Thingies for Dummies

David Thomas
Author
dtho@itu.dk

Egil Hansen
Author
ekri@itu.dk

ABSTRACT

Author Keywords
Guides, instructions, authors' kit, conference publications.

ACM Classification Keywords
H.5.2 Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Design, Documentation, Economics, Experimentation, Human Factors, Languages, Management, Measurement, Performance, Reliability, Security, Standardization, Theory, Verification.

INTRODUCTION
More than 20 years after Weiser first defined ubiquitous computing [7] we have yet to see smart homes become commonplace. We believe this is largely due to smart home technology being inaccessible and too complicated for everyday users, and both price, perceived lack of advantages, and challenges in learning new technology scare users away.

In this paper we propose a smart home infrastructure that allow homeowners to cheaply bring new smart home technology into their existing homes in small batches, technology that homeowners will realistically be able to install and manage themselves. Our proposed infrastructure bases itself on hardware that exists in most homes today – Wifi and smart phones – and uses simple and open protocols like HTTP and the WebSocket protocol to provide the communication backbone of the infrastructure. Each individual device, or "Thingy", as we have dubbed them, is supposed to be simple to install and use, but provide value nonetheless. We call this concept *Thingies for Dummies*[1].

Edwards and Grinter describe, in their 2001 paper[3], seven challenges facing the adoption of smart home technology[2], and we believe that through the overarching theme of 'simple' in our proposal – simple infrastructure and simple Thingies – we are able to answer the first six challenges, at least partly. Simplifying does imply giving up on the dream of a fully automated intelligent home for the time being, but our proposed infrastructure does not prevent a home from becoming gradually more automated or intelligent as developers become better at utilizing the platform.

When we started doing research for this project we first wanted to see if we could come up with a few use cases where some smart home technology would add so much value, that a normal homeowner would go out and invest time and money to retrofit the home. Through our informal questioning of friends, colleagues, and family, it quickly became clear that promise of the classic intelligent appliances we see in many office buildings that automatically adjusts their operations through context awareness[3] did not have a strong enough appeal. The only thing that really resonated was various remote control scenarios. Examples included double checking if all the lights are turned off at home, remotely starting the washer when leaving from work so clean cloth is ready to be hung up to dry when the user gets home, and remotely unlocking the front door to the home if a friend is waiting and the owner is running late. With that in mind, we created the following scenario to guide our design process:

After putting his children to bed, Peter is ready to go to sleep as well. He sits on his bed and picks up his phone and navigates to his Home Remote app, where he selects the "Lock down" preset. Once selected, the app sets all the homes actuators into their specified lock down mode, i.e. all doors are locked, lights are turned] off, curtains are drawn, and the array of sensors in the house reports back what they see. Peter scrolls through the list and sees a warning indicating an open window in the kitchen. Peter stands up and walks out of the bedroom towards the kitchen. As he walks with his phone in his hand, the management software turns on the light around him and unlocks the doors he is near, while keeping everything else in lock down mode. Once the

[1]This is a reference to the *For Dummies* book series that tries to present non-intimidating guides to various topics and that always contains the subtitle "A Reference for the Rest of Us!".

[2]Challenge One: *The 'Accidentally' Smart Home*. Challenge Two: *Impromptu Interoperability (islands of functionality)*. Challenge Three: *No Systems Administrator*. Challenge Four: *Designing for Domestic Use*. Challenge Five: *Social Implications of Aware Home Technologies*. Challenge Six: *Reliability*. Challenge Seven: *Inference in the Presence of Ambiguity*.

[3]Appliances such as HVAC, lights fixtures, window curtains, and automatic doors.

Submitted for evaluation in the Pervasive Computing Lab class at IT University of Copenhagen, Spring 2012.

window in the kitchen is closed, the sensor indicator on his phones turns green Peter returns to his bedroom for a good nights sleep.

Even if this scenario does not describe a remote control scenario where the user is many kilometers away, it is still remote by our definition. We consider anything remote if the user is able to control a Thingy without being able to observe how it affects its surroundings. For example, if user A is sitting in room A, and remotely controlling a lamp in room B, and user A is unable to see what is going on in room B, i.e. see the physical lamp turn on and off or see if there are people or other appliances in room B that react to the lamps state change, then it does not matter if room A and room B are right next to each other or on opposite sites of the word, it is remote controlling scenario because user A has the same information available in both cases.

In the next section we will describe our proposed infrastructure in details and discuss the design. Following that, we will present related work and end with our implementation, evaluation and discussion of the prototype we built.

THINGES FOR DUMMIES INFRASTRUCTURE

The key design decision for *Thingies for Dummies* is that it should be simple and inexpensive to get started retrofitting an existing house with thingies and that it should be independent of specific hardware and software platforms. This means not requiring a dedicated, centralized controller, and instead to rely on the existing infrastructure in the home for handling communication and control, i.e. a Wifi network and smart phones. By doing so, we limit the amount of new hardware required to get started, since most homes will already have a Wifi network and at least one smartphone. This lowers the price and reduces the environmental impact.

In the rest of this section we will describe each of the critical elements in our proposed infrastructure that enable this and discuss our design decisions.

Networking, protocols, and Security

With so many networking technologies care must be taken before choosing one above another, however, we arrived at our choice, Wifi, pretty fast. One of our primary requirements was that the network should exist in most homes already. This eliminates quite a few choices right away, leaving us with power lines, which the X10 protocol uses, cellular networks such as 3G, phone line wiring and Wifi. Using power lines, which usually reaches all rooms in a home, has problems with the boundary of the network where since data signals can travel out of the home. It is also inherently a wired network, which means an extra bridge device is needed between the power lines and mobile devices if those are to interact with devices connected to it. The phone lines inside the home has similar problems, and is also far less ubiquitous in the home, with most homes only having a few phone plugs. This leaves cellular networks, which are indeed wireless, but also owned and maintained by carrier companies, that charges fees for the data that travels on their network. This makes cellular networks less ideal choices, landing us squarely on Wifi as the preferred option.

There are many advantages to choosing the Wifi as the basis for our infrastructure and a few drawbacks. Users get to reuse their existing Wifi in their home, thus saving money and time setting up a new device. Wifi also adds, assuming it is protected with a passphrase, a virtual boundary to the smart home, a boundary that users understand because they understand that you cannot access the Wifi without the passphrase. Wifi also have the advantage of being a high bandwidth network which enables thingies such as high resolution CCTV cameras.

The biggest drawback related to Wifi is that it is power hungry compared to other technologies such as ZigBee. However, there is a general focus from the Wifi radio chip makers to lower power consumption due to the massive surge of mobile devices the last few years, so the drawback will matter less as Wifi radios become more power friendly. In addition, for many of the scenarios we have found users to be interested in, Thingies will have a permanent power source. In the case where low powered sensor thingies are required, we expect these to use a protocol like ZigBee[1] and be bundled with a special purpose Thingy which act as a bridge between the individual ZigBee enabled Thingies the Wifi network. The bridge Thingy could also perform other tasks so the extra cost would be negligible.

We also discussed the problems with oversaturation and general instabilities of the average home Wifi network. This is of course very dependent on the model of the wireless access point used. Better models with good Wifi radios, antennas, faster CPUs, and enough memory, can handle around 25 concurrent connected clients, while lesser models choke at at half as many connections. However, since one of our design goals was that it should be as inexpensive as possible to get started with smart home technology, we do not expect a average home Wifi to get saturated right away. The scenario where a homeowner invests in over 20 new thingies at the same time might not be unlikely, but in that case we feel that investment in an extra access point to handle traffic from Thingies is probably also within the budget. It is common knowledge among Wifi experts that Wifi clients also have a tendency to drop their connection from access points more often if using the higher bandwidth 802.11n and 802.11g protocols compared to the slower 802.11b. The solution here is to make Thingies use 802.11b where possible, and since only a very small subset of Thingies will require a 100% stable connection, homeowners should experience no problems since Thingies can simply reconnect if their connection drops.

Communication protocols

For communication, we use standard HTTP [4]. This opens up for a huge list of possible controllers, basically any type of computing device or library that supports HTTP. This is especially important to us since we do not want to be limited to a certain framework, language, or platform, e.g. Java[4] or

[4] `http://java.com`

Android[5]. Other great properties of HTTP is that it is fairly lightweight and stateless protocol, which simplifies things considerably on the Thingies. HTTP also supports our goal of an open infrastructure; it might very well be the best supported protocol among developers, and this enables the average app developer to build new functionality on top of the infrastructure on their platform of choice. Another benefit is that HTTP communication is very unlikely to be blocked in firewalls compared to other less used protocols, so there is a much bigger chance that things will just work.

However, HTTP's statelessness [4] is also a weakness in common smart home scenarios, where a context change in one Thingy should trigger an event in another Thingy. With basic HTTP, a Thingy (client) waiting for a state change has to continuously poll the other Thingy it is monitoring to discover the state change in a timely manner. This is an waste of resources and could potentially deplete the resources of a Thingy entirely, especially if more clients are waiting for changes. Clearly, relying only on HTTP alone does not scale. Several solutions to this problem were considered, the biggest contender to be considered being multicast. The advantages of using multicast is that Thingies can simply announce state changes on the network and it would not matter if zero, one, or hundreds of clients were interested, the amount of resources required to do so would be the same. Ignoring the problems associated multicast on Wifi, i.e. lost multicast packets are not retransmitted at the MAC layer [2], the reason we did not choose multicast was that the protocol works on a lower level compared to HTTP, and that would limit the number of supported development platforms. We wanted to stay in the Web-stack of technologies so we chose the WebSocket protocol. In the recent years, the WebSocket protocol has been standardized by the Internet Engineering Task Force (IETF)[5] and is in the process of being standardized by the World Wide Web Consortium (W3C)[6]. The WebSocket protocol supports multiplexing bi-directional, full-duplex communications channels over a single TCP connection and is already supported by the latests versions of most major browsers[6], with numerous frameworks and libraries also on the way. The WebSocket Protocol was essentially designed to be a solution for the scenario where a server needs to push information to a client[5] which web developers have been struggling with for many years, so while it might not scale as well as multicast, we feel this is the best overall approach.

Service Discovery

Most home networks do not have traditional unicast DNS server set up, so we need a different way to discover Thingies connect to the network, both during installation and also if and when their IP address changes. We have decided to use mDNS/Bonjour[7] since it offers wide support across platforms and is relatively lightweight.

[5] `http://android.com`

[6] The WebSocket protocol is supported from Chrome 16, Firefox 11 and Internet Explorer 10.

[7] Bonjour is available on `https://developer.apple.com/opensource/`

Security considerations

The basic trust model for Thingies for Dummies is based on the security of the home's Wifi. Assuming the Wifi is probably configured using WPA2 with AES encryption and a reasonable passphrase, there is a clear boundary for the smart home, and an adversary would not be able to pick up data sent to and from Thingies without being connected to the network. This also means, that anyone on the network have full control over the Thingies attached to it, which can be a problem in some cases. We have discussed the possibility of adding an extra layer of security that would handle such a case. If the concern is only that unauthorised users cannot perform certain actions on a Thingy, i.e. only change state, not update configuration, then we need a way to authenticate each user. This could be done using certificates that clients use to authenticate with. The certificate would be generated during the initial configuration of a Thingy, and depending on the number of authorisation levels needed for a Thingy, e.g. read only, update state, and update configuration, one or more unique certificates would be generated. Then it would be up to the controller devices used during the initial configuration to distribute the generated certificates to authorized users. However, it is not enough only to authenticate users, since the communication between users and Thingies are still being transmitted over an open channel which can allow an adversary to perform replay attacks. The solution to this is to secure the communication channel between users and Thinges, i.e. using HTTPS. Adding the extra level of security does raise the hardware requirements on Thingies which increases cost and power consumption, so an environment that supports both "open", low risk Thingies and "secure" Thingies should provide the best of both worlds.

REFERENCES

1. Z. Aliance. ZigBee zigbee aliance. `http://www.zigbee.org/`. [Online; accessed 1-May-2012].

2. R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye, R. Ramjee, and L. Ravindranath. Dircast: A practical and efficient wi-fi multicast system. In H. Schulzrinne, K. K. Ramakrishnan, T. G. Griffin, and S. V. Krishnamurthy, editors, *ICNP*, pages 161–170. IEEE Computer Society, 2009.

3. W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. pages 256–272. Springer-Verlag, 2001.

4. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.

5. I. E. T. F. (IETF). IETF the websocket protocol. `http://tools.ietf.org/html/rfc6455`. [Online; accessed 4-May-2012].

6. W. W. W. C. (W3C). W3C the websocket api. `http://www.w3.org/TR/websockets/`. [Online; accessed 4-May-2012].

7. M. Weiser. The computer for the 21st century. *Scientific American*, 3(3):3–11, 1991.