My IoT Puzzle: Debugging IF-THEN Rules Through the Jigsaw Metaphor

Fulvio Corno[1], Luigi De Russis[1], and Alberto Monge Roffarello[1]

Politecnico di Torino, Corso Duca degli Abruzzi, 24 Torino, Italy 10129
{fulvio.corno,luigi.derussis,alberto.monge}@polito.it

Abstract. End users can nowadays define applications in the format of IF-THEN rules to personalize their IoT devices and online services. Along with the possibility to compose such applications, however, comes the need to debug them, e.g., to avoid unpredictable and dangerous behaviors. In this context, different questions are still unexplored: which visual languages are more appropriate for debugging IF-THEN rules? Which information do end users need to understand, identify, and correct errors? To answer these questions, we first conducted a literature analysis by reviewing previous works on end-user debugging, with the aim of extracting design guidelines. Then, we developed *My IoT Puzzle*, a tool to compose and debug IF-THEN rules based on the Jigsaw metaphor. *My IoT Puzzle* interactively assists users in the debugging process with different real-time feedback, and it allows the resolution of conflicts by providing textual and graphical explanations. An exploratory study with 6 participants preliminary confirms the effectiveness of our approach, showing that the usage of the Jigsaw metaphor, along with real-time feedback and explanations, helps users understand and fix conflicts among IF-THEN rules.

Keywords: End-User Debugging · Internet of Things · Trigger-Action Programming · Visual Languages.

1 Introduction

The potential of the Internet of Things (IoT) is being increasingly recognized [7]: people daily interact with a growing number of Internet-enabled devices [13] in many different contexts, ranging from smart homes to smart cities. The IoT ecosystem is nowadays further enriched by online services such as messaging platforms and social networks [1]. By means of an Internet connection, users can therefore access a complex network of smart objects, either physical or virtual, able to interact and communicate with each other, with humans, and with the environment. Nowadays, end users can personalize such a complex network by programming the joint behavior of their IoT devices and online services. Several works in the literature demonstrate the effective applicability of End-User Development (EUD) techniques [23] for the creation of applications in various domains [22,11], including the IoT [36,14]. Particularly in this context, professional programmers cannot foresee all the possible situations end users may encounter when interacting with their IoT ecosystem. By placing the personalization of IoT devices and online services in the hands of end users, i.e., the subjects who are most familiar with the actual needs to be met, EUD is a viable way to make IoT applications comply with users' expectations [14]. Typically, users who want to personalize their IoT devices and online services can exploit the trigger-action programming approach, as implemented in popular visual programming platforms such as IFTTT[1] or Zapier[2]. Through trigger-action (IF-THEN) rules such as *"if the Nest camera in the kitchen detects a movement, then send me a Telegram message"*, users can connect a pair of 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.

Along with the possibility to create such rules, however, comes the need to debug them. Despite apparent simplicity, in fact, trigger-action programming is often a complex task for non programmers [17], and errors in trigger-action rules can lead to unpredictable and dangerous behaviors such as a door that is unexpectedly unlocked. One of the most urgent challenges is, therefore, to provide users with tools to avoid possible conflicts [5] and assess the correctness [12] of the developed applications. Unfortunately, even if few recent works started to explore end-user debugging in the IoT [8,26], open questions still remain. Which visual languages are more appropriate for debugging rules? Which information do end users need to understand, identify, and correct errors?

To answer these questions, we firstly conducted a literature analysis by reviewing previous works on end-user debugging in different contexts, with the aim of extracting design guidelines. Then, we used the extracted guidelines to implement *My IoT Puzzle*, a tool to compose and debug IF-THEN rules based on the Jigsaw metaphor. The tool interactively assists users in the composition process by representing triggers and actions as complementary puzzle pieces, and by providing real-time feedback to test *on-the-fly* the correctness of the rule under definition. Puzzle pieces, for example, deteriorate over time according to their usage (Figure 2), while the tool is able to warn users in case of conflicts, namely infinite loops, inconsistencies, and redundancies (Figure 3). Furthermore, the tool empowers end users in resolving problems through textual and graphical explanations. Following the Interrogative Debugging paradigm [18], for instance, the tool is able to answer questions such as *"why it is not working?"*, thus providing the user with a textual explanation of the detected problem (Figure 4). An exploratory study with 6 participants preliminary confirms the effectiveness of our approach. During the study, each participant used *My IoT Puzzle* to compose a set of different IF-THEN rules that generated different conflicts. By collecting quantitative and qualitative measures, we observed that participants appreciated the intuitiveness of the adopted visual languages, including the Jigsaw metaphor. Furthermore, the provided feedback and explanations helped them understand, identify, and correct the conflicts they encountered.

[1] `https://ifttt.com`, last visited on February 26, 2019
[2] `https://zapier.com`, last visited on February 26, 2019

2 Background

Following the explosion of the IoT, in the last 10 years several commercial platforms for end-user personalization such as IFTTT or Zapier were born. The aim of such platforms is to empower end users in customizing the behavior of IoT devices and online services through the trigger-action paradigm, typically. Despite the trigger-action programming expressiveness [2] and popularity [12], the definition of trigger-action rules can be difficult for non-programmers. Platforms like IFTTT have been criticized since they often expose too much functionality [17], and they adopt technology-dependent representation models that force users to have a deep knowledge of all the involved devices and online services [12,9]. As a result, users frequently misinterpret the behavior of IF-THEN rules [4], often deviating from their actual semantics, and are prone to introduce errors [16].

Therefore, one of the most urgent challenges in EUD solutions for personalizing IoT ecosystems is to provide users with tools to avoid possible conflicts [5] and assess the correctness [12] of the developed IF-THEN rules. In this context, work on end-user debugging is still in its early stage. While the majority of previous studies focus on mashup programming [6], spreadsheets [15], and novice developers [18], only a few recent works started addressing the problem of end-user debugging in the IoT. In EUDebug [8], in particular, the authors integrated an end-user debugging tool on top of IFTTT. EUDebug exploits a user interface modeled after IFTTT to warn users when they are defining any troublesome or potentially dangerous behavior. Through a formalism based on Petri Nets and Semantic Web ontologies, EUDebug is able to detect 3 types of problems among trigger-action rules: *loops*, *redundancies*, and *inconsistencies*. In *My IoT Puzzle*, we used the same approach for identifying and displaying problems between the composed rules. A similar tool for end-user debugging in this context is ITAD (Interactive Trigger-Action Debugging) [26]. In addition to warn users in case of rule conflicts, ITAD allows the simulation of trigger-action rules in fixed contexts.

In this work, we stem from both EUDebug and ITAD, i.e., the first two works that investigated how to detect conflicts and simulate IF-THEN rules, for taking a step forward: with *My IoT Puzzle*, our aim is to understand how we can make debug of trigger-action rules more *understandable* by end users.

3 Literature Analysis and Design Guidelines

To reach our goal, we firstly reviewed previous works on end-user debugging in different contexts, with the aim of extracting design guidelines (Table 1). The analysis was guided by the following research questions:

RQ1. Which information, e.g., feedback and explanations, do end users need to understand, identify, and correct errors in trigger-action rules?

RQ2. Which visual languages are more appropriate for debugging trigger-action rules?

3.1 End-User Debugging: How to Avoid and Correct Errors (RQ1)

Debugging is the process of finding the cause of an identified misbehavior and fixing or removing it. Different previous studies, e.g., [27,19], investigated how developers try to fix bugs, and discovered many slow, unproductive strategies. If it is challenging for programmers, the debugging process can become an insurmountable barrier for end users. In different contexts, ranging from spreadsheets [15] to mashup programming [6], studies have demonstrated that end users try to fix problems by following a "debugging into existence" approach [34], i.e., they continuously twist and fiddle their solutions until the failure "miraculously" goes away. Cao et al. [6], however, demonstrated that, if prompted with the right information, end users are also able to *design* applications and programs. In the context of mashup programming, for example, they proposed to add micro-evaluations of local portions of the mashup during the implementation phase, with the aim of reducing the effort of connecting the run-time output with the program's logic itself. We envision similar approaches also for our context, i.e., IF-THEN rules for personalizing IoT devices and online services. By providing real-time feedback during the composition of trigger-action rules, an EUD tool may empower users in frequently testing the correctness of their solutions (**GL1**), thus allowing them to update *on-the-fly* problematic rules (**GL2**), Table 1. This may increases the chances of fixing possible conflicts [6].

Previous studies on end-user debugging also highlight the benefits of providing users with textual and graphical explanations, to represent the run-time behaviors of the defined programs and their possible problems [24,25] (**GL3**). Indeed, Ko et al. [18] discovered that programmers' questions at the time of failure are typically one of two types: *"why did"* questions, which assume the occurrence of an unexpected run-time action, and *"why didn't"* questions, which assume the absence of an expected run-time action [18]. The same authors extended the Alice programming environment [38], a platform for creating interactive 3D virtual worlds, to support a "whyline" that allows users to receive answers concerning program outputs. Their work opened the way for a new paradigm, named *Interrogative Debugging*, that has been adopted in many different works on end-user debugging, ranging from tools to support more experienced developers [20] to interactive machine learning [21]. As preliminary suggested by Manca et al. [26], the Interrogative Debugging paradigm may effectively help end users debug their trigger-action rules (**GL4**). The event-driven nature of trigger-action rules, in particular, naturally leads to questions such as *"why this action have been executed?"* or *"why this event did not trigger?"*

3.2 Visual Languages for End-User Development (RQ2)

Besides the question of identifying which information end users need for debugging trigger-action rules, another important question is which visual languages are more appropriate in this context. Despite visual programming languages strive to simplify the intricate process of programming, in fact, they need to be tailored towards the domains in which they will be used [31]. The most common visual languages adopted in End-User Development tools can be categorized into 3 main categories: a) form-filling, b) block programming, and c) data-flow.

Form-filling visual languages, also known as wizard-based languages, are extensively used in commercial platforms such as IFTTT and Zapier [12]. Also EUDebug [8] and ITAD [26], i.e., the first two works that explore end-user debugging in the context of trigger-action rules, exploit wizard-based interfaces. To compose applications with the form-filling approach, be they rules or other types of programs, the user makes use of menus and fields to be completed. Tools that exploit form-filling visual languages, in particular, guide the user through a predefined, bounded procedure, by reducing the user interaction in completing a series of forms step-by-step. Despite form-filling approaches have been proved to be intuitive and easy to use for simple use cases, their closed form can be perceived as restrictive [30,29].

Another popular approach in End-User Development is block programming. A popular example of the approach can be seen in Scratch [32], a block-based visual programming language targeted primarily at children. With block programming, users can connect blocks of different sizes and shapes by dragging and dropping them on a work area. Differently from form-filling approaches, tools based on block programming are less restrictive, and stimulate the user creativity. One of the most appreciated ways of representing blocks, in particular, is the Jigsaw metaphor. Here, blocks are represented as puzzle pieces that can be combined on the go, thus decreasing the learning curve and motivating users to explore the underlying tool. An application example is Puzzle, a visual environment for opportunistically creating mobile applications in mobile phones [11]. We envision that block programming approaches based on the Jigsaw metaphor could be easily adapted to the composition of IF-THEN rules for IoT personalization (**GL5**).

Finally, the last category of visual languages commonly adopted in EUD is data-flow. Differently from the previous approaches, which were useful for simple use case such as the composition of a single rule, the process-oriented nature of data-flow programming languages makes them one of the best choice to represent complex use cases [3]. Process-oriented notations have been employed to provide increased expressiveness while still retaining easy-to-comprehend visualizations [33,10]. The expressiveness of such notations, however, is often coupled with complex user interfaces [3]. This makes them difficult to be used at composition time, but useful to visualize complex information such as triggers, actions, and their relationships. For this reason, we envision that a data-flow visual language could be adopted for representing the behavior of multiple trigger-action rules (**GL6**), with the aim of helping users understand and identify unwanted run-time behaviors.

4 My IoT Puzzle: Design and Implementation

We integrated the extracted guidelines (Table 1) in *My IoT Puzzle*, our tool for composing and debugging IF-THEN rules. Under the hood, the tool exploits the

Table 1. The design guidelines extracted by reviewing previous works on end-user debugging in different contexts.

Guideline	Description
GL1	A debugging tool for IF-THEN rules should empower users in frequently testing their solutions, e.g., by providing real-time feedback about possible run-time problems the rules may generate.
GL2	During the debugging of trigger-action rules it is important to provide users with tools for updating *on-the-fly* their solutions, e.g., to remove possible errors during the rule composition process.
GL3	In case of problems, a debugging tool for IF-THEN rules should provide users with textual and graphical explanations about the run-time behavior of the defined applications.
GL4	The Interrogative Debugging paradigm, with which users can ask questions like *"why something happens?"*, can be easily adapted to the event-driven nature of trigger-action rules.
GL5	Block programming based on the Jigsaw metaphor is understandable and easily adaptable to the composition of trigger-action rules.
GL6	The data-flow visual language is suitable for representing complex information such as the run-time behavior of a set of trigger-action rules.

EUDebug server [8], thus allowing the composition and the debug of IFTTT rules. Thanks to the RESTful API exposed by the server, the tool is able to detect the following problems among trigger-action rules:

- **Loops** arise when multiple rules are continuously activated without reaching a stable state. 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.
- **Redundancies** arise when rules that are activated at the same time have replicated functionality. 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.

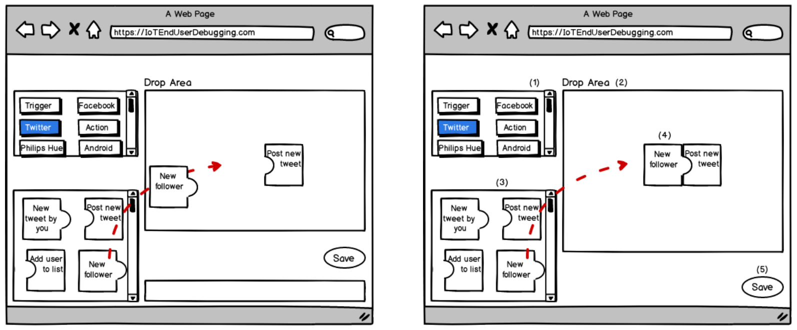
  Here, the three rules are executed at the same time because the first two rules share the same trigger, while the second rule implicitly activates the third rule. They produce two redundant actions, i.e., the first and the third rule post the same content on Twitter.
- **Inconsistencies** arise when rules that are activated at the same time try to execute contradictory actions. 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.

The user interface of *My IoT Puzzle* has been implemented with the the Angular framework[3], by exploiting the jQuery[4] and Bootstrap[5] libraries. The interface iteratively assists end users in composing and debugging IF-THEN in 3 main phases, i.e., *composition*, *problem detection*, and *problem resolution*.

Composition. Trigger-action rules are composed through a block programming approach based on the Jigsaw metaphor (**GL5**). To design the composition metaphor 3 researchers produced and evaluated different mockups.



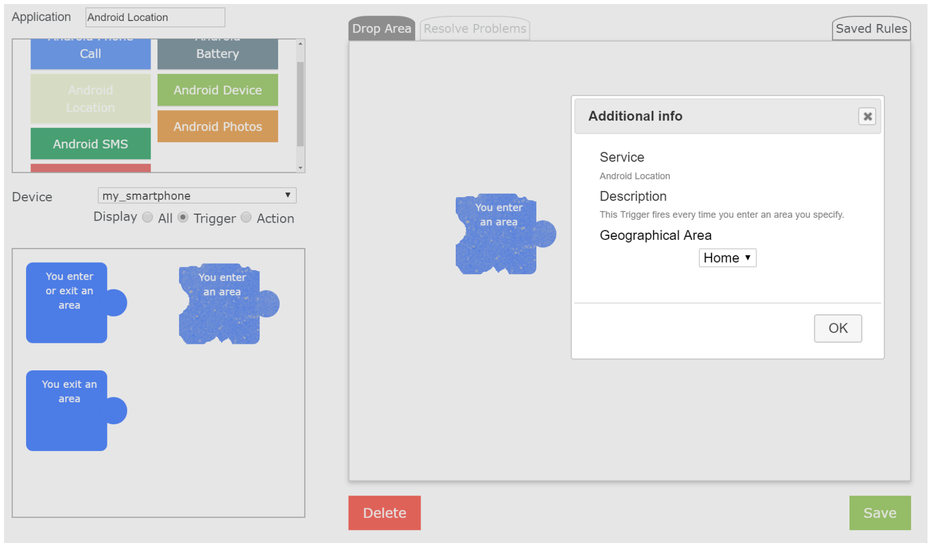
(a) (b)

Fig. 1. Two mockups priduced to design the the composition methaphor.

Figure 1 shows an example of the produced mockups: to avoid complex solutions, we decided to use 2 types of puzzle pieces, only, one for triggers and one for actions. Triggers and actions are therefore represented as complementary puzzle pieces that can be dragged and dropped in a Drop Area.

Figure 2 shows an example of the *composition* phase. Mary selects a device (her *Android* smartphone) on which monitoring an event, and drops a specific

[3] `https://angular.io/`, last visited on February 26, 2019
[4] `https://jquery.com/`, last visited on February 26, 2019
[5] `https://getbootstrap.com/`, last visited on February 26, 2019



Additional info

Service
Android Location
Description
This Trigger fires every time you enter an area you specify.
Geographical Area
Home

OK

Fig. 2. Mary, the user of our example, starts to compose a new rule by dragging a new trigger on the Drop Area. The tool provides Mary with an initial feedback: the piece of puzzle is worn, since it has been already used in other rules.

trigger on the drop area (*"You enter an area"*). Then, she completes the trigger details by specifying the geographical area of her home. The tool uses initial feedback to preliminary allow the user assess her solution (**GL1**). Indeed, due to the complementary nature of the puzzle pieces, some wrong operations are prevented by construction: pieces of the same type, e.g., two trigger pieces, cannot be connected. Furthermore, as shown in Figure 2, the dropped trigger piece is worn, since it has been already used in other rules. In *My IoT Puzzle*, in particular, puzzle pieces deteriorate over time according to their usage history. Using the same trigger in multiple rules, in fact, means that the involved rules will be executed at the same time, thus increasing the chances of introducing conflicts such as redundancies and inconsistencies.

Problem Detection. The *problem detection* phase starts every time that *My IoT Puzzle* detects loops, inconsistencies, or redundancies during the *composition* phase. Figure 3 shows the *problem detection* phase experienced by Mary. Having defined the trigger, she selects her kitchen *Philips Hue* lamp. She connects to the *"You enter an area"* trigger an action (*"Turn off lights"*) that is inconsistent with some previously saved rules. Therefore, the system warns Mary with a red feedback (**GL1**), and allows her to get more information on how to solve the issue.

Problem Resolution. The *problem resolution* phase helps users understand and fix conflicts detected during the previous phases. Figure 4 shows the phase as

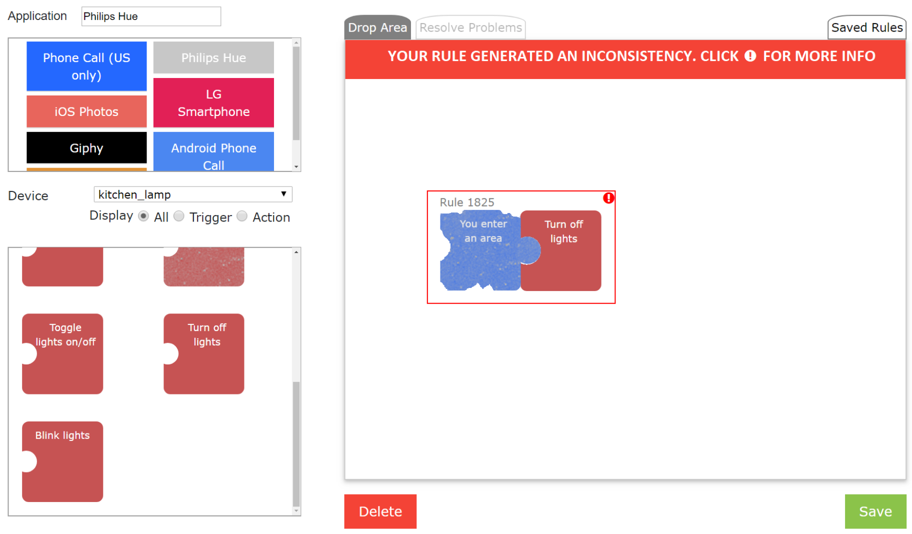


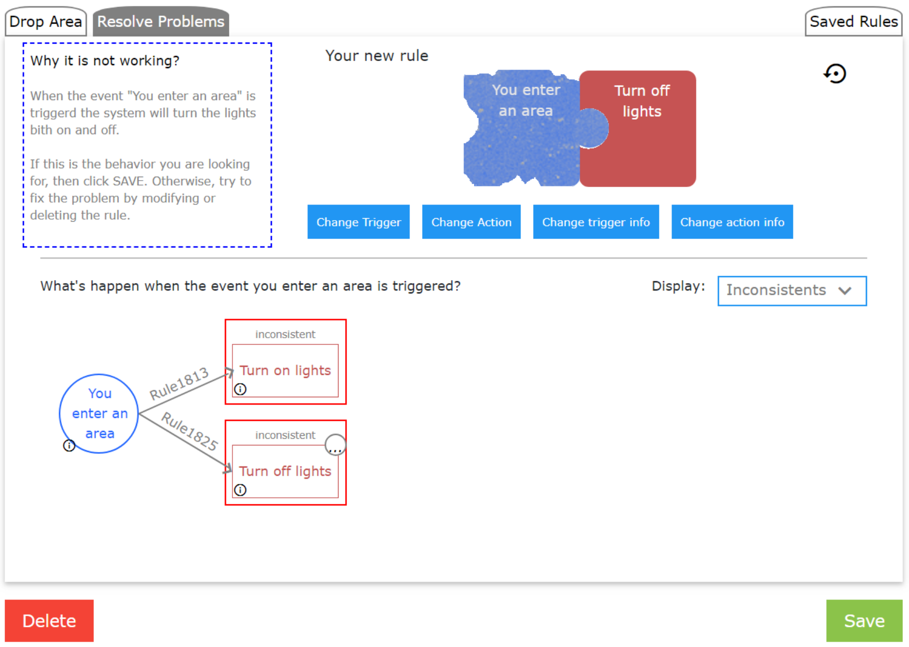
Fig. 3. Mary connects to the trigger an action that is inconsistent with some previously saved rules. The system warns Mary with a red feedback.

experienced by Mary. She can see textual and graphical explanations of the detected inconsistency (**GL3**). For graphically explaining the problem, a data-flow visual language is used (**GL6**). Instead, the textual explanation follow the Interrogative Debugging paradigm (**GL4**), by explicitly describing *why* the problem is happening. Both the textual and graphical explanations, in particular, show that it already exists a saved rule that shares the same trigger, i.e., entering the home geographical area, but with an inconsistent action, i.e., turning on the kitchen *Philips Hue* lamp. Mary has the possibility of updating on-the-fly the problematic rule by changing the trigger, the action, and/or the related details (**GL2**).

5 Evaluation

We preliminary evaluated *My IoT Puzzle* through an exploratory study with 6 participants. Our aim was to assess whether the different features offered by *My IoT Puzzle*, ranging from the provided feedback to the graphical and textual explanations, helped participants correctly understand and fix potential conflicts in trigger-action rules.

Study Procedure. We recruited 6 university students (3 males and 3 females) with a mean age of 21.5 years (SD = 2.88) who had very limited or no experience in computer science and programming: on a Likert scale from 1 (No knowledge at all) to 5 (Expert), participants declared their experience with programming



Why it is not working?

When the event "You enter an area" is triggerd the system will turn the lights bith on and off.

If this is the behavior you are looking for, then click SAVE. Otherwise, try to fix the problem by modifying or deleting the rule.

What's happen when the event you enter an area is triggered?

Fig. 4. By opening the Resolve Problems area, Mary can see textual and graphical explanation of the inconsistency, and she can resolve the problem by changing the rule.

(M = 1.16, SD = 0.40) and with the trigger-action approach (M = 1.00, SD = 0).

We brought each participant to our lab for a 45-minute session using *My IoT Puzzle*. At the beginning of the study, participants were introduced to trigger-action programming and the evaluated tool with an example of a rule composition. We then presented a task involving the composition of 12 rules (*composition* phase). Rules were presented one at a time on a sheet of paper in a counter-balanced order, and were artificially constructed to generate 2 inconsistencies, 2 redundancies, and 1 loop[6]. When *My IoT Puzzle* highlighted some problems (*problem detection* phase), participants were free to decide whether to save, update, or delete the problematic rule (*problem resolution* phase). All the sessions were video recorded for further analysis.

Measures. We quantitatively measured the number of problematic rules that were **saved**, **updated**, or **deleted**. Furthermore, after each highlighted problem, we asked participants to qualitatively provide an **explanation** for their choices. When they decided to update or delete a rule that generated a problem, for

[6] A detailed description of the rules used in the evaluation can be found in the Appendix.

example, they had to demonstrate to understand the problem by retrospectively explaining why the rule generated the issue. At the end of each session, we asked participants to quantitatively evaluate, on a Likert scale from 1 (Not understandable at all) to 5 (Very understandable), the **understandability** of a) the visual languages and feedback used in the *composition* and *problem detection* phases, b) the textual explanations, and c) the graphical explanations in the *problem resolution* phase. Finally, we performed a debriefing session with each participant.

5.1 Results

Table 2 reports the quantitative measures collected during the evaluation. In total, participants saved a rule that generated a problem in a limited number of cases, i.e., 3 out of 30 (10%), thus preliminary demonstrating that *My IoT Puzzle* helped them in identifying problems in trigger-action rules. In 2 cases, the saved rule generated a redundancy, while in the remaining case a participant saved a rule that generated a loop. Participants deleted a rule that generated a problem in 18 cases out of 30 (60%), while they updated and successfully fixed the problem in 9 cases out of (30%). By analyzing the type of the problems, we found that rules that generated loops or redundancies were deleted most of the time (66.66% and 75%, respectively), while rules that generated an inconsistency were more frequently updated (58.33%) than deleted (41.67%). Such results are promising and suggest that feedback and explanations effectively assisted participants in understanding the highlighted problems. Both loops and redundancies, indeed, result in some functionality that are replicated, thus motivating the deletion. Inconsistencies, instead, are typically caused by a mistake over a set of rules with different and specific purposes, thus making the "update" choice the most appropriate.

Table 2. The number of times a rule that generated a problem was deleted, updated, or saved in the study.

	Rules	Deleted	Updated	Saved
Loop	6	4 (66.66%)	1 (16.67%)	1 (16.67%)
Redundancy	12	9 (75%)	1 (8.33%)	2 (16.67%)
Inconsistency	12	5 (41.67%)	7 (58.33%)	0 (0%)
TOTAL	30	18 (60%)	9 (30%)	3 (10%)

We further investigated the results by analyzing the qualitative explanations given by the participants in case of a detected problem. We first tried to understand whether the participants who saved a problematic rule were aware of what would happened in the real world, or whether they simply made a mistake, thus unconsciously introducing a potential conflict at run-time. Both the users that saved a rule that generated a redundancy provided a sound explanation. When 2 rules simultaneously turned on the same lamp with different colors, for example, P1 said *"I don't care about the color, the important thing is that the lamp is turned on."* On the contrary, P3 failed in providing an explanation for saving the rule that generated a loop. She said *"I don't know how to solve the problem. I would save the rule, and then I would try the involved rules in the real world, to see what happens."* For what concerns the problematic rules that were deleted, participants provided a sound explanation in 17 cases out of 18 (94.44%). Only in 1 case a participant discarded a rule without providing any explanation. Finally, participants made a reasonable change in all the rules that were updated, by successfully fixing the problem and by providing a sound explanation.

The promising results arising from the interaction between participants and *My IoT Puzzle* are confirmed by the answers they provided at the end of the study. Participants positively evaluated the understandability of the *composition* and *problem detection* phases (M = 4.50, SD = 0.54), and the understandability of the textual (M = 4.50, SD = 0.81) and graphical (M = 4.50, SD = 0.30) explanations in the *problem resolution* phase. Finally, users provided interesting suggestions to improve *My IoT Puzzle*. P1, for example, focused on the *composition* phase, by suggesting the possibility of composing multiple rules at the same time. P4 and P5, instead, focused on the *problem resolution* phase, and they asked to introduce recommendations and suggestions for updating problematic rules. P4, in particular, said that suggestions such as *"try to replace the trigger X with the trigger Y"* would allow non-expert users to better understand and fix the problem.

Discussion. Results are promising and demonstrate that *My IoT Puzzle* is helpful for correctly identifying and fixing potential problems in trigger-action rules. Results also confirm that the design guidelines presented in Section 3 are valuable. The provided information and the exploited visual languages effectively helped users understand, identify, and correct errors in trigger-action rules.

The Jigsaw metaphor, for example, was appreciated by the participants, and turned to be easy to use and understand: all the participants composed the proposed trigger-action rules without any problem. Also the feedback used in the *composition* phase turned to be useful for preliminary assessing the correctness of trigger-action rules. When using a worn piece of puzzle, for example, P3 said *"now I'm going to make a mistake, I need to stay focused."* Furthermore, if the typical reaction to an highlighted problem was a mix of surprise and uncertainty, the *problem resolution* phase progressively made participants aware of the detected conflict: in most of the cases, the provided feedback and explanations allowed them to successfully fix the problem, either by deleting or updating the rule that generated it. This confirms that the usage of different representations of the same information facilitates users in analyzing problems [35].

6 Conclusions and Future Works

Users are making use of End-User Development tools for trigger-action programming to personalize their IoT devices and online services. Which information do end users need to understand, identify, and correct errors in this context? Which visual languages are more appropriate for debugging trigger-action rules? In this paper, we investigated such questions by presenting *My IoT Puzzle*, a tool to compose and debug IF-THEN rules, based on the Jigsaw metaphor. The tool follows design guidelines extracted from a literature analysis. It interactively assists users in the composition process by representing triggers and actions as complementary puzzle pieces, and it allows end users to debug their rules through different real-time feedback, textual and graphical explanations. Results of an exploratory study with 6 participants preliminary suggest that the adopted visual languages are easy to understand. Furthermore, results show that users can successfully identify and fix conflicts between trigger-action rules with the help of textual and graphical explanations.

Acknowledgments. The authors wish to thanks Alessia Carosella for conducting the literature review and implementing the user interface of *My IoT Puzzle*.

Appendix: Rules Used in the Evaluation

The following trigger-action rules were used in the evaluation of *My IoT Puzzle*:

R1. *If* your Android smartphone detects that you enter the home area, *then* turn on the Philips Hue kitchen lamp.
R2. *If* your Android smartphone detects that you enter the home area, *then* turn off the Philips Hue kitchen lamp.
R3. *If* your Android smartphone detects that you enter the home area, *then* turn on a color loop on the Philips Hue kitchen lamp.
R4. *If* a new photo is added to the “ios” album on iOS Photo, *then* add the file on the “drpb” Dropbox folder.
R5. *If* a new photo is added to the “drpb” Dropbox folder, *then* upload the photo on Facebook.
R6. *If* there is a new photo post by you on Facebook, *then* add the photo to the “ios” album on iOS Photo.
R7. *If* your iPhone detects that you exit the work area, *then* lock the SmartThings office door.
R8. *If* the SmartThings office door is locked, *then* arm the Homeboy office security camera.
R9. *If* the Homeboy office security camera is armed, *then* unlock the SmarThings office door.
R10. *If* a new song is played on Amazon Alexa, *then* post a tweet with the song name on Twitter.
R11. *If* a new song is played on Amazon Alexa, *then* save the track on Spotify.

R12. *If* a new track is saved track on Spotify, *then* post a tweet with the song name on Twitter.

The rules generate 2 inconsistencies, 2 redundancies, and 1 loop:

- R1 and R2 generate an **inconsistency**, because they share the same trigger while producing contradictory actions on the same device;
- R7 and R9 generate an **inconsistency**, because they produce contradictory actions on the same device and are activated nearly at the same time, since R7 activates R8, and R8 activates R9;
- R1 and R3 generate a **redundancy**, because they share the same trigger while producing two similar actions on the same device;
- R10 and R12 generate a **redundancy**, because they produce similar actions on the same online service and are activated nearly at the same time, since R10 and R11 share the same trigger and R11 activates R12.
- R4, R5, and R6 generate an infinite **loop**, because R4 activates R5, R5 activates R6, and R6 activates R4;

References

1. Akiki, P.A., Bandara, A.K., Yu, Y.: Visual simple transformations: Empowering end-users to wire internet of things objects. ACM Transactions on Computer-Human Interaction **24**(2), 10:1–10:43 (Apr 2017)
2. Barricelli, B.R., Valtolina, S.: End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, chap. Designing for End-User Development in the Internet of Things, pp. 9–24. Springer International Publishing, Cham, Germany (2015)
3. Brich, J., Walch, M., Rietzler, M., Weber, M., Schaub, F.: Exploring end user programming needs in home automation. ACM Transaction on Computer-Human Interaction **24**(2), 11:1–11:35 (Apr 2017)
4. Brush, A.B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., Dixon, C.: Home automation in the wild: Challenges and opportunities. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2115–2124. CHI '11, ACM, New York, NY, USA (2011)
5. Caivano, D., Fogli, D., Lanzilotti, R., Piccinno, A., Cassano, F.: Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. Journal of Systems and Software **144**, 295–313 (2018)
6. Cao, J., Rector, K., Park, T.H., Fleming, S.D., Burnett, M., Wiedenbeck, S.: A debugging perspective on end-user mashup programming. In: 2010 IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 149–156 (Sept 2010)
7. Cerf, V., Senges, M.: Taking the Internet to the Next Physical Level. IEEE Computer **49**(2), 80–86 (Feb 2016)
8. Corno, F., De Russis, L., Monge Roffarello, A.: Empowering end users in debugging trigger-action rules. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. CHI '19, ACM, New York, NY, USA (2019), in press
9. Corno, F., De Russis, L., Monge Roffarello, A.: A high-level semantic approach to end-user development in the internet of things. International Journal of Human-Computer Studies **125**, 41 – 54 (2019)

10. Dahl, Y., Svendsen, R.M.: End-user composition interfaces for smart environments: A preliminary study of usability factors. In: Marcus, A. (ed.) Design, User Experience, and Usability. Theory, Methods, Tools and Practice. pp. 118–127. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
11. Danado, J., Paternò, F.: Puzzle: A mobile application development environment using a jigsaw metaphor. Journal of Visual Languages and Computing **25**(4), 297–315 (Aug 2014)
12. Desolda, G., Ardito, C., Matera, M.: Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. ACM Transactions on Computer-Human Interaction **24**(2), 12:1–12:52 (2017)
13. Evans, D.: The Internet of Things: How the Next Evolution of the Internet Is Changing Everything. Tech. rep., Cisco Internet Business Solutions Group (2011)
14. Ghiani, G., Manca, M., Paternò, F., Santoro, C.: Personalization of Context-Dependent Applications Through Trigger-Action Rules. ACM Transactions on Computer-Human Interaction **24**(2), 14:1–14:33 (2017)
15. Grigoreanu, V., Burnett, M., Wiedenbeck, S., Cao, J., Rector, K., Kwan, I.: End-user debugging strategies: A sensemaking perspective. ACM Transaction on Computer-Human Interaction **19**(1), 5:1–5:28 (May 2012)
16. Huang, J., Cakmak, M.: Supporting mental model accuracy in trigger-action programming. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 215–225. UbiComp '15, ACM, New York, NY, USA (2015)
17. Huang, T.H.K., Azaria, A., Bigham, J.P.: 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. pp. 1555–1562. CHI EA '16, ACM, New York, NY, USA (2016)
18. Ko, A.J., Myers, B.A.: Designing the whyline: A debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 151–158. CHI '04, ACM, New York, NY, USA (2004)
19. Ko, A.J., Myers, B.A., Coblenz, M.J., Aung, H.H.: An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Transactions on Software Engineering **32**(12), 971–987 (Dec 2006)
20. Ko, A.J., Myers, B.A.: Finding causes of program output with the java whyline. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 1569–1578. CHI '09, ACM, New York, NY, USA (2009)
21. Kulesza, T., Burnett, M., Wong, W.K., Stumpf, S.: Principles of explanatory debugging to personalize interactive machine learning. In: Proceedings of the 20th International Conference on Intelligent User Interfaces. pp. 126–137. IUI '15, ACM, New York, NY, USA (2015)
22. Lee, J., Garduño, L., Walker, E., Burleson, W.: A tangible programming tool for creation of context-aware applications. In: Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 391–400. UbiComp '13, ACM, New York, NY, USA (2013)
23. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End User Development, chap. End-User Development: An Emerging Paradigm, pp. 1–8. Springer Netherlands, Dordrecht, Netherlands (2006)
24. Lim, B.Y., Dey, A.K.: Toolkit to support intelligibility in context-aware applications. In: Proceedings of the 12th ACM International Conference on Ubiquitous Computing. pp. 13–22. UbiComp '10, ACM, New York, NY, USA (2010)

25. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2119–2128. CHI '09, ACM, New York, NY, USA (2009)
26. Manca, M., Fabio, Paternò, Santoro, C., Corcella, L.: Supporting end-user debugging of trigger-action rules for iot applications. International Journal of Human-Computer Studies **123**, 56 – 69 (2019)
27. Myers, B.A., Ko, A.J., Scaffidi, C., Oney, S., Yoon, Y., Chang, K., Kery, M.B., Li, T.J.J.: Making End User Development More Natural, pp. 1–22. Springer International Publishing, Cham (2017)
28. Namoun, A., Daskalopoulou, A., Mehandjiev, N., Xun, Z.: Exploring mobile end user development: Existing use and design factors. IEEE Transactions on Software Engineering **42**(10), 960–976 (Oct 2016)
29. Reisinger, M., Schrammel, J., Fröhlich, P.: Visual end-user programming in smart homes: Complexity and performance. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 331–332 (Oct 2017)
30. Reisinger, M.R., Schrammel, J., Fröhlich, P.: Visual languages for smart spaces: End-user programming between data-flow and form-filling. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 165–169 (Oct 2017)
31. Repenning, A., Sumner, T.: Agentsheets: A medium for creating domain-oriented visual languages. Computer **28**(3), 17–25 (Mar 1995)
32. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y.: Scratch: Programming for all. Commun. ACM **52**(11), 60–67 (Nov 2009)
33. Rietzler, M., Greim, J., Walch, M., Schaub, F., Wiedersheim, B., Weber, M.: homeblox: Introducing process-driven home automation. In: Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication. pp. 801–808. UbiComp '13 Adjunct, ACM, New York, NY, USA (2013)
34. Rode, J., Rosson, M.B.: Programming at runtime: Requirements and paradigms for nonprogrammer web application development. In: Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments. pp. 23–30. HCC '03, IEEE Computer Society, Washington, DC, USA (2003)
35. Subrahmaniyan, N., Kissinger, C., Rector, K., Inman, D., Kaplan, J., Beckwith, L., Burnett, M.: Explaining debugging strategies to end-user programmers. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 127–136. VLHCC '07, IEEE Computer Society, Washington, DC, USA (2007)
36. Ur, B., Pak Yong Ho, M., Brawner, S., Lee, J., Mennicken, S., Picard, N., Schulze, D., Littman, M.L.: 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. pp. 3227–3231. CHI '16, ACM, New York, NY, USA (2016)
37. Ur, B., McManus, E., Pak Yong Ho, M., Littman, M.L.: Practical trigger-action programming in the smart home. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 803–812. CHI '14, ACM, New York, NY, USA (2014)
38. User Interface Group, U.: Alice: Rapid prototyping for virtual reality. IEEE Computer Graphics and Applications **15**(3), 8–11 (May 1995)