A Debugging Approach for Trigger-Action Programming

A Debugging Approach for Trigger-Action Programming

Luigi De Russis
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino, Italy 10129
luigi.derussis@polito.it

Alberto Monge Roffarello
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino, Italy 10129
alberto.monge@polito.it

Figure 1: Our debugging tool. Before saving a trigger-action rule, users are informed about possible problems that the rule may generate.

CHI'18 Extended Abstracts, April 21–26, 2018, Montreal, QC, Canada
ACM 978-1-4503-5621-3/18/04.
https://doi.org/10.1145/3170427.3188641

Abstract
Nowadays, end users can customize their technological devices and web applications by means of trigger-action rules, defined through End-User Development (EUD) tools. However, debugging capabilities are important missing features in these tools that limit their large-scale adoption. Problems in trigger-action rules, in fact, can lead to unpredictable behaviors and security issues, e.g., a door that is unexpectedly unlocked. In this paper, we present a novel debugging approach for trigger-action programming. The goal is to assists end users during the composition of trigger-action rules by: a) highlighting possible problems that the rules may generate, and b) allowing their step-by-step simulation. The approach, based on Semantic Web and Petri Nets, has been implemented in a EUD tool, and it has been preliminary evaluated in a user study with 6 participants. Results provide evidence that the tool is usable, and it helps users in understanding and identifying problems in trigger-action rules.

Author Keywords
End User Development; Trigger-Action; Debug; Internet of Things

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous

Introduction and Motivation

The potential of the Internet of Things (IoT) is nowadays being increasingly recognized, and its adoption already helps society in many different ways, i.e., with applications for the individual and for the industries as well. As reported in recent studies, e.g., [4], the IoT ecosystem can be defined from a wide perspective, by including not only physical devices, but also web applications such as messaging services and social networks. In this domain, End-User Development (EUD) tools empower users to define joint behaviors between sets of devices and web applications, on the basis of their personal needs. Such behaviors are typically composed through a set of trigger-action rules such as *"if the Nest camera in the kitchen detects a movement, then send me a Telegram message."* Despite apparent simplicity, the composition of such rules may be a complex task for end users [5], and the advent of new interconnected objects and web services raises new challenges. One of the most important is the need of assessing the correctness of trigger-action rules defined by users. Errors in trigger-action rules, in fact, can lead to unpredicted and dangerous behaviors: while posting a content on a social network twice could be considered a minor issue, in a smart home scenario wrong rules could unexpectedly unlock a door, thus generating a security threat. Unfortunately, even if debugging features could facilitate the adoption of EUD tools in the real world [3], relatively little work has been done in this field. In this paper, we present a novel debugging approach for trigger-action programming to assists end users during the composition of trigger-action rules. The goal is to properly warn users when they are defining any troublesome or potentially dangerous behavior, providing mechanisms to understand why such problems can happen. For this purpose, we define a formalism based on Semantic Web and Petri Nets to model the execution of trigger-action rules, and we integrated it in a tool (Figure 1) that: a) highlights any possible problems that the rule which is being defined may generate, and b) allows a step-by-step simulation of the rules that generated the problems. A preliminary evaluation with 6 participants demonstrate the usability of the tool, and suggests that the tool helps end users understand and identify problems in trigger-action rules.

Related Works

Relatively little work has been done to insert debugging features for end users that aim to jointly customize their devices and web applications on the basis of their personal needs. Cao et al. [1], for example, explore end users' debugging strategies in mashup programming, while Ko and Myers [6] propose an interrogative debugging interface for the Alice programming environment. None of such works, however, include IoT devices or is related to trigger-action programming. Moreover, using rules in "critical" systems implies that the system's behavior must be thoroughly analyzed, while providing end users with tools and methods for validating their rules could facilitate the adoption of EUD tools at a large-scale [3].Research targeting rule analysis has mainly been performed in the area of databases. Li et al. [7], in particular, propose a Conditional Colored Petri Net (CCPN) formalism to model and simulate Event-Condition-Action (ECA) rules for active databases. Some other previous works analyze rules in the context of smart environments, e.g., [9]. Typically, they adopt formal verification techniques to check the consistency of a set of defined rules, and employ predefined use cases to validate the algorithms. The goal of our work is different: instead of verifying "off-line" a set of already defined rules, we aim at assisting end users during the definition of their rules.

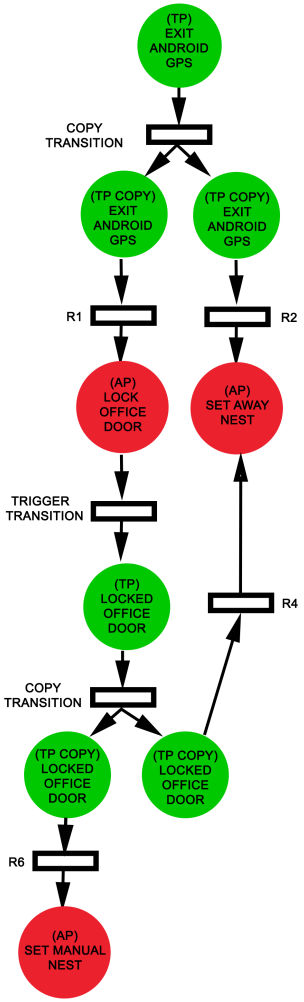

Figure 2: Part of the net that includes R1, R2, R4 and R6. Trigger Places are in green, Action Places in red.

Debugging Trigger-Action Rules

Problems in Trigger-Action Rules

Informed by the literature [9], we define three class of problems that can occur in sets of trigger-action rules: loops, inconsistencies, and redundancies. *Loops* occur when a set of trigger-action rules are continuously activated without halting by reaching a stable state. *Inconsistencies* happen when rules that are activated at (nearly) the same time[1] may try to execute contradictory actions. *Redundancies* occur, instead, when two or more rules are activated at (nearly) the same time and their functionality is replicated. The following set of trigger-action rules contains all the three problems:

R1. *if my Android GPS detects that I exit the home area, then lock the entrance door*
R2. *if my Android GPS detects that I exit the home area, then set the Nest thermostat to Away mode*
R3. *if I add a photo on my iOS library, then share it on Facebook*
R4. *if the entrance door is locked, then set the Nest thermostat to Away mode*
R5. *if I share a photo on Facebook, then save it on my iOS library*
R6. *if the entrance door is locked, then set the Nest thermostat to Manual mode*

R1, R2, R4, and R6 may be activated at nearly the same time (because R1 activates R4 and R6) and contain two inconsistent actions, i.e., *set the Nest thermostat to Away mode (R2 & R4)* and *set the Nest thermostat to Manual mode (R6)*, and two redundant actions, i.e., both R2 and R4 have the action*set the Nest thermostat to Away mode*.

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

Furthermore, R3 and R5 produce an infinite loop that constantly save and post the photo on iOS and Facebook.

Semantic Colored Petri Nets

We define a Semantic Colored Petri Nets (SCPN) formalism to model the execution of trigger-action rules. Petri nets can be used as a tool for describing distributed, concurrent, and asynchronous systems such as a rule-based system. Petri nets for describing rules have been mainly adopted in the area of active databases [7]. We adopt a similar approach for our context, i.e., end-user debug of IF-THEN rules. The main difference with other Colored Petri Net based approaches is that we use a semantic model to generate and analyze the net. Furthermore, each token of the net contains semantic information, i.e., the “color”, that allows the inference of more information from the simulation of the net. The exploited semantic model is EUPont [2], an ontological high-level description of trigger-action programming, that describes physical devices and web applications on the basis of their categories and capabilities, i.e., offered services. For each trigger or action, in particular, the ontology provides information about their final functionality, the service by which they are offered, and any relationship with other triggers or actions, e.g., the fact that the action *turn on the lamp* intrinsically activates the trigger *the lamp has been turned on*.

Figure 2 shows a partial view of the net built starting from the previous example (R1, R2, R4 and R6). The set of rules are modeled as transitions. Triggers and actions are modeled as places, i.e., *Trigger Place (TP)* and *Action Place (AP)*. A token in a *Trigger Place*, for example, means that all the rules that share that specific trigger are activated. When a trigger is in common between more than one rule (as in R1 and R2), the associated places are duplicated and connected through a *Copy Transition*. When there is a token in

the original place, the *Copy Transition* simply replicates it in each copied place. *Trigger Places* and *Action Places* can be connected each other through:

- **Rule Transition.** A connection between a trigger and an action by means of a trigger-action rule. *Rule Transitions* model the rule defined by the user. They remove a token from a *TP* and generate a new token in an *AP*.
- **Trigger Transition.** A connection used when an action of a rule triggers the event of another rule. *Triggers Transition* are extracted from the semantic information contained in EUPont. They remove a token from an *AP* and generate a new token in a *TP*.

Rule Analysis with SCPN

To detect loops, inconsistencies, and redundancies in trigger-action rules at composition time, we first translate the rules in the corresponding SCPN. Then, possible loops are detected by performing a depth-first search on the net. If the net is loop-free, i.e., its execution terminates in a finite number of transitions, the net is executed and analyzed. For the execution, a SCPN offers many possibilities. To identify problems in rules at composition time, for example, the initial marking can be a single token in the *Trigger Place* related to the rule that is being defined. When the net execution terminates, all the tokens in the net are analyzed to find inconsistencies and redundancies.

By analyzing the semantic information of the tokens, along with their position in the net, an *inconsistency* is found if there are at least two actions that: a) act on the physical object or web application, and b) are classified under contrasting EUPont functionality. A *redundancy*, instead, is found if there are at least two actions that: a) act on the physical object or web application, b) share the same EUPont final functionality. Finally, the nature of a Petri net opens up many opportunities, e.g., to simulate rules at real-time.

Implementation

We implemented the SPCN in a web-based tool for composing trigger-action rules. We exploited a large dataset of more than 200,000 trigger-action rules extracted from IFTTT [8] to define available triggers and actions. The tool assists end users in two ways, i.e., problem checking and step-by-step explanation. Users are informed of any loops, inconsistencies, or redundancies that the rule which is being defined may generate (Figure 1). By clicking an “explanation” button, users can inspect the problem by simulating the involved trigger-action rules step-by-step.

Preliminary evaluation

We preliminary evaluated our debugging approach in a user study by focusing on the usability and understandability of the implemented tool at composition time, leaving for future works the evaluation of the SCPN approach for run-time check and simulation of large sets of rules. We think that a user-centered approach is fundamental to understand whether end users are able to deal with debugging features such as those implemented in our tool. In particular, our aim was to investigate whether end users are able to **RQ1)** understand the concepts of loop, inconsistency, and redundancy in trigger-action programming, and why specific rules may generate such problems; and **RQ2)** understand and use the tool, including the step-by-step simulation.

We recruited 6 participants with a mean age of 20 years (SD = 1.26, range: 19-22) by sending emails to students coming from various backgrounds, i.e., biology, law, education, and aerospace engineering. In the recruitment process, we excluded users who had previous experience in computer science and programming.

	L	I	R
#	6	12	12
Discarded	5	12	5
Saved	1	0	7
Explanations	4	4	5

Table 1: The table reports the total number of times (#) participants encountered a Loop (L), an Inconsistency (I), or a Redundancy (R) in the study, and the number of time they discarded or saved the involved rules, along with the number of times they used the explanation button.

Study Description

At the beginning of the study, participants were introduced to the trigger-action programming and to the realized tool with an example of a rule composition. To investigate **RQ1**, participants were not introduced to the problems that rules may generate. The evaluation consisted of a single task. Participants were asked to compose 12 trigger-action rules of which 5 caused a problem with respect to the already defined rules (1 loop, 2 inconsistencies, and 2 redundancies). Rules were presented to the users one at a time in counterbalanced order. When the EUD tool highlighted some problems, they had to decide whether to save the rule or not. Before deciding, they could optionally use an explanation button to simulate the rules that generated the problem step-by-step.

Measures

We collected quantitative and qualitative measures, such as the number of saved and discarded rules and the number of times participants used the explanation button. When the composition of a rule generated a problem, we asked participants to answer some open questions by voice. In particular, when the participants decided to discard the rule, they had to demonstrate they understood the problem by retrospectively explaining why the rule generated the issue (*explanation*). On the contrary, if they decided to save the rule, they had to justify their choice (*justification*). At the end of the study, participants were asked to evaluate the comprehension of the tool on a Likert-scale from 1 (Not understandable) to 5 (Very understandable).

Results

The debugging perspective was successfully used in the study by all the participants, and the answers to the final questions about the comprehension of the tool (M = 3.75; SD = 0.96) give us a positive feedback on its usability (**RQ2**). Table 1 reports the quantitative measures collected during the study. On average, each participant discarded 3.67 rules out of the 5 that generated a problem, while they saved 1.34 rules even if the tool highlighted a problem. By analyzing these participants' choices, we found that 5 users discarded the rule that generated a loop, and all 6 users discarded both rules that generated an inconsistency. This suggests that participants were aware of the "danger" caused by such problems. The qualitative data further confirm that participants understood the concept of loop and inconsistency (**RQ1**). All 6 users, in fact, successfully provided a sound explanation in case of inconsistencies, while only 1 participant (the one that saved the rule anyway) had difficulty in understanding the concept of loop and provided a wrong justification. Not surprisingly, results suggest that the loop was the most complex concept to understand: in 4 cases out of 6 participants discarded the rule that caused the loop by providing a right explanation only after having used the step-by-step explanation. Conversely, on a total of 12 redundancies, users decided to save the rule in 7 cases. All the participants demonstrated they understood the redundancy concept (**RQ1**), and the provided justifications confirm that redundancies were perceived as less "dangerous" than loops and inconsistencies, and acceptable in some cases. For example, by looking at a redundancy that simultaneously posted on Twitter two tweets about the same topic, i.e., *"I listened to the new song of Ed Sheeran on YouTube"* and *"I listened to the new song of Ed Sheeran on Spotify"*, 2 users said *"this is not a problem for me, the two tweets are different so I want to save the rule anyway."*

Conclusion and Future Works

We presented a debugging approach to assist end users in defining trigger-action rules for customizing their smart devices and web services based on Semantic Web and Petri Nets. A preliminary evaluation provided interesting insights

about the usability and the understandability of the debugging approach and the related concepts. Open questions still remain to guide future works. How would end users actually debug their trigger-action rules, and which strategies would they adopt? To what extent they would benefit from problem checking and simulation features with a large set of rules?

REFERENCES

1. 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. DOI: http://dx.doi.org/10.1109/VLHCC.2010.29

2. 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. DOI: http://dx.doi.org/10.1109/MC.2017.4041355

3. 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. DOI:http://dx.doi.org/10.1145/3057859

4. 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. DOI:http://dx.doi.org/10.1145/3057861

5. 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. DOI: http://dx.doi.org/10.1145/2851581.2892502

6. 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. DOI: http://dx.doi.org/10.1145/985692.985712

7. 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. DOI: http://dx.doi.org/10.1109/TSMCC.2007.897329

8. 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. DOI: http://dx.doi.org/10.1145/2858036.2858556

9. 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. DOI: http://dx.doi.org/10.1007/s40860-017-0036-z