THE HOME NEEDS AN OPERATING SYSTEM

[1]SagarChoudhary , [2]Sandeep Giri ,[3]Chunchun Singh

[1,2,3] Student (B.tech 5th sem) Department of Computer Science Engineering
Dronacharya College of Engineering, Gurgaon, (India).

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 homewide operating system. A HomeOScan 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 HomeOSand our experience with developing applications for it.

General Terms: Design, Human Factors, Management
Keywords: Home networks, operating systems

I 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 . 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 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, ZWave) and research efforts. 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) and research efforts. 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 HomeOScan 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 HomeStorethat 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 HomeOSis 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.

We thus approach the development of HomeOSas 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. HomeOSshould 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 Home OS borrow liberally from existing OSes. Home OS logically centralizes the control plane of the home network 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 HomeOSusing a single, simple abstraction. For this, we extend Accent ports. Our ports enable the exchange of typed messages and can be queried for their functional description. HomeOSdoes 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 HomeOSis 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.

II 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

Twelve homes have been running Home OS for 4 – 8 months. We did not actively recruit homes but many approached us after becoming aware of the system. We limited our initial deployment to 12 homes. Ten of them had no prior experience with home automation. Beyond providing the software and documentation, we did not assist users in running or managing Home OS. These homes are using a range of devices including network cameras, webcams, appliance and light controllers, motion sensors, door-window sensors and media servers and renderers.

Organic growth What our users found most attractive was being able to start small and then expand the system themselves as desired. At first, they typically did not know what they wanted and only discovered what they found valuable over time. Home OSlet them start small (at low cost) and extend as needed. It thus provided a system that was much more approachable than commercial systems today that require thousands of dollars upfront. It was also more likely to satisfy users by allowing them to evolve it to meet their needs rather than requiring them to make all decisions during initial installation.

Indeed, all users employed an organic growth strategy. One user started running HomeOSwith only one network camera to view his front yard on a smartphone while away from home. He later added two more cameras—a webcam and a network camera from a different vendor—and was able to continue using the same application without modification. He then added two sensors to detect when doors were opened so that he could be notified when unexpected activity occurred. This used our door-window monitoring application sends email notifications, which can contain images from any cameras in the home. The user later added two light controllers and another application to control them. What started off as simply wanting to see his front yard from work evolved into a notification system and lighting control.

Diagnostic support in interoperability protocols On the negative side, at least two homes had problems diagnosing their deployments. For instance, when applications that use Z-Wave devices behaved unexpectedly, users could not easily tell if it was due to code bugs, device malfunctions or poor signal strength to the device. Disambiguation requires effort and technical expertise (e.g., unmount the device, bring it close to the controller, and then observe application behavior).

This difficulty is an instance where the added complexity of network devices, in contrast to directly connected peripherals, becomes apparent. Countering it requires diagnostic tools but they are hard to build today because

interoperability protocols have limited diagnostic support. We thus recommend that device protocols be extended to provide diagnostic information. Even something akin to ICMP would be a step forward.

2.2 The Developer Perspective

We gave the HomeOSprototype to ten academic research groups, for use as a platform for both teaching and research on home applications. As part of this program, 42 undergraduate and graduate students developed tens of HomeOSapplications and drivers.

They extended HomeOSin several directions. They wrote drivers for new devices including energy meters, different network cameras, appliance controllers and IM communication. They wrote new applications such as energy monitoring, remote surveillance, and reminders based on face recognition. The PC-like abstractions of drivers and applications enabled them to build software quickly and in reusable modules. Moreover, as a testament to the flexibility and extensibility of its architecture, we were not required to—and did not— modify HomeOSto support these development efforts.

As an example, one group extended HomeOSto support the Kinect RGB-D camera and built an application which allowed users to control lights via gestures (Figure 7). They were able to do this without having to wait for Kinect integration with a large commercial system (e.g., Control4) and were able to get it to interact with existing devices. Another group built an application that plays audio reminders based on who is recognized on cameras. This works with webcams, security cameras, Kinect or IP cameras and with any device HomeOS can play audio through (right now PCs, DLNA TVs, and Windows Phones).

Figure 7: A student demonstrating how to turn on and off lights via gestures with a Kinect

This underlines the power HomeOSgives to application developers to easily span multiple types of devices (security, PC, phone, entertainment, etc.). Commercial systems today support only a subset of devices related to their target scenario (e.g., security systems focus on cameras and motion sensors).

Layering and programmability Developers who wrote applications found the protocol independence of the APIs appealing. Developers who wrote new drivers for devices with existing DCL modules (e.g., a Z-Wave appliance controller) liked that they did not have to concern themselves with the low-level connectivity details and could instead focus exclusively on device semantics.

Interestingly, developers who extended HomeOSto devices without an existing DCL module (e.g., ENVI energy meters started by building one module that spanned both the DCL and DFL. For them, the split was unnecessary overhead as only one device used the connectivity protocol. However, in one case a group had to

support multiple devices with the same connectivity protocol based on IP to Z-Wave translation. This group found value in separating functionality across two layers indicating it was not just an artifact or our experience.

Hardware-software coupling Our developers sometimes wanted to use device features that were not exposed to third-parties over the network. For instance, one developer wanted to insert a text notification on a TV without otherwise interrupting the on-screen video. Today, some set-top boxes have this capability (e.g., for cable TV operators to signal caller identity of incoming calls), but they do not expose it to third-party software.

This points to an inherent advantage of vertically integrated software—being able to better exploit device capabilities—that open systems like HomeOSlack. This is unsurprising in retrospect as the closed nature of current solutions and devices is what HomeOSattempts to combat. However, the systematic way vendors can expose device capabilities in HomeOSshould encourage them to make their capabilities available to applications.

Media applications and decentralized data plane A few developers had difficulty in writing media applications. HomeOScentralizes the control plane but not the data plane to avoid creating a performance bottleneck. If two devices use the same protocol, we assume that they can directly exchange data. Thus, we assume that a DLNA renderer can get data directly from a DLNA server once provided with a media URI. The DLNA protocol turns out to not guarantee this because of video encoding and/or resolution incompatibilities. While we currently use heuristics to provide compatible formats when transcoding is available, they are not perfectFor reliable operation, we also plan to use HomeOSas a transcoding relay (thus, centralizing the data plane and more closely mirroring the PC abstraction) when data plane compatibility between nodes is not guaranteed. As high-quality open source transcoders exist, the main technical challenge is to generate profiles of what input and output formats devices support. This requires parsing device protocols like DLNA. Although this means violating HomeOS's agnostic kernel, we believe that media applications are common and important enough to justify an exception.

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.

III HOME OS AND HOME STORE

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

The goals of HomeOSare 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. HomeOSis 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; HomeOSwill 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 HomeOSbe coupled with a HomeStoreto simplify the distribution of applications and devices. The HomeStoreverifies 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 HomeStorecan recommend window controllers if there exists an application that combines those window controllers with the user's existing thermostat.

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

IV CHALLENGES

Home technology faces three main challenges today

4.1 Management Unlike other contexts (e.g., enterprise or ISP networks), the intended administrators are nonexpert users. But the management primitives available to users today were originally designed for experts. As a result, most users find them hard to use. Worse, devices often need to be individually managed and each comes with its own interface and semantics, rather than having a single, unified interface for the home.

The management challenge is particularly noteworthy when it comes to security and access control where users are frequently forced to choose between inconvenience and insecurity. When they are unable to easily and securely configure guest access for devices (e.g., printers) on their home networks, they either deny access to guests or completely open up their networks.

4.2 Application development Users want to compose their devices in various ways and software should be able to do just that, but heterogeneity across homes makes it difficult to develop such application software. We identify four primary sources of heterogeneity.

- *Topology:* Devices are interconnected in different ways across homes. Some homes have a Wi-Fi-only network while others have a mix of Wi-Fi, Ethernet and Z-Wave. Further, some devices use multiple connectivity modes (e.g., smartphones switch between home Wi-Fi and 3 G).
- *Devices:* Different devices, even of the same type, support different standards. For example, light switches may use Z-Wave, ZigBee or X10; and TVs use DLNA, UPnP A/V or custom protocols.

- *User control:* Different homes have different requirements as to how activities should occur . Some homes want the Xbox off after 9 PM and some want security cameras to record only at night. • *Coordination:* If multiple tasks are running, simultaneous accesses to devices will inevitably arise. Such accesses may be undesirable. For instance, a climate control application may want the window open when a security application wants it closed.

4.3 Incremental growth Users frequently want to grow their technology incrementally, as their preferences evolve. Such growth is difficult today because users cannot tell if a given piece of technology will be compatible with what they currently have. This difficulty corners them into buying from one vendor (creating lockin), seeking expensive professional help, and making significant upfront investments (e.g., buying a home-wide automation system with many features before knowing which features fit their lifestyle). Supporting incremental growth is further complicated by the rapid innovation in hardware and software; users' existing systems frequently do not support these new technologies.

V PREVALENT ABSTRACTIONS

Today, home technology can be seen as presenting one of two abstractions to users and developers. The first is the appliance abstraction that provides the same interface that a monolithic, fixed-function device would. It is used for most home security and home automation systems where the set of devices and tasks are both closed. This has the advantage of offering (potentially) simpler user interfaces and simpler integration across the set of involved devices. However, it inhibits extensibility and application development because integration with thirdparty devices and software is typically not possible. As a result, the security, audio-video, and automation systems are mutually isolated in many homes.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. HomeOSmust 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.

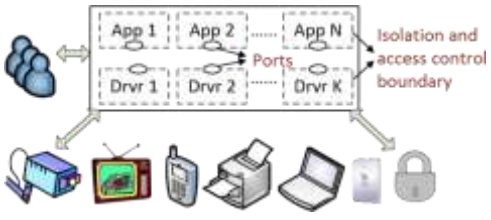

Figure 1: An overview of HomeOS.

5.1 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.

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

Logical centralization is enabled by drivers that logically incorporate the device into HomeOSindependent 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.

HomeOScan 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 HomeOSshould be close to a plugand-play model.

As its core features, HomeOSoffers: *i*) driver and application modules; *ii*) a "port" abstraction for exposing functionality and communication; and *iii*) access control for users and modules. HomeOShas other facilities, such as device discovery, but those can be easily replaced by different versions. HomeOSis built on a modern language runtime (the .NET Framework). All modules are implemented in a typesafe 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 HomeOSor 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. 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, which allows typed messages to be exchanged.

We thus decided to use ports but HomeOSports 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=("onoff", 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.

HomeOSdoes 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.

Based on these requirements, HomeOSaccess 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".

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 priin the access rules.

Runtime access is implemented using capabilities. Modules obtain the capability to access a port via HomeOSwhich 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.

VI 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 HomeOSafter 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 selfdescribing, 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 HomeOSis 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 HomeOShas adequate performance. On a 3 GHz dual core machine, the requestresponse 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.

VII 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. 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. 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) 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.* 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, 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 . 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.

VIII CONCLUSIONS

We believe that the combination of HomeOSand HomeStorecan 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 HomeOSas 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.

This experience also reveals gaps where we could not cleanly implement the abstraction due to limitations of device protocols (e.g., little support for diagnosis and incompatible implementations across vendors) or due to limited features being exposed by devices over the network. We plan to address these limitations in the future.

More broadly, our hope is that this work spurs the research community to further explore the home as a future computing platform. While we cannot outline a complete agenda for work in this area, we point out two fruitful directions based on our experience:

8.1 Foundational services Over the years PC applications have come to expect some essential services that the OS provides (e.g., a file system). Are there similar services for the home environment? Such services should not only be broadly useful but also almost universally implementable. For instance, consider occupancy information—which rooms are currently occupied by people. It can benefit many applications (e.g., lighting control, thermostat control, and security), but depending on the devices in the home, it may be difficult to infer reliably (e.g., motion sensors can be triggered by pets; cameras are more reliable). Making occupancy an essential service requires each home to possess the necessary devices, thus increasing the cost of a basic HomeOSinstallation. (This is akin to PC or smartphone OSes specifying minimum hardware requirements.) Thus, careful consideration is needed to determine which services a system like HomeOSshould provide in all homes.

8.2 Identity inference Some desired reactions to physical actions in the home depend on the identity of the user or who else is around. For instance, users may want to play different music based on who entered and turned on the lights, or parents may not want their children to turn on the Xbox in their absence. Currently, HomeOScan either not support such policies (lightswitches have no interface to query user identity) or support them in an inconvenient manner (ask parents for their password). A promising avenue for future work is to build nonintrusive identity inference (e.g., using cameras in the home, or users' smartphones), and then allow users to express policies based on that inference. A key challenge in realizing this system is to maintain safety in the face of possible errors in identify inference.

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] 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.

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

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

[14] Control4 Home Automation and Control. http://www.control4. com.

[15] Crestron Electronic: Home automation, building and campus control. http://www.crestron.com.

[16] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and V. Bahl. The home needs an operating system (and an app store). In HotNets, 2010.

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

[18] W. K. Edwards, R. E. Grinter, R. Mahajan, and D. Wetherall. Advancing the state of home networking. Communications of the ACM, 54, 2011.

[19] 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.

[19] M1 Security & Automation Controls. http://www.elkproducts. com/m1_controls.html.

[20] HomeOS. http://homeos.codeplex.com.