Automatic Trigger Generation for Rule-based Smart Homes

Chandrakana Nandi Michael D. Ernst
University of Washington
{cnandi, mernst}@cs.washington.edu

Abstract

To customize the behavior of a smart home, an end user writes rules. When an external event satisfies the rule's trigger, the rule's action executes; for example, when the temperature is above a certain threshold, then window awnings might be extended. End users often write incorrect rules [16]. This paper presents a technique that prevents *errors due to too few triggers* in the rules. The technique statically analyzes a rule's actions to determine what triggers are necessary.

We implemented the technique in a tool called TrigGen and tested it on 96 end user written rules for openHAB, an open-source home automation platform. It identified that 80% of the rules had fewer triggers than required for correct behavior. The missing triggers could lead to unexpected behavior and security vulnerabilities in a smart home.

Keywords Security, Home automation, Trigger action programming, Static analysis

1. Introduction

Most home automation platforms support end-user customization: a user writes rules to determine what actions should be taken by what device under what conditions [21]. For example, Samsung SmartThings [8] allows users to create automation rules through the "SmartApps" feature, while Apple HomeKit [4] allows users to set conditions that govern when an action should take place.

A rule has two main components: triggers that cause a rule to be fired, and actions to be executed when a rules fires. This is also called Trigger Action Programming (TAP) [24]. Listing 1 shows an example of a rule.

Even though TAP is the most commonly used and practical approach for home automation [13, 24], end-users often make errors in writing trigger-action programs [16, 25]. In a smart home with multiple interacting devices, an error in one rule can cause unexpected behavior or security vulnerabilities in another part of the house.

This paper addresses errors in writing triggers. Our approach eliminates a certain category of error in the rules—*errors due to too few triggers*. We have built a static analysis that determines a necessary and sufficient set of trigger conditions for the rules. These inferred triggers can be compared to the user-written triggers to indicate whether there are too few triggers. Other possible uses for the technique are to detect unnecessary triggers or to free users from the need to write triggers.

```
rule "Update max and min temperatures"
when
  // trigger block:  1 or more triggers
  Item Temperature changed
then
  // action block:  1 or more actions
  postUpdate(Temp_Max,
    Temperature.maxSince(now.toDateMidnight))
  postUpdate(Temp_Min,
    Temperature.minSince(now.toDateMidnight))
end
```

Listing 1: A rule that updates the maximum and minimum temperature values for the current day.

We implemented a tool, TrigGen, and ran it on 96 home automation rules written by end-users for the openHAB framework [6]. Following is a summary of TrigGen's outputs:

1. TrigGen generated a correct set of event-based triggers for 91 rules (95%). It failed to produce any output for the remaining 5 (5%) rules.
2. For 77 of the 96 rules (80%), the user had written an insufficient number of event-based triggers, which could lead to inconsistent behavior.

The contributions of this paper are the following:

1. Identification of end-user written automation rules as a source of errors and security vulnerabilities, due to *too few triggers*.
2. A static analysis tool, TrigGen, to determine a necessary and sufficient set of triggers, based on the actions written by end-users. TrigGen can be used to
   - automatically generate all event-based triggers thereby entirely freeing the user from writing them,
   - identify missing triggers, and
   - eliminate unnecessary triggers.
3. An evaluation of TrigGen on real end-user written rules for home automation in the openHAB framework. It 1) generated a correct set of event-based triggers for 95% of the rules and 2) revealed that the user-written event-based triggers were insufficient for 80% of the rules.

2. Motivating Examples

The rule in listing 2 was written by an end user [2]. The name of the rule is `Away rule`. An `Item` represents the state of a device in the house. A rule can read or write to an item in order to interact with the device. The state of an item can be changed 1) by direct physical interaction of the end-user with the device (pressing a switch), 2) by the device sensing a change in the value of some property it measures (e.g., a thermostat sensing change in temperature), 3) by the end-user through the UI of the openHAB application on a smart phone or on a desktop, or 4) as an effect of a rule firing.

PLAS'16 October 24, 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2993600.2993601

```
rule "Away rule"
when
  Item State_Away changed
then
  if (State_Away.state == ON) {
    if (State_Sleeping.state != OFF) {
      postUpdate(State_Sleeping, OFF)
    }
  }
end
```

Listing 2: Rule for setting the Away or Sleeping state.

```
rule "Christmas lamps on"
when
  Item State_Sleeping changed from ON to OFF
then
  if (Auto_Christmas.state == ON) {
    if (State_Sleeping.state == OFF &&
        State_Away.state == OFF) {
         sendCommand(Socket_Livingroom,ON)
         sendCommand(Socket_Floor,ON)
    }
  }
end
```

Listing 3: Rule for turning on lights during Christmas.

This rule (listing 2) is supposed to ensure that when `State_Away` is `ON`, `State_Sleeping` is `OFF`. If `State_Away` is `ON`, it indicates that the home inhabitants are away. If `State_Sleeping` is `ON`, it indicates that the inhabitants are sleeping in the house. If the intention of the user behind this rule is to ensure that `State_Away` and `State_Sleeping` are not `ON` simultaneously, then this rule violates the user's expectation—the rule fires when `State_Away` changes but not when `State_Sleeping` changes. This allows both values to be set to `ON` at the same time, which deviates from the expected behavior of the rule. The correct version of the rule should include *both* `State_Away` and `State_Sleeping` as triggers.

Consider another example of a rule written by an end-user [2] shown in listing 3. The action block of the rule indicates that the Christmas lights should be turned `ON` when both `State_Away` and `State_Sleeping` are `OFF`. The rule is only fired when `State_Sleeping` changes to `OFF`. Thus, if `State_Sleeping` is already `OFF` because the inhabitants are away (i.e. `State_Away` is `ON`), once they come home, the lights will not turn on automatically. If the intention of the user behind writing this rule is to ensure that whenever people are home and awake during Christmas, the lights must be on, then this rule deviates from this behavior.

These examples show that even if the action block of a rule is implemented correctly, inadequate triggers can lead to too few firings of the rule. This may give rise to security vulnerabilities (as described in section 3) or unexpected behavior. To solve this problem, we propose a technique that can automatically generate event-based triggers and also detect missing ones. It works by statically analysing the abstract syntax tree (AST) of the code in the action block.

3. Example attack

Since TrigGen can prevent incorrect behavior due to *too few triggers*, it can also help eliminate security loopholes in the house that arise due to *too few triggers*. Our approach can prevent any attack that relies on an incorrect number of rule firings.

Example attack. Smart homes often have a visitor notification system that sends a message to the owner's smart phone when someone is near the house. Consider the rule shown in listing 4 that deactivates the visitor notification system when everyone is sleeping in the house and activates it at all other times. This ensures that when

```
rule "Visitor notification system rule"
when
  Item State_Sleeping changed
then
  if (State_Sleeping.state == ON) {
      postUpdate(Notification_System, OFF)
  } else {
      postUpdate(Notification_System, ON)
  }
end
```

Listing 4: Rule for activating the visitor notification system in the house.

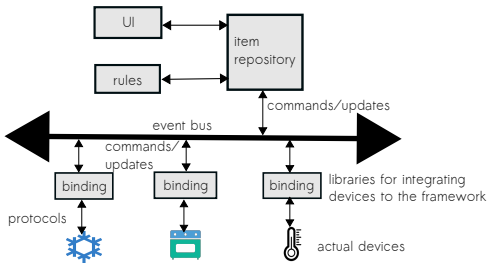

Figure 1: openHAB framework.

inhabitants are away, they are aware of any visitors, but at the same time, they are not woken up while they are sleeping, every time someone in nearby. If someone is indeed trying to enter while they are sleeping, a separate burglar alarm in the house goes off to wake them up. Since the `Away rule` in listing 2 allows `STATE_SLEEPING` to be set to `ON` when `STATE_AWAY` is `ON` (by any of the four ways described in section 2), it will wrongly deactivate the visitor notification system by triggering the rule in Listing 4, when in reality, the inhabitants are away and not sleeping inside the house. This will prevent the home inhabitants from knowing if there is a visitor near the house while they are away.

Threat model. Our approach generates the trigger conditions from the action block of the rules, so it assumes that the action block is written correctly. We trust that the devices in the house are not compromised and once they receive a command, they execute it. We also trust the rule engine and the home automation OS for which the rules are written.

4. openHAB background

openHAB [6] is an open-source vendor-agnostic software framework for integrating smart devices in a home. It supports 135 technologies, including more than 50 devices, cloud services like Twitter, DropBox, and Google Calendar, and multiple communication protocols [9]. It can be run on Linux, Windows, and MacOS X as well as on embedded platforms such as Raspberry Pi. It has apps for Android and iOS. The Android app has about 50,000 downloads in the Google Play store with a rating of 4.4/5.

Figure 1 shows the main components of the openHAB framework. The item repository stores information about the devices as explained in section 4.1. It is connected to the UI and the rule base. As shown in the figure, there are two types of events that are communicated by the event bus. 1) *Commands* cause actions or change device states. Examples are `ON`, `OFF`, `UP`, and `DOWN`. 2) *Updates* inform the UI component or the rule engine or other devices about changes in the state of a device. Bindings are libraries that connect physical devices to the openHAB framework.

End-users can write configuration files to customize the home. The main configuration files include a *rule* file containing the automation rules and an *item* file containing device information.

```
<item> ::= <type> <name> ["label"] [<icon>] [(group1,
group2, ..., groupn)] [binding]
```

```
Switch DemoSwitch "Switch"
Contact Window_Bedroom "Bedroom" (Bedroom,
Windows)
```

Listing 5: Syntax of item definitions and two examples.

```
Group All
Group GF(All)
Group GF_Living(GF)
Group Lights(All)
Dimmer Light_GF_Living(GF_Living,Lights)
```

```
Lights.members.forEach(light|postUpdate(light,ON))
```

Listing 6: Code snippet from an items file (above) showing grouping of items (GF stands for Ground Floor) and use of the group in a rule (below).

For our research, we required access to the item files and the rule files.

4.1 Items

An item [3] is a representation of a device installed in the house. Every item has a type that indicates the values it can hold and the commands it can receive. Items can also be grouped together—all lights in the living room can be in one group so that they can be controlled together. An item may belong to multiple groups. Listing 5 shows the syntax of an item definition (contents in [ ] are optional) and two entries in a sample item file.

4.2 Rules

Listings 1, 2, 3, 4, 7, and 8 show examples of rules. Rules are fired by a rule engine. For our research, we assume that the rule engine fires one rule at a time. A rule has two parts: a *trigger block* and an *action block*.

The *trigger block* can have one or more triggers. There are three types of triggers: event-based, temporal, and system. Event-based triggers fire when the state of a device changes. For example: `Item State_Away changed`. Temporal triggers fire at specific times and system triggers fire at system startup and shutdown. Event-based triggers are the focus of this research. TrigGen automatically generates only event-based triggers, and in the rest of the paper, the term trigger refers to an event-based trigger.

The *action block* or *script* describes what a rule should do when its triggers are fired. It is written in a Java-like language called Xbase.

4.3 Actions

Actions are predefined methods that can be used in the *action block* of a rule. The openHAB runtime provides a core set of actions [1], and manufacturers can implement more for specific devices. Two important core actions for the purpose of our analysis are the event bus actions: `sendCommand(String itemName, String commandName)` and `postUpdate(String itemName, String stateValue)`.

5. Errors due to too few triggers

Problem Definition. An insufficient number of triggers in a rule leads to fewer firings, which can cause unexpected behavior or security vulnerabilities.

Goal. Our goal is to prevent end-users from making errors due to too few event-based triggers.

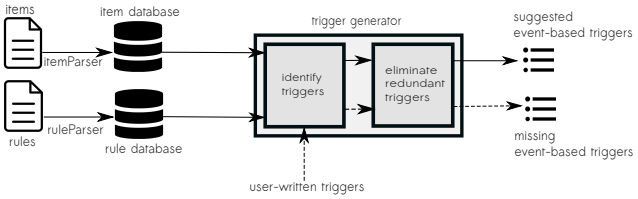

Figure 2: Architecture of the TrigGen tool.

Approach. We developed a static analysis tool—TrigGen—that automatically generates a set of necessary and sufficient event-based trigger conditions from the actions written by the end-users. Figure 2 shows the design of our tool. It takes as input a *rule* file and an *item* file. The technique is explained below.

1. By statically analysing the AST of the script, S, TrigGen identifies all the items that appear in the action block of a rule, R. This is an exhaustive list of all potential event-based triggers. Let this list be T.
2. Naively adding all $t \in T$ as triggers of R would unnecessarily fire R too many times and make it complicated. Hence, TrigGen applies an elimination technique to get rid of triggers that are not required. The following definitions are used in the elimination algorithm.

   Definition 1. An item is *live* [11] in R if its value may be read before it is written to in the rule script, S.

   Definition 2. An event-based trigger is *redundant* if inclusion of the trigger in a rule *never* changes the state of any item or the value of any program variable involved in S when the rule is fired solely due to it. Otherwise, the trigger is *non-redundant*.

   An item d which is not *live* in R is not included in T because it is redundant.

   Theorem. If d is not live in R, then it is a redundant trigger.

   Proof. Let us assume that d is not redundant. That means it must change the value of some item or program variable in the rule in some firing. Let this item or variable be a. Since d is not live, its value is never read before it is written to. The following cases are possible:

   - d is a constant, i.e. it is never assigned to in S. In this case, d as a trigger does not change the value of a because d's value is always the same.
   - d is a function of other items, $i_1, i_2, ..., i_n$ and program variables $v_1, v_2, ..., v_m$ in R, i.e. d's value depends on the values of $i_1, i_2, ..., i_n$ and $v_1, v_2, ..., v_m$. Let $I_l \subseteq \{i_1, i_2, ..., i_n\}$ be a set which contains all the live items in $\{i_1, i_2, ..., i_n\}$. In this case, d as a trigger will still not affect the value of a because d's own value depends on the values of items $i_1, i_2, ..., i_n$ — a will change only if the value of some $i \in I_l$ changes. Hence, only adding all $i \in I_l$ as triggers would be sufficient.

   This shows that d never causes the value of a to change. Therefore, our assumption that d is not redundant is wrong. Hence, d must be redundant. ■

5.1 Enumerating groups

Items can be grouped so that they can be controlled together if needed—the rule scripts can perform operations on all the member items of a group. Listing 6 shows examples of groups specified in an item file and a code snippet from a rule file showing how they are used in a rule. A group can have multiple items and one item can belong to one or more groups. Groups can also be contained in other groups. For such rules, TrigGen generates triggers by enumerating

```
rule "Air garage"
when
  Item Temp_G changed or
  Item Humidity_G changed
then
  if (Temp_G.state != null &&
      Humidity_G.state != null) {
      var String msg = Temp_G.state.toString()
            + Humidity_G.state.toString()
      if (Air_G_Message.state != msg) {
        postUpdate(Air_G_Message,msg)
      }
  }
end
```

Listing 7: Rule for updating the status message for the air quality in the garage.

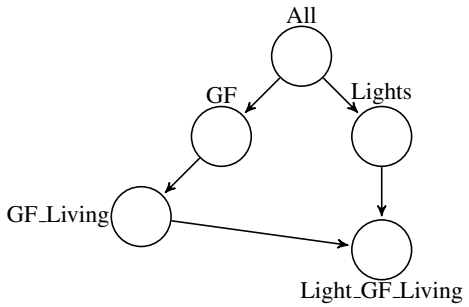

Figure 3: Graph representing the item grouping shown in listing 6. The arrows point from a parent group to a child group or a member item.

all members of a group by converting it to a graph problem. It performs a *depth first search* to determine all the groups to which an item belongs. Figure 3 shows the graph representation of the groups in listing 6.

5.2 Conflict resolution

We define two or more rules to be conflicting if they modify the state of the same item. In that case, TrigGen warns the user about a potential conflict. As an example, consider the rule written by an end-user in listing 7. Before `Air_G_Message` is updated (in `postUpdate`), its current value is checked against `msg`. This rule could conflict with another rule that writes to `Air_G_Message`. TrigGen detects such conflicts and warns the end-user about the possibility of a conflict. It is then up to the user to determine the severity of the warning and whether the rules need to be corrected.

6. Implementation

We implemented TrigGen in ∼3500 LOC of Java. It can be used directly on a rule file without any pre-processing or annotations from the end-user. Figure 2 shows the design of TrigGen. Both rules and items have their own domain specific languages (DSLs)—we used the respective grammars provided by openHAB and the Xtext parser generators [10] to generated parsers for them.

TrigGen extracts item information such as item names, types and groups they belong to, and stores them in an item database. It uses this information to identify items in the rules—it visits the nodes in the AST of the rule's script to extract the names of the items appearing in assignments, as actual method arguments, conditions in if-statements, closures and all other language constructs supported by the rule DSL. It stores this information in a rule database.

The next step is trigger generation where TrigGen first adds all the items naively to a list of potential triggers. It analyzes this list to eliminate the redundant triggers as explained in section 5 to generate a list of necessary and sufficient triggers.

Action	Execution time
generate all triggers	6.9 s
detect missing triggers	7.3 s

Table 1: Execution time (in seconds) summary of TrigGen for 96 rules for 1) generating all event-based triggers, and 2) identifying missing event-based triggers by comparing with user-written triggers.

TrigGen then compares the list of generated triggers to the triggers written by the end-users to detect the missing/extraneous ones. As explained in section 5.2, TrigGen also identifies rules which might conflict with each other due to modifying the state of the same device.

7. Experimental Evaluation

We evaluated TrigGen on 96 end-user written rules. We obtained the rules from links provided on the openHAB wiki [2, 7]. We ran TrigGen on a machine running 64-bit Ubuntu 14.04 LTS with 2.6 GHz quad core processor. Table 1 shows the time taken for TrigGen to complete executing. For 91 out of the 96 rules (95%), TrigGen suggested a list of all required event-based triggers. Its output differed from the user-written event-based triggers for 77 rules (80%). By manual inspection, we found that the event-based triggers that TrigGen detected as *missing* were true positives. For 18 rules (19%), TrigGen generated warnings about potential conflicts. 11 out of these (61%) were correct warnings as per our description of conflict in section 5.2. The other 7 (39%) were false positives.

Figure 5 shows the output of TrigGen for generating a necessary and sufficient set of triggers. For the majority of the rules, TrigGen suggested 2 triggers and detected 1 missing trigger. The large number of triggers which were determined for some rules (as shown in figure 5) was due to the grouping of items. TrigGen expanded the groups as explained in section 5.1 to enumerate all the items and then applied the technique described in section 5 for identifying the non-redundant triggers.

7.1 Precision and recall

We define the *goal set* of triggers for a particular rule as the set of all *non-redundant* triggers. Precision and recall for TrigGen's trigger suggesting feature (p_s and r_s respectively) are then defined as:

$$p_s = \frac{|\{\textit{non-redundant triggers}\} \cap \{\textit{generated triggers}\}|}{|\{\textit{generated triggers}\}|}$$

$$r_s = \frac{|\{\textit{non-redundant triggers}\} \cap \{\textit{generated triggers}\}|}{|\{\textit{non-redundant triggers}\}|}$$

where $\{\textit{generated triggers}\}$ is the set of triggers generated by TrigGen. Precision and recall for user written triggers (p_u and r_u) are defined similarly with *generated triggers* replaced by *user-written triggers*. Figure 4 shows the precision and recall values for the user written triggers. For TrigGen's missing trigger detection feature, we define the *goal set* of triggers as:

$$\{\textit{missing non-redundant triggers}\} = \{\textit{non-redundant triggers}\} - \{\textit{user-written triggers}\}$$

The definitions of p_m and r_m respectively are:

$$p_m = \frac{|\{\textit{missing non-redundant triggers}\} \cap \{\textit{missing triggers}\}|}{|\{\textit{missing triggers}\}|}$$

$$r_m = \frac{|\{\textit{missing non-redundant triggers}\} \cap \{\textit{missing triggers}\}|}{|\{\textit{missing non-redundant triggers}\}|}$$

p_s	r_s	frequency
1	1	91
not defined	0	5
p_m	r_m	frequency
1	1	77
not defined	0	19

Table 2: Precision and recall of the output of TrigGen for 1) suggesting all event-based triggers, represented by p_s and r_s respectively, and 2) detecting missing event-based triggers, represented by p_m and r_m respectively. Precision = `not defined` and recall = `0` indicate that the output set was empty.

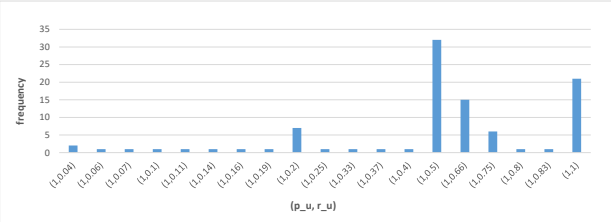

Figure 4: Precision and recall of user written event-based triggers.

```
rule "Select Radio Station"
when
  Item Radio_Station received command
then
  if (receivedCommand == 0) {
    playStream("http://mp3-live.swr3.de/swr3_m.m3u")
  }
end
```

Listing 8: Example of a rule for which TrigGen failed.

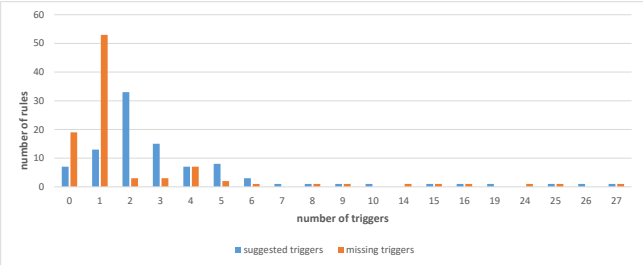

Figure 5: Summary of the output of TrigGen for suggesting a set of necessary and sufficient triggers (in blue) and detecting missing triggers (in orange). The x-axis shows the number of suggested triggers and the y-axis shows the number of rules.

where $\{missing\ triggers\}$ is the set of triggers identified as missing by TrigGen. Table 2 shows the precision and recall values of the output of TrigGen.

7.2 Limitations

TrigGen failed to suggest the event-based triggers for 5 rules which did not have references to the trigger items in the rule script. This is because TrigGen uses the AST of the rule script to extract items that can be potential triggers; if there is no reference to an item in the script, TrigGen cannot detect that trigger. Listing 8 shows an example of one of these rules.

8. Related work

Security goals and potential vulnerabilities in smart homes have been investigated in several papers [12, 14, 19, 23]. Denning et al. [12] identified assets within a home that are susceptible to security attacks and suggested security goals for smart homes. Ur et al. [23] conducted a case study of three specific smart devices to analyze the state of access control for home automation. Mennicken et al. [19] studied the importance of human-home-collaboration in making smart homes more accessible.

Recently, Fernandes et al. [15] did a security analysis of several apps based on Samsung SmartThings and discovered that many of them unnecessarily granted full access to the devices in the house. While they aimed at identifying security flaws in the SmartThings framework itself, our aim is to assist end-users in writing correct automation rules and we do so by automatically generating the trigger conditions. Further, instead of analysing apps for home automation, we analyzed end-user written automation rules.

Some work has been done on detecting conflicts in trigger-action programs [17, 18, 20, 22]. TrigGen's main capability is to automatically generate the correct triggers for a rule although it can also identify rules which may have potential conflicts. None of the previous tools can automatically determine trigger conditions.

The usefulness of TAP for customizing smart homes has been studied by Ur et al. [24] and Dey et al. [13]. Their findings motivated us to focus on end-user written rules for home automation—they showed by conducting user studies that about 80% of the automation requirements of the users could be represented by trigger action programs [13] and even non-programmers could easily learn TAP [24].

Huang et al. [16] conducted user studies on TAP to identify inconsistencies in interpreting the behavior of trigger action programs and errors in writing them and found that often the interpretation of a rule by a user is different from the semantics of the rule.

Some recent work [16, 24] observed that the IFTTT framework [5] which is also used for writing automation rules only allows a single trigger and is not sufficient for expressing complex automation rules for smart homes. This motivated us to evaluate our tool on the openHAB framework which has a more expressive rule language allowing multiple triggers.

9. Conclusions

End-users have a major role in home automation—they decide the automation rules for the devices. Confirming previous work, our research shows that these end-user written rules are often error-prone. We observed that a common error made by end-users while writing the rules is having an insufficient number of triggers, leading to fewer firings of the rules than necessary. To prevent this problem, we developed TrigGen, which automatically generates a set of necessary and sufficient triggers based on the actions in a rule. TrigGen correctly identified missing triggers in 77 out of the 96 real home automation rules that we analyzed. TrigGen reduces the burden on end-users by assisting them in writing the rules correctly, and can reduce the incidence of unexpected behaviors and security vulnerabilities.

Acknowledgments

John Toman helped with the Xtext language tools. Talia Ringer and the anonymous reviewers gave helpful feedback on the manuscript.

References

[1] Actions available in scripts and rules. `https://github.com/openhab/openhab/wiki/Actions`. Accessed: May 2016.

[2] Configs, Tools & Icons. `http://www.intranet-of-things.com/software/downloads`. Accessed: August 2016.

[3] Explanation of items. https://github.com/openhab/openhab/wiki/Explanation-of-items. Accessed: August 2016.

[4] Homekit Securely control your home. http://www.apple.com/ios/homekit/?cid=wwa-us-kwg-features. Accessed: May 2016.

[5] IFTTT. https://ifttt.com/recipes. Accessed: June 2016.

[6] openHAB empowering the smart home. http://www.openhab.org/. Accessed: August 2016.

[7] openhab home. https://github.com/openhab/openhab/wiki. Accessed: August 2016.

[8] Smart Home. Intelligent Living. https://www.smartthings.com/. Accessed: May 2016.

[9] Suported technologies. http://www.openhab.org/features/supported-technologies.html. Accessed: May 2016.

[10] Xtext. Language Engineering For Everyone. https://eclipse.org/Xtext/. Accessed: August 2016.

[11] F. E. Allen and J. Cocke. A program data flow analysis procedure. *Commun. ACM*, 19(3):137–, Mar. 1976.

[12] T. Denning, T. Kohno, and H. M. Levy. Computer security and the modern home. *Commun. ACM*, 56(1):94–103, Jan. 2013.

[13] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In *Proceedings of the 4th International Conference on Pervasive Computing*, PERVASIVE'06, pages 254–271, Berlin, Heidelberg, 2006. Springer-Verlag.

[14] N. Dhanjani. *Abusing the Internet of Things: Blackouts, Freakouts, and Stakeouts*. O'Reilly Media, Incorporated, 2015.

[15] E. Fernandes, J. Jung, and A. Prakash. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, May 2016.

[16] J. Huang and M. Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 215–225, New York, NY, USA, 2015. ACM.

[17] H. Luo, R. Wang, and X. Li. A rule verification and resolution framework in smart building system. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 438–439, Dec 2013.

[18] C. Maternaghan and K. J. Turner. Policy conflicts in home automation. *Comput. Netw.*, 57(12):2429–2441, Aug. 2013.

[19] S. Mennicken, J. Vermeulen, and E. M. Huang. From today's augmented houses to tomorrow's smart homes: New directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 105–115, New York, NY, USA, 2014. ACM.

[20] M. Nakamura, H. Igaki, and K. ichi Matsumoto. Feature interactions in integrated services of networked home appliance. In *Proc. of Int'l. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05*, pages 236–251. IOS Press, 2005.

[21] M. W. Newman. Now we're cooking: Recipes for end-user service composition in the digital home, 2006. Position Paper–CHI 2006 Workshop IT@Home.

[22] Y. L. Sun, X. Wang, H. Luo, and X. Li. Conflict detection scheme based on formal rule model for smart building systems. *IEEE Trans. Human-Machine Systems*, 45(2):215–227, 2015.

[23] B. Ur, J. Jung, and S. Schechter. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)*. HUPS 2014, July 2013.

[24] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 803–812, New York, NY, USA, 2014. ACM.

[25] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3227–3231, New York, NY, USA, 2016. ACM.