The Home Needs an Operating System (and an App Store)

Colin Dixon[†] Ratul Mahajan Sharad Agarwal

A.J. Brush Bongshin Lee Stefan Saroiu Victor Bahl

Microsoft Research [†]University of Washington

Abstract— We argue that heterogeneity is hindering technological innovation in the home—homes differ in terms of their devices and how those devices are connected and used. To abstract these differences, we propose to develop a home-wide operating system. A HomeOS can simplify application development and let users easily add functionality by installing new devices or applications. The development of such an OS is an inherently inter-disciplinary exercise. Not only must the abstractions meet the usual goals of being efficient and easy to program, but the underlying primitives must also match how users want to manage and secure their home. We describe the preliminary design of HomeOS and our experience with developing applications for it.

Categories and Subject Descriptors

C.2.4 [Distributed systems] Network operating systems

D.4.6 [Security and protection] Access controls

H.1.2 [User/Machine systems] Human factors

General Terms

Design, Human Factors, Management

Keywords

Home networks, operating systems

1. INTRODUCTION

The vision of smart, connected homes has been around for well over two decades. In this vision, users easily perform tasks involving diverse sets of devices in their home without the need for painstaking configuration and custom programming. For example, imagine a home with remotely controllable lights, cameras, windows, and door locks. It should be easy to set up this home to automatically adjust windows and lights based on the outside temperature and lighting or to remotely view who is at the front door and open the door. While modern homes have many network-capable devices, applications that coordinate them for cross-device tasks have yet to appear in any significant numbers.

We posit that the core issue is a failure to deal with heterogeneity across homes. Homes differ in terms of their devices and inter-connectivity as well as preferences for how various activities should be conducted [14]. Application developers are thus not only plagued by having to support many distinct devices, but also build configurability flexible enough to meet the demands of a majority of users. It should thus come as no surprise that there are few applications for the home today, save those provided by device vendors. But vendor applications often provide access to their own devices with little or no cross-device capabilities. For instance, electronic locks come with custom software [20] but little support for extensibility. Such vertical integration by individual vendors discourages device composition.

Current approaches for enabling cross-device tasks fall on the two ends of a spectrum. At one end are the efforts to improve basic device interoperability through standards (e.g., DLNA [10], ZWave [23]) and research efforts [11]. However, device interoperability alone is insufficient. Applications also need to support user preferences and coordinate device access. For instance, a security task may want to keep the windows closed at the same time as an energy conservation task wants to open them. Interoperability itself does not provide mechanisms to resolve such conflicts forcing the applications to provide it themselves. Such coordination needs significant engineering.

At the other end are monolithic systems that tightly integrate multiple devices for specific cross-device tasks. They include commercial security systems (e.g., ADT [1]) and research efforts [6]. However, such systems are hard to extend (especially by users) with new devices or tasks.

We argue for a fundamentally different approach for organizing home networks: the development of an operating system for the home. By masking heterogeneity across homes through appropriate abstractions, a HomeOS can greatly simplify application development. Further, users can manage their homes as a connected ensemble, by specifying their access control preferences globally. They can also easily enable new capabilities by installing new applications or devices. To simplify this task, inspired by Apple's App Store, we propose a HomeStore that is coupled with HomeOS. It helps users find applications that are compatible with their devices and find devices to enable desired tasks that cannot be supported by their existing devices alone.

While many OSes have been developed in the past, a unique aspect of HomeOS is that its success hinges on not only the suitability of its programming abstractions but also how well it lets users manage and secure their home networks. Unless it has the right primitives at its core, no degree of post-facto sophistication in user interfaces will be effective [2, 18].

We thus approach the development of HomeOS as an interdisciplinary exercise that spans human computer interaction (HCI) and network systems. Our intent is to learn the mental models of users and then design compatible abstractions. Towards that goal, we are conducting field visits to homes with automation systems. While our visits covered many aspects of home technology, in this paper, we focus on access control. We find HomeOS's access control needs to be different than those found in traditional OSes. It needs to incorporate a notion of time to allow users to restrict access to certain devices, for instance, at night. It needs to treat applications as first-order security principals to let an application be restricted from accessing certain resources, independent of the user that activates it. HomeOS should also provide users an easily understood view of security settings.

We capture these requirements by formulating access control policies as Datalog relationships on applications, devices, users, and time. The use of Datalog and the absence of complex primitives such as dynamic delegation means that access verifications are simple (and fast) queries. Datalog queries also provide users with different desired perspectives on their policies, e.g., list applications that can ever access the door lock.

Many additional abstractions in HomeOS borrow liberally from existing OSes. HomeOS logically centralizes the control plane of the home network [15, 13] and uses "drivers" to abstract the low-level details of devices and connectivity.

In place of the multitude of inter-process communication modes supported by current OSes, we build HomeOS using a single, simple abstraction. For this, we extend Accent ports [19]. Our ports enable the exchange of typed messages and can be queried for their functional description. HomeOS does not need to understand the semantics of this description. Such a design choice lets new devices and applications to be easily added in the future.

To evaluate our design, we have set up a testbed with a diverse mix of devices and are building applications that compose them in various ways. For instance, one of our applications composes a smartphone, a camera, a light, and face and speech recognition; another composes lights and speakers in multiple rooms and a media server. Preliminary experience shows that building applications using HomeOS is simple and that the applications have adequate performance for interactive use. We also plan to conduct usability studies of our system once its development matures.

2. THE STATE OF THE HOME

While each home is different, the dominant paradigm for technology in the home can be summed as follows. Users buy individual devices such as PCs, smartphones, game consoles, TVs, and cameras. Each device comes with its own application software that runs on the device or on a PC or smartphone. This software rarely leverages the functionality of other devices in the home.

The current paradigm is highly undesirable for users, application developers, as well as device vendors.

2.1 The user perspective

From the perspective of users, today's home networks have two significant shortcomings.

Integration hurdles: Despite investing in several sophisticated devices, it is hard for users to fully exploit the functionality that their devices are collectively capable of. Cross-device compatibility occurs only when two devices happen to implement the same standard (e.g., DLNA), are part of a monolithic multi-device system (e.g., cameras and locks for security), or are from the same vendor (e.g., XBox and Zune). No single vendor is even close to manufacturing all the types of devices found in homes today.

This lack of cross-device functionality frustrates users. For example, in our field visits, one user commented: "A lot of the products are kind of new and not an integrated vendor. It's like the Control4 [a home automation system] stuff is, you know, integrated but [then] you're locked in to their stuff. But if you're not locked into these things it's kind of hard to get those things to work together. So it's kind of a task to keep it all integrated and working."

Management nightmare: Each device comes with its own configuration and access control tools, and users must get familiar with different interfaces and semantics. With many devices present, this can be a nightmare [14].

2.2 The developer perspective

Given the potentially huge market, one might expect that writing applications for the home would be lucrative. However, there are few independent developers in this domain. Most software is provided by device vendors to use the functionality of their devices in certain fixed ways.

While writing applications for a specific home might not be hard, because homes are highly heterogeneous, the challenge is to write applications that work across a range of homes. We identify four sources of heterogeneity that drive up the engineering investment needed to make an application broadly useful.

Device heterogeneity: Different devices, even of the same type, support different standards. For example, light switches may use ZWave, ZigBee, or X10.

Topological heterogeneity: Devices are connected in different ways. Some homes have a simple topology based only on WiFi, but some have a mix of WiFi, Ethernet, and ZWave. Further, some devices may have different connectivity modes at different times. For instance, smartphone can use the home WiFi or the 3G network.

Heterogeneity in user preferences: Different homes have different preferences as to how various activities should occur [14]. For example, some might want the XBox to not run after 9 PM, and some might want the security camera to record only at night.

Coordination heterogeneity: If multiple applications are running, cases where two of them want to access a device at the same time may arise. Such simultaneous access may be undesirable in some cases. For instance, the lighting control application may want to open the window while the security application may want to close it.

2.3 The vendor perspective

Vendors (especially smaller ones) want their devices to become broadly adopted especially since users desire device integration inside their homes. However, heterogeneity hurts vendors as well. As a result, they tend to vertically integrate devices and software, to provide a robust experience to users independent of the environment. Abstracting heterogeneity in the home will likely reduce vertical integration and enable device composition as well as reuse across multiple tasks.

3. HomeOS AND HomeStore

To address the problems above, we call for a different paradigm for home networking. In particular, we propose the development of a HomeOS and a HomeStore. This section outlines our vision. Later sections describe the challenges and our efforts towards realizing this vision.

The goals of HomeOS are to simplify the management of home networks and the development of applications. It accomplishes these goals as follows. First, it provides one place to configure and secure the home network as one connected ensemble. Users do not have to deal with multiple different interfaces and semantics.

Second, it provides high-level abstractions to applications. Developers do not have to worry about low-level details of devices and about device inter-connectivity. HomeOS is responsible for enforcing user preferences for device access and coordination, which does not have to be supported by individual applications. For example, if a user dislikes noise at night, she can disable night-time access to all speakers; HomeOS will then automatically deny access to all applications that try to use the speakers.

With HomeOS, users enable new tasks by installing new home applications. Because homes are heterogeneous, this process must be streamlined such that users do not inadvertently install applications that will not work in their homes. For instance, if an application for keyless entry requires a fingerprint scanner, users without such devices should be warned against purchasing such an application.

Inspired by the iPhone model, we propose that HomeOS be coupled with a HomeStore to simplify the distribution of applications and devices. The HomeStore verifies compatibility between homes and applications. Based on users' desired tasks, it recommends applications that work in their homes. If a home does not have devices required for those tasks, it recommends appropriate devices as well. For instance, if a user wants integrated temperature and window control, the HomeStore can recommend window controllers if there exists an application that combines those window controllers with the user's existing thermostat.

In addition, the HomeStore can perform basic quality checks and support rating and reviewing to help identify poorly engineered applications and devices. We do not intend for the HomeStore to become the sole gatekeeper for home applications. Towards this end, we allow for multiple HomeStores, and users can visit the one they trust most.

4. CHALLENGES

There are many technical challenges in realizing the vision above. In this paper, we focus on two main ones.

The first is to design appropriate primitives that enable users to effectively manage and secure their home networks. Security must be a prime focus because applications are contributed by independent developers. Applications should not be able to, for instance, inadvertently open the door at night or make home videos public.

The primary requirement for these primitives is their compatibility with users' mental models. Recent studies show that the primitives in current OSes are insufficient even for settings [2, 18] simpler than the ones HomeOS targets. We are conducting field studies to understand what primitives are likely to work well in this setting.

The second challenge is to develop abstractions for communication and security against which applications are written. These abstractions must be flexible enough to abstract heterogeneity, yet simple enough to develop against.

A key requirement for HomeOS abstractions is that they not hinder future innovation. It should be easy to support application and device types that do not exist yet. It should also be easy for applications to exploit new or differentiating features of devices; otherwise vendors will have little incentive to invest in new features or make their devices compatible with HomeOS. This requirement implies that HomeOS must make minimal assumptions about the the nature of devices and applications. More assumptions imply a greater chance that a future technology will be hard to support. It follows that HomeOS should not rely on understanding the semantics (e.g., content formats) of device functionality or of messages exchanged between devices and applications.

Enabling future innovation and simplifying application development can be in conflict. In general, richer abstractions lead to easier application development but simpler ones make fewer assumptions. Our design errs towards simpler abstractions to not hinder future innovation.

5. PRELIMINARY DESIGN OF HomeOS

We now describe our progress towards meeting the challenges above. In addition to developing our system, we are conducting a field study of homes to inform our design.

5.1 Field study

We are visiting households with home automation (e.g., remote lighting control, security cameras and automatic door locks). While we expect HomeOS to enable tasks not possible today, these households can give us first hand insight into how they use the current technology and the challenges they face. We recruited households with a range of systems such as Elk M1, Control4, and Leviton.[1]

Each visit had two main parts: i) questions about their technology and experience with it; and ii) introduction to the HomeStore concept and questions about applications and access control. Each visit took roughly 2 hours. We analyzed the interviews using affinity diagramming [3].

We have completed visits to six homes (12 people) in the USA. While we continue additional visits, insights from these initial visits have already influenced our design. For example, we observed the unique configurations in each home and the challenge of integrating devices. Given that four of our homes had installed their own systems, but still struggled, it seems that integration and management tasks are challenging even for highly technical users.

Our visits reveal that households want access control primitives that differ from those present in traditional OSes. First, when speaking with participants about how they manage access to devices today, it became apparent that they wanted a notion of time. Parents mentioned restricting children from using certain devices after certain times (e.g. "If my daughter wanted watch [Curious] George at 11 o'clock at night, I wouldn't want to do that"). While social interaction may suffice in some situations, many parents asked for technical means. A richer notion of time was also desired in order to allow households to grant a variety of access durations for guests (e.g. a few hours to babysitters, and a few days to house guests).

Second, asking about concerns when buying applications highlighted a desire for the ability to limit applications to certain devices. One participant said "I don't want to grant it [the application] access to everything, just my laptop." A participant in a different home commented about another application: "if it said my DVR and my TV I would say fine, ... if it had my phone or my computer I would want to be able to choose [what it can access]." This observation motivates our design choice to treat applications as first-order security principals. In current OSes, users and resources (e.g., files) are the typical security principals and applications simply inherit users' privileges.

Finally, people showed clear differences between their level of sensitivity for different devices and a strong desire to ensure the security of some devices. For instance, a participant with electronic door locks said he had not hooked up remote access because he was not 100% certain of its security. HomeOS must be able to provide users a reliable, easy-to-understand view of their security settings, with provisions for being able to focus on the settings for sensitive devices.

[1]While such technology has been around since 1970s, the number of installations is quite small. Approximately 215K systems were shipped worldwide in 2008 [17].

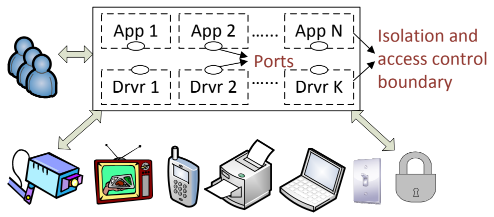

Figure 1: An overview of HomeOS.

5.2 System design

We describe now the design of HomeOS, limiting ourselves to a high-level overview. While many key pieces of our design are in place, it is far from complete and we continue to refine it based on our development experience. Figure 1 shows an illustration of HomeOS. In its first iteration, we envision it as running on an always-on appliance in the home.

HomeOS is a logically centralized system to allow for greater flexibility in control policies and avoid the fragility of distributed logic [13]. The data plane is not centralized. When devices share the same interoperability protocol (e.g., DLNA-capable network TV and media server), data streams flow directly between devices.

Logical centralization is enabled by drivers that logically incorporate the device into HomeOS independent of network topology. From the perspective of applications all devices connect to HomeOS. directly, which greatly simplifies application development. Conventional control of devices (e.g., turning off a light switch) is still permitted; drivers detect and notify interested applications of such events.

HomeOS can work with existing devices, as long a driver can communicate with them. Any protocol, whether proprietary or standardized, may be implemented between a driver and its devices. Drivers may be written by device vendors or independent developers. If a corresponding driver is available, adding a device to HomeOS should be close to a plug-and-play model.

As its core features, HomeOS offers: i) driver and application modules; ii) a "port" abstraction for exposing functionality and communication; and iii) access control for users and modules. HomeOS has other facilities, such as device discovery, but those can be easily replaced by different versions. HomeOS is built on a modern language runtime (the .NET Framework). All modules are implemented in a type-safe manner, using a language supported by the runtime.

Modules Driver and application modules are isolated in HomeOS, so that a poorly written module cannot impact HomeOS or other modules. To achieve isolation along with performance that is adequate for interactive use, we use a lightweight isolation boundary called "application domains" provided by our runtime. Direct interaction is not allowed across domains, and only typed objects can be communicated through pre-defined entry points.

Ports Modern OSes have a variety of communication abstractions such as pipes, signals, sockets, shared memory, and more. Instead, we want a single, simple abstraction that meets our needs (§4). While the simplest possible abstraction is the Unix pipe, it is too simple; for instance, it only allows file-like data to be exchanged. Perhaps the next simplest one is the Accent port [19], which allows typed messages to be exchanged.

We thus decided to use ports but HomeOS ports differ from Accent in one key way: they can be queried for their functional description and location (e.g., living room). A module that wants to expose certain functionality registers port(s) with HomeOS. Modules can query registered ports and decide which ones to use. Unless access is restricted (see below), modules can make use of a port's functionality by sending and receiving messages.

A port is functionally described in terms of *roles* and *controls*. Roles are text strings that express a general functionality, and controls are typed points of sensing and actuation within a port. For instance, a dimmer in our testbed is described as <roles="lightswitch", "dimmerswitch">, <controls=("on-off", binary, readable, writable), ("intensity", range:1-100, readable, writable)>. This port functions as a light and a dimmer switch, and it has two controls of types binary and range. Modules read from, write to and subscribe to changes from controls by sending corresponding messages to the relevant port.

HomeOS does not need to understand the semantics of a port's functionality; only modules that want to use it need this ability. For example, only camera-based applications need to understand the functional description of cameras. New devices or device features can be supported by adding roles or controls to the functional description, without modifications to HomeOS.

Access control Our field study showed that access control in the home should have a notion of time and deem applications as independent security principals. We also wanted primitives whose semantics can be easily explained to users. This is difficult in the presence of complicated primitives (e.g., dynamic delegation) in modern OSes [8].

Based on these requirements, HomeOS access control policies are Datalog rules of the form (p, g, m, Tw_s, Tw_e, pri, a, T_s, T_e), which states that port p can be accessed by users in group g, using module m, in time window from Tw_s to Tw_e, with priority pri and access mode a. Time window is modulo a week, to let users specify recurring policies by which something is allowed, for instance, on Sundays 7-9 PM. T_s and T_e are absolute times when the rule should be activated and deactivated. They are used to grant temporary access, e.g., to guests. Groups such as "kids," and "parents" are defined separately. Priorities are used to resolve conflicting access to the same port. Access mode is one of "grant," "ask user," and "grant but notify admin" [2, 18].

Any access not explicit in the rule database is not allowed. When installing a module, users specify what it can access, when, and how (which can be changed later). To simplify this process, modules suggest what they need. In the event that there is a potential for conflict between the newly-installed and existing modules when accessing a port, the user is asked to rank the modules based on their desired access priority. We infer a partial ordering of all modules based on this ranking and use it to fill in values for pri in the access rules.

Runtime access is implemented using capabilities [16]. Modules obtain the capability to access a port via HomeOS which performs the access check and installs the capability on the target port. Capabilities have an expiration time based on the rules. If rules change, capabilities are revoked by uninstalling them from the target port.

We find great value in expressing access control as Datalog rules. Evaluating access legality is a Datalog query, which is fast despite there being many dimensions in each rule. Further, by keeping these policies straightforward and direct, we can provide users a reliable view of their security settings. They can ask questions such as "which modules can access the door?", "which devices can be accessed after 10 PM?", or "can a user ever access a device?" Such questions can be answered through a user interface that constructs Datalog queries based on users' input.

Access control also forms the basis for privacy in HomeOS. Modules cannot access sensory data from inaccessible devices. Additionally, the wide-area network is treated as another port; so, unless explicitly allowed, modules cannot relay information from the home to the outside world. (Software upgrades occur through HomeOS, without modules needing network access.) Thus, our current design coarsely controls privacy at the granularity of modules. In the future, we will consider finer-grained control by labeling data [21, 22].

6. CURRENT STATUS

We have implemented the design above and are currently evaluating the ease of developing applications using our abstractions and the performance of our system. We also plan to evaluate the usability of HomeOS after developing a more complete version with appropriate user interfaces.

To evaluate HomeOS, we have set up a testbed with a variety of devices found in today's homes and are implementing a range of applications and drivers. Thus far, we have implemented drivers for DLNA (a media standard), ZWave (a home automation standard), a video camera, and a Windows Mobile smartphone.

We have written three applications that use multiple devices: $i)$ a "sticky media" application that plays music in the parts of the house where lights are on and stops elsewhere; $ii)$ a "two-factor authentication" application that uses audio from the smartphone and an image from the front-door camera to turn on lights when the two inputs match a user; and $iii)$ a "home browser" to view and control through user interfaces all devices in the home. Because ports are self-describing, the browser enables control without understanding device semantics. Due to space constraints, we omit the details of these applications.

Briefly, we find that application development in HomeOS is relatively straightforward. Even though each application uses at least four devices, implementation took less than 3 hours and 300 lines of code each. Most of the effort went towards implementing application-specific logic for composing devices. Further, logically dividing functionality between drivers and applications created a natural division with easily reused code in the driver and more specific code in the application (as in current OSes, but not in homes).

Initial experiments show that HomeOS has adequate performance. On a 3 GHz dual core machine, the request-response latency across modules, which includes crossing the isolation boundary twice, is 4 ms. This latency is much lower than the 100 ms guideline for interactive use, which leaves enough time even if messages need to be relayed to devices by their drivers [12].

7. RELATED WORK

Our work builds on previous systems to enable rich applications in the home. We divide them into two categories. The first category is that of systems and standards that focus on interoperability [10, 23, 4, 11]. While interoperability helps with device heterogeneity, it is insufficient by itself. It does not help with heterogeneity in topology and user preferences or with coordinating device access across applications.

The second category is monolithic systems [6]. These systems are not easy to extend with new devices or applications because they do not separate the applications and the programming platform. They are also not useful for homes that do not have the minimum set of devices needed to enable their programmed applications.

Modern home automation systems (e.g., Control4 [9]) allow some extensibility but are close to being monolithic. They work only with devices that implement a particular standard (e.g., ZigBee for Control4). Further, they allow only a limited form of programming, based on rules such as upon event E, do task T; all expected events (e.g., a button press) must be declared in advance. This framework cannot express many desired tasks.

Calvert *et al.* [7] detail management challenges for the home network. Like us, they then argue for centralization. In contrast to their proposal, we focus on simplifying application development as well, do not centralize the data plane, and do not require device modifications. We also develop access control primitives and communication abstractions suitable for the home environment.

With a motivation similar to ours, researchers have proposed OSes over multiple devices in other domains. One such domain is ubiquitous computing environments or collaborative workspaces [5], where the goal is to simplify application development over devices such as displays and whiteboards. Another is enterprise networks, where the goal is to simplify the management of switches [15]. We aim to handle the complexities specific to the home environment. For instance, unlike collaborative workspaces, homes need to restrict individual users and applications; and unlike enterprise networks, homes have a richer set of devices.

8. CONCLUSIONS

We believe that the combination of HomeOS and HomeStore can create a new wave of innovation in the home. It provides a platform for developers to easily write novel applications and for users to easily add new devices and applications. We approached the design of HomeOS as an interdisciplinary exercise using field visits to learn first hand the requirements of users. The visits revealed notable differences from current OSes in how users would like to secure their home networks. Our design reflects their requirements and is driven by simplicity and room for future innovation. Early experience with developing applications that compose many devices validates our design choices.

9. REFERENCES

[1] Home security systems, home security products, home alarm systems - ADT. http://www.adt.com/.

[2] L. Bauer, L. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea. A user study of policy creation in a flexible access-control system. In *CHI*, 2008.

[3] H. Beyer and K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, 1998.

[4] Open source - Apple developer. http://developer.apple.com/opensource/.

[5] J. Borchers, M. Ringel, J. Tyler, and A. Fox. Stanford interactive workspaces: A framework for physical and graphical user interface prototyping. *IEEE Wireless Communications. Special Issue on Smart Homes*, 2002.

[6] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. A. Shafer. EasyLiving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing*, 2000.

[7] K. L. Calvert, W. Keith, E. Rebecca, and E. Grinter. Moving toward the middle: The case against the end-to-end argument in home networking. In *HotNets*, 2007.

[8] A. Chaudhuri, P. Naldurg, G. Ramalingam, S. Rajamani, and L. Velaga. EON: Modeling and analyzing access control systems with logic programs. In *CCS*, 2008.

[9] Control4 home automation and control. http://www.control4.com.

[10] DLNA. http://www.dlna.org/home.

[11] W. K. Edwards, M. W. Newman, J. Z. Sedivy, T. F. Smith, D. Balfanz, D. K. Smetters, H. C. Wong, and S. Izadi. Using SpeakEasy for ad hoc peer-to-peer collaboration. In *CSCW*, 2002.

[12] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer. Using latency to evaluate interactive system performance. In *OSDI*, 1996.

[13] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5), 2005.

[14] R. E. Grinter, W. K. Edwards, M. Chetty, E. S. Poole, J.-Y. Sung, J. Yang, A. Crabtree, P. Tolmie, T. Rodden, C. Greenhalgh, and S. Benford. The ins and outs of home networking: The case for useful and usable domestic networking. *ToCHI*, 16(2), 2009.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM CCR*, 38(3), 2008.

[16] H. M. Levy. *Capability Based Computer Systems*. Digital Press, 1984.

[17] S. Lucero and S. Schatt. Home automation and security. ABI Research, 2009.

[18] M. L. Mazurek, J. Arsenault, J. Breese, N. Gupta, I. Ion, C. Johns, D. Lee, Y. Liang, J. Olsen, B. Salmon, R. Shay, K. Vaniea, L. Bauer, L. F. Cranor, G. R. Ganger, and M. K. Reiter. Access control for home data sharing: Attitudes, needs and practices. In *CHI*, 2010.

[19] R. F. Rashid and G. G. Robertson. Accent: A communication oriented network operating system kernel. In *SOSP*, 1981.

[20] Schlage LiNK. http://link.schlage.com/.

[21] S. VanDeBogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazieres. Labels and event processes in the asbestos operating system. *TOCS*, 25(4), 2007.

[22] N. Zeldovich, S. Boyd-Wickizer, and D. Mazieres. Securing distributed systems with information flow control. In *NSDI*, 2008.

[23] Z-Wave.com - ZwaveStart. http://www.z-wave.com.