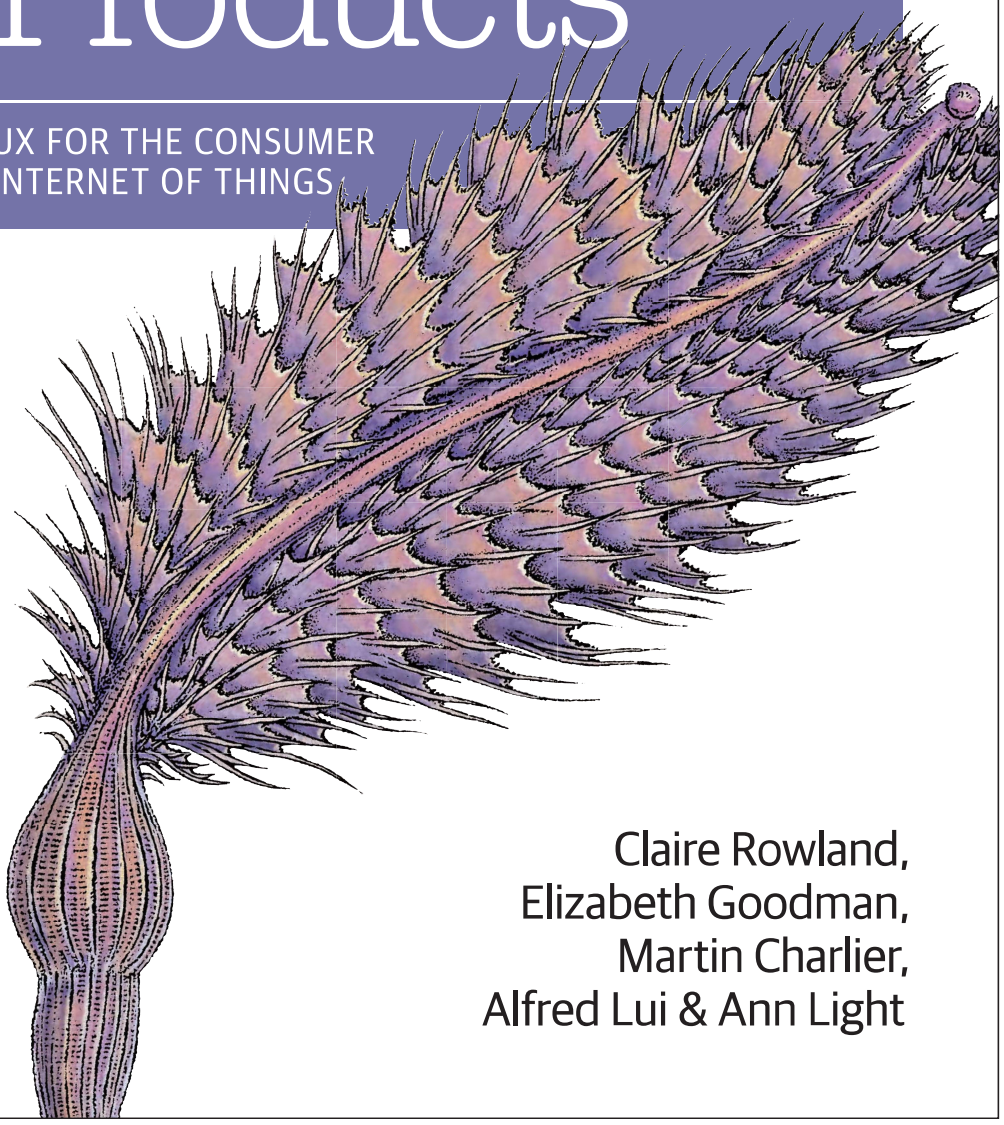
O'REILLY®

Designing Connected Products

UX FOR THE CONSUMER INTERNET OF THINGS



Claire Rowland,
Elizabeth Goodman,
Martin Charlier,
Alfred Lui & Ann Light

Designing for Connected Products: UX for the Consumer Internet of Things

4

Product/service definition and strategy

Claire Rowland

Introduction

We all aspire to create the killer product or service that people want to buy and love using. The key to this is ensuring that the product solves an actual problem that people have, in a way that appeals to them. At a pinch, it might provide them with something new and wonderful that they never knew they needed. It sounds simple and obvious, but it can be remarkably difficult to get this right. Right now, the IoT market is skewed towards innovators and early adopters. There's huge potential to create great new products for consumers, but they may have to contend with new types of complexity.

This chapter introduces:

- Productization as part of IoT design *(see page 2)*.
- What makes a good product for different audiences *(see page 7)*.
- How products differ from tools *(see page 14)*.
- What makes a good product *(see page 19)*.
- Building service offerings around products *(see page 28)*.
- Business models in IoT *(see page 35)*.

This chapter addresses the following issues:

- Why a clear value proposition is a prerequisite to great UX design *(see page 4)*.
- Why products designed by and for innovators aren't necessarily right for general consumers *(see page 7)*.
- Why consumers want products, not tools *(see page 14)*.
- Why it's important to design the service offering around a product *(see page 33)*.
- How business models can shape UX *(see page 35)*.
- How digital business models may start to appear in real world products *(see page 37)*.

Making good products

What is productization?

Productization is the extent to which the supplier makes the user value of the product explicit and easy to understand. Compelling products don't just look good or otherwise fuel some underlying need for status (although those things are often important). They make it immediately apparent to their intended audience that they do a thing of real value for them: preferably something new than serves a previously unmet need.

Nest is probably the most famous IoT productization success story. Consumers were resigned to thermostats and smoke alarms being ugly, annoying boxes with usability flaws. It hadn't occurred to most people that they could be better. Nest products promise to do the job better than most of the competition, in the form of attractive and desirable hardware that users are happy to have on show at home *(see figure 4.1)*. Of course, they are premium products with a premium price tag. The point here is not that all products should be expensive, but that a good product should fulfill a clear need for the target audience, with a usable and appealing design. **This is the product's *value proposition*: the user's understanding of what the product does for them and why they might want it.**

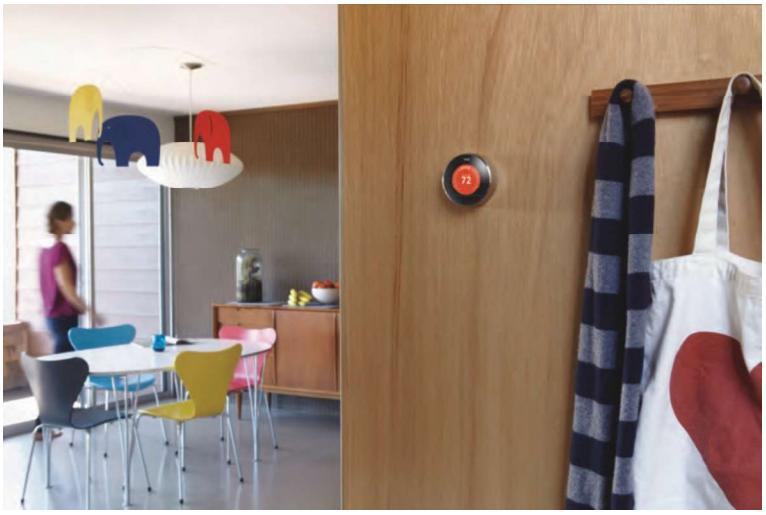


Figure 4.1: Nest thermostat shown in home (image: Nest)

"Never underestimate the power of a simple explanation, or a product that looks nice. If people can understand it, they can want one for themselves. They're not scared of it. It stops being a weird thing that geeks do." Denise Wilton, designer (and former creative director of design agency BERG)[1]

Products can be services

When we talk about IoT, we tend to focus on the edge devices: the activity monitors, thermostats, connected pet feeders, and more. This is especially true when the devices themselves look novel (such as the Nabaztag rabbit shown in chapter 2) or striking (such as the Nest thermostat).

But while the devices are a key part of the UX, they are not the whole picture. They are all dependent on an internet service. This makes the user's relationship with the product much more dynamic. Instead of the traditional one-off purchase of a traditional physical product, the user interacts with the provider on an ongoing basis. The user's experience isn't just shaped by the device, it's shaped by the whole service. There might not even be a physical product at all: just as you can now pay for Dropbox storage or personal fitness

[1] From a talk at UX Brighton, November 2012

training, so you may pay for software or storage to help you make the most of connected devices, or personalized health or energy saving advice based around data gathered from your devices.

Author's note: In this book, we use the term 'product' loosely to refer to a packaged set of functionalities that solves a problem for people or fits neatly into their lives. That could be a physical device, a service, or frequently a combination of both.

Why is this in a UX book?

To some of you, this may seem outside the remit you normally associate with UX design. You may work in a company where productization is handled by product management, or perhaps marketing. In others, it might be considered strategic design. UX is not always involved in identifying the opportunity and framing the solution. But most UX designers would walk over hot coals to be involved from the start, especially if they have first hand knowledge of user needs from conducting research.

Whoever is responsible for it in your organization, it provides the strategic foundation for UX design. **It's not possible to design a great product or service experience if users don't want, or understand, the service in the first place.**

Value propositions help sell products. But they also drive UX. A clear proposition helps users decide whether to buy it in the first place, but also helps frame their mental model of the system and what it does *(see figure 4.2)*. When users are confident that they understand what the system does for them, they have a good basis for figuring out how it works (the conceptual model), and then how to use it (the interaction model). All the clever design in the world can't overcome a murky or unappealing value proposition.

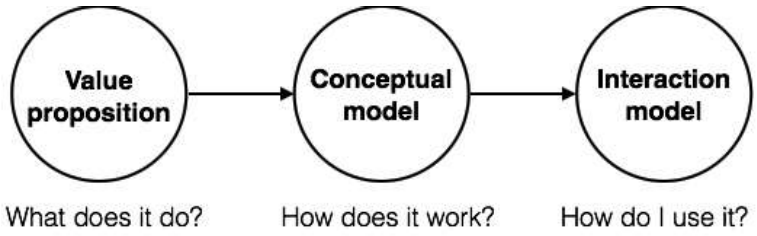


Figure 4.2: A good clear value proposition is fundamental to a great UX.

Why is this in an *IoT* book?

Productization is of course not a challenge that is unique to IoT. It is included in this book as it is a particular challenge for the consumer IoT field right now. Many products and services aren't yet offering good, practical solutions for proven consumer problems. Even where they are, the value isn't always apparent from the product itself or clearly stated in terms target users would understand.

This isn't a criticism of the many clever and talented people working in this field. Most of them are aware that consumer experience is a challenge.

It's a result of the novelty and inherent complexity of the products and services. We're still figuring out what we can do with the technology, and we're asking users to wrap their brains around some novel devices and capabilities.

It also reflects that new technology products and services are often conceived and developed by people with an engineering mindset who value highly configurable functionality. These initiatives can often seem complex and unclear in purpose to consumers because, in trying to do so much, they fail to communicate a clear value for using the service.

There is, of course, a market for products developed to meet the needs of highly technical users. There's also great value in products and services that help a wider range of people move beyond passive consumption of technology and learn how to construct their own solutions. For example, *If This Then That* offers an accessible way to coordinate different web services and even connected devices *(see figure 4.3)*. This is functionality that would previously only have been available to those with good programming skills.

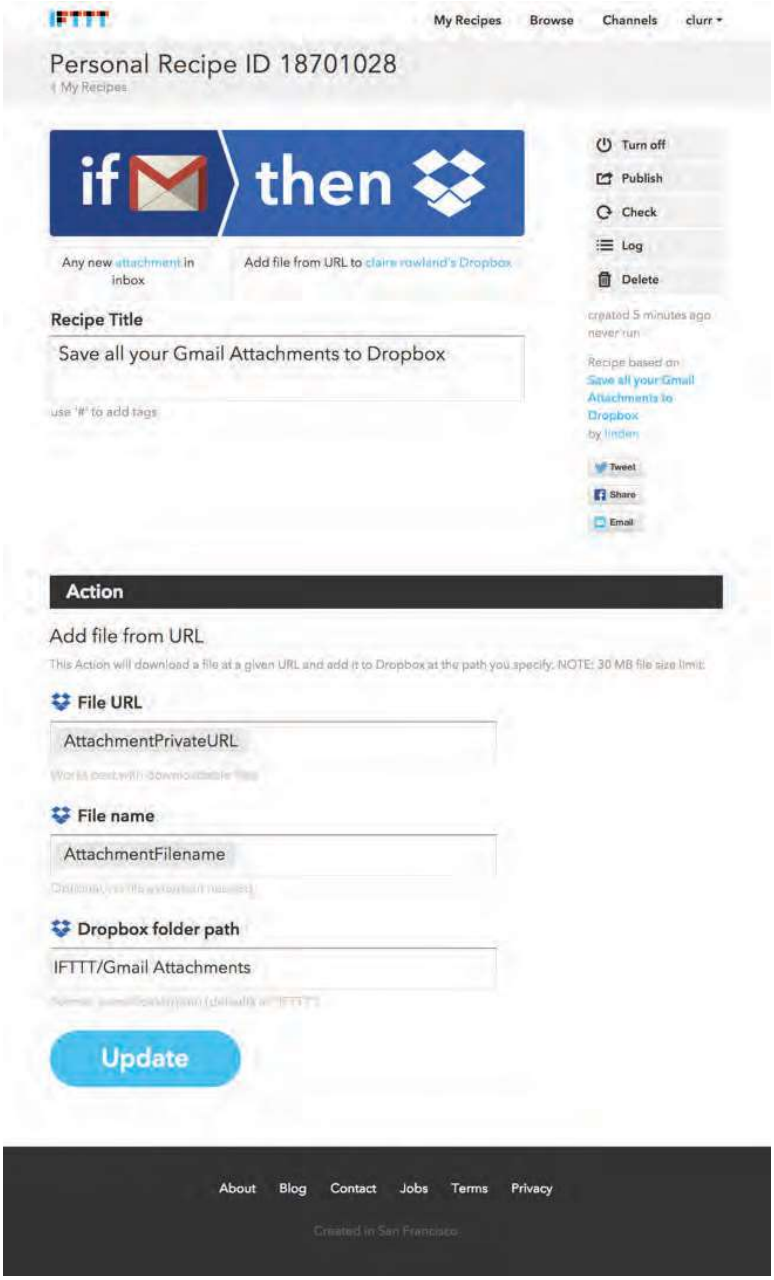


Figure 4.3: An If This Then That recipe for saving Gmail attachments to Dropbox

But the bigger challenge is in creating products and services that work for mass-market consumers. For this audience, the functionality – what the system does and how to use it - should be transparent. The underlying technology should be invisible. The user should be able to focus on getting the benefit from the product that they were promised, not on configuring it and maintaining it.

From innovation to mass market

The primary focus of this book is on creating consumer IoT products and services. In this section, we take a brief look at how technological innovations cross over into the mass market and consider what lessons there may be in here for IoT.

Innovators are not consumers

In 1962, the sociologist Everett Rogers introduced the idea of the technology lifecycle adoption curve, based on studies in agriculture[2]. Rogers proposed that technologies are adopted in successive phases by different audience groups, based on a bell curve (see figure 4.4). This theory has gained wide traction in the technology industry. Successive thinkers have built upon it, such as the organizational consultant Geoffrey Moore in his book 'Crossing the Chasm'[3].

In Rogers's model, the early market for a product is composed of innovators (or technology enthusiasts) and early adopters. These people are inherently interested in the technology and willing to invest a lot of effort in getting the product to work for them. Innovators, especially, may be willing to accept a product with flaws as long as it represents a significant or interesting new idea.

The next two groups - the early and late majority - represent the mainstream market. Early majority users may take a chance on a new product if they have seen it used successfully by others whom they know personally. Late majority users are skeptical and will adopt a product only after seeing that the majority

[2] Everett M Rogers, 2003, 'Diffusion of Innovations' (5th edition), Simon & Schuster.

[3] Geoffrey Moore, 1991, 'Crossing the Chasm', HarperBusiness.

of other people are already doing so. Both groups are primarily interested in what the product can do for them, unwilling to invest significant time or effort in getting it to work, and intolerant of flaws. Different individuals can be in different groups for different types of product. A consumer could be an early adopter of video game consoles, but a late majority customer for microwave ovens.

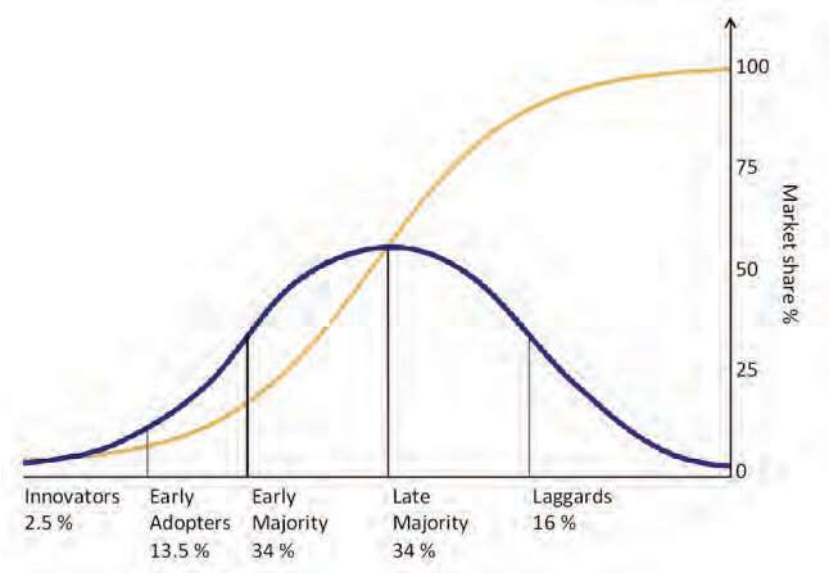


Figure 4.4: The diffusion of innovations according to Everett Rogers. The blue line represents the successive groups adopting the technology, the yellow line the market share (Image: Tungsten, via Wikicommons).

Geoffrey Moore identified a 'chasm' between the early adopter and early majority market (which he called visionaries and pragmatists). These groups have different needs and different buying habits. Mainstream customers don't buy products for the same reasons as early adopters. They don't perceive early adopters as having the same needs as themselves. Mainstream customers may be aware that early adopters are using the product. But this will not convince them to try it out themselves unless they see it as meeting their own, different, needs. So products can be successful with an early market, yet fail to find a mainstream audience.

An example of this in the IoT space is the home automation market. Systems such as those based on the power line protocol X10 have been around for close to 40 years. (Early examples ran over electrical power lines and analogue phone lines). The example in figure 4.5, from 1986, shows a system that allowed users to program and remotely control their heating, lighting and appliances over a (landline) phone. These are all applications that still seem novel and innovative to us; they would have excited the innovators of the 1980s even more.

Figure 4.5: Advertisement for X10 Powerhouse for the Commodore 64, from the January 1986 edition of Compute! Magazine (image via commodore.ca).

However, home automation remained a niche market. It was expensive. It required significant technical skill to set up and maintain. Even those mainstream consumers who had heard of home automation did not see much value in programming their heating, lighting and appliances. Had it been more affordable or easier to use, more people might have been willing to try it out. But only now are consumers starting to see the utility of connected home products. This is arguably driven by the rise of the smartphone, giving us a metaphor for the 'remote control for your life'.

What's different about consumers?

Mainstream consumers are now more aware of connected devices, but they need to be convinced that these products will actually do something valuable for them. A product that appeals to an audience that loves technology for its own sake cannot simply be made easier to use or better looking. To appeal to a mass-market audience, it may need to serve a different set of needs with a different value proposition. Chapter 5, Understanding Users, covers learning about user needs and some of the special considerations you might encounter when designing for IoT.

Mass-market product propositions have to spell out the value very clearly. Users will be subconsciously trying to estimate the benefit they'd get from your product as offset by the cost/effort involved in acquiring, setting up and using it, and you need to be realistic about the amount of effort they will be prepared to invest in your product. The further along the curve they are, the more users need products with a clear and specific value proposition, which require little effort to understand or use. And they have a very low tolerance for unreliability. Your product has made a promise to do something for them, and it must deliver on that promise.

This is not simply a question of lacking technical knowledge, and certainly not of users being dumb. That 10-step configuration process to set the heating schedule might seem trivial in the context of your single product. But it can feel overwhelmingly complex in the context of a busy life with many other more pressing concerns. For this reason, consumers tend to be most attracted by products that seem as if they will fit into their existing patterns of behavior and don't require extra effort. For example, ATM cards and mobile phones

were arguably successful because they reduced the need to plan ahead in daily activities (getting cash from the bank, or arranging to meet).

Value propositions for IoT

The guidelines above can of course be applied to any type of product or service. But connected products can be complex and often do novel things that are hard to communicate succinctly.

Core value propositions should be straightforward, e.g. a company offering smart meters may promise to "tell you where your energy spend is going", which is relatively simple. A good test of an IoT product proposition is that end users should not need to focus on its connectivity or onboard computing: it should just make sense.

But there may be complicating factors that users need to understand before buying. You may have to explain which other systems can interoperate with yours, or who owns the user's data and what they can do with it. (The technology and value of interoperability is discussed in further detail in chapter 10). You might have to guarantee how far into the future you will maintain the internet service (if your company is acquired, goes bust or discontinues the product).

The entrepreneur and academic Steve Blank describes 4 types of market in which a product can operate[4] (see figure 4-6). The type of market influences how you position the value of your product. Below, we look at what this might mean for IoT products:

[4] Steve Blank, 2005: "The Four Steps to the Epiphany: Successful Strategies for Products that Win" (K&S Ranch Press)

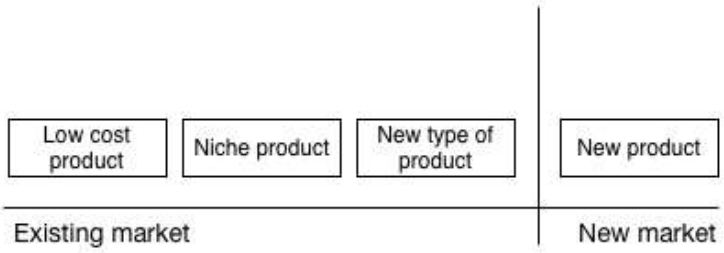


Figure 4-6: 4 types of market in which a product can operate

A new product in a new market

Embedded connectivity and intelligence will fuel the appearance of new classes of product and new markets. In consumer terms, the challenge is often to convince users of your vision. You have solved a problem they didn't realize they had, or had just accepted as 'the way things were'. The Glowcaps pill bottle top, mentioned in chapter 2, reminds users to take their medication and helps the patient's doctor track how frequently it is taken.

A new type of product in an existing market

Here, the challenge is to convince users that your product is the best solution to the problem. Perhaps it has better features or better performance. In IoT, these products may be familiar physical devices newly enhanced with sensing or connectivity (e.g. the Withings bathroom scales). Users need to understand the value that is added by the enhancements, such as easier weight tracking. They need to decide whether it's something they want, especially if it costs extra.

It might also be a technology that offers a step change in experience design. For example, airport terminals can be large and confusing. You would normally rely on signage to find your way around, but this isn't always clear, consistent or guaranteed to tell you what you need as and when you need it. You don't want to miss your flight, but nor do you want to end up sitting around at the gate for too long because you were cautious and got there too early. Apple's iBeacons technology (described in chapter 2) offers precise indoor location. Several airlines have been trialing the use of iBeacons to provide passengers with in-context information and directions (see figure 4.7).

Passengers can be directed to the correct gate more easily, based on their current location in the airport. If they are running late but are very close to the gate, knowing their location might help the crew decide to wait. And if their plane is delayed, the app could provide them with a voucher to a nearby restaurant or café.

Figure 4.7: Illustration of an airport iBeacon trial (Image: SITA).

A low cost entrant to an existing market

The falling cost of embedded computing enables cheaper alternatives to systems that used to be prohibitively expensive. For example, Lowes Iris (see figure 4.8) and Smart Things offer DIY home automation kits at a far lower cost than professionally installed systems. You may be aiming the system at people who could not previously afford this category of device, or trying to convince those who could that you're offering a worthwhile saving. Either way, it's important to convince users that the system performs the basic functionality just as well as more expensive options. Any compromise needs to be something that doesn't matter too much. You need to be clear upfront how you have achieved the cost saving: is the hardware cheaper? Does the system

involve more work from the user (e.g. DIY setup)? Does it provide them with less personal (e.g. automated or lower bandwidth) customer service?

Figure 4.8: Lowes Iris Safe and Secure DIY home security kit (hub, motion sensor, two contact sensors, alarm keypad). (Image: Lowes).

A niche entrant to an existing market

Augmenting an existing product type with connectivity and potentially intelligence can create opportunities to address previously unmet user needs in an existing market. It may target a niche with specialist interests: for example, an energy monitoring system designed for those who generate their own electricity and may sell it back to the grid. Or it may introduce a premium product for those willing to pay more. The Nest thermostat offered the first intelligent heating solution with high-end hardware and polished UX design in a market previously dominated by ugly, unusable plastic boxes. This reshaped consumer expectations of what a heating controller could be, even in the part of the market that couldn't or didn't want to pay extra for a Nest.

Tools vs. products

For some specific connected devices, like a heating controller, there's a close mapping between function and value. It's easy for people to understand what it does. That's not enough to make it a *good* heating controller. But it's pretty

clear what it does, and why you might want it. It will keep the house at a comfortable temperature and, perhaps, save money. Devices that are enhanced versions of pre-existing product types, like bathroom scales or baby monitors, have the advantage of being recognizable as things that meet a defined, familiar set of needs. You may have to convince customers as to why that product benefits from connectivity. And you may have to address concerns they have about adding connectivity or technical complexity to the product, such as security, privacy or usability. But at least the product is familiar.

Mass-market consumers, in areas in which they do not have deep technical or domain knowledge, generally expect a product to come designed and engineered to fulfill a specific need. The Nest Protect smoke detector and carbon monoxide alarm is a good example of a *product*[5]. The marketing website focuses on the ways in which it is a better safety alarm (see figure 4.9). Connectivity is only mentioned at the end, to say you'll be alerted on your phone if there's a problem when you're away from home.

[5] Nest Protect has suffered from some interaction design problems. A Heads Up feature originally allowed users to disable false alarms (such as those caused by burnt toast) by waving at the alarm. But no-one had thought that, in the case of a genuine fire, users might also wave their arms (in panic). The alarm was therefore too easy to disable. Units were recalled and Heads Up was deactivated. But the Protect is still a good example of a clear product concept.

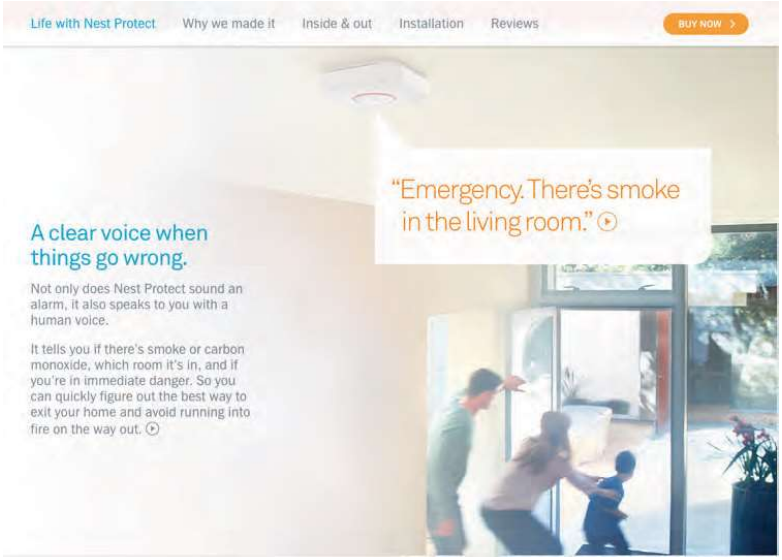


Figure 4.9: Excerpt from the Nest Protect marketing website. (Image: Nest)

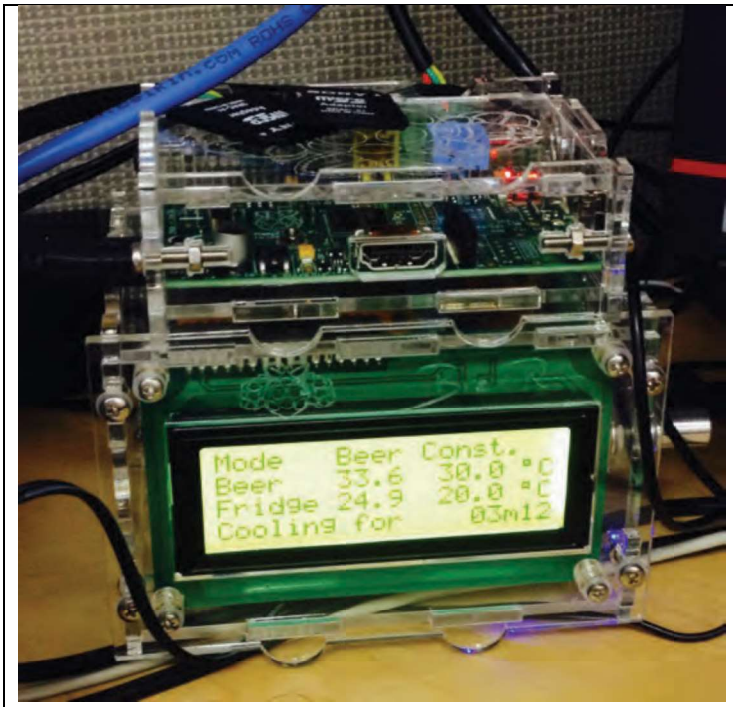
But many IoT services and devices can be configured to meet all kinds of needs. The onus is on the user to define their own needs and configure the device (or service) to achieve them. These are not products, but *tools*. Tools are often general-purpose devices, such as contact or motion sensors. The device has no inherent value to the user. The value comes when they are applied to solve a particular need, such as detecting intruders in the home, or warning you that you left a window open.

The Belkin WeMo smart plug (see figure 4.10) is a tool. It can be used to turn power to any appliance on and off remotely from a smartphone, or using an automated schedule. But it's up to the user to define their own problem, realize that a smart plug could help, and configure it to solve the problem. An imaginative leap is required. In reality, many smart plugs end up being used on lamps. In our own research, users struggled to think of other uses for them (although ensuring hair straighteners/curling tongs were turned off was popular).

Figure 4.10: WeMo smart plug and app

Services can be tools as well. The aforementioned If This Then That (which can also be used to control WeMo smart plugs), aims to make it easier for non-technical users to link up and program devices and services.

Tools aren't bad. They can be very powerful for users with technical or domain knowledge. Users who have the time and motivation to configure a system to meet their own very specific needs and aren't daunted by the need to learn the system may really enjoy this process. This could be the home brewer who enjoys rigging his or her own fermentation chamber out of an old fridge (see figure 4.11). Or a horticulturalist might be motivated to learn about the technology to configure a remotely controlled plant monitoring, watering and feeding system. Tools give us the possibility to be creative and take control of our environment.



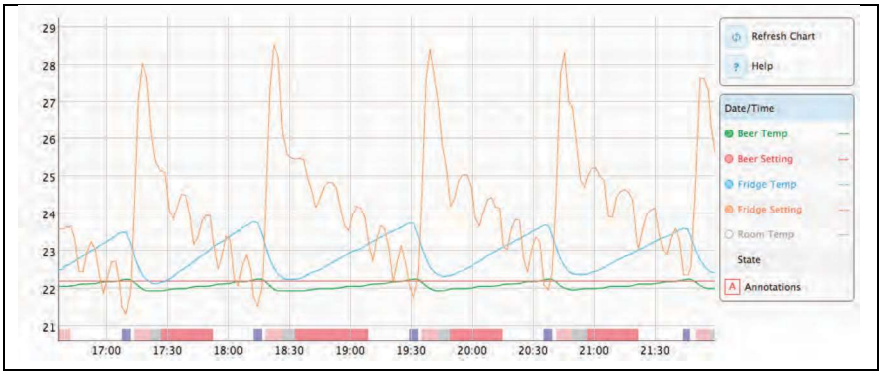


Figure 4.11: BrewPi is a fermentation temperature controller for brewing beer or wine. Running on a Raspberry Pi computer and Arduino[6], it comes with a kit to convert a standard home fridge or freezer into a fermentation chamber and is controllable via a web interface. (Images: Anthony Plunkett).

The IoT market, to date, has tended to create tools for innovators and early adopters. In an immature market that is exploring possibilities, that's fine. But it has tended to assume that the way to reach a mass audience is to make better-designed tools.

You can't turn a tool into a million-selling product just by making it usable. The WeMo plug comes with a well-designed smartphone app that walks users through the setup process fairly clearly and makes it easy to set up rules to control the plug. But the onus is still on the user to use the plug creatively. It's not actually the *plug* they want to control: it's the appliance. Controllable plugs are simply a first step in the journey towards controllable appliances.

In spring 2014, WeMo released a controllable appliance: the WeMo Crock Pot slow cooker (see figure 4.12). This allows the user to control the temperature and cooking time of a Crock Pot remotely from a smartphone app. Slow cookers might not be for everyone, but the context of use is a perfect fit for connectivity and remote control. Their value proposition is convenience: the meal that cooks itself while you're out all day. Remote control increases that convenience by allowing you to adjust the timing if you're home late. And

[6] At the time of writing the Arduino model is being phased out for a newer version based on the Spark Core development board.

being able to keep an eye on the device alleviates any anxiety about leaving a hot thing unattended in an empty house. It may be a niche appliance, but it's a well-formed product solution.

Figure 4.12: WeMo Crock Pot and smartphone app.

Mass-market consumers don't necessarily lack the knowledge, skill or imagination to solve their own problems. They may be perfectly capable of doing so but simply lack the time or have other priorities. At best they might only have time to solve a few of them.
There is a rich market for products that solve their problems for them!

What makes a good product?

Good products seem to appeal to common sense, and new good products are often greeted with the reaction 'well why didn't someone think of that before?'. But developing good products can be far harder than our 20/20 hindsight might lead us to think. This section looks at the general qualities of a good consumer product before considering what features come with IoT.

The product solves a real problem people have (and makes this clear)

Most products are acquired in order to solve a problem for the user. A good definition of the problem, and the audience, are essential to creating a clear value proposition. This is the definition of what your product does for people, and why they might want it.

A clearly communicated value proposition is fundamental to user experience. When people come across a product (or service), they try to form a quick judgment about its purpose, and whom it is for. If it's not immediately clear what the value proposition is, it may be dismissed: either because it is too hard to figure out, or because it does not appear to do anything of value for that person at that time. Worse, potential users may wrongly assume it is able to fulfill a purpose for which it is not really suited and waste time and/or money on a fruitless endeavor. (You may be happy to take their money in the short

term, but over time too many unhappy customers will damage your reputation!).

It's all too easy to end up with a poor or unclear value proposition despite good intentions. This is often the result of failing to identify the right problem for the right audience. You might have added features to show off what the system can do, or because they are simple to build, dictated too much by the capabilities of the technology at the expense of the original purpose and user needs. Or maybe there are competing interests involved in feature scoping. It's common for systems to try to do lots of things. That may create a great tool for early adopters who like to tinker and customise, but it risks muddying the value proposition for a mass-market audience. Imagine you're making a wrist-top device for outdoor pursuits like hiking or climbing. The core features are an altimeter, barometer, compass, and perhaps GPS. It might be quite straightforward softwarewise to add on a calendar, to do list and, maybe, games. You can probably imagine a situation in which someone, somewhere, might use those features. But you'll be at risk of obscuring the key purpose of the device: helping users find their way and stay safe. Too much flippant functionality might even undermine the perception that the device offers good quality in its core functionality. And it will make it harder for users to access the key features they most want and need.

If your device can fulfill multiple purposes for the user, you'll have to invest extra effort in helping users understand its value. A home contact sensor is a generic piece of hardware with no inherent value to the user. The value is in the function it enables: used to detect when an intruder has forced a door open, or when a medicine cabinet has been opened. Early adopters may love the flexibility to use the sensor as a tool that can do all kinds of things. But you'll have to help mass market users understand what it could be for. For example, your app might offer specific window or cupboard alarm functionality to go with the device, even if these do much the same thing under the hood.

Connected products intended for the mass-market need to demonstrate a clear advantage over any predecessors. Connected things are not inherently better than non-connected things, just because they are connected. Despite being demo-ed at consumer electronics fairs year after year, the much-maligned internet fridge concept has so far felt like a solution in search of a problem. Research shows that people can imagine using intelligent fridges that provide

information about their contents, nutrition and health, but this has not translated into demand.[7] Tasks such as managing shopping lists and looking up recipes simply don't feel as if they require a new, fridge-based screen. The idea of the fridge that automatically orders more shopping when goods run out is fraught with potential for irritating errors. If you have to make the fridge sync with your calendar or heating thermostat to see when you're on holiday in order to stop your regular milk order, maybe it's just simpler to buy your own milk after all.

Connected sensors enable many kinds of data in the world to be captured, quantified and made visible. Fitness tracking and energy monitoring (see e.g. figure 4.13) are obvious consumer examples of this. But beware you're not just counting things. Data should be used to provide genuine insights that users can act on.

[7] Matthias Rothensee, User acceptance of the intelligent fridge: empirical results from a simulation, IOT'08 Proceedings of the 1st international conference on The internet of things, 123-139

Fig. 4.13: The Efergy energy monitoring service helps users understand their electricity consumption. (Image: Efergy)

For more information on designing with data, see chapter 13.

Connectivity can enable remote control of devices. The core value of connected sockets and door locks is usually remote control (see figure 4.14).

Fig 4.14 The August door lock, app and hub (plugged into outlet).

Connected home systems that allow automated rules to be created are examples of products whose main value is in automation (see figure 4.16). Intelligent systems such as the Nest thermostat may promise to do the job (such as setting a heating schedule that best fits home occupancy) better than a human.

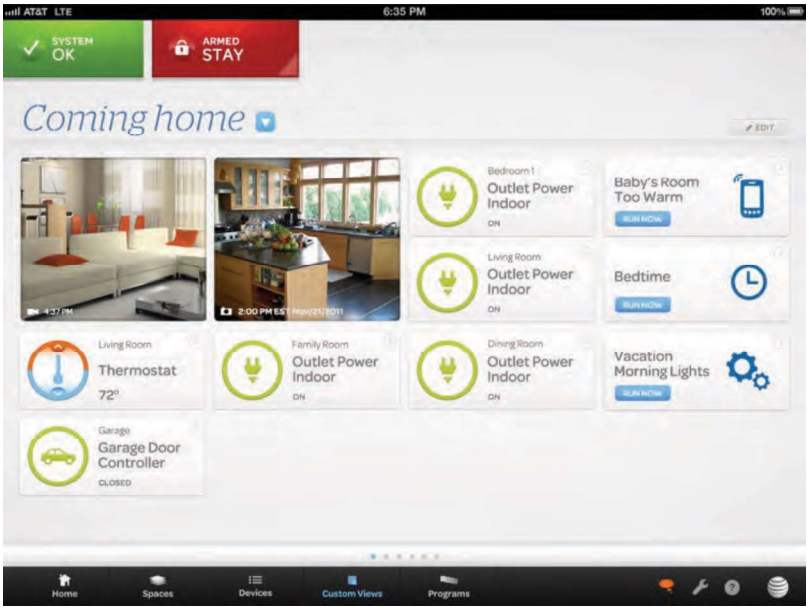


Fig. 4.15: An automated 'coming home' smart rule in the AT&T Digital Life tablet app

Tags or sensors embedded in objects allow them to be trackable and identifiable. The FedEx SenseAware service (figure 4.16) embeds a multi-sensor device inside sensitive shipments (such as medical supplies), allowing the sender to track the location of a parcel and the temperature, light levels, humidity and atmospheric pressure to which it has been exposed. If any of these fall outside a set range, a replacement parcel can be dispatched.

Fig 4.16: FedEx Senseaware sensor and web app

It goes almost without saying that your system needs to be reliable enough to fulfill its promise. Glitches and outages are inevitable in most systems and early adopters will forgive these more readily. But if there are contexts of use in which you cannot afford failure, the product must be 100% reliable. For example, emergency alarms for elderly or vulnerable people must always work. You'll need a backup power supply and connectivity (see figure 4.17), and regular checks to ensure these work.

Fig 4.17 The hub of the Scout security system has a backup battery and 3G cellular chip so it won't stop running during power and internet outages.

The product comes at a cost (financial, or effort exerted) which seems in proportion to the perceived value

A good product needs to balance the cost and effort required from the user against the value it delivers. If the value is very high, users may be prepared to pay more, or invest more time in configuration.

Determining a price point is a tricky matter in itself. You'll have to consider manufacturing costs, competition and market conditions, and what users are prepared to pay.

You'll also need to consider the cost to the user of switching from whatever they were using previously. Household technology, like heating and alarm systems, tends to last years and users won't want to replace working boilers, sensors or other kit at great expense without a significant benefit[8]. If you can support retrofit – new technology that can easily be integrated into old systems – without greatly increasing the cost of your product, you'll increase the potential market for the product.

In the context of UX, the perceived cognitive effort to use your product and the time it will take to get it set up and working affect who will buy it, and why. Be careful in your judgment here. In the thick of a project when you are excited about your idea, it's easy to overestimate how motivated users are to invest time in your product.

Smart homes are a typical example here. It's been possible to connect up lighting, heating, appliances and entertainment systems for around 40 years, as we saw earlier. But you needed to be an enthusiast to set it up and program it (or wealthy enough to pay someone else to do that). A niche of users has taken great pride in their automated homes, but others have found them fraught with support issues, technology failures, and a poor fit with the needs of other guests and residents. Mass-market users often view home automation with suspicion: home is a very personal context, and one in which we are often loath to introduce novel technologies that might break our established routines. Most of us don't want to have to do a load of programming just so we can turn lights on and off. We manage that well enough already and it's an effort to switch unless the benefits are really evident.

Adding extra cognitive effort to everyday tasks is a common risk. The UX strategist Scott Jenson proposes the idea of the 'surprise package': the mature consumer product that is 'enhanced' technologically, turning it back into an early adopter product. As Jenson puts it: "Companies take product concepts

---

[8] C.F. the model of shearing layers, which describes buildings as a set of components that evolve and obsolesce over different timescales. 'Services', like HVAC and plumbing, are expected to last 7-15 years. This concept originates from architect Frank Duffy and was developed by Stewart Brand in his book 'How Buildings Learn: What Happens After They're Built'(1994, Viking Press).

that are now far into the laggard range of stability and established behavior, and they change the product significantly. … The new product is effectively repositioned 'back to the front' of the curve, creating a high-tech product that can only be used or appreciated by the forgiving and accomplished early-adopter group of consumers. This is where much of the consumer backlash appears, as safely mature and benign products such as TVs, radios, thermostats, home phones and even cars are turned back into early adopter products, and then sold to an unsuspecting laggard audience." [9]

TV is a great example. TV used to be an instant-on experience. We may have had less choice of channels and no on-demand services, but you could be watching *something* within a second or two of turning it on. It can now take minutes. You may be faced with software updates for your set-top box and/or connected TV (perhaps for apps you don't even want but can't delete), then minutes of navigating around a program guide or on-demand service using a cheap remote control poorly equipped for the job.

If your product is replacing an existing consumer product or way of getting something done, pay attention to what was good about the old way of doing things. Try to preserve that and enhance the experience, rather than adding new complexity.

The product is pleasing to use

The hard-headed cost/benefit analysis is important for any product, but the best products speak to us on an emotional level too. This is partly about aesthetics, but it's not just about bolting pretty design on top of functionality. We form an integrated impression of the functionality and design of the product, and how well that fulfills our practical and emotional needs and fits (and perhaps communicates) our sense of who we are[10]. Figuring out the right

---

[9]Scott Jenson, 2002, 'The Simplicity Shift' (Cambridge University Press). Available from http://www.jensondesign.com/The-Simplicity-Shift.pdf

[10] Lionel Tiger's 'The Pursuit of Pleasure' is an interesting viewpoint on the anthropology of what makes products appeal. (Lionel Tiger, 1992, 'The Pursuit of Pleasure'. Boston: Little, Brown 1992).

experience is about design as well as product strategy, which is covered in more depth in chapter 5, 'Translating Research into Product Definitions'.

Services in IoT

Devices and services

At the start of the chapter, we set out that an IoT 'product' is frequently a hybrid of physical device(s) and service provision. At the very least, the connections that keep the connected device connected are services and there may be others to consider in making a product good.

When people buy a product, they expect to have the right to use it for as long as they like. When the product is dependent on an internet service, there is a reasonable expectation that that service will continue to be available, for taking it away would render the product at worst useless and at best limited. After all, you would not expect your home heating or lighting to cease to function because the company that produced the original system had gone out of business, or no longer wished to support you. (There is an inherent tension here between the old world of physical products, and the new world of internet/web services. On the web, new services appear and old ones are 'sunsetted' on a regular basis. This is acclaimed as progress. A physical product is likely to come with expectations that it will last for at least a few years. If the service stops working, the lifespan of the device is shortened, creating landfill (and unhappy users). Service providers have a responsibility to ensure that they are able to maintain and improve their internet services, so that the product has a reasonable lifespan.)

The service forms part of that experience. The relationship between the device and service can vary. <-EXPLAIN MORE

Connected heating controllers and door locks are examples of systems where the *device* is likely to be the focus: we might call them *service-enabled devices*. Users view the device as the most salient part of the system. For example, Nest users are likely to say 'I have a Nest', referring to the

thermostat to represent the entire system. (It's far less likely you'd hear someone saying 'I use Nest', 'I have Nest or 'I have a Nest system'.) Because the device is so central to the UX, users will have high expectations of its design and functionality. The service enables remote access and smarter functioning, but in the user's mind it is a way to control the device (Seefigure 4-18).

4-18 – example of produdct advert with device front and centre, e.g. Nest?

A security alarm is an example of a system where the *service* is the focus: we might call it a *device-enabled service*. The alarm service is what users care about. The sensors and other devices are generally low profile and most of the intelligence sits in the internet service or gateway software. You could add or swap out devices without affecting the core functions of the service.

Key factors that indicate that the service may be the focus of your user experience, not the device itself, may be that:

- Interactions are distributed across multiple devices, so no single device is the center of attention
- Most functionality lives in the cloud service or gateway software (perhaps because local devices don't have much computing power); and/or
- Devices can be added, removed or swapped without changing the core functioning of the system;

As the UX expert Mike Kuniavsky describes it, the device is an *avatar* for the service.[11]
The Oyster travel card (see figure 4.19 below) is a stored value contactless smart card used on London public transport. It can hold various types of tickets or a credit balance for travel on the underground, trains, buses, trams and boat services. ('Stored value' means that the credit is notionally held on the card itself, rather than in a separate account, as with a debit card.)

Passengers add tickets or money to the card itself via online purchase, ticket machines at stations or by setting up regular debits from their bank accounts.

---

[11] Mike Kuniavsky, 'Smart Things: Ubiquitous Computing User Experience Design', Morgan Kaufmann 2010

They swipe the card on a reader at the start and end of journeys to validate their tickets or deduct credit. The Oyster saves time and money processing ticket office transactions and reduces the number of paper tickets. To encourage use, fares are substantially cheaper than paper tickets.

Figure 4.19: the London Oyster card

The Oyster card itself is not much of a smart object. It's just a piece of plastic containing an RFID chip and a small amount of memory. The RFID chip passes a unique ID to a reader when a passenger swipes in. The memory holds information about the tickets or money stored on it, so the reader does not need to contact the back office service in real time every time the user swipes the card. This speeds up the rate at which passengers can pass through ticket barriers, which is vital during rush hour. Readers transmit transactions to the back office in batches.

The Oyster card is an icon of London life, but it is really just an avatar for the service. Without the card readers or the ability to top it up it wouldn't be much use to you. The Oyster service involves smooth coordination between many different channels, such as the Transport for London website, the ticket machines, ticket offices and shops that sell top ups, the readers themselves, and the back office systems (see 'Service ecosystems', below).

Technically, the Oyster card itself is not even an essential part of the service: Oyster can also be used via NFC enabled phones and bankcards. In future, the dedicated Oyster card might even disappear, but the service will remain. But services are intangible, and avatars can provide a concrete, tangible focus that helps us understand the service.

Right now, IoT systems are still pretty novel and not well understood, at least by consumers. It's easy to look to individual devices as a handle to understand the system, whether or not this is accurate. (We've heard smart meter users refer to the in-home counter-top display as the 'smart meter', and the actual smart meter as the 'computer under the stairs': see figure 4.20). You might need to play up the role of the devices in communicating what your system does (*presenting it as a service-enabled device*), just to help consumers understand it.

Figure 4-20: In-home display

Over time, as we all become more accustomed to the products around us having intelligence and connectivity, our ability to understand connected products as *services* without depending on physical manifestations may become more sophisticated. The idea of a heating system without a visible controller, or a door lock without a visible lock may seem strange right now, but in time, as long as they work, we might be more open to such things.

It will probably always be appropriate for some systems to have highly visible devices, and for some to focus more on service design (see figure 4.21). The key is to pitch your UX to best suit your product, and the needs of your users.

If your service is the focus of the UX, you can still make beautiful devices but make sure the service design is at least as good. And if the device is the hero of your UX, make sure it's attractive, usable and does what it needs to do, elegantly.

4-21: Diagram or table: service enabled device vs device enabled service.

Service ecosystems

Services are delivered through the interactions of networks of people, organizations, infrastructure and physical components. The devices, and even the digital components, are only part of the experience. A part of the Oyster experience is the interactions you may have with station staff when buying a ticket or asking for help. In order to help you they will have been trained to provide good customer service, but they will also need access to good information about the transactions on your card and system information. Making this whole system work smoothly is a lot more complicated than just making cards, machines and a website: it requires someone to take a holistic view of how the service is experienced, and make sure all the components work reasonably smoothly together (see figure 4-22).

Figure 4.22: the London Oyster ecosystem

Complex IoT services, such as a connected home, require the co-ordination of multiple devices working together, perhaps even using a degree of intelligence to automate some functions without explicit user instructions (for example,

turning off the electricity supply if a gas leak is detected). There may be multiple digital interfaces. There may even be co-ordination between multiple digital and physical services, for example, a heating system may use data from a 3rd party weather service, and the user might have the option of taking out a service and repair contract.

This is called an ecosystem.

If you're designing a service, you'll need to take a more complete view of all the parts of your service, and the relationships between them. For example, you may need to think about handling software and component upgrades across your devices. Ovens may require new controller boards, and washing machines may demand firmware upgrades (see figure 4.23). You'll need to design processes for handling these issues with consumers.

Figure 4.23: The Samsung WW9000 connected washing machine supports over the air firmware updates. (Image: Samsung).

There may well be customer support, marketing, sales, and perhaps professional installation and maintenance too.

"You see companies that have poached Apple designers, and they come up with sexy interfaces or something interesting, but it doesn't necessarily move the needle for their business or their product. That's because all the designer did was work on an interface piece, but to have a really ***well-designed product*** *in the way Steve would say, this 'holistic' thing, is everything. It's not just the interface piece. It's designing the right business model into it. Designing the right marketing and the copy, and the way to distribute it. All of*

those pieces are critical." Mark Kawano, founder of Storehouse and former Apple User Experience Evangelist[12]]

Building a service offering

Thinking at the service experience level encourages us to take a broader look at user needs, not just the interactions with the website, mobile app or embedded devices. Chapter 5 explores understanding user needs in more detail. In this chapter we'll consider how this might create new opportunities to look at the wider service package you may offer customers.

For example, take home security. We typically think of alarm systems as something that makes a loud noise and act as a visible deterrent on the home. A connected system can tell you when someone is breaking in, and perhaps film them. But if you're not able to get home, and no-one else can respond, you can't do anything about it. Connectivity has alerted you to the problem but also enabled you to feel powerless to act.

In this case, you might have the option of paying for a professional monitoring service. The security firm (with your permission) can view your cameras before sending someone out.

IoT services often provide opportunities to capture data, which can be used to improve the service offered, perhaps through better customer service or smarter support. For example, diagnostic data about the functioning of a boiler could be used to identify systems at risk of breakdown and schedule an engineer visit *before* they fail. You might even package the cost of the service contract with the monthly fee for maintaining the service. Users may be happier to pay for something like this than to cover the cost of maintaining your internet servers!

Professional installation or configuration may also be an opportunity for complex systems such as home automation technology. Time It Right[13],

---

[12] Mark Wilson, 4 Myths about Apple Design, From an Ex-Apple Designer, Fast Company July/August 2014 (http://www.fastcodesign.com/3030923/4-myths-about-apple-design-from-an-ex-apple-designer)

[13] http://autotimeonline.com/

designed for the needs of the Orthodox Jewish community, is a culturally specific example (see figure 4.24) There may in future even be a role for independent IoT 'plumbers' who specialize in helping consumers install, maintain and repair connected home systems: a kind of role the technology blogger Anil Dash refers to as 'blue collar coders'[14].

Figure 4.24: Time It Right comes with a professional installation and configuration service (Image: Autotime).

Another service opportunity may involve helping users secure lower prices or better service from 3rd parties, or otherwise benefit from the data that comes from your system. In the sustainable housing development of Little Kelham in Sheffield, northern England residents have smart meters to track energy use and band together to bulk buy electricity (see the connected home case study between chapters 4 and 5).

Personal services, such as installation or intensive customer support, are not necessary for all services and may not be practical within your business model. But it's worth considering the bigger context of user's expectations around the service you're offering, and how their needs will change over time, to make

[14] http://dashes.com/anil/2012/10/the-blue-collar-coder.html

sure your service isn't missing something they think they need, or to spot opportunities to improve the overall experience.

Business models

Establishing the relationship between your system and the surrounding service is, in part, deciding on your business model. Put crudely, this can be summarized in two questions: what will people pay for? And what do you need for production to be sustainable?

What is a business model?

A business model is the blueprint for how a business creates value for customers, and makes money. For example, a classic business model is the 'bait and hook' one used by printer manufacturers. They charge a relatively low price for the initial hardware but make money on toner cartridges.

The model maps out how the business will make money, either from increasing revenue (selling more) or decreasing costs. Increasing revenue can be approached by:

- Generating new business from new customers
- Generating *more* business from existing customers

And even a not-for-profit organization needs a sustainable business model in order to survive.

How do business models affect UX?

Business models shape the way users perceive the value of the service and the fairness of pricing. This can make the product proposition more or less appealing. Users will approach the product or service with a positive, trusting mindset, or a more skeptical or even negative one. This sets the tone for the rest of their interaction.

For example, the major energy companies in the UK have recently come under pressure for perceived unfair pricing practices. All are rolling out smart energy meters, which generate data that can be used to offer customers tips on saving energy and therefore money. But customers who feel that prices have not been

set fairly treat this money saving advice with skepticism. This has a knock-on effect on the perceived UX of the energy saving service. Issues of trust need to be tackled upfront in the design, perhaps through presenting pricing in more transparent ways.

Device and service models

In the traditional product business model, the provider charges once, upfront, for hardware. This is how we are used to buying, for example, cars, or household appliances. For a long time, this was also how we bought software.

In service models (of which many digital models are examples), the provider charges for ongoing service provision. Music subscription services such as Spotify or storage like Dropbox are examples of service business models. A non-connected product can be supplied as a service; for example, renting a car through a traditional rental company is a service. The customer pays for use of a car, not for ownership of a particular car. Adding connectivity to objects increases the potential for service models.

The choice of business model is a balancing act between where the customer perceives the value to be and what they expect to pay for, and what it costs you to provide. IoT is new, which means that what the customer expects to pay for is not always a reflection of the costs you may incur!

For example, your connected heating controller requires an internet service to provide you with remote control via your smartphone app. That costs money to maintain, especially when you consider that the lifespan of a heating controller might be 10 years or more. The provider might factor that cost into the price they charge you for the product upfront, but that might make it more expensive compared to competitor products. Or they might choose to charge you a regular fee for ongoing service provision. But they might find that customers aren't used to paying an ongoing fee to keep their heating working! In that case, the company might try to add value to this service, by turning it into a platform that also supports other devices (like connected lighting or energy monitoring), or adding extra service components, like a maintenance contract.

As discussed above, the perceived value may rest in different places for different types of connected device/IoT system. Is it the overall service users think is most valuable, or one or more of the devices that deliver it?

IoT is immature and there's perhaps a tendency for users to focus more on devices, as these are novel (as we suggested earlier). Service providers may focus on more on services, as that's where the potential for long-term customer engagement sits. You may think that you are offering a health management service, but if your users see beautiful bathroom scales then persuading them to pay for an ongoing service may be tough. The key for a business model is that customers feel they are paying a fair price (whether in terms of money or sharing their data) for the value they receive, and you are making the money you need.

Bringing digital business models to physical products

Building digital services around physical products enables suppliers to apply novel business models, more commonly found in the digital realm, to physical products.

Combining physical devices and services is likely to lead to some interesting, novel and disruptive business models that challenge our preconceived ideas of what it is to own and use a product.

Business models we are accustomed to in the digital realm might make their way into the physical world. For example, we are accustomed to using websites that are free at point of use but make money from selling eyeballs to advertisers. It's not a huge stretch to imagine that physical devices might be given away in exchange for advertising or user data. The ubiquitous computing researcher Pertti Huuskonen jokes about the freemium fridge. Your supermarket gives you a free fridge with a screen that forces you to watch an advert every time you open it. Or the fridge might track the products you put in it and eat, and where you bought them, and share that data with the supermarket, and other advertisers. Users could buy their own fridges, essentially paying for ad free experiences or privacy, or get free or cheaper

appliances in exchange for their eyeballs and data. In future, privacy may be a rich person's luxury[15].

More positively, there are benefits to digital service business models. There are opportunities to develop ongoing relationships with customers, understand more about the people who buy your products and what they do with them, and tailor services better to their needs. Users are accustomed to (and mostly comfortable with) web-based services that capture and store information about them to provide a better service. The sensing and processing capabilities of IoT devices open up potential to extend these personalized services to the physical world.

You may be able to capture user behavior that wasn't previously visible, perhaps in real time: e.g. how people are using energy via smart metering, identifying and tracking people in a physical space, or monitoring traffic levels via aggregated data about the density and speed of movement of drivers' smartphones. Knowing how often or intensively a product is used, and what for, enables you to tailor the service, sell supplies, improve the next version, or offer additional services. For example, some Nespresso coffee machines now come with a SIM card, allowing Nespresso automatically to send more capsules when the customer is running low (see figure 4.25 below).

---

[15] Pertti Huuskonen, personal communication and 2007, 'Run to the Hills! Ubiquitous Computing Meltdown", Proceedings of the 2007 conference on Advances in Ambient Intelligence, Pages 157-172 (available at http://www.cs.swan.ac.uk/~csmax/csrsRG.pdf)

Figure 4.25: The Nespresso Zenius coffee machine comes with a SIM card. (Image: Nestle Nespresso)

You could charge users based on their behavior. For example, car insurance policies such as Insure the Box in the UK, base pricing on actual driver behavior rather than demographics. This may benefit responsible drivers who are in a demographic category considered to be at high risk of accidents, such as under 25s.

Products can proactively maintain themselves or provide data to enable smarter support: as discussed earlier, boilers could identify when they are developing a fault and be serviced before they break down; appliances could share fault data with the manufacturer so customer support can help users diagnose and fix more of their own problems. It is possible to vary pricing based on time of use, for example charging more for electricity at certain times of day to manage demand.

A good UX design needs to balance the needs of the business with the needs of the user. Even if you are not shaping the business model of the product you are working on, you at least need to understand it. The best business models serve the interests of customers as well as the business. The car insurance example above provides cheaper insurance to drivers who would otherwise be penalized on grounds of age, but also allows the company to reduce costs and uncertainty through more accurate risk profiling. User and business needs can be in tension: for example, a connected home service might rely on heavy upselling of new products. Here, design can make the difference between the upsell advertising being either useful or at least minimally intrusive, or downright irritating or something that causes users to stop using the system altogether.

Summary

A clear *value proposition* ensures users understand what your product does, and whether they want it. This is essential in order for them to understand how it works, and how to use it.

Innovators and early adopters are inherently interested in technology and forgiving of imperfections. Mass-market consumers often have different needs.

Many IoT systems are *tools*: they require the user to frame their own problem and configure the system to solve it. Consumers tend to look for *products* that promise to solve a particular problem for user and come already configured to do that. They expect the cost and effort of using the product to be in proportion to the value it brings them.

IoT creates new opportunities for information gathering, sharing, remote control and automation. But there are common pitfalls that can limit a product to early adopter markets. In particular, be careful of introducing *new* complexity to mature consumer products.

The UX of an IoT product might be focused around the device or the service. All IoT systems depend on some kind of digital service, and perhaps offline service components too, like professional installation, maintenance or customer support helplines. Ensuring these work well together is an important part of the overall UX.

Business models shape the way users perceive the value of the service and thus the UX. Bringing connectivity and intelligence into devices may lead to digital business models appearing in the physical world.

9

Cross-device interactions and interusability

Claire Rowland

Introduction

In systems where functionality and interactions are distributed across more than one device, it's not enough to design individual UIs in isolation. Designers need to create a coherent UX across *all* the devices with which the user interacts. That means thinking about how UIs work together to create a coherent understanding of the overall system, and how the user may move between using different devices.

This chapter explores *interusability* - the user experience of interconnected devices and cross-platform interactions – and how to make a bunch of diverse devices feel like they are working in concert.

This chapter introduces:

This chapter addresses the following issues:

- What makes a cross-device system feel coherent *(see page 4)*
- Why it's complicated to understand how an IoT system works, and how we might help users with this *(see page 7)*
- Deciding on the best way to distribute functionality between different devices in the system *(see page 14)*.
- Determining which UI elements and interactions need to be consistent across devices, and which don't *(see page 25)*.
- Dealing with data and content synchronization issues in the UI *(see page 31)*.
- Designing interactions that require switching between devices *(see page 39)*.

Cross-platform UX and usability

Many of the tools of UX design and HCI originate from a time when an interaction was usually a single user using a single device. This was almost always a desktop computer, which they'd be using to complete a work-like task, giving it more or less their full attention.

The reality of our digital lives moved on from this long ago. Many of us own multiple internet-capable devices such as smartphones, tablets and connected TVs, used for leisure as well as work. They have different form factors, may be used in different contexts and some of them come with specific sensing capabilities, such as mobile location.

Cross-platform UX is an area of huge interest to the practitioner community. But academic researchers have given little attention to defining the properties of good cross-platform UX. This has left a gap between practice and theory that needs addressing.

In industry practice cross-platform UX has often proceeded device by device. Designers begin with a key reference device and subsequent interfaces are treated as adaptations. In the early days of smartphones this reference device was often the desktop. In recent years the 'mobile first' approach has

encouraged us to start with mobile web or apps as a way to focus on optimizing key functionality and minimize 'featuritis'. Such services usually have overarching design guidelines spanning all platforms to ensure a degree of consistency. The aim is usually on making the different interfaces feel like a family, rather than on how devices work together as a system.

This works when each device is delivering broadly the same functionality. Evernote, eBay and Dropbox (see figure 9.1) are typical examples: each offers more or less the same features via a responsive website and smartphone apps. The design is optimized for each device, but provides the same basic service functionality (bar a few admin functions that may only be available on the desktop).

Figure 9.1: Evernote offers broadly the same service functionality across different device types

But this approach breaks down when the system involves very diverse devices with different capabilities working in concert. In IoT, many devices do not even have screens, or an on-device user interface. Multiple devices may have UIs with very different forms or specialized functionality (see figure 9.2). Even if the UI is only on one device, the service still depends on all the devices working together in concert.

Figure 9.2: The Smart Things ecosystem contains a range of specialized devices that complement each other.

It's not possible to design a system like this by thinking about one device at a time: this is likely to create a disjointed experience.

In order to use it effectively, the user has to form a coherent mental image of the overall system. This includes its various parts, what each does and how different objectives can be achieved using the system as a whole. Traditional single-device usability doesn't tell us very much about how to do this.

What is interusability?

Charles Denis and Laurent Karsenty first coined the term 'inter-usability' in 2004 to describe UX across multiple devices[1]. Conventional usability theory is under-equipped to cope with cross-platform design. However, one 2010 paper by Minna Wäljas, Katarina Segerståhl, Kaisa Väänänen-Vainio-Mattila and Harri Oinas-Kukkonen proposes a practical model of interusability[2].

Wäljas et al propose that the ultimate goal of cross-platform design is that the experience should feel *coherent*. Does the service feel like the devices are working in concert, or does the UX feel fragmented?

They define three key concepts for cross-platform service UX, which together ensure a coherent experience:

- **Composition**: how devices and functionality are organized
- Appropriate **consistency** of interfaces across different devices
- **Continuity** of content and data to ensure smooth transitions between platforms

The paper was published in 2010 and the services evaluated (including Nike+ and Nokia Sportstracker) now inevitably feel a little dated. But we have found the model still holds up well in our own work designing IoT services, and it's a key reference for the rest of this chapter.

---

[1] Denis, C. and Karsenty, L. (2005) Inter-Usability of Multi-Device Systems – A Conceptual Framework, in Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces (eds A. Seffah and H. Javahery), John Wiley & Sons, Ltd, Chichester, UK

[2] Wäljas, M., Segerståhl, K., Väänänen-Vainio-Mattila, K., Oinas-Kukkonen, H.: Cross-platform service user experience: a field study and an initial framework. In: Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services, MobileHCI 2010, p. 219. ACM, New York (2010). I'll refer to this paper several times in the rest of this chapter as Wäljas et al, although I understand that Katarina Segerståhl was the primary researcher. Her PhD, available at http://herkules.oulu.fi/isbn9789514297274/isbn9789514297274.pdf, builds on the same concepts.

Conceptual models and composition

In this section, we'll look at two related concepts that help us design systems spanning multiple devices.

Conceptual models refer to the way humans understand the overall system (and its interfaces) to work. Users need some understanding of how the system works in order to figure out how to interact with it. As we saw above, *composition* is a dimension of interusability. It refers to the way user-facing functionality is distributed between different devices: which device does what. The two concepts are related in cross-platform design: understanding which device does what is part of forming an effective conceptual model.

Conceptual models

The user model and the design model

The conceptual model may refer to the way the user understands the system, or the way the designers (or engineers) think about the system. Users develop a mental model of the system (a *user model)* that enables them to understand what it does, how to interact with it, and how it will behave. At first, this will be based on prior experience of other systems or similar activities. Over time they will develop the model through their experiences with the system itself. The way the designers or engineers think about the system will be reflected in the *design model* (this distinction was defined in Don Norman, 1988)[3].

As Norman puts it: 'The problem is to design the system so that, first, it follows a consistent, coherent conceptualization - a design model – and second, so that the user can develop a mental model of the system – a user model – consistent with the design model.'

The similarity between the design model and the user's mental model is a core determinant of usability in any system, not just IoT. How easy is it for the user to figure out how to achieve a particular goal using the system (which Norman

---

[3] Norman, Donald (1988). *The Design of Everyday Things*. New York: Basic Books.

refers to as bridging 'the gulf of execution'[4])? How easy is it for the user to understand what the system does in response ('the gulf of evaluation')? (See figure 9.3).

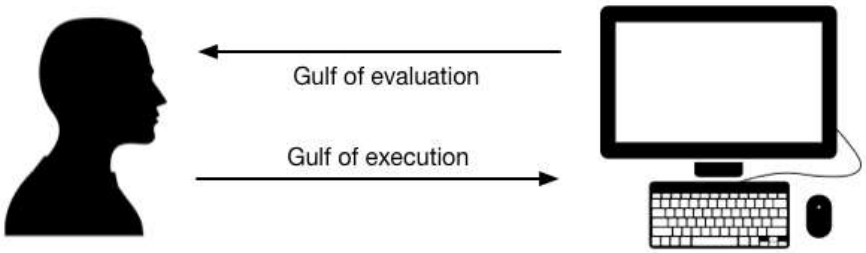


Figure 9.3: The gulfs of evaluation and execution. (User icon by Simon Child, computer icon by Alyssa Mahlberg, both from the Noun Project)

Ideally, the user model maps closely onto the design model. But frequently, the design model may not be a good fit for what the user wants to do, or users may only partially understand it. If the system doesn't conform to any prior expectations, users must develop a new mental model, based on trying to infer the design model. Users learn about the design model through the interface, behaviors of the system and documentation – which Norman refers to as the *system image* (see figure 9.4).

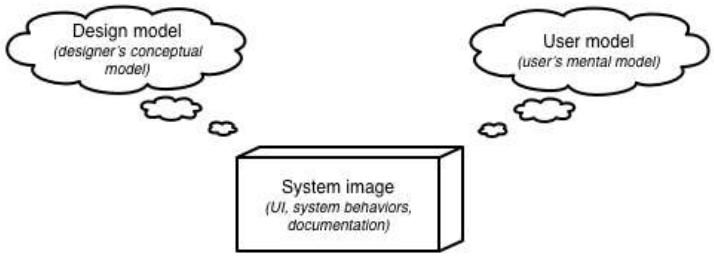


Figure 9.4: Diagram: the user, system and design model (redrawn from http://www.jnd.org/dn.mss/design_as_communication.html)

[4] Chapter 3 'Cognitive Engineering' in 'User Centered System Design: New Perspectives on Human-Computer Interaction', ed. Norman and Draper, Lawrence Erlbaum Associates, 1986

You can't design the user's mental model directly. But you can design the system image to convey the design model clearly (see figure 9.5). You should also define your design model explicitly; to make sure it's clear, consistent and not overly technical or complex for your audience.

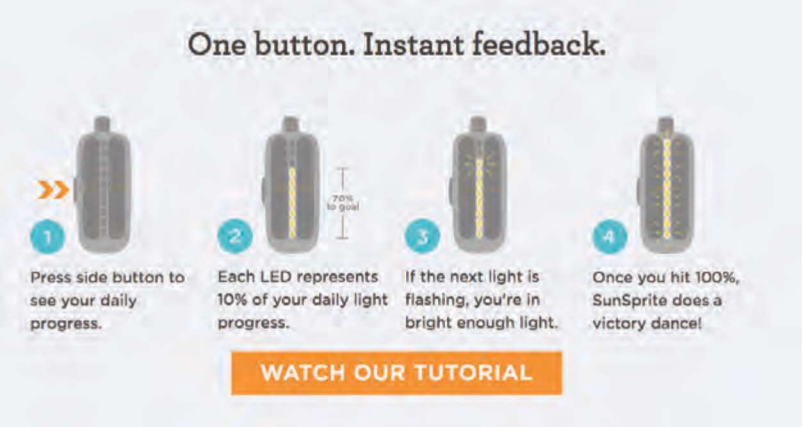


Figure 9.5: A system image: The Sunsprite Tracklight helps users monitor their daylight exposure. The instructions explain what it does, how to interact and data is displayed using the LEDs[5]*. (Image from* [1] *https://www.sunsprite.com/tracklight/).*

Multi-device services are conceptually more complex

Back when Norman first wrote about conceptual models, a system was generally a software application running on a standalone computer. Multi-device services make conceptual models more complicated. There are not just more interfaces, but more places where processing and functionality can live and where data can be stored. Because there are more nodes and connections, there are more points of failure and ways to fail. This is often where complexity is exposed: when the system is working well, it may not matter where your data or preferences are stored. But when parts or connections fail, the user has to understand something about how the system works in order to understand what is happening and why.

Take the example of a lighting system. The mental model of a lamp is simple: it has power, a switch, a fitting and a bulb (see figure 9.6). If the lamp doesn't work it's probably because the power has failed or the bulb has blown.

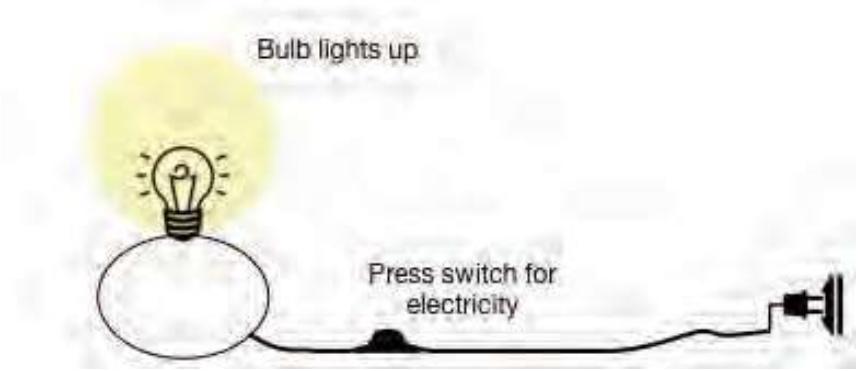


Figure 9.6: A conventional lamp has a simple conceptual model

A connected lighting system has bulbs, switches, fittings and power too (typically either the bulbs or switches will be connected). It also has an internet service, probably hosted remotely. It has a smartphone app and perhaps a web app too. It probably also has a gateway device. It has more parts (see figure 9.7) and more different kinds of part. It can also do more. It may run automated rules to turn lights on and off at certain times, or when certain trigger events happen (such as the security alarm being activated). The intelligence that controls the system may live in several places: in the bulb or switch itself, in the gateway, in the internet service, or even in the smartphone app.

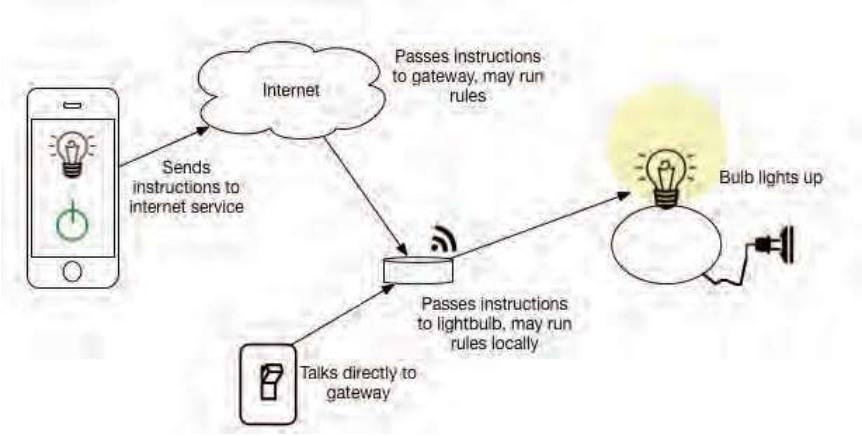


Figure 9.7: connected lighting can do more than conventional lighting, but the conceptual model is more complex. (Lightbulb by Marek Polakovic, wireless icon by Spencer Harrison. plug by Vithin Viswanathan, light switch by Sarah Hutchinson, all from the Noun Project.) MAY NEED WIFI GATEWAY ADDING

When everything is working well and connected, users don't need to concern themselves with which code is running where. But if a part of the system develops a fault or loses connectivity or the network is slow, the impact will depend on what that device is doing. Will the lights (or intruder alarm, or heating) stop working if the user's phone battery runs out or they have no signal? Or will they keep running even if the user cannot remotely access the home at that point? What if the home internet connection goes down? Will lighting rules continue to work locally? If they are stored in the gateway or edge device, they will. If they are stored in the internet service or smartphone, they will not (see figure 9.8).

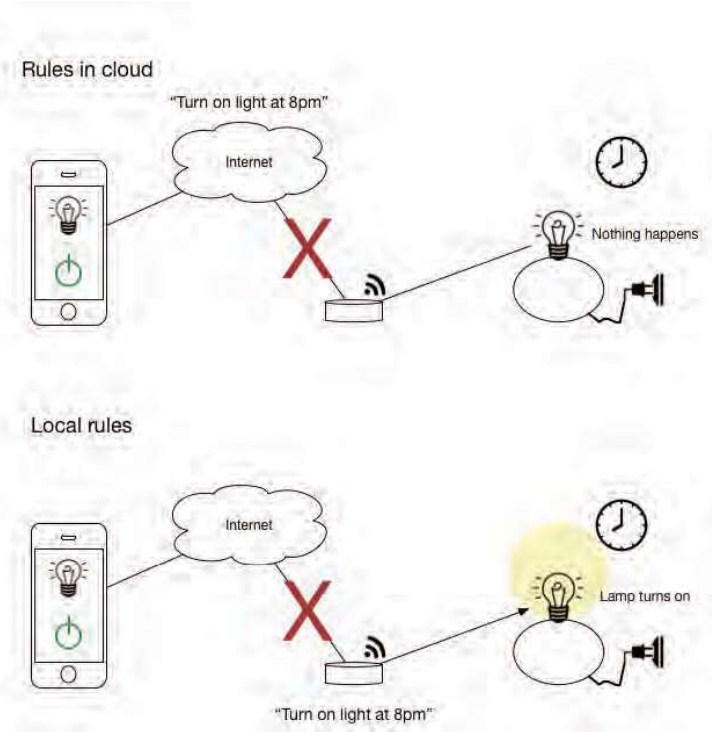


Figure 9.8: Home automation routines stored in the cloud will not run if the internet connection goes down. If they are stored locally, they will continue to run (although the user won't be able to see this or control devices remotely). (Clock icon by Christoph Robausch from the Noun Project) BOTH NEED WIFI GATEWAY ADDING

How you choose to distribute system intelligence is a system architecture issue, as discussed in chapter 3. What's most appropriate for your system will depend on what it does and your users' expectations. An intruder alarm should not fail completely because the internet went down. But it's not a disaster if your energy monitoring system is occasionally unavailable for short periods of time, as long as data is not lost.

The new challenge for UX is that this is a lot of complexity that users didn't previously have to worry about (see the discussion of the 'surprise package' in chapter 4). There are two ways to deal with this complexity: you can explain it, or try to hide it.

Although BERG are now defunct, the BERG Cloud bridge, the gateway for Little Printer, was a simple example of a device that explained how the system was working.. It had LEDs to show whether the device had power and an Ethernet connection, and upstream and downstream connectivity (see figure 9.9). It was labeled to explain that upstream meant that the bridge could see the BERG cloud internet service, and downstream meant that the ZigBee network used to connect to local devices was running. The gateway was communicating the system image.

Figure 9.9: the BERG Cloud Bridge.(Image: BERG).

The more complex the system, the more overwhelming it may be to explain in detail. In that case, it would be better to allow the user to work from a simplified mental model. Automatic gearboxes are complex mechanical systems that require only a simplified mental model in order to use (see figure 9.10). But this is a hard trick to pull off. The gearbox is mature technology and only performs one basic function. Also, consumers have lots of prior experience of driving cars on which to draw. IoT technology is newer, and often does things that are less familiar to users. Many IoT systems also have multiple functions, so it can be harder to reduce them to a simplified conceptual model.

Figure 9.10: automatic gearbox controls hide the complexity of the system behind a simplified conceptual model

One example of a product with a simplified conceptual model is Apple iBeacons (introduced in chapters 2 and 4). Imagine a user walks into a store for which they have a an app installed, their location is detected using iBeacons, and they are sent a push notification about a special offer. It's good enough for that user to understand that the store knows they are there, thanks to the beacons, and the store has sent them a message.

Apple's own description of iBeacons is "a new class of low-powered, low-cost transmitters that can notify nearby iOS devices of their presence[6]". This can be read as implying that the beacons notify the iOS device of its location. This is a good enough model but it skims over some complexity.

What actually happens is: the beacon broadcasts a unique ID. The iOS device detects the ID, and looks up its location in an online database. It notes its own position by its proximity to the beacon. In looking up the beacon, it informs Apple of its location, and *Apple* then send the push notification.

9-11: iBeacons diagram: what the user needs to know, and what's actually happening

UX researchers at Ericsson have suggested that it is particularly difficult for users to understand *networks* of devices. Their informal research indicated that users currently think of connections between devices as being 'invisible wires'. As Ann Light, co-author of chapter 15, puts it: "Most people are disposed to think of things, not links; of nodes rather than relations"[7]. But this way of understanding is not helpful in making sense of complex networks with many interconnections and interdependencies[8]. **To understand a system, users must understand the links as well as the nodes.**

In 1983, the HCI specialist Larry Tesler (then at Apple) proposed the Law of Conservation of Complexity. Interviewed in Dan Saffer's book 'Designing Interactions', Tesler says: "I postulated that every application must have an inherent amount of irreducible complexity. The only question is who will have to deal with it[9]." His point was that shielding users from complexity would involve extra work from designers and developers.

---

[6] https://developer.apple.com/ios7/, which now redirects to iOS8, original text retrieved via http://en.wikipedia.org/wiki/IBeacon#cite_note-5

[7] In conversation.

[8] Joakim Formo, The Internet of Things for Mere Mortals, http://www.ericsson.com/uxblog/2012/04/the-internet-of-things-for-mere-mortals/

[9] Dan Saffer, Designing for interaction, 2006, New Riders. Larry Tesler interview available at http://www.designingforinteraction.com/tesler.html

User understanding will improve over time with familiarity, but only if we, as designers, help them with clear system images. We need to figure out what complexity users will need to deal with, and where products and tools can be simplified. As a general rule, if the task or activity the user wants to perform is complex or requires a high level of skill, it's appropriate for the user to engage with that complexity. Or perhaps it's a job for a professional. If the task or activity can be expressed simply but the technology is complicated, there's a good case for designing around a simplified mental model.

We don't yet know what that looks like for IoT. For starters, if you need to explain to the user that part of the system is not working, it's important to explain why and what this means. For example, if you are alerting them that the security alarm has lost internet connectivity, you might choose to tell them that the cameras and alarm sounder are still active but that they will not receive alerts. Or if the user is travelling in a different timezone, you might want to show the current time at home on the heating control app, to indicate that schedule changes are based on the local timezone of the controller, not where they are now.

Composition

Patterns of composition

Composition refers to the way the functionality of a service – especially the user-facing functionality - is distributed across devices.

Good composition distributes functionality between devices to make the most of the capabilities of each device. Designers should take into account the context in which each device will be used, and what users expect each to do.

There are some common patterns to composition. Web services delivered across smartphones, desktops, tablets and connected TVs are often *multi-channel*. Each device provides the same, or very similar, functionality (in other words, there is a high level of *redundancy* between devices. Kindle, Netflix (see figure 9.12), BBC iPlayer, Facebook and eBay are all examples. Devices with small screens or limited input capabilities may only provide a subset of key functionality. But each device offers a similar basic experience of the service. Many users won't own or use all the possible devices on which the service could be used and this doesn't matter. It's perfectly possible to use the service via a single device and still have a good experience.

Figure 9.12: Netflix is a multi-channel service

However, many IoT services run on a mix of devices with different capabilities. Service functionality and user interactions will be distributed across different devices. Some functionality may be exclusive to a specific device. For example, in the Withings ecosystem, only the scale can measure body mass and only the blood pressure monitor can measure blood pressure. Some devices may be custom designed for the service. We've seen many examples of these already throughout the book, from connected door locks to smart watches and thermostats.

For reasons of cost, or practicality, these may have limited inputs and outputs that are quite different from a conventional 'computer' UI. A door lock may have a keypad and handle and perhaps an LED to show whether it is connected or not. It probably doesn't have a screen. In these systems, we have to figure out which device handles which functionality. Each device (and the internet service itself) may have a different role in terms of providing user interactions, connectivity, information gathering, processing or display. In the terminology of Wäljas et al. this is a *cross-media* system (figure 9.13 is an example).

Figure 9.13: Withings is a cross-media ecosystem

For example, a heating service as shown in figure 9.14 below may comprise:

- A boiler that heats the water
- An in-home controller/thermostat that tells the boiler when to switch on or off (this may be a separate thermostat and programmer, or just one device).
- A gateway that provides a low power connection to the controller and bridge out to the internet
- A cloud service that stores user account information and remote access to the system
- Smartphone and web apps that connect to the cloud service.

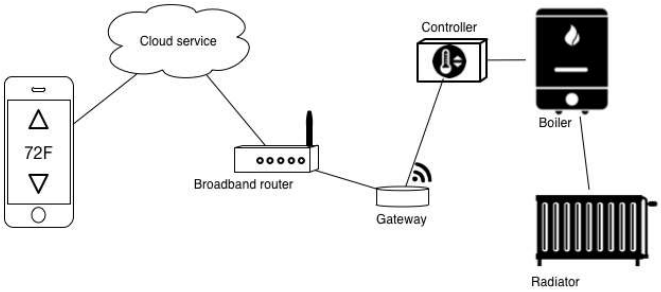


Figure 9.14: system diagram of a heating system (Thermostat by Iconathon, boiler by Axeny Virtinsky, radiator by Jose Hernandez from the Noun Project)

Both the Tado and British Gas Hive systems work in this way, but user-facing functionality is distributed differently. The Tado thermostat/heating controller has almost no UI (see figure 9.15). User can view the current temperature and set or alter the setpoint, but most interactions are handled on the smartphone[10]. This may keep manufacturing costs down. Smartphone interfaces are much cheaper to develop than physical interfaces, as components like screens and buttons are relatively expensive. It's an elegant choice for a small household occupied mainly by smartphone owners but there are trade-offs. If you don't have your phone to hand, or the battery is dead, or you're a guest in the house without access to the phone UI, you have limited control.

[10] The first generation Tado controller had no onboard controls at all: all interactions were via the smartphone.

Figure 9.15: The Tado heating controller and smartphone app

The Hive Active Thermostat heating controller is a standard thermostat with on-device controls (see figure 9.16). The device is designed to be competitive on cost with non-connected heating controllers. So interaction design has to work within the constraints of an LCD screen and limited number of buttons. However heating can also be controlled by phone and web apps, which are probably easier for most people to use than the hardware. This means that heating controls are available to anyone in the house, whether they have access to the smartphone app or not.

Figure 9.16: The Hive Active Thermostat heating controller and smartphone app (Images: British Gas).

Some devices may not support any user interactions at all (see figure 9.17). Some devices may be simple sensors which simply provide data to the service, as in an air quality monitoring system. In this case, you may simply hand off all functionality onto a single mobile or web app. Although the overall service may be complex, the web or smartphone UI in this case is in some ways simpler to design as there is only one interface to consider. (Chapter 8, Interface and interaction design, considers the pros and cons of handling functionality via a mobile device versus a specialized embedded device).

Figure 9.17: The Greenbox garden irrigation controller is entirely controlled by a smartphone app (need alternative – 403 on greenboxhq.com).

When key tasks are available across multiple devices, users may still be able to use the service even when some devices are unavailable. For example, the Withings smartphone app can use the onboard accelerometer to measure activity, so even if the user has forgotten their dedicated activity monitor, they need not lose data (see figure 9.16).

Figure 9.18: The Withings mobile app can use accelerometer data to measure activity.

Even where different devices support the same tasks, they may be used in different situations. For example, the key advantage of connected heating systems is that the smartphone app enables control from anywhere, whether that's the other side of the world or the user's bed.

However, this isn't a recommendation to duplicate every piece of functionality across every device – redundancy isn't necessarily a good thing. Too many functions on a single device can make the UI harder to use, especially if the device has limited input/output capabilities. And user interaction components, such as screens and buttons, add significantly to manufacturing costs of embedded devices. The right decision will balance the usefulness, cost, and usability of putting various features on different devices.

For many systems, it makes sense to use a network of devices that are specialized for particular functions. For example, the Lively elderly care service uses specialized sensors to monitor the pillbox, fridge and the kitchen (see figure 9.19). This is referred to as *synergistic specificity*[11]: specialized components working together to deliver a service that is more than the sum of those components.

Figure 9.19: Lively elderly care system: safety watch for summoning help, with pedometer and medication reminders, hub, sensors for fridge and pillboxes and a custom sensor containing an accelerometer to detect movement. (Image: Lively)

Users may also want to add (or remove) devices to suit their individual needs, or combine them in different ways to fulfill different purposes – this is a *modular* system. In some cases, different devices can be used to perform different functions as part of different services. For example, a home monitoring system may offer contact, temperature, moisture, smoke and motion sensors. These could be used to detect occupancy for heating, lighting, and potential safety problems or intruders. A highly modular system can be very powerful, but may be more complex for users to understand, configure and use. To use the distinction from chapter 4, it's likely to be more of a tool

---

[11] Schilling, M. A. 2000. Toward a General Modular Systems Theory and Its Application

to Interfirm Product Modularity. Academy of Management Review, 25

for early adopters than a product that majority consumers will configure themselves. It's also more complex for designers to communicate which devices are doing what at each point.

Determining the right composition

For any service, there is often more than one possible suite of devices that could be used to deliver the service. The decision as to which is most practical will be influenced by the following factors. You may wish to prioritize, per task, which devices are optimal for each task, acceptable for the task, or not possible for this task.

What best fits the situation and context?

The most important consideration is what best fits the activity, situation and user needs. Certain devices need to live in one place where only one function is required, e.g. blind/window shade controllers or light switches. Others are used in a context that places constraints on form factors and interaction modalities. For example, climbers need their hands free, so delivering altitude, weather and location information to a wrist top makes sense. Using a mobile phone's music controls while driving would be dangerous. Key functionality should be mirrored on the car dashboard in a way that minimizes demands on attention. Features that are essential to one device may be inessential or even inappropriate for others.

The availability and reliability of the network connection is also key. If you have a good reliable connection you can afford to centralize more functionality, e.g. putting your irrigation system controls on a smartphone. If not, and you can't afford to lose access to functionality, you may need user controls (and perhaps more onboard intelligence) in the edge devices.

Can you work with pre-existing devices?

What hardware can you assume your user base already has and is familiar with using? For example, if users all have smartphones you can use these to handle complex interactions, determine location and identity, and in some cases handle local and internet connectivity.

Smartphones come with onboard sensors (such as accelerometers), which can be used for tasks such as activity tracking without additional hardware. However, custom form factors (such as wristbands) may provide a better

experience in some contexts of use, such as the forthcoming clip-on fall sensor for the Lively safety watch for older adults. Specialist equipment may also tend to offer better performance through better quality parts, such as more powerful GPS chips or better battery life.

What interaction capabilities do the various devices have (or could you cost-effectively include on a custom device?)

You may be able to consider adding or removing interaction capabilities (like screens, buttons, audio beeps or LEDs) to embedded devices. However, these typically add to production costs, so you will probably need to keep these to a minimum and offload more complex functionality onto a mobile or web UI. If you're not able to influence the design of the embedded devices, you'll have to work with the interaction capabilities you have.

You may also decide that just because a device *could* support a particular function, it does not have to. Keeping things simple may make the device interface easier to understand. For example, a heating controller with a low-resolution screen and limited buttons might be best used for status information and in-the-moment controls (turn the heating up now!). You could offload more complex tasks such as schedule setting onto a web or mobile interface. The bigger screen size and richer interaction capabilities will enable a better design. You can also provide a 'good' way to do the task on a fuller featured device and a limited or compromised version on a less capable device which must occasionally work alone. For example, an intruder alarm system may provide an easy way to view which sensors triggered the alarm on a mobile interface. The task may also need to be possible on the alarm panel via a basic LCD screen, even though this is likely to involve many more button presses, perhaps navigating menus and modal states.

Does the system need to work if some devices are unavailable?

What happens if a device is unavailable, e.g. a smartphone is lost or the battery is dead? Does it need to be used by 3rd parties who may not have access to a web or mobile app, such as visitors to the home? Can the device work offline?

How accurate does sensing need to be?

If a service needs to know the rough location of its user, a smartphone can estimate this from for GPS/celltower signals. If it needs to know which room

the user is in at home e.g. to turn the lights on and off, a smartphone could be used via Bluetooth LE connections, but will only be accurate if the user carries it with them at all times.

Do users have set expectations of devices?

Users may expect certain devices to conform to familiar form factors, or provide familiar functionality. For example, they are likely to expect a heating controller to have some way of turning the heating on, or up.

How do you balance cost, upgradeability and flexibility?

User interface components, such as screens and buttons, are expensive to add to embedded devices. You may therefore decide to limit interactions on the embedded devices themselves and do most of the interaction "heavy lifting" via mobile apps or web interfaces.

It can be difficult to add new features to devices that are already out in the field, especially if this requires modifications to the interface. Again, offloading interactions to smartphone and web apps, which can be modified relatively cheaply and quickly, may make sense.

What connectivity and power issues do you need to consider?

It may also be simpler to handle information processing in the cloud, as with the Withings scales and fitness trackers, which simply take readings, display them in real-time, but handle all other functionality in the online service. A weight and fitness service can handle temporary losses of connectivity gracefully, by storing data locally and syncing when the connection is available again. However, for other types of service this will not be acceptable, e.g. where safety or security is at stake. If a monitoring system for an elderly person loses connectivity, it might be acceptable for motion sensor data to be temporarily unavailable to the carer, as long as it is clear that connectivity has been lost and live data is not available. However, it would be completely unacceptable for the elderly person to be unable to use their emergency alarm during this time: the alarm should be able to fall back to using another form of connectivity.

In other words, the designer has to make an informed call on which tasks need to be available in different conditions: offline? With no power? Does the

system make no sense if connectivity is lost? Is a suitable fallback available? I should not have to worry about being unable to enter and leave my own house because the front door lock has lost connectivity or has no power – and many other household functions must also be taken for granted to be effective.

What is the physical context of use?

Do any parts of the system need to have particular form factors/be used in certain contexts: e.g. worn on wrist, weatherproofed, used one handed?

How central to the service are the devices?

Are devices central to the conceptual model, or not? This may not affect the distribution of functionality, but it will affect the way in which you communicate the composition of the system.

9-20: (Summary table of composition patterns?)

Consistency

"Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions." Jakob Nielsen, 1994[12]

Consistency is well known as a general UI design heuristic. It's a simple concept to grasp. But knowing what needs to be consistent, and what does not, can be tricky. You may have to trade off one type of consistency for another. Do you make all the buttons look the same so they are easy to identify as buttons? Or does that cause confusion by implying that certain functions are similar, when in fact they are not? Too much consistency, or consistency between the wrong things, can be as damaging as too little.

---

[12] Nielsen, J. (1994b). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), *Usability Inspection Methods*, John Wiley & Sons, New York, NY.

Consistency across multiple devices

In the case of cross-platform systems, designers also need to consider consistency across different devices. Consistency works to create a sense of coherence of the overall system.

Words, data and actions that are the same across devices should be understood to be the same. Words, data and actions that are different should be understood to be different. This helps users form a clear mental model of the system and its capabilities. Knowledge that users have gained about the system from one device can be transferred to help them learn how to use other devices.

Other elements that may need to be consistent to some degree across devices include

- Aesthetic/visual design (to make the devices look, feel and sound like a family)
- Interaction architecture (how functionality is organized) and
- Interaction logic (how tasks are structured or the types of control used).

Guidelines for consistency

Use consistent terminology

As a rule of thumb, the highest priority is to use consistent wording across devices. This ensures that data and actions across different platforms are understood to be the same thing. Whatever the display capabilities of each device, you can always give functions or data the same label even if you can't make them look the same.

For example, imagine you are working on a connected heating controller that offers 3 mode options: ON (heating is on continuously), AUTO (heating is running to a pre-programmed schedule), and OFF (see e.g. figure 9.21). These are already set in the fixed segment LCD display and cannot be changed. You might think that ON and AUTO are not the clearest terms. You'd prefer to change them to CONTINUOUS and TIMER or SCHEDULE in the smartphone app interface. However, this would create a disconnect: users then have to understand that ON and CONTINUOUS are the same thing.

Users are often intimidated by heating controllers and expect them to be confusing. And they might not have a strong enough mental model of the system to infer that CONTINUOUS and ON are the same thing. Having tested systems with similar issues, we found it was more important that these options were consistently named across devices. The value of better terminology in the mobile app is undermined if users don't understand that the functions are the same.

Figure 9.21: The Hive Active Heating controller is an example of a device with fixed segment LCD labels mapped to physical buttons.

Follow platform conventions

The second priority is for each UI to be consistent to the platform conventions of the device.

Mobile OS UI conventions are well documented in styleguides, e.g. the iOS Human Interface Guidelines[13] and Android Design[14]. There are some key

---

13 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

differences, for example Android users may expect contextual menus on long press, a convention which does not exist on iOS, where the same menu would be displayed on another screen (see figure 9.22).

Figure 9.22: Android and iOS screens from the same mobile app showing different platform conventions

In general, for mobile devices and others that have established platform conventions, following these conventions will make it much easier for users to use your app even if it means some things are done in a different way than they are on any specialized connected devices. A heating control app for iOS must be a good iOS app as well as recognizably part of the heating service. It does not have to be a skeuomorphic representation of a heating controller. But this works both ways: a heating controller does not have to pretend to be an iPhone just because it has an iPhone app. The goal is to ensure that interfaces are appropriate to each device, yet also feel like parts of a coherent service.

Interaction elements such as buttons, menus and switches should be recognizable according to the platform conventions of the device. An iOS button should generally follow conventions for button styles on iOS rather than trying to look like the physical button on an embedded device.

Most importantly, specific interaction controls should be optimized to the platform. If a physical control uses a wheel, slider or handle, there is no need to replicate this on a small touchscreen device where it could be harder and more imprecise to use.

For example, the Nest thermostat incorporates a rotating bezel, which is used (amongst other things) to increase and decrease the temperature. The bezel makes a clicking noise as it is rotated. But the iOS app up/down controls are arrows. The designers could have shown a representation of a bezel onscreen for the user to tap and drag around. But the arrows are a much more efficient and precise control on a touchscreen (see figure 9.23). Adjustments to domestic temperature are typically within a degree or two, so precision is important to avoid overshooting.

---

[14] https://developer.android.com/design/index.htmlio

Figure 9.23: The Nest thermostat and iOS app (showing Celsius temperatures).

Aesthetic styling

A consistent visual and aesthetic style across all platforms reinforces the perception of a coherent service. Consistent fonts and colors across devices are nice but may not always be practical. For example, the Nest thermostat uses the same font and colors to indicate temperature on the wall thermostat as the iOS app. But this isn't always going to be possible: a cheap monochrome LCD screen won't support a choice of fonts anyway. Replicating the LCD font on a web or smartphone app may impact readability. It's also a definite retro statement that may not be the look you're after. It's nearly always aesthetically clunky to make a screen design resemble a physical device.

Audio is another way to use aesthetic design to create a sense of coherence. Tapping the down/up arrows on the Nest smartphone app produces the same clicking noise per increment as the bezel on the wall thermostat. This is an elegant touch that adds a common aesthetic to each interaction without

intruding on usage. It adds to the sense that the devices are a family and helps users form a conceptual model of how the system works.

Where visual elements also convey meaning it is vital that they are used in the same way. This is called *semantic consistency*. To continue with the heating example, you may use red/orange/blue colors to indicate temperature. Or a particular icon might indicate that the water tank is heating up. The icon may be higher resolution on devices with better screens, but it must be recognizably the same thing (see figure 9.24).

Figure 9.24: An icon (tbc) shown on a smartphone screen and Pebble watch

Interaction architecture and functionality

Interaction architecture is the logical hierarchy (or other structure) of the UI as mapped to the controls. This is likely to be less consistent across devices and more platform-dependent. Devices may have different functions in the service. Even where there is an overlap between functions, they may be optimized for different purposes. A wall thermostat might be optimized for small adjustments and switching mode (e.g. turning on the hot water). It might need to support changing the heating schedule, but that's always going to be a better experience on the mobile or web app. In optimizing the thermostat for quick adjustments, the designers might knowingly create a less-good UX for schedule changes. But they might view this as acceptable if users are likely to change schedules on a smartphone or website anyway.

As devices are used for different things, it's not necessarily desirable to group functions in exactly the same way. For example, a mobile or tablet screen can provide one touch access to many functions, facilitating a broad, shallow functional hierarchy. Fitting the same functions into a heating controller with an LCD screen plus 3 buttons may require a narrower, deeper hierarchy.

You may also need to use modes, in which the same buttons perform different actions in different states. Modes are typically more difficult to use, but they may be an essential compromise if you're stuck with the hardware (see e.g. figure 9.25). Structure your mobile or tablet app to be a great solution for that device, and don't let it be constrained by the limitations of the embedded device.

Figure 9.25: heating controller with modal functions

UIs on different devices don't all have to have the same features, but where they do, the functionality should be consistent. For example, if a heating controller supports a 6 phase schedule (6 phases throughout the day) but the companion phone app only supports 4, users will wonder what happened to the other two (see figure 9.26).

9.26 – Diagram: 6 versus 4 phase operation…

Consider the most likely combinations of devices

As a designer, you may have to think about design across a large ecosystem of devices. Users may not have all of these. Focus your effort to achieve consistency on the combinations of devices users are most likely to have. To stick with the example of a heating system, all your users might have a controller and smartphone app, but few will regularly use both iOS and Android apps. So it's important that the smartphone apps are both appropriately consistent with the controller. It's less important that knowledge users acquire from using one smartphone app is transferable to the other. For example, the location of the menu button, or the way that system settings are grouped and accessed, need not be the same across mobile platforms but should conform to the platform conventions (as discussed above under 'Follow platform conventions'). Few users will use both and those who do are likely to be familiar with both conventions.

Continuity

What is continuity?

In the film industry, continuity editing ensures that different shots flow in a coherent sequence, even if they were filmed in a different order. It would be disrupting to the narrative if a character's hairstyle changed within a scene, furniture moved around, or a broken window was suddenly intact again[15].

In cross-platform interaction design, continuity refers to the flow of data and interactions in a coherent sequence across devices. The user should feel as if

[15]Spotting continuity errors in movies is a sport. Sharp-eyed viewers share the errors they have spotted on websites such as http://www.moviemistakes.com,.

they are interacting with the *service* through the devices, not with a bunch of separate devices. There are two key components here. Data and content must be synchronized, and cross-device interactions must be clearly signposted. In my experience, some of the biggest usability challenges in IoT are continuity issues.

Data and content synchronization

It sounds obvious that different device UIs should each give the same information on system state.

Kindle Whispersync is a great example of synchronization. You can switch between reading on different devices - even swapping between the book and the audiobook - and your place in the book is always up to date (see figure 9.27).

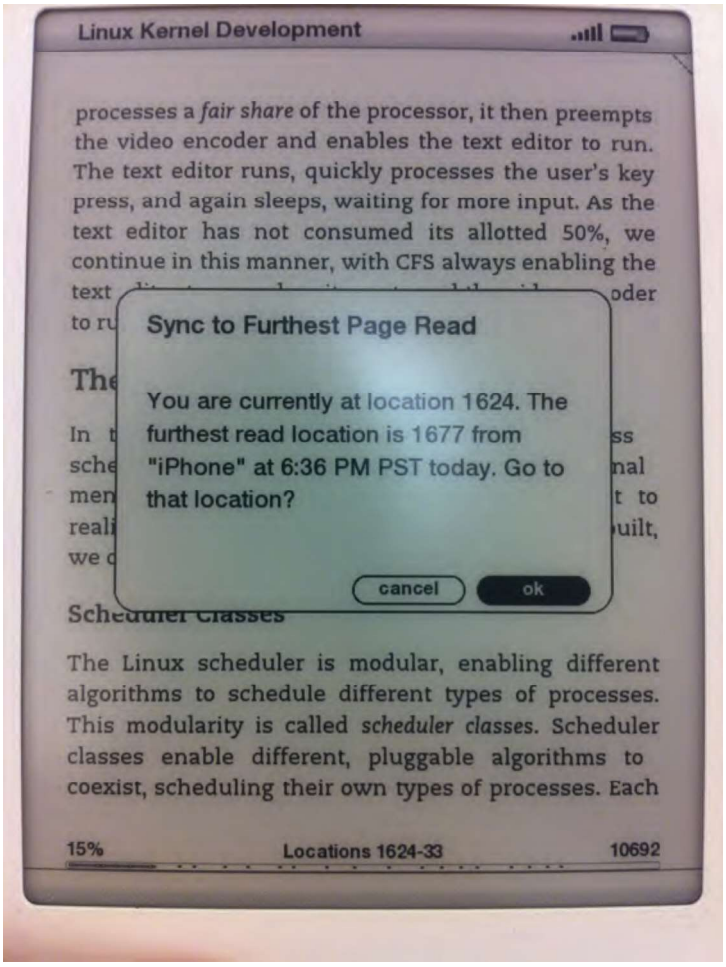


Figure 9.27: Kindle Whispersync UI dialog (Image by Kei Noguchi).

You'd expect this from any other connected device too. For example, your wall thermostat says it's 21C and the heating is on. You'd therefore expect your smartphone heating app to say the same thing, and not to tell you that it's actually 22C and the heating is off.

If you turn the heating off from the wall, then you'd expect the smartphone app to reflect that change in state right away. Right?

Unfortunately in IoT, as discussed in chapters 2 and 3, this isn't always possible. Devices that need to conserve power, such as those that run on batteries, often cannot maintain constant connections to the network as this

uses a lot of power. Instead, they will connect intermittently, checking in for new data. This can cause delays and result in situations where some interfaces do not reflect the 'correct' state of the system. Network latency is also an issue: it's possible for the user to know that something has worked before the UI does. For example, they may be physically sitting near a light that they have just turned on and have to wait for a smartphone app UI to tell them what they already know.

To return to the heating example: in the UK, it's common for heating controllers to run off a battery[16]. So a heating controller may need to connect via a low powered network like ZigBee to a gateway, and only connect intermittently to check in for new instructions. There might be a delay of perhaps two minutes between a setting being changed on the smartphone app and the heating controller receiving that instruction.

This causes discontinuities in the UX. If a user changes the settings on the smartphone app (say, turning the temperature up from 19C to 21C), there may be a period of up to two minutes before the heating controller checks in to the service and receives the updated instruction. During this period, the phone UI could show that the system is set to 21C, and the controller UI will show that it is set to 19C. If the user is standing in front of the controller with the smartphone app, they will see two conflicting pieces of information about the current status of the system (see figure 9.28). This violates one of the most fundamental of Nielsen's usability heuristics, visibility of system status: "The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."[17]

---

[16] UK heating engineers prefer battery powered wireless controllers as they can be installed easily and sited anywhere without risk that rewiring will be needed. In the UK, mains power is 240V AC and any mains electrical work must be done by a qualified electrician. Even replacing an existing mains controller in the same location requires an electrician. This isn't an issue in the US, where HVAC controllers typically run on a special low voltage circuit, making them safe for homeowners to install themselves. This means that in the US, it's feasible to offer a controller that maintains a constant connection to a WiFi network, and the system can always be in sync.

[17] Nielsen, J. (1994b). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), *Usability Inspection Methods*, John Wiley & Sons, New York, NY.

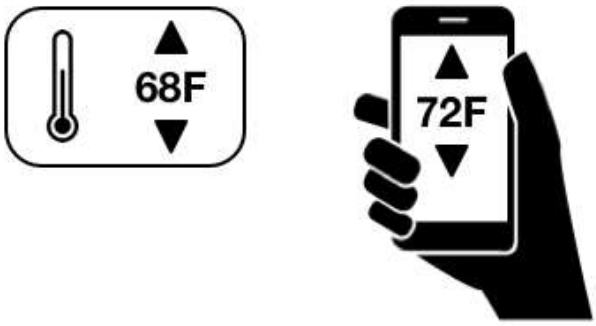


Figure 9.28: In some situations, devices may temporarily report conflicting information about the state of the system. (Hand/phone by Siddarth Dasari, thermometer by Saneef Ansari)

What can you do about this? You could fix this by making the controller check in more frequently. But that would run the battery down within days. Users don't expect to have to change batteries in heating controllers several times a week. So that's not practical.

Your next option is to consider how you can design the smartphone UI to account for this two minute period. There are two possible approaches.§

Firstly, you could show the updated settings the user wanted to apply: the temperature setting of 21C, even though it might (for a short time) give a misleading impression of the system state. If the instruction cannot be applied for some reason (e.g. temporary internet outage at the property), you can alert the user and then revert the UI to the old state. In essence, you pretend that it has worked while you wait for confirmation from the controller.

Instagram employ similar 'white lies' to make their mobile app feel more responsive. For example, Instagram registers likes and comments in the app UI while the request is still being sent to the service. The user is notified if the action fails. They call this 'performing actions optimistically'[18] (see figure 9.29).

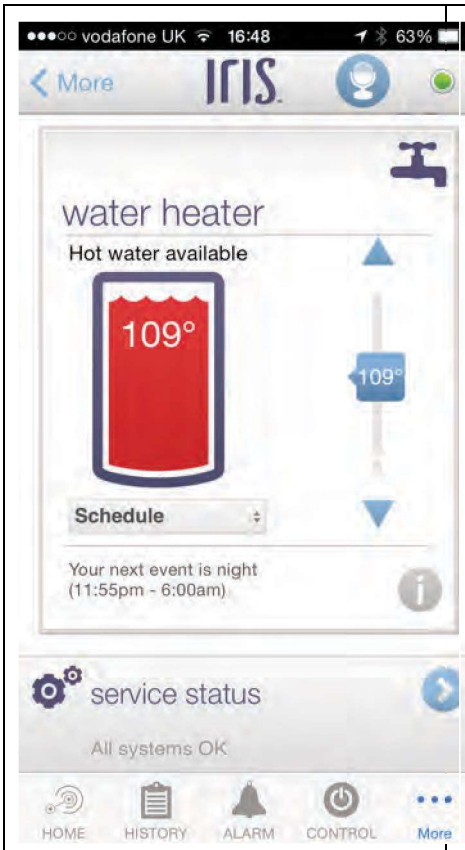
---

[18] Krieger, M 'Secrets to Lightning-Fast Mobile Design', Warm Gun conference 2011. https://speakerdeck.com/mikeyk/secrets-to-lightning-fast-mobile-design

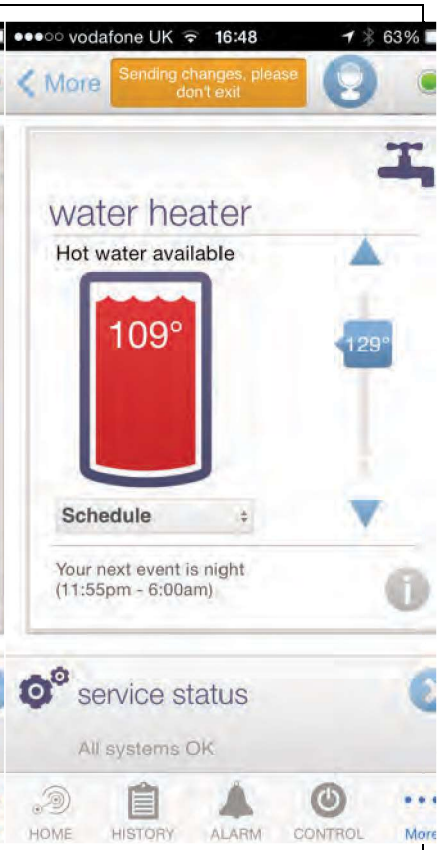
Figure 9.29: Instagram registers likes or comments in the UI even while the status bar spinner shows the request is technically still being sent

When everything works OK with our heating example, the responsiveness allows the user to feel as if they are interacting with the *service*, not just the phone UI. They have direct control of the heating.

The second approach is to be more transparent about what is technically happening. You show the instruction as being in the process of being sent. This is the approach used by the Lowes Iris system (as also shown in chapter 3): a status message at the top of the screen is shown to indicate that an instruction is being sent. When confirmation is received that the controller received the instruction, a confirmation message is displayed (see figure 9.30).



The temperature is set to 109F. The user turns this up to 129F



The system registers that the new temperature should be 129F, and that the current temperature is still 109F. A status message indicates that the instruction is still being sent.

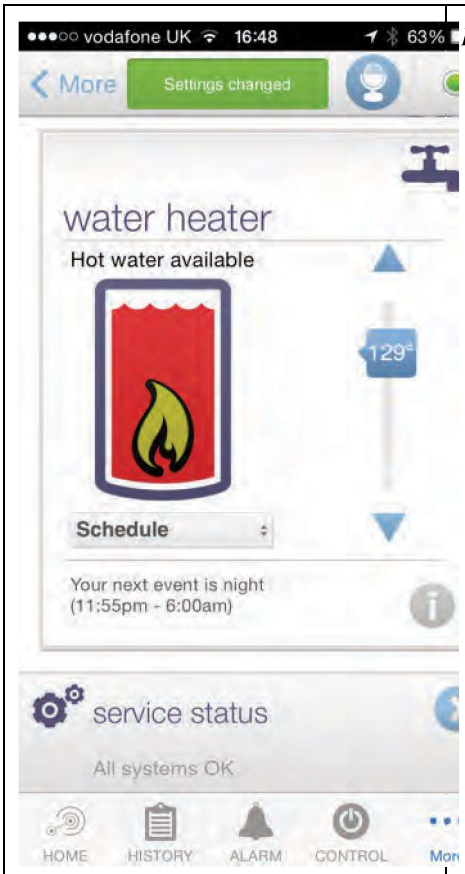


Image 1:

A status message confirms that the target temperature has been changed. The water heater has turned on to heat the water to 129F.

Figure 9.30: The Lowes Iris water heater UI, showing the status message

Here, the UI is showing the data as being in the process of being sent. In essence, the system is saying to the user: 'thanks for your instruction, let me see whether I can do that'. This requires the user to know a little more about how the system works in order to understand *why* the instruction isn't just acted upon. It also introduces the possibility of failure to every interaction, successful or not.

There are no standards here yet, and no right or wrong answer for every situation. In our first example, the primary use case for the system is to support remote access. If the user is out and about and turns the heating on remotely using the phone app, the 2 minute delay is not noticeable. The house will be warm when the user gets home. Even if the user is in, heating is the type of system that operates on a timescale of hours, so unlike a light switch, down to the second responsiveness isn't necessarily needed. If the user is standing in front of the heating controller with the smartphone app, it may be confusing, but the compromise may be acceptable.

In other situations, any delay or uncertainty about whether a command has been executed might be dangerous. For example, a person who presses an emergency alarm button must be absolutely confident their call for help has been sent and received. In this case, the UI should not make it appear that the system has received and acted on their command until it has definitely done so.

The frequency with which data is synchronized around the system can heavily shape the user value of the service. For example, dual fuel smart meters monitoring gas and electricity usage may report data at different frequencies for each fuel. The device monitoring electricity usage can run on mains power, so it can report data every few seconds. However, it would be dangerous to place a mains powered electrical device on a gas pipe. So the gas monitoring device will be battery powered. To maintain acceptable battery life, gas data will be reported less frequently than electricity data, perhaps only every 30 minutes.

With live electricity data, users can turn devices on and off and use a display (see figure 9.31) or smartphone to view almost immediately the energy impact each device had. The system can be used to understand the energy consumption of specific appliances and behaviors, such as boiling a kettle, or turning on a clothes dryer.

Figure 9.31: an in-home display

With gas data in half hour chunks, it's harder for the user to relate consumption to specific gas consuming activities in the home. You can't see the immediate impact of turning on a gas cooker. Was the last half hour's consumption high because the oven was on, or because the heating or hot water was in use? So the system data does not directly answer the question

‘how much gas does my oven consume’? As relatively few activities in the home run on gas as compared to electricity, it’s possible for users to make some rough guesses from the data. For example, if no one is at home but gas is being consumed, that might mean that the heating is on (and thus that the schedule should be changed). But for more detailed insights, the system would need to analyze longer-term patterns in gas consumption data and estimate likely usage by appliance.

When it’s not possible for all system data to be perfectly synchronized or ‘live’, it’s important to indicate how old data or status information may be. For example, you might show a timestamp for a sensor reading, or the time that the latest status information was received.

In the energy monitoring example above, it’s important that users understand that the two energy readings are not equally ‘live’. You could display a timestamp for each reading, but you might also choose to display information in a different format. You might use a line graph for electricity (because you have near continuous readings) but a bar chart for gas, where readings are only intermittent. (See e.g. figure 9.32).

Figure 9.32: gas vs electricity displays showing timestamps. (mock up)

It’s important to ensure that system status information is as accurate as it *needs* to be for the context of use. In a safety critical system, it should be clear when data may be out of date or an instruction may not yet have been received or acted on. A remote door locking system should not pretend that it has locked the door until proven otherwise! Perhaps the biggest challenge design-wise is how to design these behaviors for a system that needs to do multiple things with different responsiveness demands, such as heating, lighting and safety alarms. It’s safest to err on the side of communicating what is actually happening, but in some circumstances that may feel inelegant.

Handling cross-device interactions and task migration

Cross-device interactions require users to switch between devices in order to achieve a goal. Examples might include syncing data from a wearable fitness tracker to a smartphone, or connecting home sensors to a gateway.

Transitions between devices should be smooth and well-signposted. The word ‘seamless’ is often used in cross-platform UX, but it’s probably misleading.

Where a task requires the user to interact with more than one device, they *need* to be aware of the seams: the different role of each device, and the point at which the handover happens. This is especially important to help reinforce the user's mental model of the system, and what each part does. The less familiar they are with it (e.g. during setup when devices are new and unfamiliar), the more explanation is required. Below, we set out some key requirements for effective, usable cross-device interactions.

In the first place, the user needs to know that they *need* to switch to another device to complete their intended task. They may have to identify the correct device from amongst several: for example, there may be several identical light bulbs. Then they need to know what they're being asked to do, and any information that's needed to interact effectively with the other device. They also need to know why they're being asked to switch. For example, are they transferring data, or pairing the devices?

For example, the Misfit Shine syncing process tells the user to place or tap the Shine on the iPhone screen (see figure 9.33). (Data is transferred over Bluetooth LE but the sync is initiated by the phone recognizing the Shine on the touchscreen).

Figure 9.33: Misfit Shine syncing (https://www.youtube.com/watch?v=wmUOczrb9J4 - http://i.ytimg.com/vi/wmUOczrb9J4/maxresdefault.jpg)

The Bluetooth pairing process to connect a Jaguar car to a smartphone displays a 4 digit code on the dashboard that needs to be entered in the phone (if not already displayed) (see figure 9.34). (See chapter 12, Key interactions, for more on Bluetooth pairing interactions).

Figure 9.34: Jaguar syncing with Bluetooth device (low res grab from jaguar.com, get high res when Allan back from holiday).

The user also needs to know what reaction to expect from the other device. This is especially important if it has a very limited UI. For example, hitting a button on the web UI may make an LED flash for two seconds on a motion sensor to help you know which one it is, but you need to know where to look (see figure 9.35).

Figure 9.35: Identifying a sensor from the AlertMe web interface

If the interaction is not an integral part of a process (i.e. not something the user *has* to do), provide enough context/content to enable them to decide whether

it's important right now. For example, the Pebble Smartwatch can notify the user of new emails, texts and Twitter alerts, and shows some of the content (see figure 9.36) . The user might not be able to see the whole message, but there's usually enough information to decide whether it's important to get out the phone and read the whole thing there and then. A wearable that only tells you if you have a message, and not who it is from or what it might be, would not offer much over the phone's audio alert or vibrate function.

Figure 9.36: A Pebble notification

With a multi-device interaction, it is very easy to lose track of your progress in a task, or for one or more devices to lose connectivity. Where possible, design for interrupted use. Try to avoid locking users into lengthy processes (such as setup) which must be completed in one sitting or in a specific order. Provide some flexibility: if the user has to break off and return later, don't lose their progress - allow them to resume part way through. Guide them back to the parts that need to be completed when they return. For example, a home automation system setup process might require users to associate a gateway with an online account and then pair devices. If the user is interrupted after creating the online account but before pairing the devices, make sure that when they log in there is a clear route to resume and add devices, not just a blank screen! (See chapter 12, Supporting key interactions, for more information on designing effective setup experiences).

Broader contexts of interusability

This chapter has focused on cross-device digital interactions. However, these principles can also be applied to more holistic service design thinking across both on- and offline interactions. This broadens the focus of UX to include marketing and sales materials which set expectations of what the product does, packaging and setup guides which shape initial impressions of the UX, to customer support.

As with cross-device interaction design, the individual parts can be good, but if they don't work together well the overall experience can still be unsatisfactory or confusing.

You might consider composition when figuring out which setup instructions to put onscreen and which in a print booklet. You would need to consider

consistency of language, information graphics and aesthetics across online and print materials. And you might also need to consider the continuity of any processes that require users to refer between materials. Again, setup is a key example: if your instruction booklet says 'now the LED will blink for 2 seconds', that's a pointer to look at the other device. (Setup is covered in more detail in chapter 10, Key interactions). Setting user expectations accurately is also a form of continuity: if your marketing materials highlight a feature, it should be easy to find the UI. If not, that's a form of discontinuity.

The interusability model may not be complete for the broader service context, but I have found it useful for thinking about interactions than span digital and non-digital media.

Chapter summary

Conventional usability/UX is concerned with interactions between a user and a single UI. Interusability deals with interactions across multiple devices. The aim is to create a coherent UX across the whole system even when devices have very different characteristics.

Users need to form a clear mental model of the overall system, although it can be challenging for them to understand the interconnections between devices.

Designers need to distribute functionality between devices, to suit the capabilities of each and context of use (*composition*).

They also need to determine which elements of the design should be *consistent* across which parts of the system, for example terminology, platform conventions, aesthetic styling and interaction architecture.

Data and content can sometimes be out of sync around the system, causing *continuity* issues. Designers may need to find creative ways of dealing with this in the UI. When interactions begin on one device and switch to another, clear signposting is needed.