Empowering End Users in Debugging Trigger-Action Rules

Fulvio Corno
Politecnico di Torino
Torino, Italy
fulvio.corno@polito.it

Luigi De Russis
Politecnico di Torino
Torino, Italy
luigi.derussis@polito.it

Alberto Monge Roffarello
Politecnico di Torino
Torino, Italy
alberto.monge@polito.it

ABSTRACT

End users can program trigger-action rules to personalize the joint behavior of their smart devices and online services. Trigger-action programming is, however, a complex task for non-programmers and errors made during the composition of rules may lead to unpredictable behaviors and security issues, e.g., a lamp that is continuously flashing or a door that is unexpectedly unlocked. In this paper, we introduce *EUDebug*, a system that enables end users to debug trigger-action rules. With EUDebug, users compose rules in a web-based application like IFTTT. EUDebug highlights possible problems that the set of all defined rules may generate and allows their step-by-step simulation. Under the hood, a hybrid Semantic Colored Petri Net (SCPN) models, checks, and simulates trigger-action rules and their interactions. An exploratory study on 15 end users shows that EUDebug helps identifying and understanding problems in trigger-action rules, which are not easily discoverable in existing platforms.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; *User studies*; • **Software and its engineering** → *Visual languages*.

KEYWORDS

Internet of Things, Trigger-Action Programming, Petri Nets, Semantic Web, End-User Debugging

ACM Reference Format:
Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. Empowering End Users in Debugging Trigger-Action Rules. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3290605.3300618

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05…$15.00
https://doi.org/10.1145/3290605.3300618

1 INTRODUCTION

The Internet of Things (IoT) is a well-established paradigm that already helps society with applications for the individual and for the community [13]. People, in fact, interact daily with a growing number of Internet-enabled devices [19] in many different contexts, ranging from their smart homes to smart cities. Furthermore, the IoT ecosystem can be defined from a wide perspective [1, 21], by including not only physical devices, but also online services such as messaging platforms and social networks. The result is a complex network of smart objects, either physical or virtual, that are able to interact and communicate with each other, with humans, and with the environment. In this complex domain, professional programmers cannot foresee all the possible situations end users may encounter when interacting with their IoT ecosystem, and existing software development cycles are still too slow to respond to user needs [21].

To solve these issues, End-User Development (EUD) can put the personalization of IoT ecosystems in the hands of end users, i.e., the subjects who are most familiar with the actual needs to be met. Several works in the literature demonstrate the effective applicability of EUD techniques for the creation of personalized applications in various domains [15, 18, 21, 29, 36, 37], including the IoT. Nowadays, end users who want to personalize their IoT ecosystem can take advantage of visual programming platforms such as IFTTT[1] or Zapier[2]. In such platforms, users can program the joint behavior of their devices and online services by defining trigger-action rules such as *"if the Nest camera in the kitchen detects a movement, then send me a Telegram message."*

Despite apparent simplicity, trigger-action programming is often a complex task for non-programmers [24] and one of the most important and urgent challenges is the need to avoid possible *conflicts* [10] and to assess the *correctness* [17] of trigger-action rules. Errors in this context, in fact, can lead to unpredictable and dangerous behaviors [7]: while posting a content on a social network twice could be considered

[1] https://ifttt.com, last visited on September 18, 2018
[2] https://zapier.com, last visited on September 18, 2018

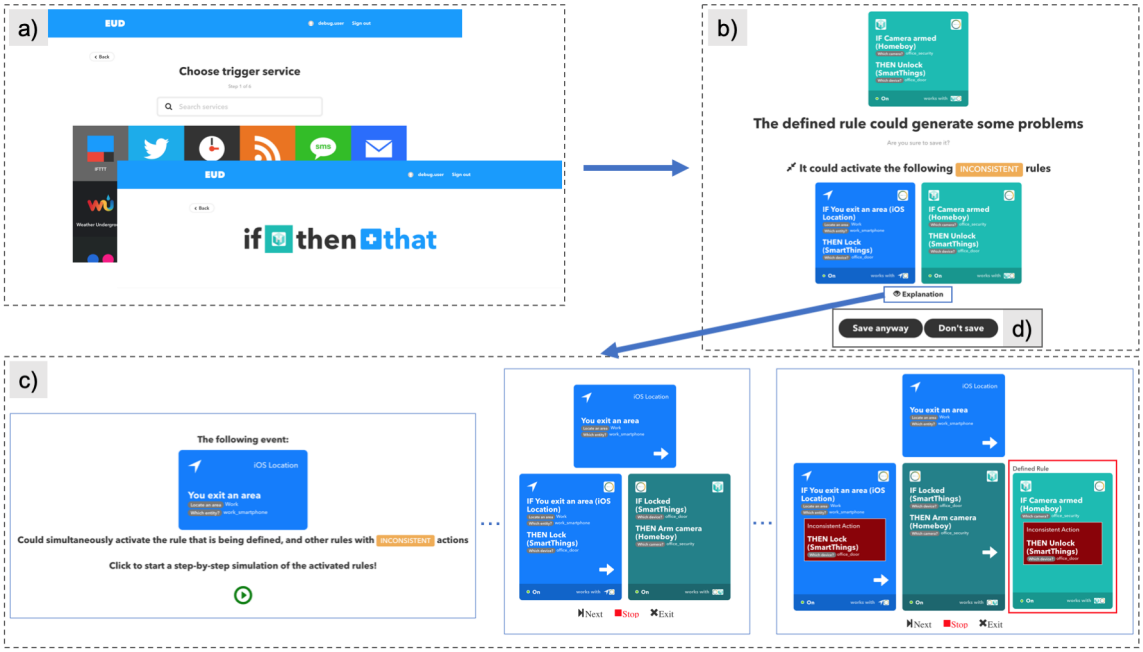

Figure 1: EUDebug is a system for debugging trigger-action rules that allows the user to: a) compose a new rule; b) view any problems that the rule may generate; c) further investigate each problem with a step-by-step simulation; and d) edit the rule to fix the problem or save it anyway.

a trivial issue, wrong rules could unexpectedly unlock the main door of a house, thus generating a security threat.

In this paper, we introduce *EUDebug*, a system that enables end users to debug their trigger-action rules. The goal of EUDebug is to properly warn users when they are defining any troublesome or potentially dangerous behavior, according to two strategies: (i) by assisting them in identifying rule conflicts, and (ii) by helping them foresee the run-time behavior of their rules through step-by-step simulation. Figure 1 shows a sample usage scenario:

a) The user composes a new trigger-action rule in a web-based application modeled after IFTTT (e.g., "*if* the security camera in the office is armed, *then* unlock the door"). As the user is composing the rule, EUDebug employs a novel hybrid Semantic Colored Petri Net to model, check, and simulate the rule with respect to previously defined trigger-action rules.

b) When the rule composition is completed, EUDebug highlights possible problems that the rule may generate, by providing a short explanation to the user.

c) If needed, the user can further inspect and understand the problems by asking EUDebug to perform and show a step-by-step simulation of the problematic rules.

d) Finally, the user can edit the composed rule or decide to ignore the highlighted problems, thus saving the rule in the current format.

Thanks to the Semantic Colored Petri Net, EUDebug is able to detect problems in rules that involve different types of devices but with similar functions. The semantic representation, in fact, allows the net to reason about the final goal of each object (e.g., illumination, messaging, …) to detect and show problems.

To our knowledge, EUDebug is the first attempt to provide debugging features to end users during the composition of trigger-action rules for their IoT ecosystem. The user interface of our EUDebug prototype is modeled after IFTTT and shares with it the same metaphors and the same expressiveness. This choice was made deliberately: the form-filling procedure adopted by IFTTT helps users avoid syntactical errors during the composition of a rule and, according to the results of a comparative study held by Caivano et al. [10] among different EUD platforms, IFTTT is the most accurate in terms of correct rules created, the most effective, and the least difficult to use.

To investigate whether EUDebug can help end users identify and understand problems in trigger-action rules, we ran an exploratory study with 15 university students coming from various backgrounds, but excluding those who had previous experience in computer science and programming, and with IFTTT. Each of them used EUDebug to compose 12 different trigger-action rules that generated 5 problems. They found EUDebug useful to identify and understand most of the

problems they encountered while composing trigger-action rules, which are not easily discoverable within existing platforms. In addition, they found the step-by-step simulation to be fundamental for understanding some types of problems.

Finally, this paper makes the following contributions:

- We introduce EUDebug, a system to enable end users to debug trigger-action rules in their IoT ecosystem, and we characterize the problems to be detected for trigger-action rules.
- We show two complementary strategies exploited by EUDebug towards *end-user debugging* in the IoT: to assists users in identifying rule conflicts, and to help them simulate and foresee the run-time behavior of their rules.
- We present a hybrid Semantic Colored Petri Net (SCPN) formalism, able to model, check, and simulate trigger-action rules and their interactions.
- In a study with 15 participants, we show that EUDebug helps identify and understand most of the problems that may arise in a set of trigger-action rules.

2 RELATED WORK

EUDebug lies at the intersection of research in three related areas: *(i)* trigger-action programming in the IoT, *(ii)* end-user debugging, and *(iii)* rule modeling and analysis.

Trigger-Action Programming in the IoT

With the technological advances we are confronting today, people are increasingly moving from passive consumers to active producers of information, data, and software [32], and End-User Development (EUD) approaches and methodologies have been extensively explored in different contexts, e.g., mobile environments [33], smart homes [6, 36], and web mashups [16, 35]. The explosion of the IoT further increased the need of allowing end users to customize the behavior of their smart devices and online services. Not surprisingly, in the last 10 years, several commercial platforms for end-user personalization, such as IFTTT and Zapier, were born. Such platforms typically adopt one of the most common EUD paradigm, i.e., Trigger-Action Programming. By defining trigger-action rules, users can connect a pair of smart devices or online services in such a way that, when an event (the *trigger*) is detected on one of them, an *action* is automatically executed on the second.

Despite the trigger-action programming paradigm can express most of the behaviors desired by potential users [5, 36], and is adopted by the most common EUD platforms [17], the definition of trigger-action rules can be difficult for non-programmers. Multiple studies investigated different aspects of contemporary platforms like IFTTT, ranging from empirical characterization of the performance and usage of IFTTT [31] to human factor related to their adoption in the smart home [36]. Large-scale analysis of publicly shared rules on IFTTT [37], and changes to the underlying models are proposed as well [14, 17].

These works highlight different issues and challenges: contemporary EUD platforms often expose too much functionality [24], and adopt technology-dependent representation models [14], thus forcing users to have a deep knowledge of all the involved devices and online services. Moreover, conflicts and ambiguities among rules emerged as possible challenges [36]. As a result, users frequently misinterpret the behavior of trigger-action rules [7], often deviating from their actual semantics, and are prone to introduce errors [23]. Therefore, tools for trigger-action programming in the IoT should be better tailored for end users who are not accustomed to programming. In this paper, we take a step towards such a direction with EUDebug. Systems to assist users in foreseeing the behavior of their trigger-action rules, while identifying conflicts and problematic rules, could indeed facilitate the adoption of EUD platforms in the real world [17].

End-User Debugging

Today's end-user programmers include anyone who creates artifacts that instruct computers how to perform an upcoming computation, without being necessarily interested in learning how to program. Along with the ability to create programs comes the need to debug them, and work on end-user debugging is only beginning to become established [22].

To our knowledge, EUDebug is the first attempt to provide debugging features to end users who compose trigger-action rules for their IoT ecosystem. Previous works on end-user debugging are not related to trigger-action programming nor the IoT, but mainly focus on mashup programming [12] and, especially, spreadsheets [8, 22, 27]. In using spreadsheets, in particular, users are likely to make a large number of mistakes [9, 34], and the need of supporting end-users' debugging efforts in such tools has gained interest in the last years [22].

By focusing on novice developers, instead, Ko and Myers [28] propose an interrogative debugging interface for the Alice programming environment. The interrogative debugging is a debugging paradigm by which novices can ask *why* and *why not* questions about their program's run-time failures. EUDebug is inspired by this paradigm, as it assists end users in understanding *why* their trigger-action rules may be problematic through the step-by-step simulation of the run-time behavior of the rules.

Rule Modeling and Analysis

Many prior works face the problem of formally or semi-formally verifying event-based rules with different approaches, especially in the area of databases [20, 30], expert systems [39],

and smart environments [3, 38]. Rules, indeed, have the ability to interact with each other, and even a small set of dependencies between them makes it hard (and often undecidable) the problem of predicting their overall behavior [4]. Li et al. [30], for instance, propose a Conditional Colored Petri Net (CCPN) formalism to model and simulate Event-Condition-Action (ECA) rules for active databases. Petri nets are used by Yang et al. [39] to verify rules in expert systems, and by Jin et al. [26] to dynamically verify ECA properties such as termination and confluence. In the field of smart environments, Vannucchi et al. [38] adopt formal verification methods for ECA rules, while Augusto and Hornos [3] propose a methodological guide to use the Spin model checker to inform the development of more reliable intelligent environments.

The majority of the works described above aim at checking the consistency of a set of fixed and *already* defined rules, not in real time, and employ predefined use cases to validate the algorithms. The goal of EUDebug is different: instead of performing such an "off-line" verification of rules, EUDebug aims at assisting end users *during* the definition of their own trigger-action rules. For this purpose, we empower the EUDebug interface with a novel Petri net formalism, similar to CCPN but enhanced with new elements and with semantic information.

3 CHARACTERIZING PROBLEMS IN TA RULES

To better understand which problems EUDebug should be able to detect and show to end users, we reviewed previous works on rule analysis and trigger-action programming [31, 38]. From those works, we generalize three classes of problems to be considered in trigger-action rules for the IoT: *loops*, *inconsistencies*, and *redundancies*.

Loops occur when a set of trigger-action rules are continuously activated without reaching a stable state [11, 31]. An example of a loop is:

- *if* I post a photo on Facebook, *then* save the photo on my iOS library;
- *if* I add a new photo on my iOS library, *then* post the photo on Instagram;
- *if* I post a photo on Instagram, *then* post the photo on Facebook.

Inconsistencies occur when rules that are activated at (nearly) the same time[3] try to execute contradictory actions. It is to be noticed that the typical definition is different: "inconsistencies occur when the execution order of rules may render different final states in the system" [11]. We generalized this concept to consider the entire IoT ecosystem, i.e., not only physical devices but also online services. The order of actions performed on online services, e.g., posting a content on Facebook or sending a WhatsApp message, indeed, is not really important, because they do not change the internal state of a device and they do not leave the system in a unpredictable or dangerous state. For this reason, we analyze the *meaning* of the executed actions rather than their execution order. An example of a set of rules that produces an inconsistency is:

- *if* my Android GPS detects that I exit the home area, *then* lock the SmartThings entrance door;
- *if* my Android GPS detects that I exit the home area, *then* set the Nest thermostat to Away mode;
- *if* the SmartThings entrance door is locked, *then* set the Nest thermostat to Manual mode.

Here, the three rules are executed at the same time because the first two rules share the same trigger, while the first rule implicitly activates the third rule. They produce two inconsistent actions, since they set 2 contradictory modes on the Nest thermostat, i.e., Away and Manual.

Redundancies, finally, occur when two or more rules that are activated (nearly) at the same time have replicated functionality [11]. An example of a set of rules that produce a redundancy is:

- *if* I play a new song on my Amazon Alexa, *then* post a tweet on Twitter;
- *if* I play a new song on my Amazon Alexa, *then* save the track on Spotify;
- *if* I save a track on Spotify, *then* post a tweet on Twitter.

Here, the three rules are executed at the same time because the first two rules share the same trigger, while the second rule implicitly activates the third rule. They produce two redundant actions, i.e., the first and the third rule post the same content on Twitter.

4 THE EUDEBUG SYSTEM

The goal of EUDebug is to give end users an effective way to debug trigger-action rules in their IoT ecosystem, *a)* by assisting them in identifying rule conflicts (i.e., loops, inconsistencies, and redundancies) and *b)* by allowing a further investigation of the run-time behavior of their rules in simulation.

During the rule composition phase, EUDebug automatically detects potential problems with no user intervention. At the end of the composition process, EUDebug shows any conflicts that the composed rule may generate by interacting with previously defined trigger-action rules, and allows users to further investigate *why* the problem happens. In this way, EUDebug facilitates end user debugging of trigger-action rules, rather than waiting for the problem to arise in the real world. It consists of two main components: 1) a Semantic Colored Petri Net (SCPN), which runs on a dedicated server, to

[3] e.g., when rules share the same trigger or when some rules trigger other rules

model, check, and simulate trigger-action rules, and 2) a web-based user interface for composing trigger-action rules, for showing any detected problems, and simulating step-by-step the problematic rules.

Semantic Colored Petri Net

To model and check trigger-action rules, we define a novel Semantic Colored Petri Net (SCPN) formalism. Petri nets are bipartite directed graphs, in which directed arcs connect places and transitions. Places may hold tokens, which are used to study the dynamic behavior of the net. They can naturally describe the rules as well as their non-deterministic concurrent environment [26]. We chose such an approach to allow users to simulate step-by-step the execution of their rules: by firing a transition at a time, tokens move in the net by giving the idea of a possible execution flow. As a member of Petri nets family, Colored Petri Net (CPN) [25] combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. In particular, SCPN is a Colored Petri Net similar to the Conditional Colored Petri Net (CCPN) formalism [30] proposed to model ECA rules in active databases. Differently from such a formalism, we do not consider conditions and we use a semantic model both to generate and analyze the net. Furthermore, as explained in the following, each token assumes different semantic "colors" by moving in the net. Such semantic information allows the inference of more information from the simulation of the net, i.e., to discriminate between problematic and safe rules.

Adding Semantics to Trigger-Action Rules. The novel characteristic of the SCPN formalism is the usage of Semantic Web technologies in conjunction with a Colored Petri Net. Adding semantic to IoT objects, triggers, and actions is a common approach [2]. In our case, we exploited EUPont [14] as the semantic model. EUPont[4] is an ontological high-level representation of trigger-action programming that describes smart devices and online services on the basis of their categories and capabilities, i.e., their offered services. Here, the semantic information is used to build and analyze a SCPN. Trigger-action rules are first translated into their respective semantic representation.

In detail, for each trigger or action, the ontology provides information about the device or online service by which they are offered, and any relationship with other triggers or actions, e.g., the fact that the action *turn on the Philips Hue lamp* implicitly activates the trigger *the Philips Hue lamp has been turned on*. Furthermore, triggers and actions are classified through a tree of classes that represents the final behavior they monitor, in case of triggers, or produce, in case of actions. Triggers or actions that are classified under the same EUPont classes, in particular, are similar in terms of final functionality, while triggers or actions that do not share any EUPont class are functionally contradictory. For example, the two actions *"set the Nest thermostat to Home mode"* and *"set 25 Celsius degree on the Nest thermostat"* share the same final functionality, because they are both classified under the same EUPont class, i.e., *IncreaseTemperatureAction*. Compared to these actions, the action *"set the Nest thermostat to Away mode"* is contradictory in terms of functionality, because it is classified under a different EUPont class, i.e., *DecreaseTemperatureAction*. This information is used by a SCPN to "color" the places of the Petri net and to detect inconsistencies and redundancies among rules.

Formalism. To exemplify and better explain the SCPN formalism, Figure 2 shows the net built starting from the seven rules in Table 1, with R7 being the rule in composition.

[4]http://elite.polito.it/ontologies/eupont.owl, last visited on September 18, 2018

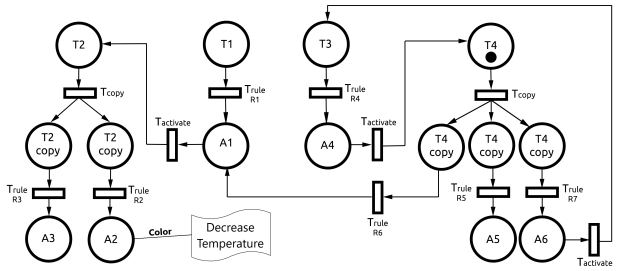

Figure 2: The SCPN generated by analyzing the rules of Table 1.

#	Trigger (if…)	Action (then…)
R1	my Android GPS detects that I exit the home area (T1)	lock the SmartThings entrance door (A1)
R2	the SmartThings entrance door is locked (T2)	set the Nest thermostat to Away mode (A2)
R3	the SmartThings entrance door is locked (T2)	turn off the Philips Hue lamp in the kitchen (A3)
R4	the SmartThings entrance door is unlocked (T3)	arm the Homeboy security camera (A4)
R5	the Homeboy security camera is armed (T4)	send me a Telegram message (A5)
R6	the Homeboy security camera is armed (T4)	lock the SmartThings entrance door (A1)
R7	the Homeboy security camera is armed (T4)	unlock the SmartThings entrance door (A6)

Table 1: The rules that generate the SCPN of Figure 2.

Triggers and actions of a given object, e.g., SmartThings, are modeled as places, i.e., *Trigger Place (TP)* and *Action Place (AP)*. For instance, in Figure 2, a *TP* is $T1$ while an *AP* is $A1$. When a trigger is in common between more than one rule (as in R2 and R3), the associated places are duplicated (e.g., $T2copy$ in Figure) and connected through a *Copy Transition*. When a token is in the original place, the *Copy Transition* simply replicates the token in each copied place. *AP*s follow a slightly different process than *TP*s: an *AP* can be reused for rules that share the same action, e.g., $A1$ models the action offered by both R1 and R6 (Figure 2). *TP*s and *AP*s can be connected each other through:

Rule Transition, a connection between a trigger and an action of the same rule. *Rule Transitions* model the rule defined by the user. They remove a token from a *TP* to generate a new token in an *AP*. In Figure 2, *Rule Transitions* are indicated with $T_{rule}Rx$ (e.g., $T_{rule}R2$ for R2).

Trigger Transition, a connection used when an action of a rule triggers the event of another rule. *Trigger Transitions* are extracted from the semantic information contained in EUPont. They remove a token from an *AP* to generate a new token in a *TP*. In Figure 2, *Trigger Transitions* are indicated with $T_{activate}$.

Finally, as exemplified for A2 in Figure 2, all the places are characterized by a semantic color that represents the semantic information associated with the corresponding trigger or action. When a token cross a place, it assumes the place color.

Rule Analysis with SCPN. To detect loops, inconsistencies, and redundancies in trigger-action rules at composition time, we first translate the rules into the corresponding SCPN. Possible loops are detected by performing a depth-first search on the net.

To detect inconsistencies and redundancies, instead, we need to execute and analyze the net. For this purpose, tokens are used as artifacts during the analysis of the net execution. For starting the execution, a SCPN offers many possibilities: in our case, to identify problems in rules at composition time, the initial marking is a single token in the *Trigger Place* related to the rule that is being defined, i.e., $T4$ for R7 in Figure 2. Then, the net is executed, and the activated transitions move the token in the net. When the token is in a *TP*, all the rules that share such a specific trigger are activated. In the first step of the execution of Figure 2, for example, the token is removed from $T4$ by the *Copy Transition* and replicated in each copy of $T4$, thus activating the Rule Transitions of $R6$, $R5$, and $R7$. In the next step, the net may execute $R6$, i.e., one of the activated *Rule Transitions*, thus moving the token from the $T4copy$ to $A1$. This simulate the execution of action associated with $A1$. Following this process, the *Action Places* crossed by the tokens during the execution (i.e., the executed actions) along with the associated semantic colors are analyzed to detect inconsistencies and redundancies. An *inconsistency* is found if there are at least two executed actions that a) act on the same device or online service, and b) are classified under different EUPont classes. A *redundancy* is, instead, found if there are at least two executed actions that a) act on the same device or web application, and b) share the same EUPont classes.

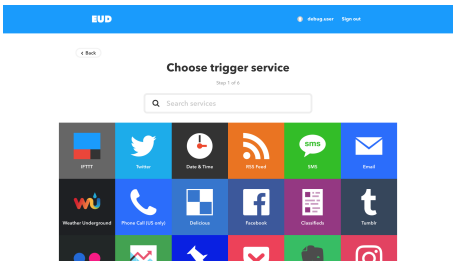

Figure 3: The user interface for composing a new trigger-action rule, showing the selection of the service to be used as a trigger.

To further exemplify, the net in Figure 2 presents a loop arising between R4 and R7 (i.e., the rule that is being defined). A redundancy and an inconsistency are also present. The former arises since the action of R5 ($A5$) contains multiple tokens at the end of the net execution, thus generating many Telegram messages (an infinite number), as the trigger of R5 ($T4$) is involved in the loop. The latter arises since two APs (i.e., $A1$ and $A6$), crossed during the execution of the net, model two inconsistent actions, i.e., "lock the entrance door" and "unlock the entrance door". By getting rid of R7, a user can eliminate all those problems.

Finally, for what concerns the rules simulation, various evolutions of the net can be executed step-by-step by randomly selecting a transition to be fired from the set of transitions that are enabled in a given moment.

EUDebug User Interface

The EUDebug user interface can be logically split in three parts: a) *Rule Composition*, b) *Problem Checking*, and c) *Step-by-Step Explanation*. The Problem Checking and the Step-by-Step Explanation interfaces implement the two adopted strategies for end-user debugging, respectively: identification of rule conflicts, and simulation of the run-time behavior.

To allow the composition of trigger-action rules, in our EUDebug prototype, we modeled the composition interface after IFTTT (Figure 3) due to the popularity of the platform [17], its ease of use and accuracy in the rule composition process [10], and the availability of real usage data [37], which we used to define available triggers and actions. In addition,

Figure 4: The Problem Checking interface showing an inconsistency between an already existent rule and the defined one.

the form-filling procedure it adopts helps users to avoid syntactical errors during the rule composition. To compose a rule, a user needs to first select which service (i.e., a supported smart device or web application) they want to use as a trigger (Figure 3). Once they select a service, they can choose the specific trigger to be used (e.g., "turned on" for Philips Hue lamps) and fill any additional information required by the trigger (e.g., which Philips Hue lamp they want to use). To define the action part of the rule, the user has to repeat the same steps.

The composed rule is, then, described according to the SCPN formalism, and analyzed by the net to look for any loops, inconsistencies, and redundancies. The results of the analysis of the SCPN are, in real time, shown to the user in the Problem Checking interface (Figure 4).

The *Problem Checking* interface shows the rule just defined by the user and any problems that the rule may generate. In Figure 4, for instance, a possible inconsistency between two rules is highlighted. To better understand the problems and to foresee the run-time behavior of the involved trigger-action rules, the user can click on the "Explanation" button to open the *Step-by-Step Explanation* interface (Figure 5). In such an interface, the user can simulate step-by-step what happens within their rule, to try to understand why the highlighted problems arise. For instance, Figure 5 shows that the event *"You exit an area"* activates a sequence of trigger-action rules that includes the rule that is being defined, and two inconsistent actions that close and open a door at the same time.

Implementation

The implementation of the EUDebug prototype consists of two main components:

Rules Server It is built in Java with the Spring framework[5]. It is composed of three modules: *Rule Service*, *SCPN Service*, and *Rule Controller*. The Rule Service offers the features needed to manage collections of trigger-action rules, i.e., to create, read, update, and delete rules through the interaction with a MySQL database. Once a rule has been completed by a user, the SCPN Service generates and analyzes the SCPN by retrieving the defined rules from the Rule Service, and by using the OWL API[6] library to extract the needed semantic information from the EUPont ontology. The same module is also responsible for the step-by-step simulation of the involved rules. Finally, the Rule Controller exposes a list of REST APIs to interact with the two services.

EUDebug Interface It is the web-based interface of the EUDebug prototype, built with the Angular framework[7]. It interacts with the Rules Server through the provided REST APIs.

5 EXPLORATORY USER STUDY

We ran an exploratory study with 15 participants to evaluate whether EUDebug helps them to a) *understand* and b) *identify* problems that may arise in their trigger-action rules. The following questions guided our study:

(1) **Understandability**. Can EUDebug help end users debug their trigger-action rules? Do they understand the involved problems and why their rules generate them?
(2) **Identification**. Is highlighting the detected problems sufficient to identify such problems, or do users need the additional details provided by the step-by-step simulation? In other words, which of the two adopted strategies is more useful?

Study Procedure

We recruited 15 university students (9 males and 6 females) with a mean age of 20.34 years ($SD = 2.50$, $range : 18 - 25$). We excluded users who had previous experience in computer science and programming. On a Likert-scale from 1 (Very Low) to 5 (Very High), participants stated their level of technophilia ($M = 3.94$, $SD = 0.80$) and technological savviness ($M = 2.67$, $SD = 0.82$). Furthermore, on a Likert-scale from 1 (No knowledge at all) to 5 (Expert), participants declared their experience with trigger-action programming ($M = 1.34$, $SD = 1.04$). We brought each participant to our lab for a 45-minute session using our EUDebug prototype on a Macbook Pro connected to an external 22-inch monitor. At the beginning of the study, participants were introduced

[5]https://spring.io, last visited on September 18, 2018
[6]http://owlapi.sourceforge.net, last visited September 18, 2018
[7]https://angular.io, last visited on September 18, 2018

Figure 5: Step-by-Step Explanation: a sequence of screenshots of the user interface related to the step-by-step simulation of the inconsistency problem of Figure 4.

ID	Trigger Service	Trigger	Action Service	Action
TA1	Android Location	You enter an area (*where:* home)	Philips Hue	Turn on lights (*what:* kitchen lamp)
TA2	Android Location	You enter an area (*where:* home)	Philips Hue	Turn off lights (*what:* kitchen lamp)
TA3	Android Location	You enter an area (*where:* home)	Philips Hue	Turn on color loop (*what:* kitchen lamp)
TA4	iOS Photo	New photo added to album (*album:* ios photos)	Dropbox	Add file from URL (*URL:* ios photo, *folder:* drpb photos)
TA5	Dropbox	New file in your folder (*Folder:* drpb photos)	Facebook	Upload a photo from URL (*URL:* drpb photo)
TA6	Facebook	New photo post by you	iOS Photo	Add photo to album (*URL:* facebook photo, *Album:* ios photos)
TA7	iOS Location	You exit an area (*where:* work)	SmartThings	Lock (*what:* office door)
TA8	SmartThings	Locked (*what:* office door)	Homeboy	Arm camera (*what:* office camera)
TA9	Homeboy	Camera armed (*what:* office camera)	SmarThings	Unlock (*what:* office door)
TA10	Amazon Alexa	New song played	Twitter	Post a tweet (*text:* I liked the Alexa song)
TA11	Amazon Alexa	New song played	Spotify	Save a track (*track:* alexa song)
TA12	Spotify	New saved track	Twitter	Post a tweet (*text:* I liked the Spotify song)

Table 2: The 12 trigger-action rules composed in the study.

to trigger-action programming and to EUDebug with an example of a rule composition. To allow us to investigate *Understandability*, participants were not introduced to the problems that rules may generate. We then presented a task involving the composition of 12 trigger-action rules, which include both smart devices and online services. The rules, reported in Table 2, generated 5 different problems (i.e., 2 inconsistencies, 2 redundancies, and 1 loop):

- **IC1**. TA1 and TA2 generate an **inconsistency**, because they share the same trigger while producing contradictory actions on the same device;
- **IC2**. TA7 and TA9 generate an **inconsistency**, because they produce contradictory actions on the same device and are activated nearly at the same time, since TA7 activates TA8, and TA8 activates TA9;
- **RD1**. TA1 and TA3 generate a **redundancy**, because they share the same trigger while producing two similar actions on the same device;
- **RD2**. TA10 and TA12 generate a **redundancy**, because they produce similar actions on the same online service and are activated nearly at the same time, since TA10 and TA11 share the same trigger and TA11 activates TA12.
- **LP**. TA4, TA5, and TA6 generate an infinite **loop**, because TA4 activates TA5, TA5 activates TA6, and TA6 activates TA4;

Rules were presented one at a time on a sheet of paper in a counterbalanced order. To make sure that all the participants experienced a given problem in the same way, however, we maintained the order within each problem, e.g., TA2 was always presented after TA1. When the EUDebug interface highlighted some problems (Problem Checking), participants were free to decide whether to save the rule or not. Our aim was to investigate whether participants understood the presented problems and their dangerousness, without forcing them to discard problematic rules. Before deciding, participants could optionally use the Explanation button to perform the step-by-step simulation of the rules that generated the problem. All the sessions were audio recorded for further analysis.

Measures. During the study, we collected the following quantitative measures:

- **S** - Number of rules that generated a problem *saved* anyway by participants, monitored for each highlighted problem, e.g., number of saved rules in case of loops.
- **D** - Number of rules that generated a problem *discarded* by participants, monitored for each highlighted problem.
- **SbS** - Number of times participants used the *Step-by-Step Explanation* when experienced a specific problem.

In addition, if participants used the Step-by-Step Explanation, we asked them:

- **SbS Motivation** - Why they decided to use the Step-by-Step Explanation.
- **SbS Usefulness** - Whether and how the explanation helped (or not) them to understand the problem.

Furthermore, when the composition of a rule generated a problem, we asked participants for their **Interpretation**. The interpretation was asked *before* the optional usage of the Step-by-Step Explanation interface. In particular, when a participant decided to *discard* a rule that generated a problem, they had to demonstrate to understand the problem by retrospectively explaining why the rule generated the issue. When they decided to *save* anyway a rule that generated a problem, instead, they had to justify their choice. In the next sections, we present and discuss the findings of the study, by organizing the discussion around the main topics that emerged from the analysis of the results. Qualitative analysis was conducted by two researchers in an iterative coding process.

EUDebug as a Helper for Understanding Problems

Differences in Users' Behavior. Most of the participants perceived EUDebug as a helper for understanding whether the highlighted problems were "dangerous" or not. Moreover, they exhibited different behaviors when facing the various problems, i.e., they considered *redundancies* as less problematic than loops and inconsistencies, at least in some specific cases.

Rule	Problem	Type	S	D	SbS
TA2	**IC1**	Inconsistency	1	14	5
TA9	**IC2**	Inconsistency	0	15	5
TA3	**RD1**	Redundancy	12	3	3
TA12	**RD2**	Redundancy	2	13	8
TA6	**LP**	Loop	3	12	7

Table 3: The number of times participants (N = 15) saved a rule (S), discarded a rule (D), or used the Step-by-Step explanation (SbS) when a problem is highlighted.

In detail, we analyzed how many times participants saved (or discarded) a rule that generated a given problem, i.e., the *S* and *D* measures. As reported in Table 3, 12 participants out of 15 (80%) discarded TA6, i.e., the rule that generated the loop L. Instead, participants discarded the rule that generated an inconsistency in the 96.67% of the cases, on average: for IC1, 14 participants out of 15 (93.34%) discarded TA2, while for IC2 all the participants discarded TA12. This seems to suggest that participants were aware of the "danger" caused by such problems. Conversely, participants discarded the rule that generated a redundancy, i.e., RD1 and RD2, only in 53.34% of cases, on average. Therefore, at least in some cases, redundancies seemed to be considered less "dangerous" and even acceptable than loops and inconsistencies.

Key Takeaway: Participants showed different perceptions among the various problems. They considered loops and inconsistency as dangerous, while they were inclined to accept redundancies.

Virtual vs. Physical Worlds. Since participants had opposite behaviors when facing with the two *redundancies*, we further analyze the collected data and the audio recording of the entire session. In fact, only 3 participants out of 15 (20%) discarded the rule that generated the RD1 problem, while 13 participants out of 15 (86.67%) discarded TA12, i.e., the rule that generated RD2. The reason for such a difference in the participants' behavior can be glimpsed by inspecting the *nature of the rules* involved in the two redundancies. In the first redundancy, considered as "acceptable" by the majority of the participants, both involved rules turned on the kitchen lamp with different colors. Instead, the second redundancy, considered as "unacceptable" by the majority of the participants, produced two similar messages on Twitter. We can preliminary conclude that redundancies in the "virtual" world, e.g., multiple messages on the web, are more annoying compared to redundancies in the "physical" world. In fact, rules in "physical" redundancies often send similar commands to a device without drastically modifying its current

state, e.g., the fact that a lamp is turned on. On the contrary, "virtual" redundancies typically result in duplicated messages and notifications, a potentially more annoying behavior.

Key Takeaway: Redundancies in online services (e.g., social networks) were considered as more annoying with respect to redundancies that involved physical devices.

Differencies in Users' Interpretation. Most of the participants give a correct *interpretation* about their choice of saving or discarding a problematic rule. However, not all the problems were equally understood, with *loops* being the most difficult problem to understand.

In details, to investigate whether participants understood the meaning of the encountered problems and why they happened, we used the *SbS* measure and the participants' *interpretations* extracted from the audio recording.

Inconsistency: For what concerns IC1, all the 14 participants that discarded TA2 provided a sound interpretation. P1, for example, said *"the rules did not have any sense. They turned the lights on and off at the same time. The two commands (turn the lights on and turn the lights off) cannot be executed at the same time."* P7, beside explaining the problem, also identified a possible alternative: *"I would have modified the trigger: this rule is ok when you exit the home area."* Only 1 participant, the one that decided to save TA2, provided an incorrect interpretation of the problem even after using the Step-by-Step Explanation. In her interpretation, in particular, she said *"I do not trust the platform, I am sure that such two rules will never be activated at the same time."* The 15 interpretations collected for IC2 are also encouraging. 11 participants, in particular, provided a sound interpretation after discarding TA9, such as *"if the door is locked, the camera is armed, but when the camera is armed, this rule unlocked the door!"* or *"this rule will unlock the door when I leave the office: not good."* The remaining 4 participants immediately discarded TA9, but they provided a misinterpretation. In their interpretation, in particular, they focused on the rule they were evaluating, only, rather than on the entire chain of rules that generated the problem, i.e., TA7, TA8, and TA9. P7, for example, said *"I did not save the rule because I want the door to remain closed"*, while P8 said *"if the camera is armed, the door must be closed."* A possible explanation can be found in their decision to discard the rule without using the Step-by-Step Explanation. On average, 5 participants out of 15 (33.34%) used the Step-by-Step Explanation (Table 3).

Redundancy: The number of wrong interpretations of redundancies is similar to the number of wrong interpretations of inconsistencies. For what concerns RD1, 13 participants out of 15 (86.67%) provided a sound interpretation. In particular, 11 participants out of 12 (91.67%) successfully provided an interpretation for their decision to save TA3 anyway. All of them declared that they were aware of what would happen, and that the highlighted issue was not a problem at all. P6 said that the color can be seen as a *"new feature"* of the first rule, while P7 asserted that *"the important thing is that the lamp is turned on, I do not care its color."* The only participant that provided a wrong interpretation was P15, the same participants that made an error for IC1. No one of the 12 participants that saved TA3 used the Step-by-Step Explanation. Instead, all the 3 participants that discarded TA3 used the Step-by-Step Explanation, and 2 of them provided a sound interpretation, while the other focused on TA3, only, by saying *"I do not want a colored light in the kitchen"*. Also for the second redundancy, i.e., RD2, no one of the 2 participants that saved TA12 anyway used the Step-by-Step Explanation, but all of them provided a sound interpretation. Instead, 11 of the remaining 13 participants (84.61%) that discarded TA12 successfully provided an interpretation. P1, for example, explained exactly what happened by saying *"When I listen to a song on Alexa, the defined rules post a tweet and save the track on Spotify. Now I'm defining a rule to post on Twitter when I saved a track on Spotify, but there is already a post on Twitter!"* The remaining 2 participants, even after using the Step-by-Step Explanation, focused on TA12, only, by saying, for example, *"it does not have any sense to post on Twitter the song you are listening"*. On average, participants used the Step-by-Step Explanation in 36.67% of cases (Table 3).

Loop: The loop LP led participants to make more errors in their interpretations. Since a loop can never be considered as "acceptable", all the 3 participants that saved TA6 failed in providing a correct interpretation. P13, for example, did not understand that the 3 involved rules would be executed infinite times, because she said *"I am sure that this problem will never occur with the rules I have defined. Moreover, such rules are useful, because the photo will be saved in 3 places at the same time."* Furthermore, also 3 of the participants that discarded TA6 provided an incorrect interpretation. The prevailing error was that participants did not understand that the involved rules would have been executed for an infinite number of times: both P1 and P12, for example, said *"I did not save the rule because otherwise the same photo would have been shared twice on Facebook."* Therefore, results suggest that the loop was the most complex concept to understand. A series of paired-samples t-test confirm this finding. In fact, the number of errors in loop interpretations was significantly higher than in redundancies ($t(14) = 2.25, p < 0.05$), while such a difference was not significant with respect to inconsistencies ($t(14) = 1.97, p = 0.06$). For the loop, the Step-by-Step Explanation was more used (7 participants out of 15, 46.67%, Table 3) than for the other problems. A possible explanation of such understandability problems is that the concept of

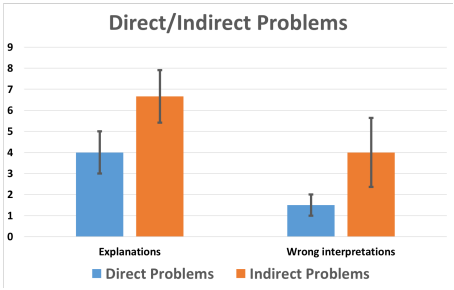

Figure 6: Average number of explanations used and average number of wrong interpretations for direct/indirect problems.

loop is strictly related to the mental model of users with a computer science background.

Key Takeaway: The loop turned to be the most difficult problem to understand, and led participants to frequently use the Step-by-Step Explanation.

Direct vs. Indirect Problems. Finally, we noticed a possible link between the "nature" of a problem and its understandability by further analyzing the number of *Step-by-Step Explanations* used and the number of wrong interpretations in each problem. In particular, when subjected to the first 2 problems, i.e., IC1 and RD1, participants used the Step-by-Step Explanation in fewer cases, and provided less wrong interpretations with respect to the other three problems, i.e., LP, IC2, and RD2. Such a difference can be associated with the nature of the problems. IC1 and RD1, in fact, are *direct* problems, i.e., problems between rules that shared the same trigger. On the contrary, LP, IC2, and RD2 are *indirect* problems, because they are caused by implicit activations between rules, i.e., an action of a rule that implicitly activates the trigger of another rule. Figure 6 visually shows the differences between direct and indirect problems, and further suggests that indirect problems are more difficult to understand, and need more efforts, e.g., a step-by-step simulation, to be identified by end users.

Key Takeaway: Indirect problems, i.e., problems caused by implicit activations between rules, were difficult to understand and identify.

Highlighting Problems or Explaining Them?

To investigate whether participants found more useful one of the two strategies adopted by EUDebug for identifying problems in trigger-action rules, we studied the correlation between the interface used, i.e., Problem Checking or Step-by-Step Explanation, and the participants' *interpretations* in case of a problem (Table 4). On average, the usage of the Problem Checking interface, only, resulted in a correct interpretation in 77.81% of cases. When participants decided to use the Step-by-Step Explanation, the percentage of correct interpretations increased to 83.78%. Such a difference is particularly evident for the loop L. Only 50% of the participants, in fact, discarded TA6 by providing a correct interpretation by using the Problem Checking interface, only. Participants that used the Step-by-Step Explanation, instead, provided a correct interpretation in 71.43% of cases. This seems to suggest that, at least in some cases, highlighting the detected problems (i.e., Problem Checking) may be *not* sufficient to allow end users in identifying possible problems in their rules, and that a step-by-step simulation of the involved rules could instead help users in understanding what happens.

	L	I	R	Total
PC success	50%	85%	84.52%	77.81%
SbS success	71.43%	80%	93.75%	83.78%

Table 4: Number of times participants provided a correct *interpretation* by using the Problem Checking interface, only (PC success), or after using the Step-by-Step explanation (SbS success).

To confirm this finding, we analyzed the participants' feedback about the usage of the Step-by-Step Explanation (used 28 times in total) by group it into several topics described below. For what concern the *SbS Motivation*, in most of the cases participants asserted that they used the Step-by-Step Explanation to "better understand the problem" (13). When subjected to IC1, for example, P1 said *"I used the Step-by-Step Explanation because I did not understand the problem. The two rules seemed the same to me."*. Similarly, P10 used the step-by-step explanation for RD2 *"to better understand the redundancy concept"*, while P15 provided the same motivation when subjected to the loop. In a considerable number of cases (8), the Step-by-Step Explanation was instead used because "the problems were composed of too many steps", i.e., rules that activated other rules. Not surprisingly, such motivation was used for indirect problems, only. In one case, for example, P14 said *"I used the Explanation because I did not understand the execution path of the rules"*, while in another case, P12 said *"I used the Explanation because I did not understand the relationship between the rules."* In the remaining cases, participants used the Step-by-Step Explanation because "they did not remember a rule they defined before" (4), "to confirm their first idea about the problem" (2), and because the "Explanation helped them before" (1).

Participants provided interesting feedback also when asked to evaluate whether and how the Explanation helped (or not) them in understanding the problems (*SbS Usefulness*). In 13 cases, participants asserted that the Step-by-Step Explanation was useful because it allowed them "to see all the involved steps." Participants provided this feedback for

indirect problems, mainly. The loop, in particular, was the problem for which this feedback was more common. P6, for example, said *"the Explanation helped me in understanding the loop because I could better see the evolution of the rules,"* while P10 pointed out that seeing the figures related to the rules one at a time helped her in understanding the problem. In other 5 cases, the Step-by-Step Explanation helped "participants to remember a rule they had defined before" (*"The Explanation helped me in understanding the problem because it told me: hey, you have defined this rule before!"*, P8). This feedback takes even more importance if we think to the real usage of an EUD platforms, where rules are defined in different moments, even months later. In other 5 cases, participants asserted that "the Explanation helped them by visually highlighting the problem" (*"The Explanation helped me to understand the problem because it visually told me what happened"*, P6). In the remaining cases, participants provided generic feedback about the usefulness of the Explanation, i.e., "it helped me in understanding the problem" (3) and "it confirmed my first idea" (2).

Key Takeaway: Highlighting the detected problem was often not sufficient, while the step-by-step simulation of the involved rules helped users understand problems.

Limitations

The main limitation of our study is that it was exploratory in nature. In addition, this study targeted users without any programming skills, only, and involved the creation of 12 trigger-action rules in a lab setting; a more ecologically-valid study would be to deploy EUDebug in-the-wild where end users could use it with their own rules. As such, our results raise the possibility of EUDebug's debugging strategies leading to a more predictable and correct usage of trigger-action programming for the IoT. These findings could inform follow-up comparative studies or future development.

6 CONCLUSION

We presented EUDebug, a system that enables users to debug, at composition time, the trigger-action rules they create for their IoT ecosystem. EUDebug highlights any possible problems that the rules may generate and allows a step-by-step simulation. It exploits a Semantic Colored Petri Net formalism to model, check, and simulate trigger-action rules. Results of an exploratory study with 15 participants suggest that end users, with the help of EUDebug, can deal with computer-related concepts such as loops, inconsistencies, and redundancies. Moreover, they are able to understand why their rules may generate a specific problem in most of the cases. Results also highlight different perceptions among the various highlighted problems, i.e., end users demonstrate to be more tolerant with redundancies than with loops and inconsistencies.

REFERENCES

[1] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2017. Visual Simple Transformations: Empowering End-Users to Wire Internet of Things Objects. *ACM Transactions on Computer-Human Interaction* 24, 2, Article 10 (April 2017), 43 pages. https://doi.org/10.1145/3057857

[2] Carmelo Ardito, Paolo Buono, Giuseppe Desolda, and Maristella Matera. 2018. From smart objects to smart experiences: An end-user development approach. *International Journal of Human-Computer Studies* 114 (2018), 51 – 68. https://doi.org/10.1016/j.ijhcs.2017.12.002 Advanced User Interfaces for Cultural Heritage.

[3] J. C. Augusto and M. J. Hornos. 2013. Software simulation and verification to increase the reliability of Intelligent Environments. *Advances in Engineering Software* 58, Supplement C (2013), 18 – 34. https://doi.org/10.1016/j.advengsoft.2012.12.004

[4] J. Bailey, G. Dong, and K. Ramamohanarao. 2004. On the decidability of the termination problem of active database systems. *Theoretical Computer Science* 311, 1 (2004), 389 – 437. https://doi.org/10.1016/j.tcs.2003.09.003

[5] B. R. Barricelli and S. Valtolina. 2015. *End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings.* Springer International Publishing, Cham, Germany, Chapter Designing for End-User Development in the Internet of Things, 9–24. https://doi.org/10.1007/978-3-319-18425-8_2

[6] Julia Brich, Marcel Walch, Michael Rietzler, Michael Weber, and Florian Schaub. 2017. Exploring End User Programming Needs in Home Automation. *ACM Transaction on Computer-Human Interaction* 24, 2, Article 11 (April 2017), 35 pages. https://doi.org/10.1145/3057858

[7] A.J. Bernheim Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu, and Colin Dixon. 2011. Home Automation in the Wild: Challenges and Opportunities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2115–2124. https://doi.org/10.1145/1978942.1979249

[8] Margaret Burnett, Curtis Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Chris Wallace. 2003. End-user Software Engineering with Assertions in the Spreadsheet Paradigm. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 93–103.

[9] R. J. Butler. 2000. Is this spreadsheet a tax evader? How HM Customs and Excise test spreadsheet applications. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. 6 pp. vol.1–. https://doi.org/10.1109/HICSS.2000.926737

[10] Danilo Caivano, Daniela Fogli, Rosa Lanzilotti, Antonio Piccinno, and Fabio Cassano. 2018. Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. *Journal of Systems and Software* 144 (2018), 295–313. https://doi.org/10.1016/j.jss.2018.06.035

[11] Julio Cano, Gwenaël Delaval, and Eric Rutten. 2014. Coordination of ECA Rules by Verification and Control. In *Coordination Models and Languages*, Eva Kühn and Rosario Pugliese (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–48.

[12] J. Cao, K. Rector, T. H. Park, S. D. Fleming, M. Burnett, and S. Wiedenbeck. 2010. A Debugging Perspective on End-User Mashup Programming. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. 149–156. https://doi.org/10.1109/VLHCC.2010.29

[13] V. Cerf and M. Senges. 2016. Taking the Internet to the Next Physical Level. *IEEE Computer* 49, 2 (Feb 2016), 80–86. https://doi.org/10.1109/MC.2016.51

[14] F. Corno, L. De Russis, and A. Monge Roffarello. 2017. A Semantic Web Approach to Simplifying Trigger-Action Programming in the

IoT. *Computer* 50, 11 (2017), 18–24. https://doi.org/10.1109/MC.2017.4041355

[15] Jose Danado and Fabio Paternò. 2014. Puzzle: A Mobile Application Development Environment Using a Jigsaw Metaphor. *Journal of Visual Languages and Computing* 25, 4 (Aug. 2014), 297–315. https://doi.org/10.1016/j.jvlc.2014.03.005

[16] Florian Daniel and Maristella Matera. 2014. *Mashups: Concepts, Models and Architectures*. Springer Publishing Company, Incorporated.

[17] G. Desolda, C. Ardito, and M. Matera. 2017. Empowering End Users to Customize Their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools. *ACM Transactions on Computer-Human Interaction* 24, 2, Article 12 (2017), 52 pages. https://doi.org/10.1145/3057859

[18] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive Prototyping of Context-aware Applications. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE'06)*. Springer-Verlag, Berlin, Heidelberg, 254–271. https://doi.org/10.1007/11748625_16

[19] D. Evans. 2011. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. Technical Report. Cisco Internet Business Solutions Group.

[20] S. Gatziu and K. R. Dittrich. 1994. Detecting composite events in active database systems using Petri nets. In *Proceedings of IEEE International Workshop on Research Issues in Data Engineering: Active Databases Systems*. 2–9. https://doi.org/10.1109/RIDE.1994.282859

[21] G. Ghiani, M. Manca, F. Paternò, and C. Santoro. 2017. Personalization of Context-Dependent Applications Through Trigger-Action Rules. *ACM Transactions on Computer-Human Interaction* 24, 2, Article 14 (2017), 33 pages. https://doi.org/10.1145/3057861

[22] Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. 2012. End-user Debugging Strategies: A Sensemaking Perspective. *ACM Transaction on Computer-Human Interaction* 19, 1, Article 5 (May 2012), 28 pages. https://doi.org/10.1145/2147783.2147788

[23] J. Huang and M. Cakmak. 2015. Supporting Mental Model Accuracy in Trigger-action Programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 215–225. https://doi.org/10.1145/2750858.2805830

[24] Ting-Hao K. Huang, A. Azaria, and J. P. Bigham. 2016. InstructableCrowd: Creating IF-THEN Rules via Conversations with the Crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 1555–1562. https://doi.org/10.1145/2851581.2892502

[25] K. Jensen. 1995. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2*. Springer-Verlag, London, UK, UK.

[26] Xiaoqing Jin, Yousra Lembachar, and Gianfranco Ciardo. 2014. *Symbolic Termination and Confluence Checking for ECA Rules*. Springer Berlin Heidelberg, Berlin, Heidelberg, 99–123. https://doi.org/10.1007/978-3-662-45730-6_6

[27] Cory Kissinger, Margaret Burnett, Simone Stumpf, Neeraja Subrahmaniyan, Laura Beckwith, Sherry Yang, and Mary Beth Rosson. 2006. Supporting End-user Debugging: What Do Users Want to Know?. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '06)*. ACM, New York, NY, USA, 135–142. https://doi.org/10.1145/1133265.1133293

[28] A. J. Ko and B. A. Myers. 2004. Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 151–158. https://doi.org/10.1145/985692.985712

[29] Jisoo Lee, Luis Garduño, Erin Walker, and Winslow Burleson. 2013. A Tangible Programming Tool for Creation of Context-aware Applications. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '13)*. ACM, New York, NY, USA, 391–400. https://doi.org/10.1145/2493432.2493483

[30] X. Li, J. M. Medina, and S. V. Chapa. 2007. Applying Petri Nets in Active Database Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 4 (July 2007), 482–493. https://doi.org/10.1109/TSMCC.2007.897329

[31] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. ACM, New York, NY, USA, 398–404. https://doi.org/10.1145/3131365.3131369

[32] Dejan Munjin. 2013. *User Empowerment in the Internet of Things*. Ph.D. Dissertation. Université de Genève. http://archive-ouverte.unige.ch/unige:28951

[33] A. Namoun, A. Daskalopoulou, N. Mehandjiev, and Z. Xun. 2016. Exploring Mobile End User Development: Existing Use and Design Factors. *IEEE Transactions on Software Engineering* 42, 10 (Oct 2016), 960–976. https://doi.org/10.1109/TSE.2016.2532873

[34] Raymond R. Panko. 1998. What We Know About Spreadsheet Errors. *J. End User Comput.* 10, 2 (May 1998), 15–21. http://dl.acm.org/citation.cfm?id=287893.287899

[35] K. T. Stolee and S. Elbaum. 2013. Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups. *IEEE Transactions on Software Engineering* 39, 12 (Dec 2013), 1654–1679. https://doi.org/10.1109/TSE.2013.42

[36] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical Trigger-action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 803–812. https://doi.org/10.1145/2556288.2557420

[37] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman. 2016. Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes. In *Proceedings of the 34rd Annual ACM Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3227–3231. https://doi.org/10.1145/2858036.2858556

[38] C. Vannucchi, M. Diamanti, G. Mazzante, D. Cacciagrano, R. Culmone, N. Gorogiannis, L. Mostarda, and F. Raimondi. 2017. Symbolic verification of event–condition–action rules in intelligent environments. *Journal of Reliable Intelligent Environments* 3, 2 (01 Aug 2017), 117–130. https://doi.org/10.1007/s40860-017-0036-z

[39] S. J. H. Yang, A. S. Lee, W. C. Chu, and Hongji Yang. 1998. Rule base verification using Petri nets. In *Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International*. 476–481. https://doi.org/10.1109/CMPSAC.1998.716699