IoT End User Programming Models

Steven P. Reiss
Department of Computer Science
Brown University
Providence, RI 02912
spr@cs.brown.edu

ABSTRACT

The advent of smart devices and sensors (the Internet of Things or IoT) will create increasing demands for the automation of devices based on sensor, time, and other inputs. This is essentially a programming task with all the problems and difficulties that programming entails, for example, modularity, feature interaction, debugging, and understanding. Moreover, much of the programming for smart devices is going to be done not by professional programmers but by end users, often end users without any programming experience or computational literacy. Our research is aimed at exploring the programming space and the associated issues using a case study of a smart sign that can be controlled using a variety of sensors. We have developed a general system for programming smart devices and, in this paper, explore a variety of different user interfaces for programming this system for our smart sign.

CCS CONCEPTS

Software and its engineering → **Software creation and management**; Software development techniques

KEYWORDS

Internet of things; end-user programming; debugging; program understanding.

1 INTRODUCTION

Imagine a "smart house" where everything is driven by software. Lights are not directly wired to switches; thermostats are advisory and not directly connect to the furnace; buttons exist to request open windows or skylights; doors may be unlocked by RFID keys (as in today's automobiles); etc. The owners of such a house would need to program it and keep the program up-to-date as new devices are added or as their needs change. Moreover, the owners are generally not going to be software engineers and will not necessarily understand programming or programming concepts.

Programming for such a smart house raises several issues. The first is what form a program for such a smart house should take. We lean toward a rule-based system since rules are generally easy to specify and can map directly to user interactions. However, the conditions and rules that are needed can be complex and involved.

The second issue is handling conflicts when creating rules and when attempting to understand what will happen based on the current set of rules. Any programming system for smart devices needs to have a scheme for addressing conflicts and needs to make such conflicts explicit and understandable to the end user.

A third issue is that the users will not create the perfect program initially. This means that the user interface for programming needs to handle debugging, particularly debugging with respect to potential conflicts.

The fourth issue is one of scale. While our simple case study only involves twenty-some rules, a smart house with tens of devices might require hundreds of rules. Can an interface be designed that can handle this large a set? Will end users be able to understand a "program" at this scale? What techniques will help in such understanding?

This paper describes our initial explorations of how to program smart devices on the scale needed for a smart house. We start by describing some of the related work in Section 2. Then we describe a case study, our smart sign, in Section 3. Next we describe what we see as a practical programming framework for smart devices based on a rule-based model in Section 4. Next we provide a set of four exploratory solutions we developed using our smart sign as an example in Section 5. We conclude by describing our experiences.

2 RELATED WORK

Pervasive computing has been touted for a long time. In the past few years, this technology has become more evident in the "Internet of Things". Here everyday devices are web-enabled so they can talk and potentially control one another. Individual smart devices, e.g. windows that close themselves when it rains, coffee makers that turn on in the morning, thermostats that program themselves, are become more common. Internet-based control of individual devices in the home is becoming common with major companies such as Verizon producing practical control systems. Smart houses were first popularized in science fiction (e.g. the television series The Jetsons), but have now become a reality. This has opened many new research questions and directions [32].

The trends we see here are moving from physical controls (such as switches and locks) to virtual controls (such as a phone app); moving from user-programmed devices to learning-based programming; and moving from explicit control to automated control. We see the trend in terms of the Internet of Things moving to devices which essentially control themselves according to the user's wishes.

Most of the techniques for programming devices and home automation are based on production systems. Production system have been widely studied over the years, are well understood, and have been used in a wide variety of applications [22]. The most widely used rule-based model for programming devices is trig-

ger-based. This is exemplified by IFTTT ("IF This Then That") [17]. It is both widely used and has been extensively studied. For example, [46] show that it is easy to learn and to program. Our rule-based approach builds on this. However, purely trigger-based approaches do not really cover all aspects and in many cases are not natural ways of specifying interactions [46,47]. This led us to develop our more general framework.

Moving from individual web-enabled devices to a larger set of interacting devices, for example, in a smart house, is non-trivial. Different sensors might trigger the device in different ways at the same time; different device settings might conflict with each other; there can be implicit constraints (e.g. do not heat and cool at the same time; do not turn on both the coffee maker and the microwave since that will blow a fuse) that should be enforced. This problem, the complexity involved in interacting devices and conflicts, has been noted by several others [3,20,24]. Others have noted that it is important in the user interface to deal with unusual situations (i.e. deviations from the routine) [12,31,49]. Our interfaces attempt to make such conflicts and interactions explicit.

The DeLorean system takes another approach to handling conflicts [9]. It assumes a rule-based language with trigger-based rules with arbitrary conditions and actions. Based on this language, they use timed-automata to let the user fast-forward their system in order to understand conflicting behaviors. This, and other model-checking based approaches are aimed more at the problem of predicting unusual behaviors while we are more interested in letting do their own exploration. Our interfaces attempt to take a more practical approach to showing conflicts, but do not preclude the use of formal checking.

Explanations of the behavior of rule-based systems are common [4,13,14]. Lim, Dey, and Avrahami [30] provide the user with explanations of the behavior of complex systems that are rule or machine-learning based and note that it can be useful to both show why an action was taken and to show why and action was not taken. Explanations for debugging are explored in WhyLine [23]. Explanations can also be used for debugging machine learning approaches [25]. Many explanations, especially for debugging, involve reachability questions [27]. Our interfaces include the ability to show what will happen and can easily be extended to show full explanations.

Learning interfaces for IoT devices are becoming more common as in the Nest thermometer [26]. The utility and practicality of such interfaces has been studied where it was shown that they should be combined with other interactive technologies for handling exceptions, providing descriptions, and user engagement [49]. Rule recommendation systems work along similar lines [48]. Our approach includes the ability to learn new rules based on user requests.

End-user programming [7,8,19,35,37,38] has a long history. Spreadsheets are the prime example of a very successful end-user programming system, employing a metaphor that is easily understood and enabling a wide range of programs. There has been significant work on debugging spreadsheets from this perspective [1,5,6,25]. Teaching programming concepts, especially to children, has been one theme for end-user programming. This started with Logo and has progressed to languages such as Alice [11], Scratch [40], Squeak [18], and similar systems [21]. A recent example is Toque [45], which uses cooking as a basis. These systems attempt to make complex constructs intuitive to the user. End-user debugging is also used for web applications [16]. Natural programming languages and end-user oriented environments are another approach [34,36]. Our approach draws inspiration from all of these.

FIGURE 1. The automated sign outside my office.

Another theme of end user programming has been programming-by-example [2,15,29,44] or programming-by-demonstration [10,33,41-43]. This has been used for IoT devices to a limited extent [28,39] Our learning interface provides an basic implementation of programming-by-demonstration for IoT devices.

3 SMARTSIGN: A CASE STUDY

I have a sign outside my office. Actually its not a sign but a inexpensive 10" Android tablet running a kiosk application. (It was originally a Bluetooth digital picture frame.) The sign displays my current status, indicating whether I am available, whether I'm with someone, whether I'm on the phone, or, if I'm out of the office, when I'm likely to be back. It does this all automatically. The sign can be seen in Fig. 1. The sign has been running continually for about 3 years at this point and serves as an interesting, well-tested, case study.

The sign software detects whether I'm in my office by seeing if it can see my phone via Bluetooth. It determines if I have a visitor by using a motion detector aimed at my seating area. It determines if I'm on the phone using an off-the-hook circuit attached to my phone line. It accesses my Google calendar to determine if I'm at a meeting or on vacation and when I'll be back. It accesses the web to determine the current weather conditions and temperature.

The sign works off a set of about twenty-five rules. The rules have a relatively simple form, for example,

IF <in_office> AND <time between 8:30am and 5pm, Monday thru Friday> THEN <I'm available>
IF <visitor> THEN <With visitor>
IF <in_office> THEN <I'm hiding>

This form consists of a set of conditions and then the action to perform when the conditions hold. In the case of the sign, the action is simply to display an image (created as an SVG diagram) which is done by updating an image for a web page. The conditions include the basic ones determined by sensors (e.g. *in_office*), time-based conditions, Google-calendar based conditions, weather-based conditions, and some artificial conditions that

were created to simplify the rules. Examples of the latter include a condition which is true if I've been out of the office for less than five minutes (likely stepped out to use the facilities or get a soda), and one that is true if I've been in at all during the day.

The rules are given in priority order, so that the first rule whose conditions are satisfied is applied. Thus, with the above three rules, the last one is applied if I'm in the office and it is not during what I consider work hours.

In addition to displaying my current status on the sign display, the system also provides my current status on my personal web site (http://www.cs.brown.edu/people/spr/status.html).

4 PROGRAMMING CONSIDERATIONS

There are many different alternatives for control languages for embedded or smart-house applications. One could start with an actual programming language, for example Python, and add basic methods that check conditions and implement controls. This is what is done, for example, in Sikuli [50] for user interface testing. However, given the difficulties involved in understanding and teaching computation, this does not seem to be a feasible approach for the average homeowner who knows nothing about programming. Alternatively, one could work in terms of finite state machines, which are a natural model for many devices. However, these again present a not-easily-understood formalism that could be confusing to non-programmers and would require extensions to handle time-based conditions.

A simpler alternative is a rule-based approach, essentially a production system. Here there are a set of rules that are checked in some order. The rules consist of a condition and action. Rules are evaluated by checking the condition, and then, if it holds, taking the corresponding action. Even with this constrained framework, there are several design alternatives that need to be considered.

The first alternative is whether rules are considered as triggers or as conditions that are (in theory at least) continuously checked. An common implementation of a rule-based approach for devices or home automation is IFTTT ("IF This Then That") [17]. This consists of conditions that are triggers and an action that should be evaluated when the trigger conditions occur. This type of rule will be helpful in a smart house (e.g. send me a text message if the burglar alarm goes off), but is not sufficient. Most rules for our sign (and eventually for a smart house) are designed to be evaluated continuously. Consider, for example, "If I'm in the office then display 'I'm available'. This is more natural than saying "When I enter the office, display 'I'm available' since the condition is designed to hold whenever I'm in the office, not just when I enter it. Moreover, other displays are possible while I'm in the office (e.g. On the Phone, With a Visitor, Hiding) and defining trigger-based transitions for all such events would be tedious and error-prone. Moreover, such events also need to consider time (the actual rule includes normal office hours), which make a trigger-based approach even more complex. Moreover, one can easily create conditions where the number of trigger-only rules could grow quite large. For example, consider N light switches and N lights where each switch turns on a different subset the lights (e.g. each light is based on OR conditions over a subset of the switches).

Given that both trigger-based and continuous rules are needed, a second design alternative involves how to handle multiple triggers and how to distinguish between trigger and continous rules. It is not uncommon for users to try to create rules of the form "If the doorbell rings at 3:00pm" [47]. Such rules will almost never be triggered since it is unlikely that someone will ring the door at precisely (to the millisecond) 3:00pm. Similarly, using trigger-like actions (e.g. send an email) under continuous conditions could result in unexpected or unwanted results.

To avoid these situations, our approach distinguishes both conditions and actions as either continuous or trigger-based. A rule can either include zero or one trigger-based condition. If it has none, then the action has to be continuous. If it has one, then the action has to be trigger-based. The front end is also aware of these constraints and enforces them as the user creates rules.

The third design alternative is what types of conditions are allowed. It is easy to code the system to allow arbitrary logical conditions, i.e. combinations of AND, OR, and NOT. However, this quickly becomes confusing without parenthesis even to the experienced programmer since there is no implicit priority ordering for these operators as there is for addition and multiplication. Moreover, creating a easy-to-use interface that supports all three also becomes difficult. Based on our initial experiences attempting to develop interfaces for our sign, we determined that a more viable alternative was just to allow AND rules. This makes the rules easier to comprehend and the interface more tractable.

A problem with only allowing AND rules is that there are conditions that are naturally OR conditions, for example turn on light 2 if either switch A is on OR switch B is on. Simple OR conditions like this can easily be handled by using multiple rules. To handle more complex situations, our experience has shown that it is often easier to create new basic conditions (effectively pseudo-sensors) than to create more complex rules. For example, we have both an On-Phone condition and a Not-On-Phone condition. For the switch, one could create a new sensor which was "Either A or B is on".

New sensors are useful for other applications as well. For example, we have defined sensors that combine other sensor states with time for use with our sign. One is whether we have stepped out of the office for less than five minutes; another is whether we have come in at all during the day. Such sensors will also be useful within a smart house (i.e. for turning off lights when there is nobody around for some time or automatically setting the alarm to external-doors only if someone is in the house at night). We note that such new sensors can be used to turn a trigger-condition into a continuous condition when needed.

The current front end allows new sensors to be defined interactively for durations (some event occurring for either more than or less than a given interval); for latches (some event occurring at all with a given timeout or reset time); and for combinations (ORs of events). Other sensors, for example those based on RSS feeds and web pages are implemented, but the front end for defining them is still missing since we have not found them particularly useful for our smart sign. The implementation also includes a generic sequence sensor that detects and ordered sequence of events and maintains a corresponding internal state (essentially a finite state machine), although again, we have not implemented the corresponding interfaces for building such sensors. More complex new sensors can be defined programmatically as needed.

A fourth design alternative involves determining what rules (assuming a priority order) should be applied as conditions

change. With the sign, it is rather simple. The first rule (in priority order) whose condition is true will affect the sign and hence it is applied, and any subsequent rules are ignored since they would also affect the sign. However, in a smart house multiple rules affecting different, independent devices might be triggered by a single condition. For example, if it starts to rain each of the skylights should be closed. In this case, it seems logical to allow multiple active rules while still maintaining rule priorities for each device. One needs to make it explicit to the user, however, what rules preclude or do not preclude others as it might not be immediately obvious (for example if both close-window and close-curtain are triggered by different rules for the same window, would both occur or only one; similarly if a rule has multiple actions associated with it and one of those actions conflicts with a prior rule, should the other actions be taken or ignored).

A smart house will require a large rule set. It is essential for understandability that this rule set be organized in a meaningful way. The common notion in programming languages is to modularize the rules. This seems a logical approach. However, there are multiple ways that such modularization can be done. For example, one could show all rules that affect a given device, or one could show all rules that are affected by a given sensor, or all the rules that are might be in effect at a given time, or some combination of these. The fifth design alternative involves how to do this modularization, whether it should be done statically (i.e. rules are defined in modules), or dynamically (where the user specifies a device or condition or time and the system creates an implicit module for corresponding rules).

Modularization of the rules has other uses as well. It can, for example, make the implicit finite state machines embodied by the various devices in the house explicit. This is done by viewing each state in the machine as a "module" and letting the user define the rule set for just that state. This requires a flexible modularization approach and might require artificial sensors that reflect the implicit state of various devices.

A sixth design alternative involves handling complex sensors. With the sign we needed special sensors to handle time and access to ones Google calendar. Time was modeled by how events (and repeated events) are typically defined in modern calendar systems. This provides an interface that people might be accustomed to, but still might be overly complex for simple conditions. Events derived from a Google Calendar were a bit more complex since we had no example to build against. Our current implementation lets the user choose specific fields of the calendar event (e.g.Where, Who, State, Visibility, Title, Which Calendar) and define strings that should or should not occur in those fields. Again, for simplicity and understandability, the calendar event matches if all specified fields are matched. For example, we can create rules for meetings that are not in my office by looking for events where the WHERE field does not contain "403" (my office number). We expect that for a smart house there will be other sensors of similar complexity that will have to be defined and handled, for example using a camera to detect motion or the presence of people in a room.

We also have implemented sensors for accessing RSS feeds and web pages. The RSS feed sensor is triggered on each new feed item and makes the title and description available to rules. The web page sensor takes a URL, a check frequency, and a CSS-style selector for the text of interest. It is used to check weather conditions and temperatures.

5 SAMPLE PROGRAMMING INTERFACES

The programming considerations cited above need to be consolidated into a user interface or programming system that can be used to effectively control our sign or a smart house. As a first step toward exploring what is needed and what might work here, we prototyped a control system and four different web-based interfaces for programming the smart sign. These interfaces are designed to be automatically generated based on a description of the underlying conditions and available actions and hence should be able to be used for a smart house, provided they scale appropriately.

5.1 The Control System

To experiment with the different end-user programming interfaces we developed an underlying control system that supports a wide variety of rule types and thus can accommodate many of the alternatives described above. The rules supported by the system are based on the notion of input sensors and output actuators. While it is simple to state that the smart house consists of separate sets of sensors and actuators, this by itself is not a realistic model. Many devices are actually both sensors and actuators. Even a simple light bulb can provide input information as to whether it is burned out or not. A dimmer light switch might include a display showing the current light setting. An alarm panel will display the current state of its sensors and the alarm state as well as letting the user change state, bypass sensors, etc. For this reason, the control system is based on devices, with each device having 0 or more sensors which are externally viewable parameters, 0 or more internal parameters (hidden state), and 0 or more actions.

The control system supports a web-callable (RESTful) API that lets us define web-based front ends that can create arbitrary rules and get information about the system. To facilitate more complex interfaces, it includes the notion of a *hypothetical world*. Such a world can be created by cloning the current (real) world or an existing hypothetical world. Within a hypothetical world, conditions and time can be set arbitrarily, and actions can be taken without actually changing the real world. This lets external tools (and hence the user) explore the effect of the rule set without actually changing any devices. This can be used, for example, to provide an interface that would show the user what happen under different conditions, something that is very useful for both debugging and understanding the potential effect of new rules.

The control system is designed to handle more than the smart sign. It includes authentication to provide limited or selective access to the rule set. It includes sensor and rule types that are not used for the smart sign, notably trigger rules (one-shot) and sensors that act as finite state automata and maintain an internal state. It can handle multiple output devices. New devices and sensors can be added dynamically.

Using this control system we have built four potential interfaces to explore different user programming models that might be used in coding a smart house. These are a programmer-oriented interface, and interface specialized to rule creation, an interface for learning rules, and an interface to explore modularity. While the interfaces have been explored for the smart sign application, they are all generated automatically from a description of the

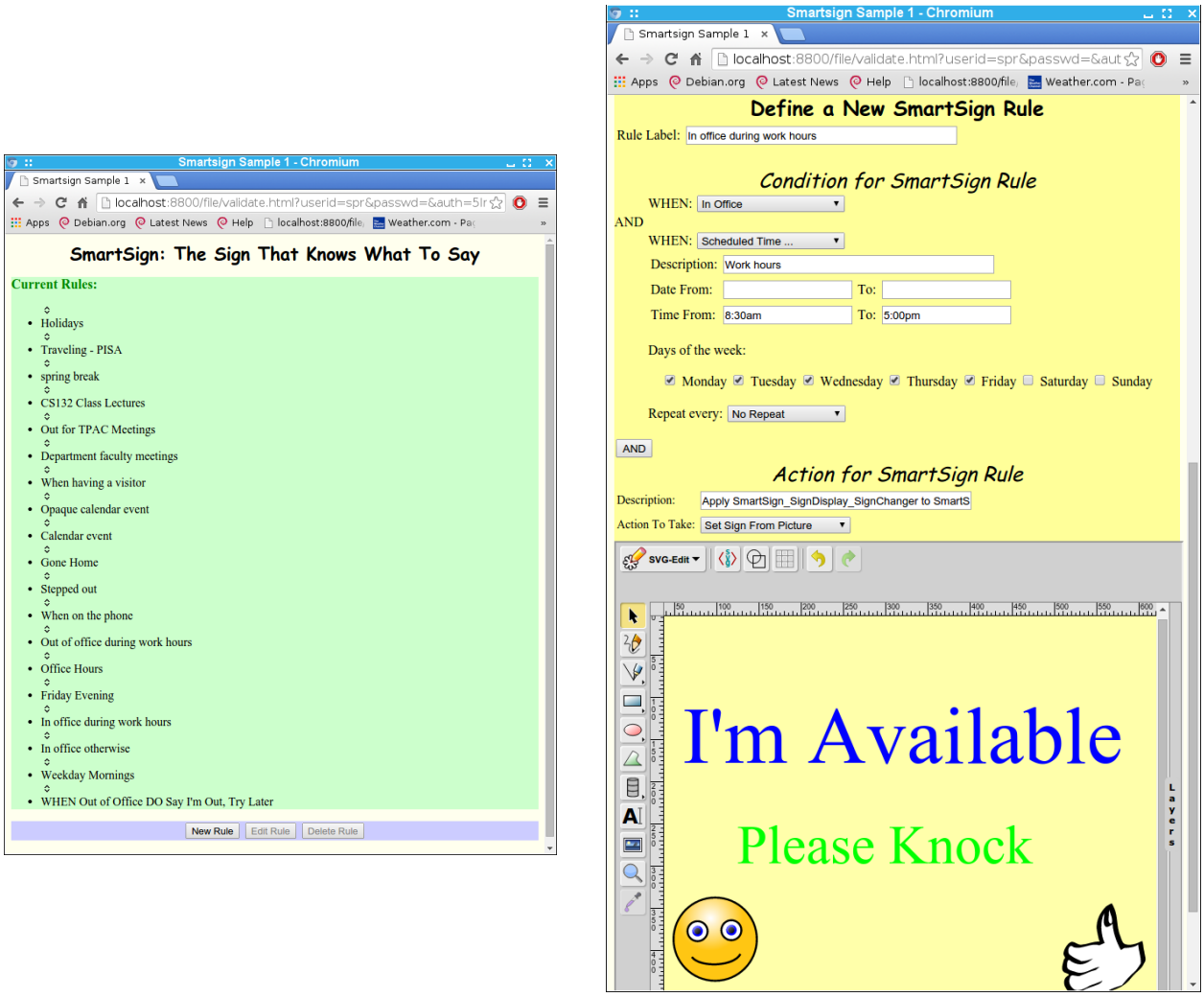

FIGURE 2. Programmer Interface to the smart sign. The image on the left shows the current rules and lets the user drag and drop rules to change priorities. The image on the right shows the interface for defining a new rule or editing an existing one.

devices and sensors and not specialized to the smart sign. We note that these are experimental prototypes and not professionally designed interfaces.

5.2 The Programmer Interface

The first interface, shown in Fig. 2, is what we call the programmer interface. It is designed for those who understand production systems and want complete control over the system. It consists of the list of all rules in priority order (using labels provided by the user when the rules were defined) where the user can drag rules around to change priorities. In addition, one can select a rule to edit, can delete a rule, and can define a new rule. Selecting "New Rule" or "Edit Rule" brings up a dynamic display seen on the right of the figure where the user can provide one or more conditions and then use an SVG editor to define the image for the sign.

This is the interface that we generally use personally when programming the sign (we are programmers after all). The rule listing part is relatively simple, and with twenty-some rules. The rules generally fit on one web page and it is relatively easy to find the rule one is looking for. The interface is not particularly good, however, if one is not already familiar with the existing rules since it provides no hints as to placement of a new rule. This has led to several occasions where we had to debug new rules because they didn't operate as intended in unusual situations. Moreover, this interface will not scale easily to systems with larger sets of rules.

5.3 The New-Rule Interface

The second interface, shown in Fig. 3, we call the new-rule interface since it emphasizes defining new rules and understanding their impact. It consists of the interface for creating a rule used in the programmer interface at the top and then a display of all the rules at the bottom that this potential new rule would conflict with and, by default, would be overridden if the rule were created as specified. The idea is that the user starts by specifying some basic trigger or condition for the new rule. As conditions are added, the set of conflicting rules is narrowed down. If the user decides that this rule should not override all of the conflicts that are shown, they can add additional conditions to specify when this rule should be applied or can select where the new rule should go in the listed hierarchy. When the user is satisfied that

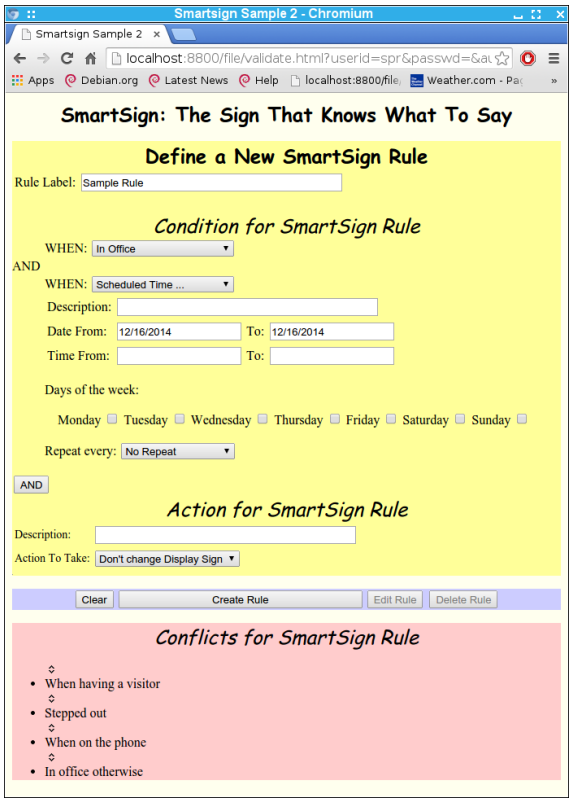

FIGURE 3 New-Rule interface to the smart sign. This interface lets the user start defining a new rule by specifying its conditions. As they do so, the set of conflicting rules is displayed at the bottom.

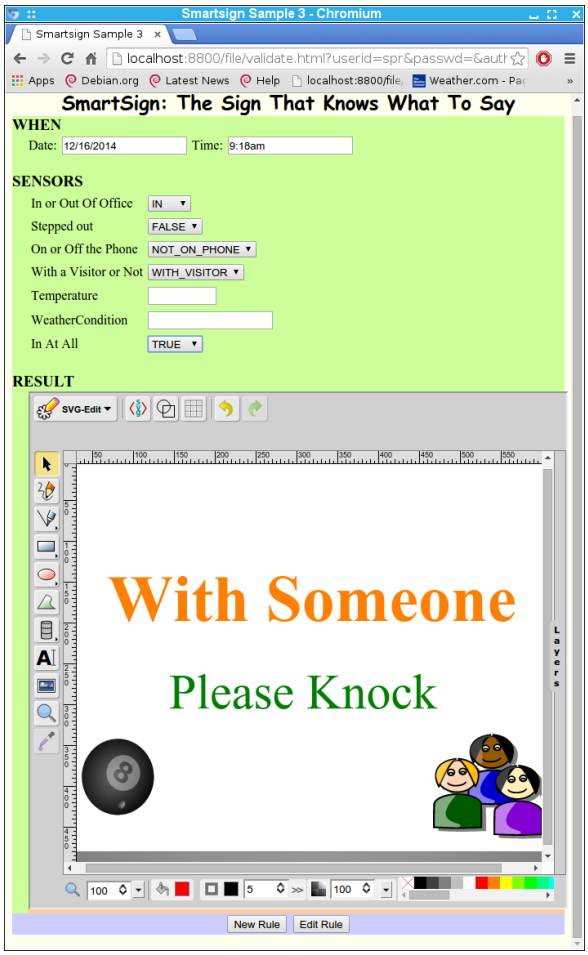

FIGURE 4The learning interface. The display shows a set of conditions (by default the current ones), and then shows what the current rules would result in under those conditions. The user can then change the display to indicate what the state should be. The system will deduce new rules based on the user specification.

the new rule is appropriate (i.e. that all the conflicting rules should indeed be overridden by this rule), they can click on the Create Rule button. Selecting an old rule before clicking this button causes the new rule to be added after that rule rather than before all the listed rules.

This interface scales somewhat better than the programmer interface. For the smart sign, it does reasonably well, because all the listed rules are essentially relevant to the sign and can quickly narrow the set of rules to understand conflicts. For a smart house, there might still be a large set of rules, many of which were irrelevant to what the user was intending. Another aspect of the interface is that the placement of the new rule is relative to the given rules, not absolute. This means that the actual priority of the new rule is not obvious to the user.

5.4 The Learning Interface

The third interface, shown in Fig. 4, we call the learning interface. It is loosely based on devices such as the Nest thermostat [26]. It starts with a display where the user can specify the time and a set of conditions. (Google calendar conditions are actually read from the user's calendar based on the time specification.) The time can be in the past or future and defaults to the current time. Based on the condition settings, the system computes what would be displayed and shows that to the user. (This uses the hypothetical worlds feature of the controller.) The user can then edit the displayed state and hit "New Rule". ("Edit Rule" changes the output for the rule that caused the display.) This causes the system to remember the conditions under which this rule should be applied. The system then builds a decision tree based on these conditions, any prior conditions that were defined using this interface, and the current set of rules defined using the other interfaces.

Creating a decision tree is done using the standard machine learning algorithm. The system uses the resultant decision tree to determine appropriate conditions for triggering all the rules created using this interface and then creates the corresponding rules with appropriate priorities. The decision tree approach is relatively straightforward, but there are unsolved problems.

The main problem is that the interface has the user specify a particular instance in time. This makes sense when exploring the rule set and for handling time-based external sensors such as cal-

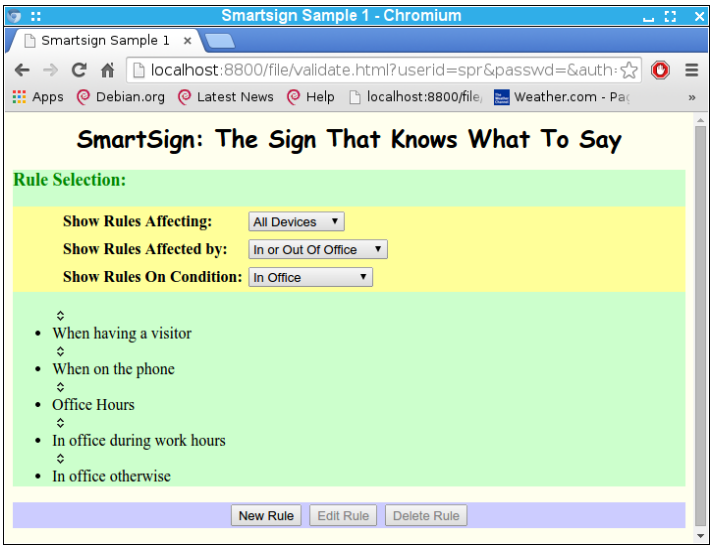

FIGURE 5 The modular interface. Here the user chooses a device, sensor or condition and the system displays the rules relevant to that object. The user can then change the relative priority of the given rules, can edit one of the relevant rules, or can create a new rule relevant to the context.

endar events. However, the rules are based on time intervals, not time instances. This leads to the difficult problem of converting the time instant from the specification into an appropriate time interval. This is currently done by keeping a history of sensor changes over time and using this history to determine what a reasonable interval should be for this rule and whether there are any repeats. Our experience to date is that this method is not particularly effective, possibly because our schedule is somewhat erratic.

Because many rules are time-based and the problem of inferring time is difficult, this interface has not been used extensively for creating rules. However, it has proven effective for understanding what this system is doing and hence for debugging purposes. One can use it to easily explore what will happen under different conditions, to see if the rules are correct, and to determine if new rules might be needed. It would be better for such exploration if it also showed which rule(s) were being triggered (although this is generally obvious for the sign from the display), and provided a means for checking why a particular rule was not triggered.

5.5 The Modular Interface

The fourth interface, shown in Fig. 5, we call the modular interface. It supports dynamic modularization of the rules sets. It starts by letting the user choose how to modularize, either by device, by sensor, or by condition, or some combination of these. (It currently does not support modularization by time.) Once the user chooses the modular condition, the rule set relevant to that condition is displayed as it was in the programmer interface. Now the user can rearrange the rules for this modularization, can edit one of these rules, or can create a new rule that is appropriate to the modularization.

This interface provides a convenient means for scaling the simple programmer interface to large sets of rules. However, it still requires understanding priorities and does not provide feedback on what will happen under different conditions or what rules conflict with each other. Another aspect of this interface that might be confusing to the user is what happens to the priority of a modular rule with respect to rules not displayed when the priority is changed.

A similar modular interface could be created for new rule and learning interfaces. Modularization in these cases would be most useful for particular devices.

6 EXPERIENCE

Our smart sign has been running for several years at this point, providing us with enough experience to understand many of the design issues and to realize potential next steps.

The current rule set is relatively stable. We edit the rules about once a month, generally to take into account travel, vacation, and recurrent events that change each semester, and then generally to reset the times associated with the affected rules. This generally involves editing existing (or previously used) rules, not creating new ones from scratch.

The underlying control system is stable and robust. The web front ends, a little less so. This is particularly true for the SVG editor, which is a bit quirky, and can create SVG that can not be converted to an image directly. The interfaces are still a bit primitive and could use a good redesign. The main continuing issues we have found are in the connectivity with the tablet display and the Bluetooth on our phone.

We see several directions for extending these efforts. One is to define meta-rules that can generate low-level rules automatically (e.g. in a room with a ceiling light and a wall switch, the switch should control the light). Another is to be able to specify properties and validate that the rule set enforces those properties. A third involves determining how to do learning effectively, particularly learning of time slots.

7 REFERENCES

1. Robin Abraham and Martin Erwig, "Goal-Directed Debugging of Spreadsheets," pp. 37-44, in *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing* (2005).
2. G. Attardi and M. Simi, "Extending the Power of Programming by Examples," in *Integrated Interactive Computing Systems*, ed. P. Degano and E. Sandewall, North-Holland (1982).

3. Kenneth H. Braithwaite and Joanne M. Atlee, "Towards automated detection of feature interactions," pp. 36-59, in *Feature Interactions in Telecommunications Systems*, ed. Weit Bouma and Hugo Velthuijsen, IOS Press (1994).

4. Andrea Bunt, Matthew Lount, and Catherine Lauzon, "Are explanations always important?: a study of deployed, low-cost intelligent interactive systems," pp. 169-178, in *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (2012).

5. Margaret Burnett, Bing Ren, Andrew Ko, Curtis Cook, and Gregg Rothermel, "Visually testing recursive programs in spreadsheet languages," *IEEE Symposium on Human- Centric Computing Languages and Environments* (September 2001).

6. Margaret Burnett, Curtis Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Chris Wallace, "End-user software engineering with assertions in the spreadsheet paradigm," *International Conference on Software Engineering 2003* (May 2003).

7. Margaret M. Burnett and Brad A. Myers, "Future of end- user software engineering: beyond the silos," pp. 201-211, in *Proceedings of the on Future of Software Engineering* (2014).

8. Margeret M. Burnett, "End-user development," in *The Encyclopedia of Human-Computer Interaction, 2nd Edition*, ed. Mads Siegaard and Rikke Friis Dam, The Interaction Design Foundation (2014).

9. Jason Croft, Ratul Mahajan, Matthew Caesar, and Madan Musuvathi, "Back to the future: Forecasting program behavior in automated homes," *Microsoft Technical Report MSR-TR-2012-131* (May 2012).

10. Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, *No Code Required: Giving Users Tools to Transform the Web*, p. 512, Morgan Kaufmann Publishers Inc. (2010).

11. Wanda P. Dann, Stephen Cooper, and Randy Pausch, *Learning To Program with Alice*, Prentice Hall Press (2008).

12. Scott Davidoff, Min Kyung Lee, Charles Yiu, John Zimmerman, and Anind K. Dey, "Principles of smart home control," *Proceedings of the 8th International Conference on Ubiquitous Computing*, pp. 19-34 (2006).

13. Kate Ehrlich, Susanna E. Kirk, John Patterson, Jamie C. Rasmussen, Steven I. Ross, and Daniel M. Gruen, "Taking advice from intelligent systems: the double-edged sword of explanations," pp. 125-134, in *Proceedings of the 16th International Conference on Intelligent User Interfaces* (2011).

14. Shirley Gregor and Izak Benbasat, "Explanations from intelligent systems: theoretical foundations and implications for practice," *MIS Quarterly* **23**(4), pp. 497-530 (1999).

15. D. C. Halbert, *An example of programming by example, Ph.D. dissertation, Department of Computer Science, University of California, Berkeley (1981).*

16. Mouna Hammoudi, Ali Alakeel, Brian Burg, Gigon Bae, and Gregg Rothermel, "Facilitating debugging of web applications through recording reduction," *Empirical Software Engineering* (2017).

17. IFTTT, "If This Then That," http://ifttt.com (2014).

18. Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay, "Back to the future: the story of Squeak, a practical Smalltalk written in itself," *Proceedings ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications ,97*, pp. 318-326 (1997).

19. Andri Ioannidou, Alexander Repenning, and David C. Webb, "AgentCubes: Incremental 3D end-user development," *J. Vis. Lang. Comput.* **20**(4), pp. 236-251, Academic Press, Inc. (2009).

20. Michael Jackson and Pamela Zave, "Distributed feature composition: a virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering* **24**(10), pp. 831-847 (1998).

21. Caitlin Kelleher and Randy Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys* **37**(2), pp. 83-137 (June 2005).

22. D. Klahr, P. Langley, and R. Neches, *Production System Models of Learning and Development*, MIT Press (1987).

23. Andrew J. Ko and Brad A. Myers, "Debugging reinvented: asking and answering why and why not questions about program behavior," *International Conference on Software Engineering 2008*, pp. 301-310 (May 2008).

24. Mario Kolberg, Evan Magill, Dave Marples, and Simon Tsang, "Feature interactions in services for Internet personal appliances," *Proceedings of the IEEE International Conference on Softare Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 2613-2618 (2002).

25. Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf, "Principles of Explanatory Debugging to Personalize Interactive Machine Learning," pp. 126-137, in *Proceedings of the 20th International Conference on Intelligent User Interfaces*, ACM, Atlanta, Georgia, USA (2015).

26. Nest Labs, "Nest Thermostat," http://nest.com/thermostat (2014).

27. Thomas D. LaToza and Brad A. Myers, "Developers ask reachability questions," pp. 185-194, in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, Cape Town, South Africa (2010).

28. Toby Jaa-Jun Li, Yuanchun Li, Fanglijn Chen, and Brad A. Myers, "Programming IoT devices by demonstration using mobile apps," *International Symposium on End User Development (IS-EUD 2017)*, pp. 3-17 (June 2017).

29. H. Lieberman, "Designing Interactive Systems from the User,s Viewpoint," in *Integrated Interactive Computing Systems*, ed. P. Degano and E. Sandewall, North-Holland (1982).

30. Brian Y. Lim, Anind K. Dey, and Daniel Avrahami, "Why and why not explanations improve the intelligibility of context-aware intelligent systems," pp. 2119-2128, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2009).

31. Sarah Mennicken and Elaine M. Huang, "Hacking the natural habitat: an in-the-wild study of smart homes, their development, and the people who live in them," *Proceedings of the 10th International Conference on Pervasive Computing*, pp. 143-160 (2012).

32. Sarah Mennicken, Jo Vermeulen, and Elaine M. Huang, "From today,s augmented houses to tomorrow,s smart homes: new directions for home automation research," *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 105-115 (2014).

33. Brad A. Myers, Brad Vander Zanden, and Roger B. Dannenberg, "Creating graphical interactive application objects by demonstration," *Proceedings UIST ,89*, pp. 95-104 (November 1989).

34. Brad A. Myers, John F. Pane, and Andy Ko, "Natural programming languages and environments," *Commun. ACM* ***47***(9), pp. 47-52, ACM (2004).

35. Bonnie A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*, MIT Press (1993).

36. Stephen Oney, Brad Myers, and Joel Brandt, "InterState: a language and environment for expressing interface behavior," pp. 263-272, in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, Honolulu, Hawaii, USA (2014).

37. John F. Pane, Brad A. Myers, and Leah B. Miller, "Using HCI Techniques to Design a More Usable Programming System," p. 198, in *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC,02)*, IEEE Computer Society (2002).

38. Fabio Paterno, "End user development: survey of an emerging field for empowering people," *ISRN Software Engineering article 532659* **2013** (2013).

39. Marissa Radensky, Toby Jia-Jun Li, and Brad A. Myers, "How end users express conditionals in programming by demonstration for mobile apps," *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 311-312 (October 2018).

40. Mitchel Resnick, John Maloney, Andres Monrow-Hernandez, and Natalie Rusk, "Scratch: programming for all," *Communications of the ACM* **52**(11), pp. 60-67 (Nov. 2009).

41. Robert V. Rubin, Eric J. Golin, and Steven P. Reiss, "ThinkPad: a graphical system for programming-by- demonstration," *IEEE Software* **2**(2), pp. 73-78 (March 1985).

42. Robert V. Rubin, Steven P. Reiss, and Eric J. Golin, "Compiler aspects of an environment for programming by demonstration," *Proceedings International Workshop on Visual Aids To Programming* (May 1986).

43. John Stasko, "Using direct manipulation to build algorithm animations by demonstration," *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, pp. 307-314 (1991).

44. Phillip Dale Summers, "Program construction from examples," Yale research report 51 (1976).

45. Sureyya Tarkan, Vibha Sazawal, Allison Druin, Evan Golub, Elizabeth M. Bonsignore, Greg Walsh, and Zeina Atrash, "Toque: designing a cooking-based programming language for and with children," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2417-2426 (2010).

46. Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman, "Practical trigger-action programming in the smart home," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 803-812 (2014).

47. Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman, "Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes," pp. 3227-3231, in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, San Jose, California, USA (2016).

48. Beidou Wang, Xin Guo, Martin Ester, Ziyu Guan, Bandeep Singh, Yu Zhu, Jiajun Bu, and Deng Cai, "Device-Aware Rule Recommendation for the Internet of Things," pp. 2037-2045, in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, ACM, Torino, Italy (2018).

49. Rayoung Yang and Mark W. Newman, "Learning from a learning thermostat: lessons for intelligent systems for the home," *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pp. 93-102 (2013).

50. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller, "Sikuli: using GUI screenshots for search and automation," *Proceedings UIST 2009*, pp. 183-192 (2009).