A First-Person Experience with End-User Development for Smart Homes

Joëlle Coutaz, James L. Crowley

In IEEE Pervasive Computing, special issue on Domestic Pervasive Computing, vol. 15, no. 2, May-June 2016

A First Person Experience with End-User Development for Smart Homes

Joëlle Coutaz, James L. Crowley
Université Grenoble Alpes

Abstract
The authors present their "lived-with" experience with an End-User Development (EUD) prototype deployed in their home and show how the results overlap and complement findings from more traditional approaches to the study of EUD for the home.

Keywords
End-User Development, Smart home systems and services, pervasive domestic computing.

Introduction

Most current HCI research on the home adopts either an anthropomorphic perspective, in which researchers conduct their investigations from the point of view of the inhabitants, or a third-person perspective, in which researchers take a view from outside the domestic space, choosing what to include to support their claims[1].
We present our experience with an alternative first person perspective, in which researchers live within their own production[1,2].

Here, we present AppsGate, an End-User Development (EUD) environment designed to empower people with tools to augment and control their home. This creative Do-It-Yourself approach provides an alternative to Artificial Intelligence (AI) where inhabitants are passive consumers of automatic services derived from data mining of plans, daily routines, and other personal information stored on the cloud. After discussing some of the research and technical challenges of developing AppsGate, we present various "lived-with" experiences—first, the developers' experiences with using their own homes as living labs, and then our experience in deploying AppsGate in our own home for more than a year. We show how our own experience helped us reflect on the evolution of our relationship with a domestic augmented space, and how our observations overlap with and complement findings from a three-week field experiment of AppsGate conducted in the home of five families external to the project.

AppsGate and its Challenges

In academic research, most EUD environments developed for the home focus on either device configuration or programming techniques, and do not include debugging aids for nonspecialists (see the "Related Work on End-User Development for the Home" sidebar for more information). AppsGate was designed using a systemic and holistic approach to provide reliable operation under real-world conditions. The goal was to support both basic, mundane activities and more creative activities, such as end-user programming.

Related work on End-User Development for the Home

End-User Development (EUD) environments provide users with tools to create entities from scratch or to reuse or modify existing entities. The tools also let users test, debug, repair, maintain, and adapt a system's functional coverage[1, 2]. Unlike users of traditional software engineering development tools, users of EUD tools are not necessarily familiar with professional software engineering practices. Their goal is not necessarily to learn programming or apply the best software engineering practices. Users of EUD environments want to get things done, primarily for themselves.[3]

With EUD environments for the home, the goal is to enable people to shape their domestic space in a flexible, reliable, incremental, and opportunistic manner. Dozens of "domestic boxes" are now available on the market. Most of these provide users with a rule-based programming tool presented in a variety of graphical notations and styles. Some domestic boxes include general-purpose scripting languages, such as Lua, or even permit the use of Visual Basic or JavaScript. The popular free-ware IFTTT[6] (If This Then That; https//ifttt.com) lets users write and share "recipes" that bring together Web services and the physical world. Thousands of these recipes are now available on the Web, resulting in an active crowd-sourcing community.

In academic research, EUD environments have rarely gone beyond proof of concept. Examples include the seminal OSCAR,[4] as well as Jigsaw,[5] CAMP (Capture and Access Magnetic Poetry),[6] and iCAP (interactive Prototyping of Context-Aware Applications).[7] More recent work, such as FedNet and its tangible tool, explores a novel approach to the deployment of sensors and services by end users using RFID cards.[8] TeC (Team Computing)[9] and DiaSuite-Pantagruel,[10] which include formal verification, open the way to real world field experiments.

Several concrete syntaxes have been proposed for programming languages in the home, with no clear advantage for any one approach. These include pseudo-natural languages (CAMP), visual languages such as the wiring diagrams of Pantagruel and TeC, and a mix of icons and text (IFTTT and a variety of Scratch-based notations[11]). Interaction techniques are typically based on drag and drop but can include examples of programming by demonstration[12] or even tangible interaction (Media Cubes[13]). These environments do not include debugging aids for nonspecialists.

References

1. M. Burnett, and B. Myers, "Future of End-User Software Engineering: Beyond the Silos," *Proc. Int'l Conf. on Software Engineering (ICSE'14),* ACM, 2014, pp. 201-211.
2. End-User Development, F. Lieberman, F. Paternò, and V. Wulf, eds.,End-User Development, F. Lieberman, F. Paternò, and V. Wulf, eds., Kluwer Academics, 2006.
3. B. Nardi, A Small Matter of Programming; Perspectives on End User Computing, Cambridge MIT Press, 1993.
4. M.W. Newman, A. Elliott, and T.F. Smith, "Providing an Integrated User Experience of Networked Media, Devices, and Services through End-User Composition," Proc. 6th Int'l Conf. Pervasive Computing (Pervasive), 2008, pp. 213–227.
5. J. Humble et al., "Playing with the Bits: User-Configuration of Ubiquitous Domestic Environments," Proc. 5th Int'l Conf. Ubiquitous Computing (UbiComp), 2003, pp. 256–263.
6. K.N. Truong, E.M. Huang, and G. Abowd, "CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home," Proc. 6th Int'l Conf. Ubiquitous Computing (UbiComp), 2004, pp. 143–160.
7. A. Dey et al., "iCAP: Interactive Prototyping of Context-Aware Applications, Proc. 4th Int'l Conf. Pervasive Computing (Pervasive), 2006, pp. 254–271.
8. F. Kawsar, T. Nakajima, and K. Fujinami, "Deploy Spontaneously: Supporting End-Users in Building and Enhancing Smart Home," Proc. 10th Int'l Conf. Ubiquitous Computing (UbiComp), ACM NewYork, 2008, pp. 282–291.
9. J.P. Sousa et al., "TeC: End-User Development of Software Systems for Smart Spaces," Int'l J. Space-Based and Situated Computing (IJBSC), vol. 4, no. 1, 2011, pp. 257–269.
10. Z. Drey and C. Consel, "Taxonomy-Driven Prototyping of Home Automation Applications: A Novice-Programmer Visual Language and its Evaluation," J. Visual Languages and Computing, vol. 23, no. 6, 2012, pp. 311–326.
11. M. Resnik et al., "Scratch: Programming for All," Comm. ACM, vol. 52, no. 11, 2009, pp. 60–67.
12. J. Chin, V. Callaghan, and G. Clarke, "An End-User Programming Paradigm for Pervasive Applications," Proc. IEEE Int'l Conf. on Pervasive Services, 2006, pp. 325–328.
13. J.A. Rode, E.F. Toye, and A. Blackwell, "The Domestic Economy: A Broader Unit of Analysis for End-User Programming," Proc. Int'l Conf. Human–Computer Interaction (CHI), 2005, pp. 1757–1760.

Figure 1 shows an example of support for basic activities—that is, for directly controlling devices using items that can be installed and removed on the fly. A palette shows the devices organized by categories; here, the "smart plugs" category is the current selection. This home includes six smart plugs, each of which has a different color. The right panel lets the user modify and observe the current state of the devices in the selected category. As the panel shows, in this case, three of the smart plugs—black, blue, and orange—are located in the kitchen, consuming a total of 283 watts. The yellow smart plug is used to control lighting in the entryway. Convenience buttons let users act on all devices in a category with one click. Here, we discuss some of the challenges of installing and monitoring devices, programming and debugging the devices, and deploying them in real world conditions.

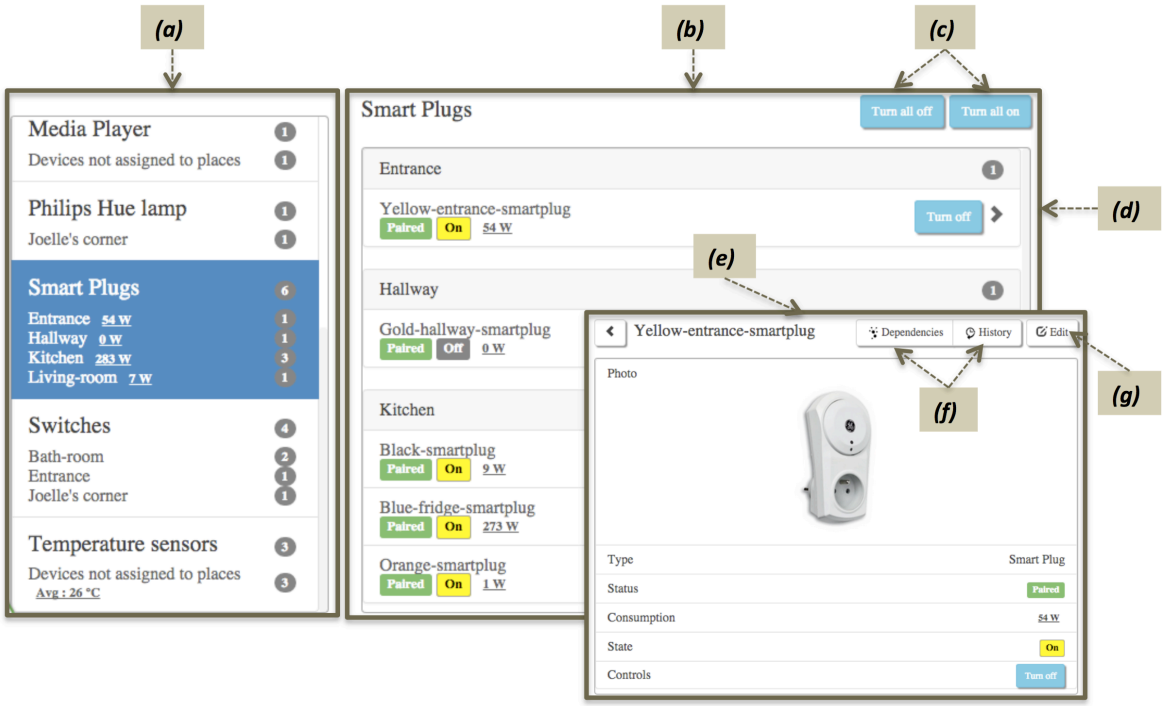

Figure 1. *Direct control of devices. (a) A palette shows the devices organized by categories, with the "smart plugs" category as the current selection. (b) The right panel lets users modify and observe the current state of the devices of the selected category. (c) Convenience buttons make it possible to act on all devices in the category with one click. (d) A ">" symbol, shown here for the yellow "entrance" smart plug, lets users view (e) a description card for each device. This card includes buttons (f) to display timelines (see Figure 2) and a dependency graph (see Figure 3) related to the device, and (g) to rename or change the location of the device.*

Installing Devices

Device installation is at the core of incremental development for the home. Research has shown that installation "enhances a user's sense of control"[3] but requires manual operations that can sometimes be excessively complex for nonspecialists. Providing users with contextual guidelines is difficult to achieve—in particular, any attempt to pair a device located out of system range, or whose energy level is too low, is problematic because the device is undetectable, thus preventing the system from providing useful feedback. AppsGate includes support for device pairing, but this support can be insufficient for naive users.

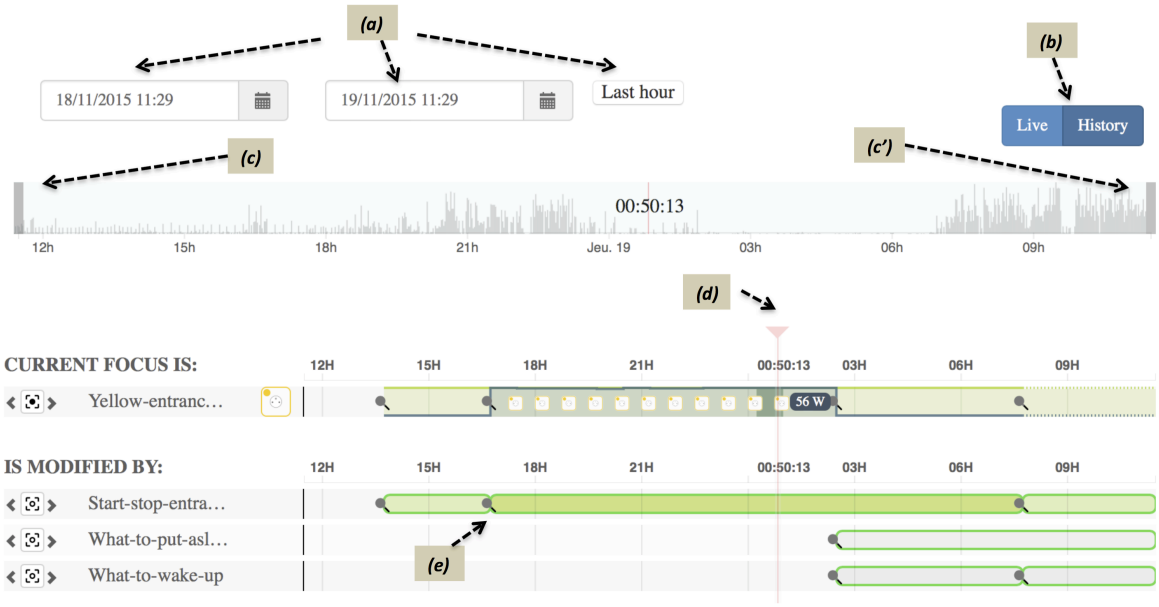

Figure 2. *Timelines provide a temporal synoptic of the state of the home, which users can browse at different levels of details. (a) Controls let users select the date and time of interest. Furthermore, timelines can be refreshed (b) automatically in real time or displayed as a snapshot. (c–c') Handles can be dragged laterally to zoom in on a particular time slot within the chronogram that shows the "heartbeat of the home." In this example, activities are concentrated in late evening and early morning. (d) The vertical ruler lets users explore states across timelines. The current position of the ruler shows that the "Yellow-entrance-smart-plug" was consuming 56 watts at 00:50:13, and that the program "Start-stop-entrance-light" was running at this time. The timelines of the programs that reference the plug are grouped together so that causalities between events and states can be detected. Clicking the (e) magnifying glass presents a list of actions executed by the program "start-stop-entrance-lighting" that changed the plug's state.*

Monitoring

Monitoring the home means observing, discovering, remembering, and understanding reasons for particular states or behavior, both at the present time and in the past. Principles from interactive visualization, such as Ben Shneiderman's mantra "overview first, zoom and filter, then details on-demand"[4], give useful guidelines but do not provide ready-for-use solutions, resulting in time-consuming developments.

In AppsGate, monitoring is supported in two complementary ways: *timelines* let users monitor home states over time (see Figure 2), and a dependency graph lets users monitor home states through relations between entities (see Figure 3). These relations, which result from programming, can be hard to conceptualize or might be forgotten. In particular, a user might forget that the state of an entity is modified by more than one program, resulting in unexpected behavior due to conflicting commands. Such relations, which can be hard to anticipate or detect, are made obvious with a dependency graph (see Figure 3).

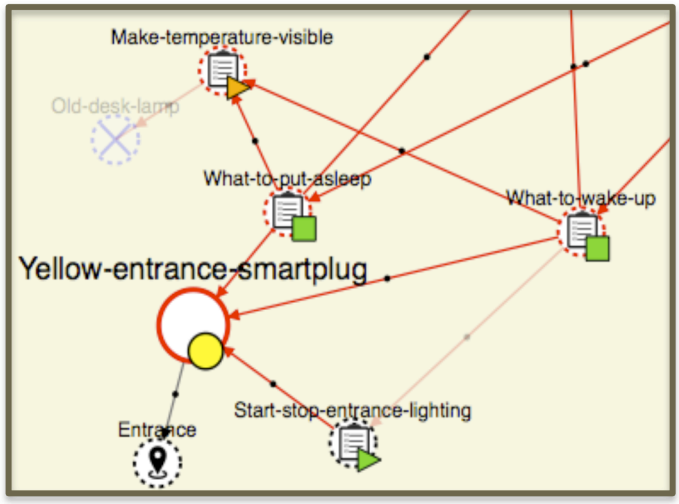

Figure 3. *An extract of the dependency graph focused on "Yellow-entrance-smartplug". This device is located in the entrance and is currently turned on (small yellow circle). It is modified by more than one program (red circle). The program "Make-temperature-visible" is running, but it references a device that is currently missing (orange triangle). The missing device is the old desk lamp (represented by the dotted blue circle with an "x" in it). The program "Start-stop-entrance-lighting" has all the resources needed and is running (green triangle). The "What-to-wake-up" and "What-to-put-asleep" programs have the resources needed but are not running (green rectangle).*

Programming

We view programming not as device configuration but as a creative activity in which people define their own semantics concerning their use of devices and services. The fundamental problem is finding the best cognitive fit between the users' semantic domain and the computer programming language domain. Research into programming languages shows that there is no ideal solution—there are only tradeoffs between interdependent and possibly conflicting factors. We have used factors from the Cognitive Dimensions framework[5] to inform our design decisions, some of which we discuss here.

Abstraction gradient and *closeness of mapping*. What are the minimum and maximum levels of abstraction? Can fragments be encapsulated? What conventions must be learned? Inhabitants can easily express their goals using rules,[6] and they group rules based on functional proximity (lighting control, for example) or routines (a morning scenario, for example).[7] Inspired by these field studies, AppsGate "fragments" are rules, conditionals, and actions on entities that exist in the problem domain (see Figure 4 for details). In turn, these fragments are encapsulated within programs that can be used as user-defined abstractions. Variables are the user-defined names of the entities that exist in the problem domain. Our choice for the implicit declaration of variables results from the tradeoff between expressive power and closeness of mapping.

As for notation (that is, concrete syntax), visual languages are appealing but do not necessarily satisfy a "close mapping."[5] Instead, a pseudonatural language is proposed, along with a syntax-driven editor to protect users from overly complex syntax conventions (see Figure 5). A "smart keyboard" displays all possible options, given the entry point currently selected and the current state of the home. With this feedforward[8] mechanism, users are not only informed in advance of the capabilities in their domestic space but are also prevented from making errors.

-- A program is either a nonempty set of imperative statements followed by a non-empty set of rules, or it is only a nonempty set of imperative statements or of rules. The Executed once IMPERATIVE-SECTION lets users express initializations. It is executed only once, when program execution is started. Rules of the Repeated RULE-SECTION are evaluated in parallel and repeatedly until program execution is stopped. Program execution can be stopped or started by end users or by any program. A program can stop itself.

```
PROGRAM ::= Executed once IMPERATIVE-SECTION {then Repeated RULE-SECTION} |
{Executed once IMPERATIVE-SECTION then} Repeated RULE-SECTION

IMPERATIVE-SECTION ::= IMPERATIVE-STMT {then IMPERATIVE-SECTION}
RULE-SECTION ::= RULE-STMT {and RULE-SECTION}
```

(a)

-- An imperative statement is either an action (such as "Switch-on Orange-smartplug"), or a conditional statement (IF or AS-SOON-AS).
-- IF and AS-SOON-AS make the distinction between conditions on states (for example, "*If* Orange-smartplug is on") and conditions on events (for example "*As soon as* Orange-smartplug is turned on").

```
IMPERATIVE-STMT ::= ACTION | IF | AS-SOON-AS
IF ::= If <state-condition> then IMPERATIVE-SECTION {otherwise IMPERATIVE-SECTION}
AS-SOON-AS ::= As soon as <event-condition> then IMPERATIVE-SECTION
```

(b)

-- EACH-TIME and WHILE rules make the distinction between triggers from events ("*Each time* Orange-smartplug is switched on") and triggers from states ("*While* Orange-smartplug is on").

```
RULE-STMT ::= EACH-TIME | WHILE
EACH-TIME ::= Each time <event-condition> then IMPERATIVE-SECTION
WHILE ::= While <state-condition> then keep <state> and as soon it is not true anymore
then IMPERATIVE-SECTION
```

-- Clause *and as soon it is not true anymore then* is intended to help people to remember to specify the behavior to be adopted for durative actions (" *While* a card is inserted in Card-reader *then Keep* every lamp of everywhere on *and as soon as it is not true anymore* Switch off every lamp of everywhere").

(c)

```
ACTION ::= <action> <selected-set-of-devices> | <command>  <a-service> | wait <a-number-
of-seconds> | start <program-name> | stop <program-name> | stop myself
```

<action>	action supported by the <selected-set-of-devices> available in the system.
<command>	command supported by <a-service> available in the system.
<selected-set-of-devices>::= all <device-type> located in <location-name> | <device-name> located in <location-name>	
<location-name> ::= everywhere | <place-name>	
<state-condition>	a simple Boolean expression whose evaluation returns TRUE or FALSE. It is composed of one symbol that denotes a <state>, one single logical operator (=, ≠, <, >), and one value that belongs to the value domain of that <state>.
<event-condition>	denotation of an event whose evaluation returns TRUE on the occurrence of that event.
<event>	instantaneous signal detected or generated by the system to express the state change of an entity.
<state>	value of an attribute of an entity.
<device-name>	user-specified string that denotes a device of the home.
<program-name>	user-specified string that denotes a program.
<place-name>	user-specified string that denotes a place of the home.

(d)

Figure 4. *Grammar of the AppsGate programming language. (a) overall structure of a program, (b) imperative statements, (c) rules, (d) actions and miscellaneous syntactic constructs.*

Secondary notations. The purpose of secondary notations is to make obscure information visible. They can help improve the "cognitive fit" but can also be used to track program execution in real time at a fine grain for debugging purposes. Rule-based languages tend to obscure the order of actions. To make program execution more explicit, arrows, along with a blue background, denote the current statements where the program is waiting (see Figure 6). Counters express the number of times an action has been executed since the program last started, and the words YES and NO are attached to "if" statements to indicate the results of the most recent evaluations of conditions.

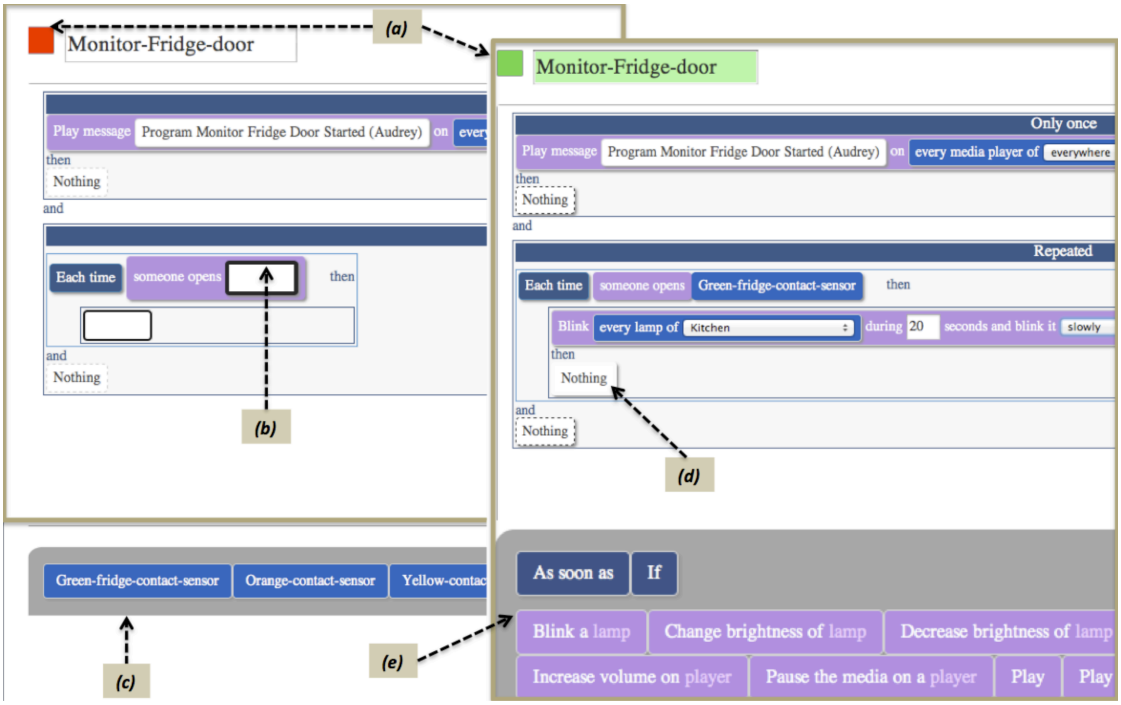

Figure 5. *The "Monitor-Fridge-door" program, edited with the AppsGate syntax driven editor. (a) Colors indicate the status: when the program is incomplete, the status indicator is red; when the program is syntactically correct, the status indicator is green. (b) Given the current insertion point, the Smart Keyboard (c) shows the list of devices that currently support an "open" action (contact sensors, for example). (d) On the right, given the insertion point, the Smart Keyboard (e) is updated according to the grammar of Figure 4 with the names of the statements and actions that currently make sense in the home.*

Debugging

Debugging is both costly and cognitively hard. The Interrogative Debugging paradigm, in which the system directly answers "why" and "why not" questions, offers a promising solution.[9] As mentioned, timelines, the dependency graph, and secondary notations include visual proactive warnings:

- red circles denote access conflicts in the dependency graph,
- an orange indicator flags programs that reference entities that are no longer available in the environment,
- red text highlights missing devices, and
- counters (which show the number of times something occurs) and arrows make program execution traceable.

However, answers to questions such as "Why is the sofa-lamp turned off?" or "Why was the sofa-lamp turned off yesterday morning?" are not explicitly provided. They must be found through exploration. The ability to ask "what if" questions is also important to support exploration by trial and error. Except for running programs using a virtual date and time, AppsGate falls short of providing a full-fledge virtual home that would act as a sandbox.

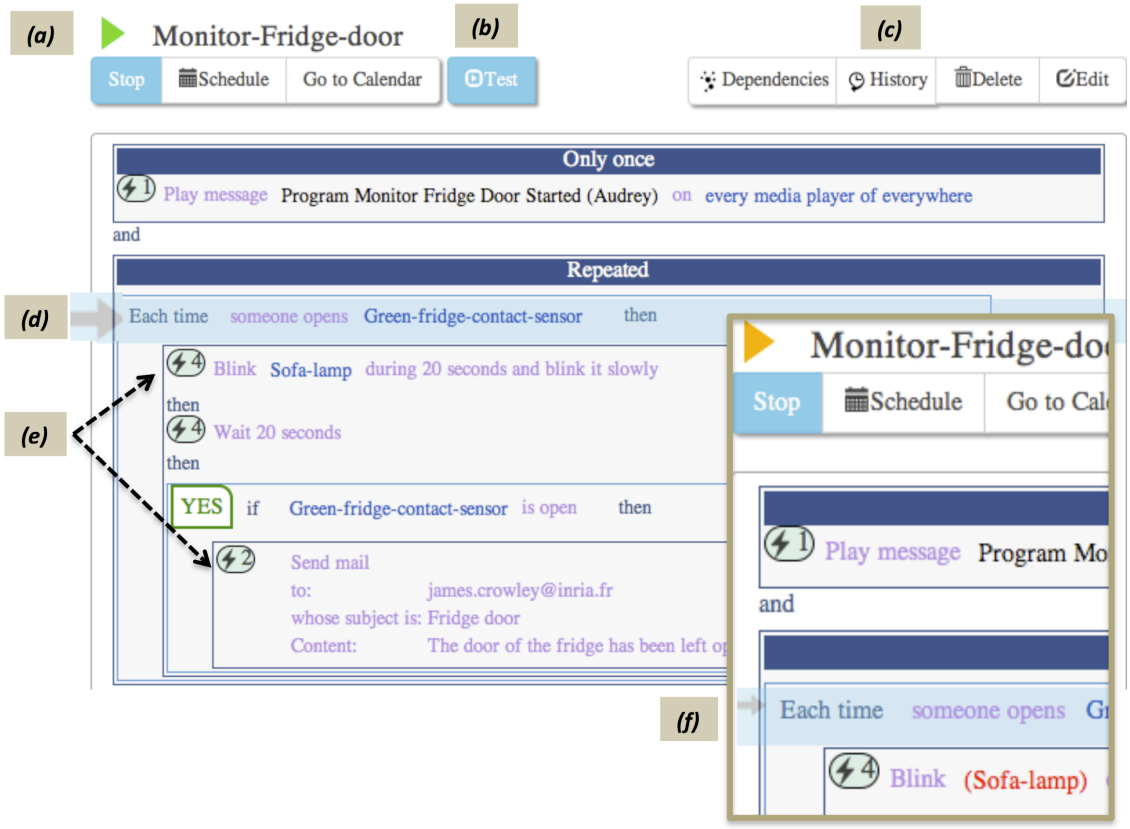

Figure 6. *Execution of the "Monitor-Fridge-door" program. (a) The green triangle indicates that the program is running and the resources needed are available. (b) Primary controls on programs start and stop execution, schedule execution as events in Google Calendar, and run the program in virtual time. (c) Secondary controls open timelines and the dependency graph focused on the program and edit or delete the program. (d) The gray arrow (enlarged here for legibility) acts as a secondary notation: the "each time… then" rule is waiting for the fridge door to be opened. (e) The "counters" also act as a secondary notation: the fridge door has been opened four times and left opened twice for more than 20 seconds. Selecting a counter will tell when this happened. (f) At the bottom right of the figure, the orange triangle indicates that the program is running, possibly with an unexpected behavior, and the red text indicates that the Sofa-lamp is currently undetectable.*

Real world conditions

Deployment under real-world conditions, incremental installation of devices and services, and meaningful feedback and feedforward are possible only if the EUD environment runs on top of a robust infrastructure capable of detecting the dynamic arrival and departure of devices and services in real time. This detection is key to supporting partial installation[3] and to attracting the user's attention to problems for trouble-shooting. The lack of a *de facto* open operating system for the home that satisfies these requirements makes the development of EUD environment both complex and time-consuming (see the "AppsGate from a Technical Perspective" sidebar).

As a result, when development occurs under severe time constraints, as was the case for AppsGate, priorities and compromises must be revised in an agile manner. Here, we discuss how the use of our personal homes helped us make informed decisions during such revisions.

AppsGate from a Technical Perspective

The AppsGate system consists of a server and a set of clients for user interaction (see Figure A). The server runs on a MiniPC or a Raspberry Pi, whereas clients may run on a variety of devices including iOS iPads, Android Nexus tablets, and laptops. A detailed description is available at https://github.com/appsgate2015/appsgate/tree/master/documentation. An on-line demonstration can be found at http://iihm.imag.fr/demos/appsgate/appsgate2015.mp4

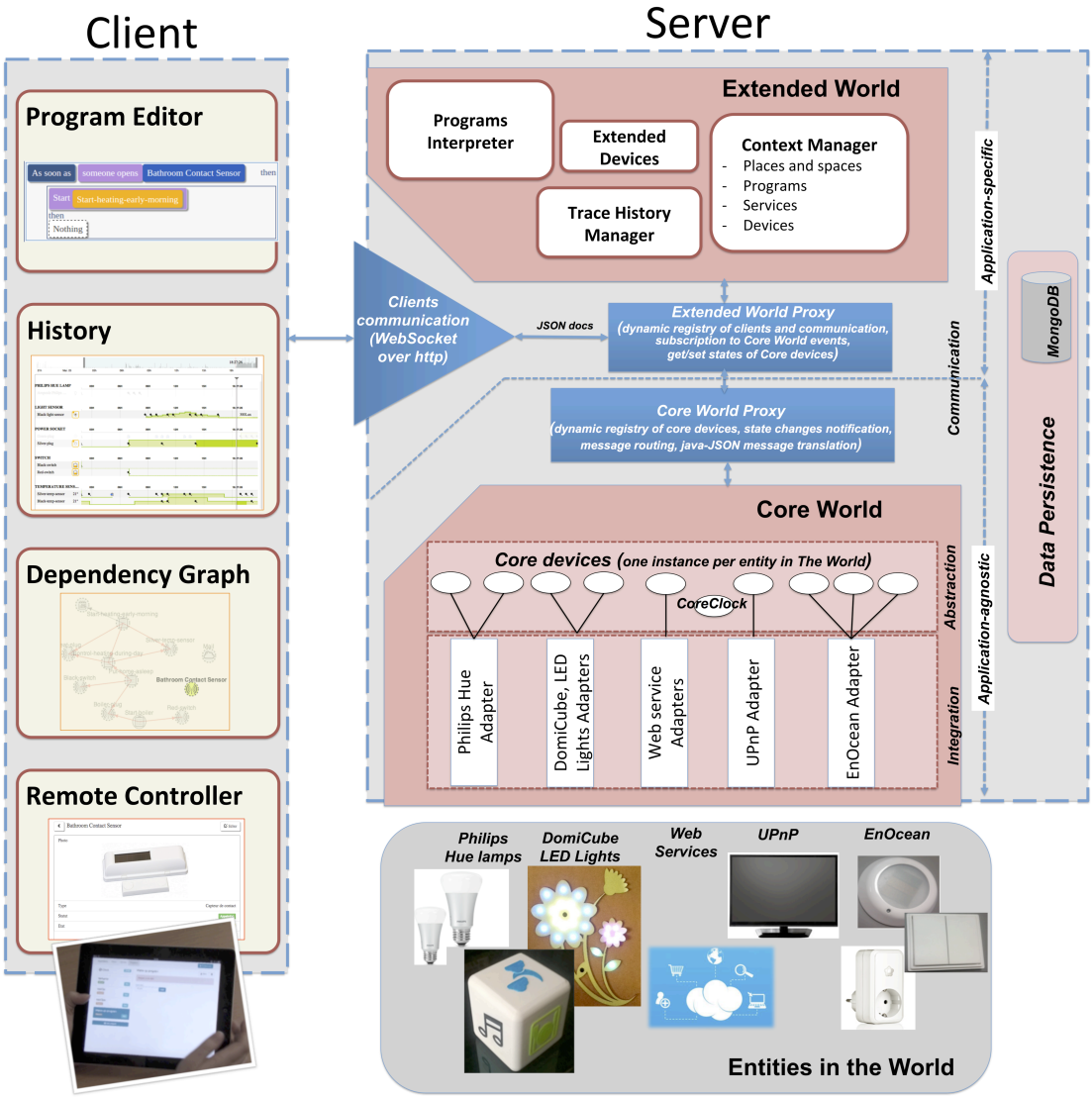

Figure A. *The AppsGAte system consists of a server and a set of clients for user interaction. Clients are Web applications implemented with HTML5, CSS, and JavaScript. The server is structured as two levels of abstraction, referred to as the "Core World", which is application-agnostic, and the "Extended World" which is application-specific.*

The server is implemented in Java and uses OSGi to support the dynamic arrival and departure of devices and third-party cloud services such as Google Calendar, Google mail, Yahoo! Weather forecast, and text-to-speech. The Web socket protocol is used to provide a full-duplex communication channel over HTTP between the server and the clients.

Over this channel, the server and the clients exchange JavaScript Object Notation (JSON) documents for both queries from clients and messages from the server.

Like most platforms developed for the "pervasive home" – such as HomeOS,[1] OpenHAB (www.openhab.org), FedNet,[2] or Diasuite[3] – the server is structured as two levels of abstraction, referred to as "Core World" (which is application-agnostic) and "Extended World" (which is application-specific). At the lowest level, the Core World operates as an integration middleware for a variety of cloud-based services, devices, and protocols including Universal Plug and Play (UPnP), Bluetooth, EnOcean, and Philips Hue Lights. A typical solution to the problem of heterogeneity is the development of an adapter for each protocol combined with an ontology that replaces physical devices and third party-services with virtual devices (denoted as "Core devices").

Devices, services, and protocols are not only heterogeneous bur also volatile. Thus, their representation in Core World appears, changes, and disappears as a result of events that happen in the world. The Core World handles these events automatically without a need to stop or reboot the system. For example, if the Philips Hue bridge detects the disappearance or arrival of a light bulb, the Philips Hue Adapter asks the Core World to destroy or create an instance that represents the light bulb. This change in the Core World is registered by a Core World Proxy that, in turn, broadcasts a JSON message to its listeners – typically the Extended World Proxy. From there, the Extended World Proxy notifies its subscribers such as the clients and components of Extended World (such as Programs Interpreter, Trace History manager, Context Manager).

References

1. N. Rosen et al., "HomeOS: Context-Aware Home Connectivity," Proc. Int'l Conf. Pervasive Computing and Comm. (Pervasive), 2004, pp. 739–744.
2. F. Kawsar, T. Nakajima, and K. Fujinami, "Deploy Spontaneously: Supporting End-Users in Building and Enhancing Smart Home," Proc. 10th Int'l Conf. Ubiquitous Computing (UbiComp), ACM New York, 2008, pp. 282–291.
3. Z. Drey and C. Consel, "Taxonomy-Driven Prototyping of Home Automation Applications: A Novice-Programmer Visual Language and its Evaluation," J. Visual Languages and Computing, vol. 23, no. 6, 2012, pp. 311–326.

Team Members' Homes as Living Labs

AppsGate was developed over a period of 30 months, ending in June 2015. The development team was composed of 11 people with distinct backgrounds: four researchers (one in autonomic computing and three in HCI), two human factors specialists, and five software engineers. The only members of the team with previous knowledge in the area of EUD for the home were two of the HCI researchers who had conducted field studies before the project started to investigate whether families are prone to envision new services by "connecting things."[10]

Early Phase: Creativity Using Team Member's Homes

During the first six months of the project, we employed a mix of traditional methods in the lab as well as in our institution's smart-home living lab. These methods included structured focus groups, creativity sessions, and scenario enactments. In addition, we performed creativity sessions in the homes of four of the team members (one house and three apartments, for a total of six sessions, three hours each, involving 18 people, including six project members). We did not initially intend to use personal examples, but these turned out to be quite valuable as the project advanced. Shared examples not only reinforced the cohesion of the group but also provided a common grounding for later discussions: examples felt more authentic than those identified in the lab.

Deployment Phase: Progressively Leaving the Lab

Lab studies are not enough to assess the soundness and added value of a smart home EUD environment. Longitudinal experiments in real-world conditions are needed. Such experiments require a level of technical robustness that is difficult to achieve. They also require recruiting users who are willing to participate over a long time period in their

own home. We have therefore employed an incremental deployment process in environments of increasing "wildness," with each deployment covering specific types of assessment and having its own criteria for proceeding to the next step:

1. *Living lab*: The first step was deployment in the smart home living lab until "minimal robustness" was achieved. This deployment was primarily under the control of the software engineers.
2. *Team member homes*: The next step comprised four-month stress tests in the homes of five team members (October 2014 through January 2015) to assess the "minimal robustness and functional coverage" to support the field studies planned for the next deployment step.
3. *External homes*: The third step comprised a three-week deployment in the homes of five families external to the project (February 2015). These families had been recruited in the area of Grenoble, France, with the following criteria: two adults possibly with children; with no programming background; living in a home equipped with an Internet gateway and a home computer; and having no prior experience with home automation. Except for one person, these families were not members of our research institution. They were compensated with a 200 € gift card.
4. *Long-term lived-with experience*: We have been conducting a lived-with experience in our own home since October 2014. This trial was motivated by both the effectiveness of step 2 and a lack of sufficient resources to perform realistic longitudinal field experiments with external families. It was also the opportunity to test the effectiveness of a first-person perspective.

Rapid deployment in external environments requires that the equipment literally fit in a suitcase. We used patafix-glue-based technology that can be easily installed and removed without damaging the domestic space. Thus, an AppsGate kit (see Figure 7) includes an Android or iOS tablet that can be used as a media player and a device to interact with the system, a Wi-Fi router, a mini PC, a Philips Hue kit, a DomiCube and its Bluetooth dongle, and 31 EnOcean wireless sensors and their dongle (temperature, luminosity, and contact sensors, as well as switches and smart plugs). The mini PC is configured with the AppsGate server as well as the Linux Advanced Packaging Tool (APT) to install new versions of the system conveniently and reliably with a single "apt-get update" command.

Deployment in the team members' homes (step 2) proved quite effective. It offered not only diverse physical settings for testing the robustness and weakness of the wireless technology but provided diverse knowledge about AppsGate. The distinct research interests of the team members helped uncover several important blocking factors. The four HCI experimenters, by cross-checking their experiences, were able to predict problems that external families might encounter. These included problems with pairing EnOcean devices, understanding the internal function of some sensors, and expressing compound conditions in programming rules. These predictions helped tune the experimental protocol for the third deployment phase. In particular, we decided to provide the external families with devices that had been previously paired in our lab.

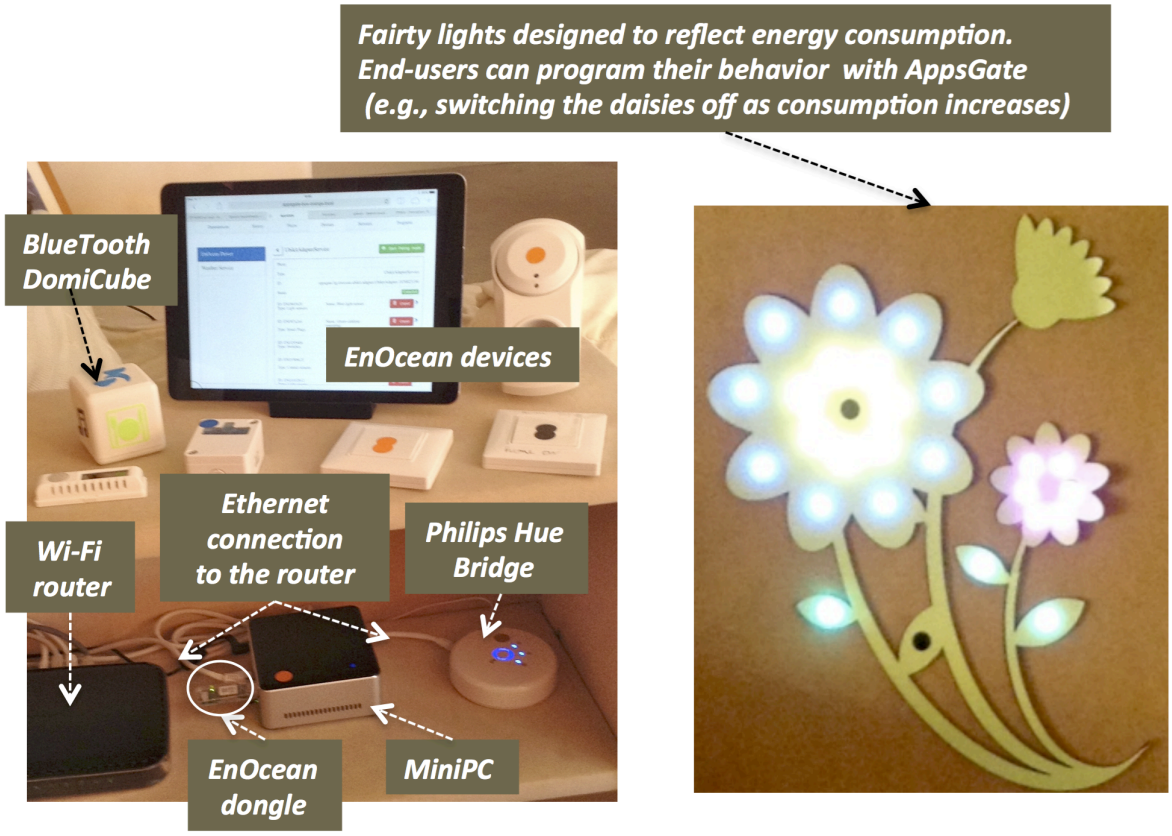

Figure 7. *The AppsGate kit. The kit includes an Android or iOS tablet, a Wi-Fi router, a mini PC, a Philips Hue kit, a DomiCube and its BlueTooth dongle, and 31 EnOcean wireless sensors and their dongle (temperature, luminosity, and contact sensors, as well as switches and smart plugs).*

The External Field Experiment

The overall objective of the external field experiments was to perform a qualitative evaluation of system utility, learnability, flexibility, and pleasurability. Before the experiment started, the five families filled in a questionnaire about their profile and their understanding of terms such as “domotics” and “program.” They also provided a floor plan of their home so we could track the locations of wireless devices as the experiment proceeded. For three consecutive weekends, a human factor specialist visited each family for 1.5 hours to answer questions and record a semi-structured interview. During each visit, the experimenter provided the family with a questionnaire, suggested new tasks to be performed during the coming week, and asked for examples of both pleasant and frustrating experiences with the system.

During the first weekend, an experimenter with technical background installed the AppsGate kit. Meanwhile, the human factor specialist presented the operation of AppsGate using printed screen shots, and asked the two adults of the family about their understanding of the symbols as well as the predictability of actions. No live demonstrations of the system were performed and no program examples were provided. Families were only provided with printed documentation describing each of the devices in the kit and a paper diary for recording experiences.

The experimental data includes five hours of audio recording of the interviews, answers to questionnaires, informal notes taken by the HCI experimenter, floor plans, screen shots of programs developed by the families, and lists of devices used. This data was analyzed by a third-party HCI expert who had no prior knowledge of AppsGate. The overall satisfaction and system understanding of the participants increased over the three weeks, starting at an average of 3.2 (standard deviation of 0.45) on a five-level Likert scale, reaching an average of 3.5 (std. 0.08) at the end of the experiment. Families used approximately 16 devices (the average was 15.6, std. 3.15), installing 1 to 3 additional devices every week. The most popular devices were switches, contact sensors, and Philips Hue lights. The most problematic were luminosity sensors, in part because of difficulty understanding the concept of lumen.

Each family developed approximately 10 programs (avg. 10.2, std. 2.66), and found 3 to 5 programs to be useful enough to execute daily. At first, programs were built to play with the system to explore the capabilities of the equipment. Families then investigated how to support their routines and improve their comfort. In the end, we identified the following themes:

- *Simplification of use*:[10] replacing the alarm clock with a Philips Hue light (one family),
- *Peace of mind*:[11] scheduling the kettle to start automatically every morning (one family).
- *Convenience*:[11] turning all lights on or off with a single switch or with contact sensors installed on doors, luminosity sensors, or geographical conditions such as “sunrise in Grenoble” (three families).
- *Comfort and leisure*: setting ambient lighting when watching TV.
- *Social gathering*: using blinking lights to notify everyone of dinner-time (one family).
- *Security and hygiene*: using lights and email to signal “out of sight” events such as wrong temperature or illumination in the baby’s bedroom (one family), a cat entering a bedroom (one family), or an entrance door being opened.
- *Making energy visible*: using the Philips Hue lights to indicate instantaneous energy consumption (one family).

Simple programs were easy to build (for all five families) with comments such as “editing programs is extremely easy; it’s well designed” and “I find this system of sentences to be very relevant.” Programming was qualified as “fun” after one week of use, providing a nice feeling of success (one family). It was easy to understand programs written by someone else. All families found that modifying programs was easy. However, as predicted in step 2 of the deployment (in team members’ homes), expressing compound conditions was difficult, and the operation of the luminosity sensors was difficult to understand. Also, there were problems with the aesthetics and form factor of the smart plugs (one family). Wireless switches posed problems because of a lack of feedback to users about whether the switch had been pressed firmly enough to be detected.

The expression of compound conditions is difficult for users without programming experience. In their field study, Justin Huang and Maya Cakmak observed that systems such as IFTTT neglect the distinction between states and events as triggers, as well as

between instantaneous, extended, and sustained actions. [12] As the grammar shows in Figure 4, AppsGate makes these distinctions explicit by proposing different statements. This issue requires additional study.

At the end of deployment, when asked "If you happen to miss AppsGate, what is the reason?" most families answered "convenience" and "personalization" as the two main reasons:

- *Convenience*: Programming lights (five families) and the use of switches (four families) – "switches are fantastic!"
- *Personalization*: For two families "it's an application that adapts to our needs since we can create our own programs" and it provides "the ability to customize our home, to adapt it to our daily life and moods".

One family was "curious to know what comes next" and all of them would have liked to keep AppsGate longer. Typically, new ideas of use emerged one week after the experiment ended (such as the use of a contact sensor to "detect when the litter of the cat should be changed").

Overall, our participating families found that AppsGate provided them with sufficient flexibility, and that it generated a number of unexpected pleasant experiences such as "our two year old daughter found it fun to turn the lights on by entering her bedroom." They found the system useful but felt that utility would be significantly enhanced if AppsGate offered more integration with the rest of the home.

We conjecture that our participating families did not live with AppsGate long enough to experience problems. Only one family member reported an unexpected situation when the "triangle [of a program] turned to orange during the night". The problem was repaired quickly as the program showed which device was missing (his wife turned off power to the Philips Hue lamp referenced in the program). A three-week experiment was insufficient for people to develop novel relationship with their augmented home, or even to assess whether AppsGate would support the "organic evolution of routines, periodic changes and exceptions."[13] Additional experience was needed.

Our Experience as End-Users

We are a married couple living in a four-room 120 square meter apartment equipped with four laptops for professional use and five iPads. One of us (Coutaz) was responsible for the scientific objectives of AppsGate. The other is a computer science professor who was not directly involved in the project.

In the first six months, we used one of the AppsGate kits that had previously been deployed with external families. As the experiment progressed, we asked for the integration of text-to-speech and the development of an energy-monitoring service. The following findings draw from the analysis of 73 hand-written notes and screen dumps recorded in a notebook from October 2014 to November 2015.

As reported in the literature on EUD environments, one of us naturally emerged as the local expert, whereas the other would occasionally ask for new programs. We have developed more than 30 programs, first for testing the system robustness, and then for supporting our daily life. We currently use 14 programs involving 25 devices. Overall, these programs were not developed to save time; they were about quality of life. Although one of the iPads is now dedicated to home monitoring, we prefer to edit programs with our laptops (this is in line with the findings of Fahim Kawsar and Alice Jane Bernheim Brush about the use of computers at home[14]).

Similarities

As with our participant families, we developed "convenience" programs and "notifiers"—for example, using flashing lamps to warn of undesirable conditions, such as the sun illuminating the refrigerator. In contrast with the pilot families, who did not express the need for reminders, we use vocal messages as a convenient means to replace post-it notes on the entrance door or the refrigerator.

In line with the ethnographic analysis of Scott Davidoff and his colleagues,[13] some programs and devices are seasonal. For example, the detector for sunlight on the fridge is needed only in winter, when the sun is low on the horizon. Other programs maintain relevance, but need periodic change, while others rapidly become obsolete. For example, the soft ambient lighting used in winter to "wake up the house" does not make sense in summer, when the sun rises at 5 a.m. After minor surgery, a program vocally reminded one of us when it was time to perform physical therapy exercises or to take medicine.

Additional findings

Some additional findings from our deployment had to do with the quantified home, privacy, the taming effect, and our time investment.

Quantified-home. We progressively switched from convenience and reminder programs to programs for a "quantified home" using the timelines (Figure 2). This led us to discover the effectiveness of our dishwasher's "eco" mode. We also learned that our refrigerator's "silent defrost" feature is actually very expensive in energy use. Our "quantified home" has also revealed an interesting lesson about privacy.

Privacy. Although our participant families mentioned watching energy consumption, they did not discover that this is also a threat to privacy. Our awareness of privacy concerns was raised by the incidental analysis of the timelines. Movements in the home, people's arrivals and departures, and meal and bed times are all quantified and made obvious, as was evident when our teenage granddaughter used our residence to organize an impromptu party while we were away for the weekend. For ethical reasons, we now shut down AppsGate when we are away for more than a day.

Meaning and complicity result from taming. As the fox says to the Little Prince lost in the desert in Antoine de Saint-Exupéry's novel, Le Petit Prince, "If you tame me, then we shall need each other" (Gallimard, 1943). Taming unfolds over time, punctuated by tiny

amounts of progress and periods of disinterest. Loss of interest in AppsGate emerged as we obtained a good model of the home behavior. However, AppsGate progressively became an integral part of our domestic space, switching from pure utility to a form of anthropomorphism. When AppsGate is turned off, we miss the serendipity of the lighting and of the vocal messages, as well as the welcoming ambiance it creates. We do not "use" technology so much as "live with it" now.

For example, our refrigerator is now more than a cold storage appliance. At first, it was an energy consumer, which the timelines helped discover. It then became a living device with its own life cycle reflected as blue, orange, and red ambient lightings. The program shown in Figure 6 was intended to make us aware of the number of times the door was opened. Actually, the program turned the refrigerator into a personalized social media: Opening the fridge door has become a convenient way to tell the other one "it's time to come home" or "I'm cooking." Sadly enough, an email sent by the fridge on 13 November 2015 at 7:15 p.m. helped us reconstruct our personal activities as the terrorist events unfolded in Paris. However, user engagement in programming takes time and effort.

Cost in time and attention. Monitoring the home, checking that the technology is working, writing and modifying programs, and transforming ideas into "AppsGate code" all add to a busy schedule. None of our families mentioned time investment, but one colleague, who had superficial knowledge of the system and to whom we proposed an AppsGate installation, replied: "I am more busy at home than at work!"

As for us, we have observed that tinkering with AppsGate can interfere with our own activities. To avoid the temptation, we sometimes shut it down when working on a deadline. Also, useful improvements to programs are sometimes put off for later when attention costs feel too high.[15] For this situation, we would like to control and program AppsGate "as we are"—that is, from where we are and without being forced to use a machine. We believe that multimodal interaction that includes spoken language will increasingly provide a desirable component for the smart home, or that the system be able to "guess our intention" and take over the programming task.

Take Away Messages

A number of take-away messages can be drawn from our experience with implementing, deploying, and living in AppsGate. These concern the methodology, technology, and synergistic combination of machine learning with EUD environments.

Methodology

In practice, recruiting participants for a long period of time is costly and difficult to achieve. A good compromise is to perform field experiments in the home of families external to the project for short periods of time (one month minimum) and combine them with a first-person approach over long periods— typically a year.

One month is the metric for evaluating the overall usability and utility of an EUD environment. It is not sufficient for assessing its capacity to accommodate opportunistic needs and seasonal uses, nor is it sufficient for the "taming effect" to occur along with its

ups and downs, including cost in attention investment. The first-person experience approach not only makes it possible to confirm findings from short field studies but also makes it possible to discover subtle problems and advantages that might otherwise take up to a year to uncover. This approach might also result in unexpected side effects for other research areas. This is illustrated by our design of the fairy lights, which not only serve as a "persuasive device" that mirrors our energy consumption (see Figure 7), but also is an intriguing conversation piece for visitors.

On the down side, there is a risk of "falling in love with your own creation." Clearly, the scientific objectivity of the researcher is at stake here. Limits of the assessment must be analyzed with honesty, and results should be confirmed by further studies with external participants. For example, given our background in computer science, we are not entitled to assess the apparatus developed for supporting debugging.

Finding Appropriate Baseline Technologies

As mentioned, implementation for real world use is always more complex and time-consuming than planned. Careful attention should be paid when selecting which middleware to use as the runtime infrastructure for the EUD environment. The same holds for sensor technologies. EUD environment developers can't provide end users with the appropriate feedback for wireless devices that don't transfer their energy level or range of sight. The hardware and software choices strongly influence the final features and services delivered.

Given the time and complexity of real-world use, it was not possible to address several worthy issues in our experiments with AppsGate. These include providing contextual help for device pairing, programming constructs to express compound conditions, augmenting the interrogative paradigm with "what if" simulation for debugging, and providing a multimodal user interface for more fluid interaction with the system. In addition, social programming is worth investigating as are possibilities for augmenting EUD environments with machine learning.

EUD Environments and Machine Learning

Two competing approaches are emerging for the development of smart-home technologies. With one approach, users are passive consumers who willingly trade their data in exchange for the convenience of smart services. This approach is compelling, both because it frees users from the challenge of configuring and maintaining systems, and because it makes it possible for established companies to apply modern machine learning and big data analysis to construct smart-home systems. The challenge for companies is to provide services that are so compelling and easy to use that end users surrender control of both system behavior and personal data. The danger is that end users will become prisoners of closed ecosystems of smart-home services subject to the dictates of large companies. Another danger is that users might lose their sense of agency.

With the other approach, users retain local control of data and services, at the cost of investing the effort required to configure and manage the smart home in a changing

landscape of devices and network protocols. For this approach, the challenge to the scientific community is to provide users with powerful EUD environments, running on top of open software infrastructures that are easily extended by nonexperts. Another important challenge to EUD is raised by machine learning, as illustrated by the Nest thermostat. However, as discussed by Rayoung Yang and Mark Newman,[16] current techniques based on machine learning can't distinguish between routines and exceptions. Significant error prediction remains.

We believe it should be possible to augment EUD with machine learning based on local data—for example, using machine learning to build procedures to recognize activities, contexts, and exceptions as additional services. With this approach, end users can incorporate elements of context and activity in their programs while retaining control of their home. One key issue here is that users must have the means to uncover the models inferred by machine-learning algorithms, so that the behavior of the inferred services can be understood, predicted, and possibly adapted using EUD.

We are currently porting AppsGate onto Open HAB, an open source middleware supported by a large community of users. This should facilitate the integration of new protocols, services, and devices. In parallel, we are deploying the current version of the system in the home of three families to compare our lived-with experience with that of people external to the project over a longer period. One family that participated in the previous three-week experiment recently moved and is eager to try the system in their new home. The other two families are retired computer scientists interested in the study of energy consumption and in the development of new devices accessible to elderly users. AppsGate provides a solid basis for such experiments.

Acknowledgments

This work was supported by the European AppsGate CA110 project and the EquipEx AmiQual4Home ANR - 11-EQPX-00. We thank Jean-Christophe Pont for his leadership in managing the AppsGate project as well as Mario Diaznava whose efforts and vision made the project possible. We are also grateful to the AmiQual4Home team for the realization of the Fairylights and the DomiCube (Nicolas Bonnefond, Stan Borkowski, and Rémi Pincent) as well as the AppsGate development team (Morgan Bidois, Sybille Caffiau, Jean-René Courtois, Rémy Dautriche, Alexandre Demeure, Elena Elias, Jacky Estublier, Thibaud Flury, Cédric Gérard, Camille Lenoir, Jander Nascimento, Kouzma Petoukhov, Patrick Reignier, Camille Roux, and German Vega).

References

1. Desjardin, R. Wakkary, and W. Odom, "Investigating Genres and Perspectives in HCI Research on the Home," Proc. Int'l Conf. Human–Computer Interaction (CHI), 2015, pp. 3073–3082.
2. M.C. Mozer, "Lessons from an Adaptive Home," Smart Environments: Technologies, Protocols, and Applications, John Wiley & Sons, 2004, Chapter 12.
3. Beckman, S. Consolvo, and A. LaMarca, "Some Assembly Required: Supporting End-User Sensor Installation in Domestic Ubiquitous Computing Environments," Proc. 6th Int'l Conf. Ubiquitous Computing (UbiComp), LNCS 3205, Springer-Verlag, 2004, pp. 107–124.
4. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," Proc. IEEE Symp. Visual Languages, 1996, pp. 336–343.
5. T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework," J. Visual Languages and Programming, vol. 7, no. 2, 1996, pp. 131–174.

6. B. Ur et al., "Practical Trigger-Action Programming in the Smart Home," Proc. Int'l Conf. Human–Computer Interaction (CHI), 2014, pp. 803–812.
7. Demeure et al., "Building and Using Home Automation Systems: A Field Study," Proc. Int'l Symp. End-User Development (ISEUD), 2015, pp. 125–140.
8. J. Vermeulen et al., "Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity," Proc. Int'l Conf. Human–Computer Interaction (CHI), 2013, pp. 1931–1940.
9. A.J. Ko and B. Myers, "Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior," Proc. Int'l Conf. Software Engineering (ICSE), 2008, pp. 76–82.
10. Coutaz et al., "DisQo: A User Needs Analysis Method for Smart Home," Proc. Nordic Conf. Human–Computer Interaction (NordiCHI), 2010, pp. 615–618.
11. A.J. Brush et al., "Home Automation in the Wild: Challenges and Opportunities," Proc. Int'l Conf. Human–Computer Interaction (CHI), 2011, pp. 2115–2124.
12. Huang and M. Cakmak, "Supporting Mental Model Accuracy in Trigger-Action Programming," Proc. 2015 Int'l Joint Conf. Pervasive and Ubiquitous Computing (UbiComp), 2015, pp. 215–225.
13. S. Davidoff et al., "Principles of Smart Home Control," Proc. 8th Int'l Ubiquitous Computing (UbiComp), LNCS 4206, Springer-Verlag, 2006, pp. 19–34.
14. F. Kawsar and A.J. Bernheim Brush, "Home Computing Unplugged: Why, Where and When People Use Different Connected Devices at Home," Proc. 2013 Int'l Joint Conf. Pervasive and Ubiquitous Computing (UbiComp), 2013, pp. 627–636.
15. Perera, S. Aghaee, and A. Blackwell, "Natural Notation for the Domestic Internet of Things," Proc. Int'l Symp. End-User Development (ISEUD), 2015, pp. 25–41.
16. R. Yang and M. Newman, "Learning from a Learning Thermostat: Lessons for Intelligent Systems for the Home," Proc. 2013 Int'l Joint Conf. Pervasive and Ubiquitous Computing (UbiComp), 2013, pp. 93–102.

The Authors

Joëlle Coutaz is a professor emeritus at Université Grenoble Alpes and founder of the Engineering Human–Computer Interaction group within the Grenoble Informatics Laboratory (LIG). Her research interests include HCI, multimodal and tangible interaction, user interface plasticity, and end-user development for smart homes and ubiquitous computing. Coutaz received a PhD in computer science and Thèse d'Etat from Université Joseph Fourier. She is a member of the ACM SIGCHI Academy and, in March 2013, was awarded the National Order of the Legion of Honor (Chevalier de la Légion d'Honneur), the highest decoration in France for her pioneering work in HCI. Contact her at joelle.coutaz@imag.fr.

James L. Crowley is a professor at Grenoble Institut Polytechnique and a senior member of the Institut Universitaire de France (IUF). He performs research on computer vision, pervasive computing, and multi-modal interaction within the Grenoble Informatics Laboratory (LIG) at the Inria Grenoble Rhône-Alpes Research Center in Montbonnot, France. Crowley received a PhD from Carnegie Mellon University. In March 2014, he was awarded the National Order of Merit (Chevalier de l'Ordre National du Mérite) for his distinguished contributions to education, research, and innovation. Contact him at james.crowley@inria.fr.