Flickr, while a TV becomes a nice Flickr display that you don't have to log into every time, and a phone becomes a convenient way to take your Flickr pictures on the road.

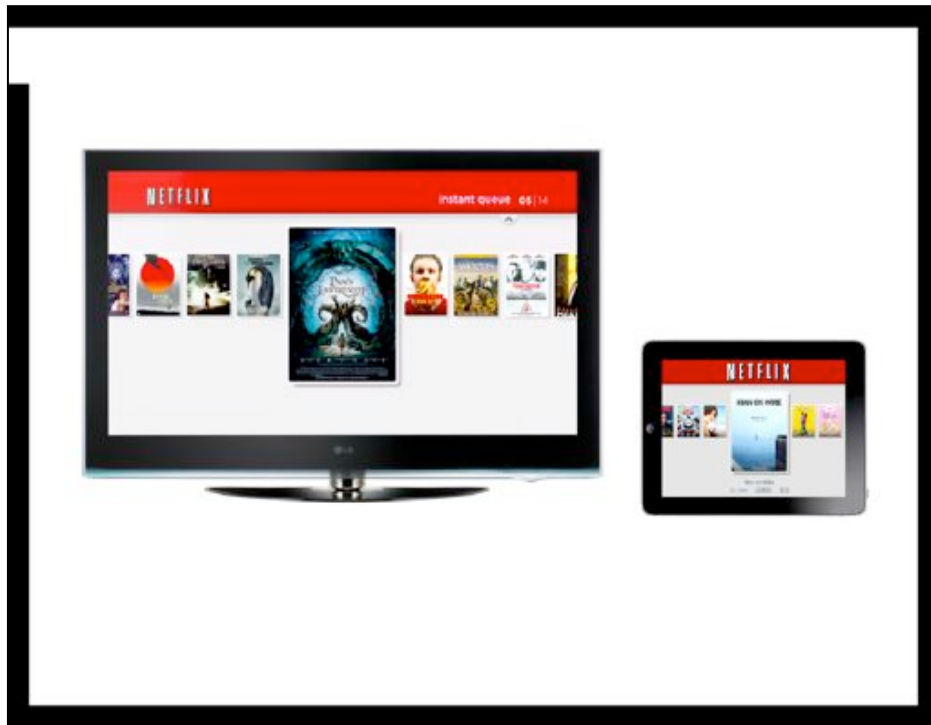
Hardware becomes simultaneously more specialized and devalued as users see “through” each device to the service it represents.

## EXAMPLE: NETFLIX



For example, you can now get Netflix on virtually any terminal that has a screen and a network connection. You can pause a Netflix movie on one terminal and then upause it on another. This seems natural. Why?





Because to the Netflix customer, any device used to watch a movie on Netflix is just a hole in space to the Netflix service. It's a short-term manifestation of a single service. The value, the brand loyalty, and the focus is on the service, not the frame around it. The technology exists to enable the service, not as an end to itself.



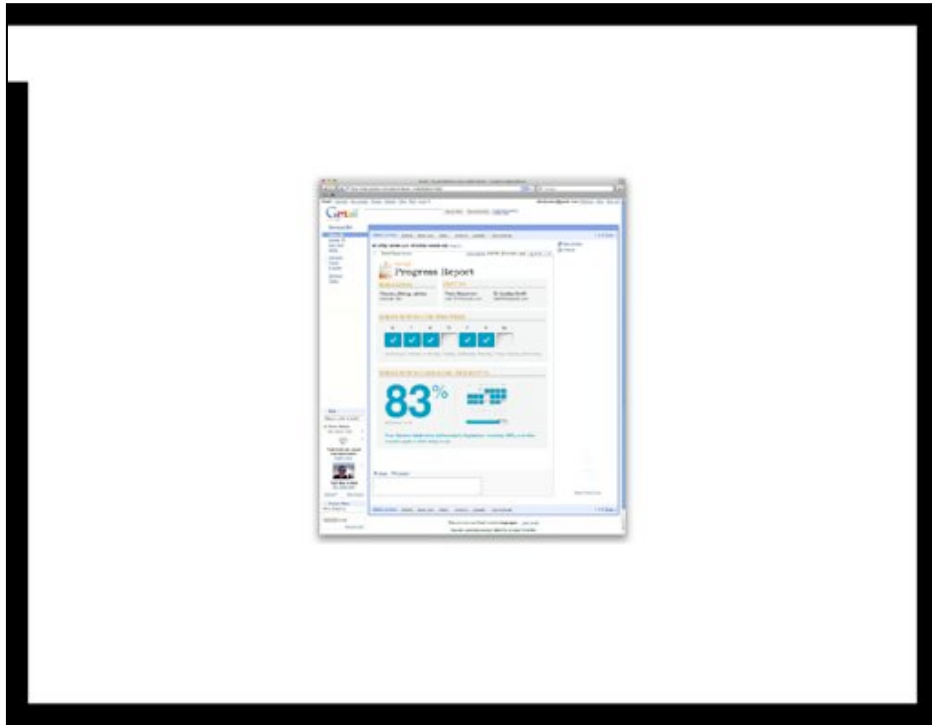
Netflix appliances are created for a single reason: to make it easier to access Netflix. That's what Roku does. It turns every terminal that's not already Netflix enabled into a Netflix terminal. The Boxee box does that for the Boxee service. The new Apple TV does it for iTunes.

## VITALITY

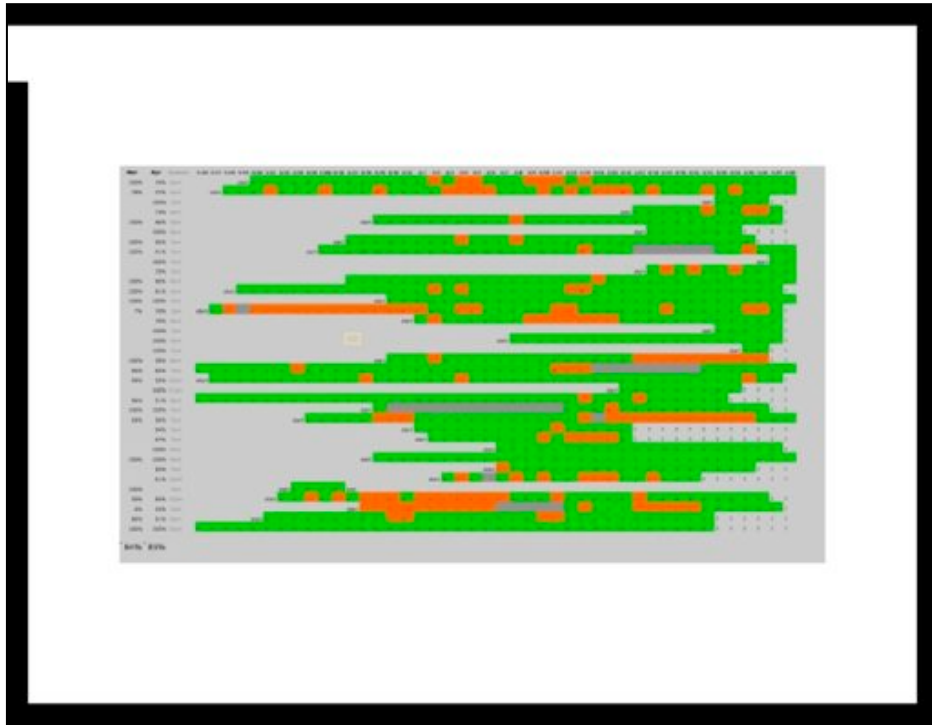


Let me give you another example. This is Vitality's Glowcap, which is a wireless network-connected pill bottle appliance that's an avatar to Vitality's service for increasing compliance to medicine prescriptions. When you close the cap, it sends a packet of information through a mobile phone-based base station to a central server and it starts counting down to when you next need to take your medicine. When it's time, it lights up the LED on the top of the bottle.

However, the real power is in the packet of data it sends. That packet opens a door to the full power of an Internet-based service. Now Vitality can create sophisticated experiences that transcend a single piece of software or a single device.

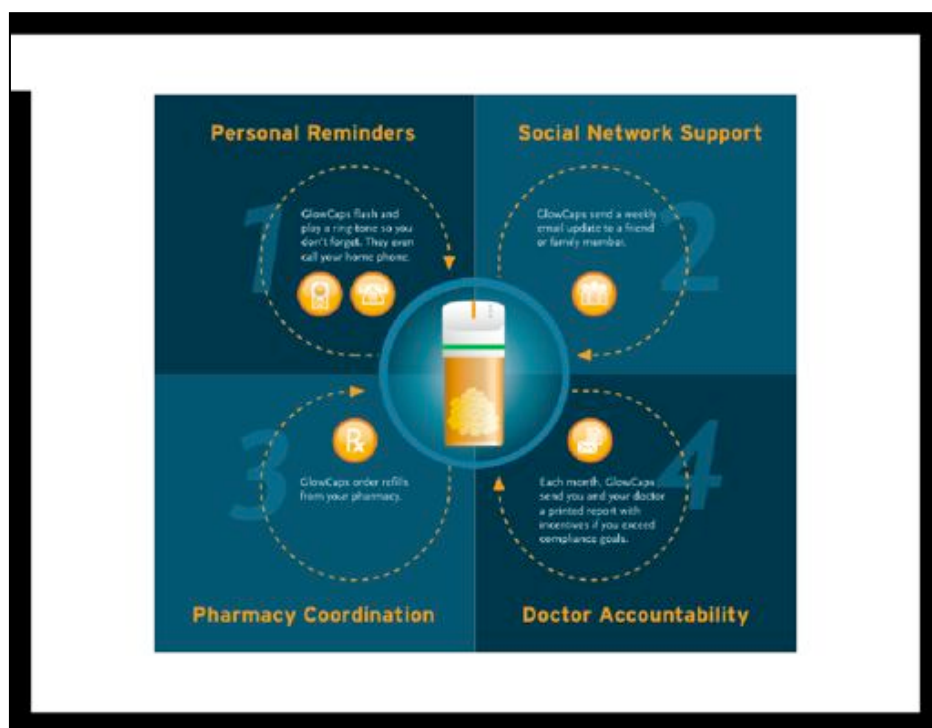


For example, another avatar of the Vitality service is an online progress report that can be used interactively or delivered by email. It's like Google Analytics for your medicine.



Health care practitioners get yet another avatar that gives them long-term and longitudinal analytics about compliance across medications and time.

To me, this kind of conversation between devices and net services is where the real power of The Internet of Things begins.



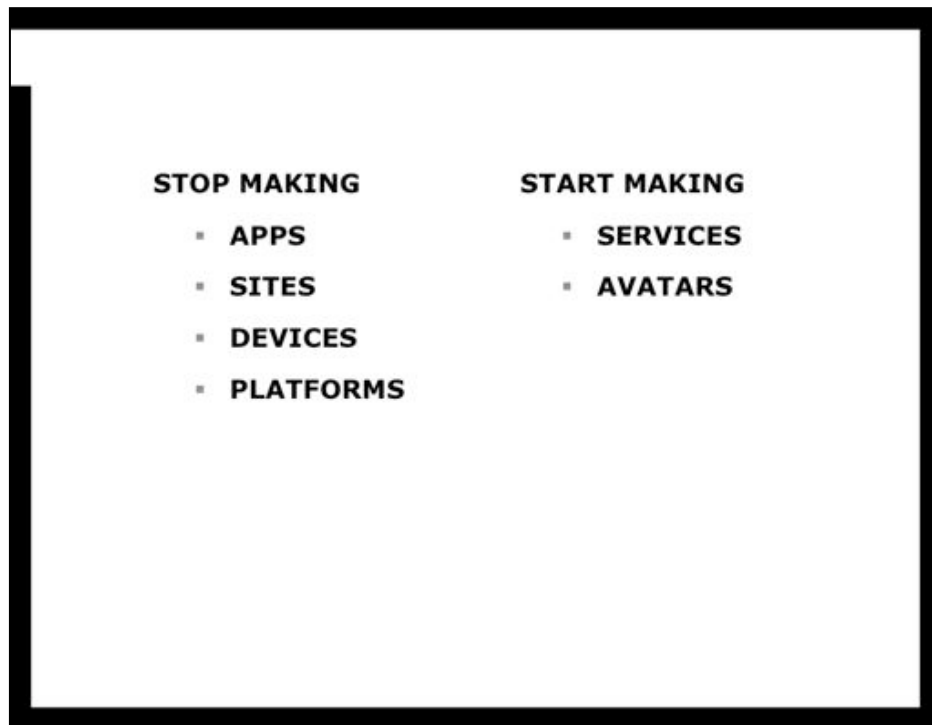
Vitality has developed a complete system around this service that includes a social component, and different avatars for patients, patients families, health care practitioners and pharmacies. Each avatar looks different and has different functionality, but they're perceived, and designed as a single system.



I think it's a model of how many everyday things are going to be designed in the future.

Soon designing objects that have significant social lives in the cloud will become just how everything is made.

Terminals, such as what we call smart phones or connected TVs, will of course be part of this world—they will be



So if you're a developer I recommend you shift your thinking away from whether to make an app, a mobile web site, a platform, or a dedicated device, and to start thinking about how you design your service, and what avatars will best facilitate that service.



## **SERVICE AVATAR OPERATING SYSTEM**

But what does the operating system that enables this kind of service avatar model look like? What is the infrastructure that makes a service avatar ecosystems functional and profitable? How does it incentivize developers to create services for it?

I made a list of the qualities of a service avatar operating system that I believe are important. And before I give you that list, I want to warn you: I'm not a mobile developer, and I'm not much of a mobile designer, but I've designed a lot of user experiences across a wide range of devices and have felt the frustration of infrastructures that prevent me from meeting my business goals, rather than supporting me as I try to build products that people will like and pay for.

So please excuse me if the following ideas are obvious, or wrong or obviously wrong, but I wanted to give you some things to think about, or argue over lunch as we contemplate the near future of digital technology.



The service avatar operating system I'm thinking about will need to support experiences that span devices as a core part of its services, on a layer that's largely invisible to users and developers, like providing network connectivity or memory management. Inter-device experiences should be part of the basic value proposition. Doing something like the Netflix pause, or how the Kindle service automatically opens to the last page you looked at, regardless of what avatar you last used, should be almost free. I should be able to take an experience on one avatar and throw it onto another one and have it just work.

The interaction techniques for this have existed for a long time. Jun Rekimoto of Sony had pick and drop--where you pick up a window from one device, "carry it" in a stylus that conceptually serves as a temporary container, and drop it on another device--working in Sony's labs in 1997. There are dozens of methods to create a single experience that transcends multiple devices, but they're all incredibly hard to implement, because they have to be done from scratch. This OS needs to make it so easy to create such experiences that you forget about it.

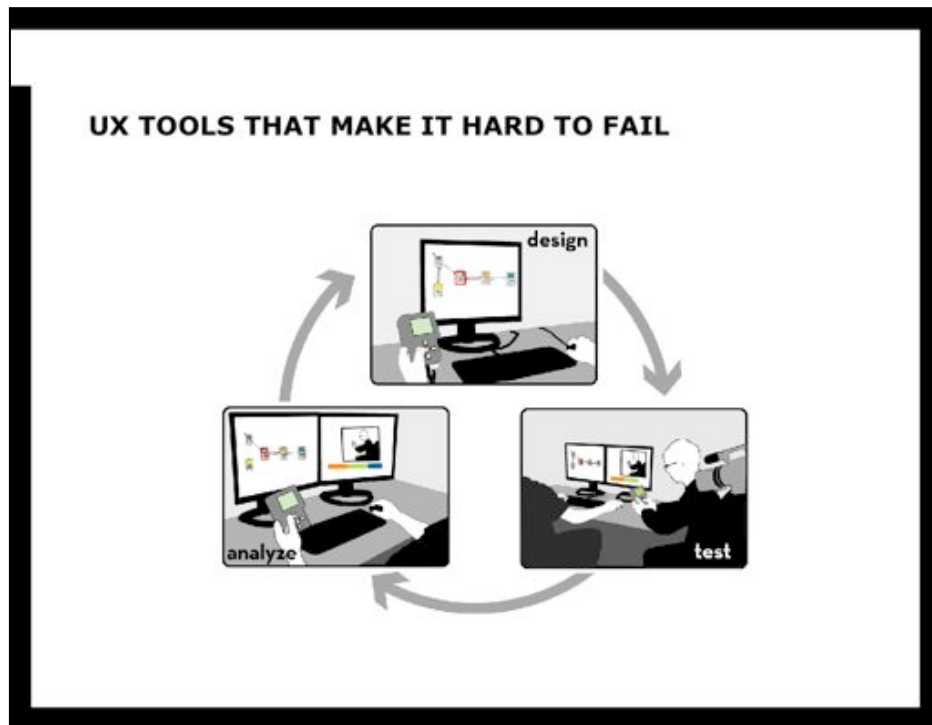
## IDENTITY AND CONTROL SHARING



As the number of devices and services increases, so does the number of people using each device or service. Problems of identity and control appear. This service avatar operating system will make it very clear about who is in control of what avatar when, how much of their personal information is being utilized at the moment, and how to shift the locus of control from one person to another. Think of how the locus of control shifts in a business meeting, or friends talking to each other on the subway.

Video games are the model here. They have been multiuser experiences for a long time. One thing to learn from video games is that this is not a n-person problem. We're not talking making systems that can scale from one person to a stadium at the blink of an eye. Instead this operating system should make it easy for, say, five people to have an experience together using a variety of in-person avatars, and perhaps a couple of dozen folks peripherally participating online.

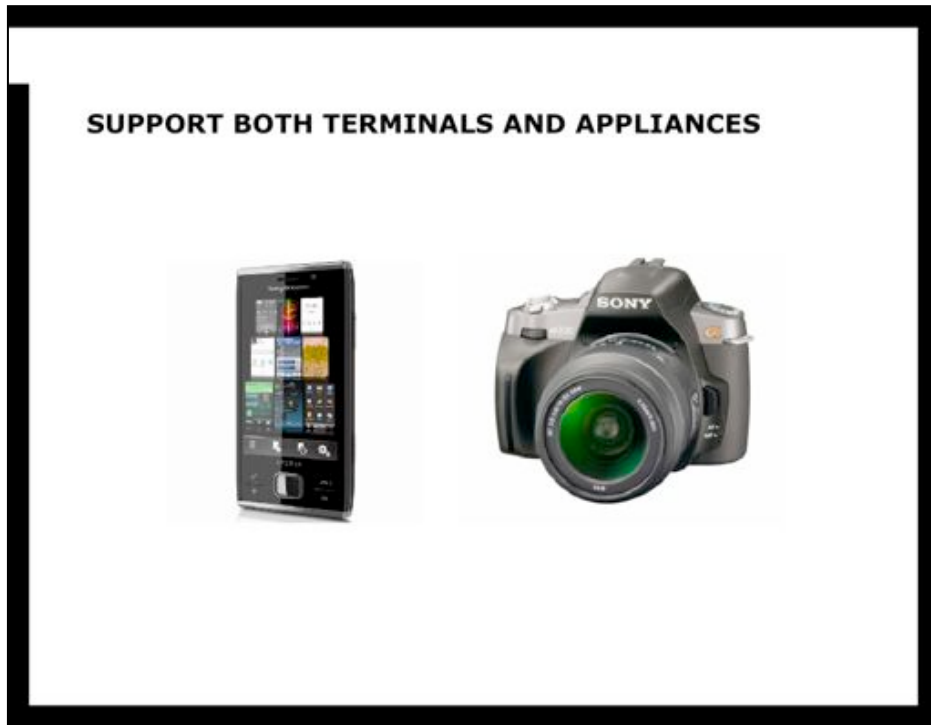
This operating system will make it easy to create such experiences by taking care of synchronizing everyone's avatars and managing identity. Smart phones, or whatever the device is that smart phones become, play a particularly important role here as proxies for identity. As personal terminals, they're the perfect locus for doing all of this identity management, even if they're not the primary avatar that the group interacts with.



Most digital experiences are designed by the people coding them. Interaction designers are few and typically overworked. A service avatar operating system will have to make it very easy for engineers to create avatar user experiences that don't suck. These experiences don't have to be super awesome, but it has to be easy to make an OK one without really thinking hard about UX design.

One way to do this is through the application of draconian standards and to have an enforcement mechanism. The other way is to make tools that make it easier to do the right thing than to do the wrong thing. Developers know what a good interface is, but they may not have the time to design an elegant one from scratch. Good, well-constrained, design tools will both enforce good experience design hygiene and describe what that is.

For service avatars this is more than just creating a widget library and a layout tool. It means creating tools that allow developers to explore experiences as they are lived in all avatars, rather than just how the pixels are laid out on one terminal or another. This is a slide of d.tools, a project that was developed at Stanford's d.school. It integrates in a single environment a hardware prototyping and programming environment, usability testing that synchronizes people's actions with video of their actions, plus a facility for analyzing the relationship between the two. It also has a great learn-by-doing system. You can, in effect, say "When I hold the device like this, and I hold this button while shaking it like this...that's how I want to trigger some effect." I think it really points the way to what a future ubiquitous computing development environment looks like.



My imagined service avatar operating system supports both terminals and avatars using essentially the same code. And I don't mean this in an abstract "hey, it has a linux kernel" way. I mean that from the developer's perspective, the same environment can be used to develop for a wide variety of devices, and the operating system abstracts the vast majority of differences away, leaving only the appliance-specific functions, which should be easy to identify.

A developer should be able to quickly deploy a new service idea on a terminal platform, say, a phone or a TV, and then, if it takes off, deploy a specialized avatar that does that one thing particularly well. Say my prosumer photo sharing app becomes a big hit; I should be able to make a deal with Sony to deploy a cobranded version of their high resolution camera that will natively runs my app and be able to release it nine months later because it uses most of the same technology as the app running on my phone. Yeah, sure, the camera also work like a smartphone, but who cares? Pretty soon anything will be able to run anything, that's to be assumed. The question is: what is it good for?

We have the technology and infrastructure for this. What's missing is an easy way to tie to together.

## RAPID ITERATION



I'm a big fan of fast failure. The faster you can make an idea fail, the faster you move on to the next idea, which will hopefully fail less fast. Eventually you get ideas that take a really long time to fail. We tend to call those successes.

In today's technology marketplace, developers should be able to succeed or fail quickly. This requires an ecosystem where new service and avatar ideas can be quickly developed, quickly deployed, easily evaluated and rapidly updated. A service avatar operating system has iteration of business models a core part of what it delivers. This is what we have learned from the success of Web 2.0 startups: fast development, deployment and usage analytics are the key to identifying the key combination of designs and features that represent market success. People talk about what pieces make up the magic of successful products and companies: is it the social graph, is it the technology, is it the design? I think that in most cases it's the ability to rapidly iterate on a business model.

This, by the way, is an image of a project by a British furniture designer [NAME?] who spent a year making one chair per day out of other chairs he found on the street.

## **MICROTRANSACTIONS**



In a service world, microtransactions are king. The current app market is flooded and apps that are getting cheaper all the time. That does not bode well for most app developers because acquiring new users becomes increasingly expensive while per-user value stays the same or falls. It's a losing game. I think this create a significant opportunity for an operating system that wants to compete with the two app-based OSes in terms of attracting developers.

If you look at a traditional service, such as a credit card, the basic service is free, but the ongoing revenue stream of small transactions creates a revenue stream that's not based on new user acquisition. Microtransaction management at the operating system level opens up the practicality of services that have exponentially growing revenue with a linear customer acquisition effort.

## ECOSYSTEM OF SERVICES



What all of this adds up to is an ecosystem where services are the key creators of value for developers. The operating system provides the necessary infrastructure for developers to build services on top of. It does not build the services for them. It serves no map tiles, or address books. Those are all services that it enables someone to build and charge for. It primarily exists to enable the easy creation, deployment, monetization and interoperation of such services and their avatars.

This model shifts the burden of identifying user needs and meeting those needs to developers, and rewards them with continuous sources of revenue.

Ultimately this shift to services and avatars is being driven by technology and people's changing use of it, but for an operating system to be successful it only needs to do one thing: it needs to create a great user experience for its developers. That means giving them great developer tools, lucrative business models and then getting out of their way.





**Mike Kuniavsky**  
mikek@thingm.com



Thank you.