Supporting End-User Debugging of Trigger-Action Rules for IoT Applications

Marco Manca, Fabio Paternò, Carmen Santoro, Luca Corcella
CNR-ISTI, HIIS Laboratory
Pisa, Italy
{marco.manca, fabio.paterno, carmen.santoro, luca.corcella}@isti.cnr.it

ABSTRACT
End users need tools to enable them to control and personalise Internet of Things (IoT) applications, which may involve hundreds of interconnected objects. Trigger-action programming has shown to be a useful support for this purpose because it allows users to easily associate dynamic events with the activation of desired effects. End User Development (EUD) tools aim to allow even users without programming experience to define the behaviour of IoT applications. However, users may define rules triggering various actions that may be in conflict, may specify rules that do not result in the intended behaviour or may define rules which will never be applied. Although such situations can often occur, there seems to be a lack of tools able to help users understand whether the specified rules actually bring about the desired behaviour and, if not, the reasons why they fail. We present an original solution for filling this gap, which takes into account the specific aspects of trigger-action rules. We describe the design and implementation of this debugging support, and then discuss the results of a first user test.

Author Keywords
End User Development; Internet of Things; Trigger-Action Rules; Debugging.

INTRODUCTION

In recent years we have witnessed the advent of the Internet of Things (Atzori et al., 2010), which has also emphasised the need to design applications able to react to events that can be generated from dynamic combinations of a variety of sensors, objects, services, devices, and people. In the Internet of Things (IoT) vision, 'smart' physical objects are networked together, able to interact and communicate with each other, with human beings and/or with the environment, to exchange data and information 'sensed' about the environment. Furthermore, they are able to react autonomously to events in the physical world, and influence it by running processes that trigger actions and perform services. According to Gartner[1], there will be nearly 26 billion devices on the Internet of Things by 2020. In this perspective it becomes important to empower end users to configure future smart environments consisting of hundreds or thousands of interconnected devices and objects, which will enable many possible interactions in a user's surroundings. In End-User Development (EUD) (Lieberman et al., 2006) the goal is to allow non-professional developers to create or modify their applications so that they can better meet their diverse and frequently changing needs. One of the approaches considered in this area is the use of rule-based systems. Some first examples were AgentSheets (Repenning, 1995) and the KidSim/Cocoa/StageCast work (Smith et al., 1994). In recent years, there has been increasing interest in using trigger-action rules for supporting EUD

[1] https://www.gartner.com/newsroom/id/2636073, Last accessed in May 2018

of IoT applications at both the commercial (e.g. Tasker[2], IFTTT[3]) and research level. This type of approach seems to be increasingly adopted. For example, IFTTT has more than 320,000 automation scripts (called "applets") offered by more than 400 service providers. The applets have been installed more than 20 million times, and more than half of IFTTT services are IoT device-related (Mi et al., 2017). In this trend one important aspect related to EUD is how people can test and possibly assess whether the modified or created behaviour of the application actually results in the expected one. This need is especially relevant in IoT domains, such as elderly assistance or the home, where incorrect behaviour of applications or actuators can even have safety-critical consequences. This issue is relevant both in single-user and in multi-user scenarios. For instance, conflicting rules can occur with a single user who might not realise that some rules can conflict under specific circumstances. In other situations, multiple users might define rules attempting to influence the status of devices or physical objects belonging to the same environment in a conflicting manner, following opposing preferences. In both cases a debugging tool could be beneficial to highlight potentially conflicting rules or show the reasons why a rule will not be triggered in specific situations. So, although this kind of support could represent an important aid for improving the correctness of applications and facilitating the appropriation (Pipek, 2005) of such tools by end users, most EUD environments do not include any debugging tool.

In rule-based environments it is possible to have conflicts when multiple rules are verified in the same time interval and their actions have contradictory effects. One of the easiest ways to resolve rule conflicts is to associate a 'priority' value to each rule, an attribute that is set when the rule is built. Once a conflict happens, the rule with the highest priority value will be triggered. If there are two verified rules having the same priority, further policies should be put in place to solve the conflict. Thus, the conflict resolution strategy based only on priority values is the simplest one but works well only if users are able to set suitable priority values at design time, which is not always easy as they might not be able to predict in advance the conflicts that will occur at runtime. In addition, the strategy of assigning priority values that are all different does not seem feasible when dealing with large sets of rules, as it would require a significant overhead on the user's side. Thus, more effective resolution strategies should be put in place, able to identify potential problems in rule execution and help find possible solutions, also involving users in the process.

In this perspective, another key point is how people can test the correctness of rules and possibly identify errors in them, e.g. triggers or actions that they might have forgotten (or inappropriately added) in the current rule specification. There are two main aspects when dealing with correctness: syntactical and semantical correctness. The first one is generally easier to ensure than the second. Thus, here we mainly focus on the semantic correctness of rules, i.e. when rules are syntactically correct, but do not behave in the expected manner. A way to reduce the likelihood of errors in the specification of rules is to allow users to *simulate* the occurrences of conditions and events, and analyse whether the relevant rules are triggered in such context. An alternative to the simulation is to *apply* them in the current (real) context of use, to understand whether they will be executed, and then identify the corresponding actions that will be carried out. However, this approach it is not immediate (e.g. think about rules involving date or time triggers) and it provides only a limited set of context options for the verification of the rules.

While testing aims at detecting possible errors in the programs built by users, debugging is the process of finding the cause of the identified behaviour errors and fixing/removing them. In particular, here we present a tool able to support a number of functionalities (simulation, interactive explanations, conflict identification/resolution), by taking into account the Interrogative Debugging paradigm (Ko and Myers,

[2] http://tasker.dinglisch.net, Last accessed in May 2018

[3] https://ifttt.com/, Last accessed in May 2018

2004), in which the system directly answers “why” and “why not” questions. In this regard, we have also considered previous studies on the use of “why” and “why not” explanations to improve the intelligibility of context-aware intelligent systems (Lim and Dey, 2010).

In this paper, after discussing related work, we provide some background information and an example scenario. Next, the approach proposed to debugging trigger-action rules is presented. We also report on a user test that shows how the introduction of such support allows users to more easily find bugs and fix them more accurately with respect to current practises. Lastly, we draw some conclusions and provide indications for future work.

RELATED WORK

We build on prior work on end user development of internet of things applications, intelligibility of context-aware systems, interactive debugging.

End-User Development of Internet of Things Applications

EUD is an emerging field that has stimulated research in various directions (Blackwell, 2002; Dax et al., 2015; Kubitza and Schmidt, 2015; Perera et al., 2015; Paternò, 2013; Pipek and Wulf, 2009; Sutcliffe and Papamargaritis, 2014; Tetteroo et al., 2015). Rule-based solutions have been considered in order to obtain EUD approaches for IoT applications. For example, various apps for customizing the behaviour of existing applications in mobile devices or Web services have been introduced, such as Atooma, Tasker, IFTTT. The underlying idea is to specify the system behaviour by using a number of if-then statements expressing how the system should behave when specific situations occur. At the research level, one of the first proposals using rules for EUD was iCAP (Dey et al., 2006), which introduced the possibility to create if-then rules to support personalization of dynamic access to home appliances. Recently, due to the importance of contextual dynamic aspects that can potentially affect the behaviour of IoT applications, rule-based approaches are receiving increasing interest. This is because end users can easily reason about context and express in rules the desired behaviour of their applications by describing how the application should react to specific events occurring in specific contexts. However, trigger-action rule-based approaches can become difficult for non-programmer users (Huang and Cakmak, 2015) because they could raise some ambiguity in their interpretation due to potential discrepancies in end users’ mental models, or, in case of complex rules, because correct formulation of logical expressions implies knowledge of some key concepts (e.g. Boolean operators, priority of operators, composition of triggers) that may not always be intuitive for non-professional developers. In addition, proposals adopting this approach are generally limited in their ability to compose events and corresponding actions: some approaches do not even support their composition at all (as happens with IFTTT), whereas end users may need to specify them (Ur et al., 2014). Therefore, further effort in enabling end users to describe complex expressions of triggers and actions should be pursued because this would provide users with the possibility to specify more flexible behaviour. TARE (Ghiani et al., 2017) provides results that are still far from satisfying all the needs of domain experts and end users in the many possible scenarios of use. In addition, especially when dealing with complex expressions of triggers (e.g. events and conditions) and actions, there are further aspects that need to be better analysed. According to (Yarosh and Zave, 2017) currently the task of reasoning and resolving ‘feature interaction’ (i.e. conflicting functions occurring in complex systems with multiple devices), is entirely left on end users and remains challenging for most of them even with a feature interaction resolution mechanism. Misunderstandings involving rules for IoT scenarios can cause undesired behaviours such as unlocking doors at the wrong time, or unintended energy waste. This requires further analysis and investigation of the various types of triggers and actions that can be included in complex expressions, so as to avoid such interpretation issues in future EUD tools. For this purpose, some authors have considered the 5W model (Desolda et al., 2017) seeking to answer five

questions: (1) Who did it? (2) What happened? (3) When did it take place? (4) Where did it take place? (5) Why did it happen? The problem of intuitive composition of logical expressions by end users has also been studied in (Metaxas and Markopoulos, 2017), where an established theory of mental models has been used to guide the design of interfaces for end user programming. According to such mental model theory, people find it easier to conceptualize logical statements as a disjunction of conjunctions (an OR of ANDs), as opposed to other logically equivalent forms. Thus, Metaxas and Markopoulos presented a tool with the aim of facilitating end-users in programming context-dependent behaviour using quite complex logical expressions. Corno et al. (2017) consider the use of ontologies to obtain high-level descriptions of the context of use and specify corresponding trigger-action rules. Although such approaches represent a useful contribution to end user programming by decreasing the cognitive load associated with the specification of complex logical expressions, we are still far from finding general solutions for these key aspects. In addition, in this area only a few contributions to some extent addressed the issue of end user debugging. The TARE editor (Ghiani et al., 2017) included a simulator that provided just a basic support and it was not empirically assessed. In addition, TARE completely lacked any debugging support and did not provide users with aid for conflict identification and resolution. AppsGate (Coutaz and Crowley, 2016) included dependency graphs representing devices and programs, their status and dependencies. Unfortunately, these types of representations have some scalability issues and in realistic cases soon become difficult to interpret due to the numerous icons and arrows they contain.

Intelligibility of context-aware systems

While context-dependent adaptation can be a powerful tool to obtain flexible solutions, it is important that users can understand how such solutions work and are able to effectively control them. Bellotti and Edwards (2001) discuss the need for intelligibility and accountability. Context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it. In addition, context-aware systems must enforce user accountability when, based on their inferences about the context, they seek to mediate user actions that impact others. Indeed, the incorporation of context-awareness raises a number of issues. For example, users are required to trust the behaviour of the system's intelligence and this requires the system to exhibit predictable behaviour and the ability to successfully and consistently achieve user's goals. Unfortunately, even an intelligent application may incorrectly support such user's goals, owing to either flawed intelligence or to incorrect contextual information (Cheverst et al., 2001). In such circumstances the user is likely to feel frustrated because the application will either appear overly prescriptive or, worse still, present incorrect adaptations. Since context-aware intelligent systems employ implicit inputs, and make decisions based on complex rules that are rarely clear to the users, Lim et al. (2009) carried out a user study to investigate how to mitigate this through automatically providing explanations answering why, why not, what if, and how to questions. They found that explanations describing why the system behaved in a certain way resulted in better understanding and stronger feelings of trust. From such studies they derived a toolkit specific for generating explanations for decision models (Lim and Dey, 2010), while in this work we consider a different domain (execution of trigger-action rules).

Interactive debugging

One of the most difficult tasks in End User Development is debugging, i.e. finding the code portion causing an unwanted behaviour in a program. Previous studies investigated how developers try to fix bugs, and discovered many slow, unproductive strategies (Myers et al., 2017, Ko and Myers, 2005; Ko et al., 2006). They found that people without programming experience tend to read their code and change things they thought might be wrong. This often introduced new defects, rather than resolving the original ones. More experienced developers use breakpoints to step through a program's execution, looking for where it deviated from the expected behaviour. Such studies led to the idea to provide tools able to answer the

typical 'why' and 'why not' questions. The first application (Ko and Myers, 2004) extended Alice (Pausch et al., 1995), an environment for creating interactive 3D virtual worlds, and was supported through a 'whyline' allowing users to receive answers concerning program outputs. The 'whyline' provides a simple graphical representation of the elementary programming steps necessary to reach a desired effect. Further studies investigated the possibility of applying this approach also to help more experienced developers debug Java programs (Ko and Myers, 2009). This approach is interesting but needs to be appropriately revised and extended in order to address trigger-action rules for context-dependent applications. The following presents a contribution to resolving such issue. Kulesza et al. (2011) use the Whyline oriented approach to ask questions about how an intelligent assistant made its prediction; it provides answers to these 'why' questions by showing the assistant's logic to look at or modify it; all the explanations are provided through bar charts visualizations. It asks questions about the intelligent assistant behaviour at run-time, while our approach supports rules debugging with respect to a simulated context indicated by the users. In the evolution of that work (Kulesza et al., 2015) propose an approach called Explanatory Debugging in which the system explains to users how it made the predictions, and the user then defines the changes to the learning system in order to obtain better predictions. This approach is based on two important concepts: *explainability* which responds to the 'why' question and *correctability* which responds to the 'what if' question. The performed tests show that participants, using the Explanatory Debugging approach, understand how the learning system operate about 50% better than the other participants. This approach is applied to the decisions and predictions taken by a machine learning system; thus end-users debug the decisions taken by an automatic system, while we want to apply similar concepts to debugging the resulting behaviour obtained through a set of trigger-actions rules defined by end-users. Moreover, in addition to the *'why'* and *'what if'* questions, we also want to address the '*why not*' question explaining to the users why their rule will not be triggered given the current simulated context values.

BACKGROUND

For clarity and self-containment goals, here we clarify some terms that will be used in the following sections to describe the proposed approach:

- A trigger-action **rule** describes a behaviour that automatically performs actions provided w specific triggers occur. A rule is composed of two parts:
    - a part dedicated to specifying trigger(s), which defines the event(s) and/or the condition(s) that activate the execution of the rule
    - and a part dedicated to specifying action(s) to carry out when the rule is verified.
- A rule is **verified** when any event(s) occur and/or any condition(s) are satisfied in its trigger part.
- A rule is **executed** when it is verified and the action part is performed in a specific context.
- An **event** is a change of state of some contextual aspect.
- A **condition** is a state of a specific contextual aspect that lasts for a period of time.
- A **trigger** can be either an event or a condition or their composition through Boolean operators (AND/OR).
- An **action** specifies what to carry out and multiple actions are executed sequentially.

A **context model** provides a description of the main aspects that characterise a context. The context model considered in this work has a hierarchical structure. At the highest level the context model is specified in terms of four **context dimensions,** which are: User, Environment, Technology, Social, while at the lowest level of this hierarchy we find **context entities**. A context model can be *generic* (i.e. includes entities that

are domain –independent) or specific (which is obtained by refinement of the generic model in such a way to include the entities that are specific for a particular context of use).

In this approach, triggers refer to events or conditions related to aspects described in the considered context model. The four main contextual dimensions consider different aspects: *User* describes some user characteristics e.g. personal data, physiological and mental state, user activity; *Environment* defines the attributes that characterize the settings where the user acts; *Technology* considers the state and characteristics of any devices, smart sensors, smart objects and appliances available in the context; *Social* concerns relationships in terms of social networks and related aspects (relationships, memberships, events).

This model is used for defining the possible triggers that can be specified through the Rule Editor. The considered context model can be edited and saved in XSD (XML Schema Definition)-based format, and it can be imported by the Rule Editor (Figure 1), which is then able to propose triggers according to the imported context model structure. This tool also provides a classification of the actions that specify the possible personalisation. Five action categories were identified: changes of the state of the appliances/smart objects/devices; changes of the aspect or content in the user interface of the target application; activation of functionalities (e.g. show the energy consumption or query the weather forecast); alarms to signal risky situations; reminders to bring users' attention to some action that should be taken. It is worth noting that such actions can be tailored according to the target application since it is the application that will apply them.

When we want to create an instance of the Rule Editor the first step is to refine the context model (which determines the hierarchies of triggers) and the actions for the target context of use and applications. This work is carried out involving relevant stakeholders. For instance, when developing the context model for some trials to carry out in an Ambient Assisted Living project, we had an interactive discussion with expert caregivers working in a Norwegian rehabilitation organisation who helped us to better identify the contextual elements to use in the trigger hierarchy. In carrying out such work we also have to consider the information that can actually be received by the available sensors and applications.

Triggers include both events and conditions, which are distinguished through the use of different keywords in the rule editing ("IF/IS" for the condition, "WHEN/BECOMES" for the events). Their different meanings can be explained by considering the following rules: IF time is after 5 pm DO ring a bell; WHEN time becomes after 5 pm DO ring a bell. Apparently these two rules seem conceptually equivalent. However, they describe different behaviours. The first rule includes a condition, and the action should be performed as long as the condition is verified: in this case the bell should ring continuously after 5 pm. The second rule defines an event, which is associated with a state change of a contextual aspect (in this case the time), thus it indicates that the bell should ring only when the time changes its state from 5.00 to 5.01.

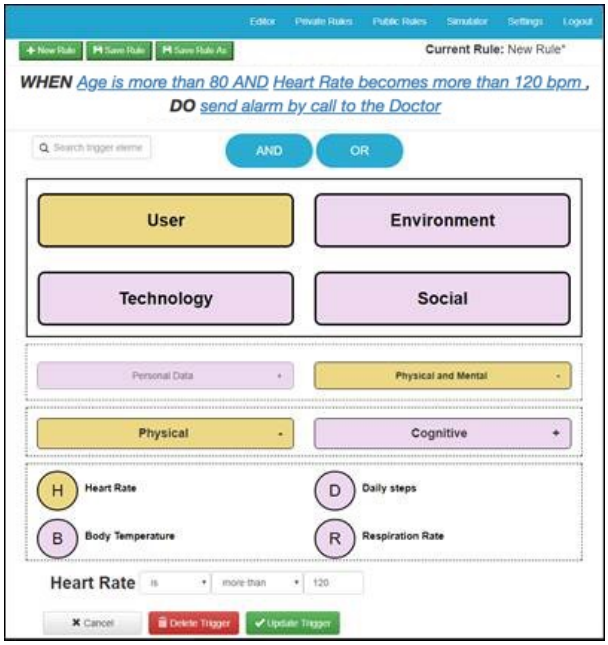

Figure 1: Rule Editor interface

The rules created through the rule editor are saved in JSON format. Figure 2 shows an example of the format used to store a rule: the first three rows represent respectively the author, the rule name, and the priority field, which is an optional field that can be used to select the rule to execute when multiple rules are triggered simultaneously. Since a trigger can be composed of multiple elements, the 'triggers' field in Figure 2 (row 5) is implemented as an array of triggers. Each element of such array is described by: an entity belonging to one of the four main logical context dimensions, its parent element in the context hierarchy, the context element representing the trigger, a field indicating if the trigger is a condition or an event (see row 13 in Figure 2), the trigger operator (more, less, equal, etc.) and the current value associated with the concerned contextual element. The trigger specification also indicates the Boolean operators used when composing events and conditions (see e.g. the rule example shown in Figure 1), if any.

Since there can be multiple actions in a rule, the "actions" field is also associated with an array. An action is described by the corresponding category (see the attribute "type" in Figure 2), a reference to the target element (in Figure 2 the action reference is defined in row 20), an operator (e.g. turn on/off, display/hide element, open, close, etc.), its actual name, as well as its parent element in the hierarchy of actions.

```
{
    "author": "anonymous",
    "ruleName": "rule1",
    "priority": 0,
    "triggers": [{
        "dimension": "user",
        "parent": "PersonalData",
        "element": {
            "realName": "Age",
            "type": "int",
            "xPath": "User/PersonalData/@age"
        },
        "isEvent": true,
        "operator": "more",
        "value": "18"
    }],
    "actions": [{
        "root": "ChangeApplianceState",
        "action": {
            "reference": "appliances/kitchen/@light",
            "realName": "light",
            "type": "custom:applianceState"
        },
        "operator": "turnOn",
        "parent": "Kitchen"
    }]
}
```


Figure 2: Rule format example

The target element of an action can be an appliance, a user interface element, the name of an external function. External functions are accessible services through specific calls. When the Rule Editor is configured it is possible to define the external services that can be accessed through the rules, and their parameters. Thus, when the user selects this type of action, the configured external services are listed, it is possible to select one of them (for example weather forecast), then the action editor shows the corresponding parameters (for example town and date) and the corresponding call to the service is stored as the action of the rule.

Previous work (Ghiani et al., 2017) was able to provide a simulator returning only the list of verified rules in a given context, which is not sufficient to understand why a rule is (or is not) verified. Since already in the evaluation reported in that paper several users had difficulties in correctly understanding the distinction between events and conditions, we have designed and developed a new tool called ITAD (Interactive Trigger-Action Debugging), which supports the Whyline approach for trigger-action rules. Thus, users can interactively ask “why” and “why not” questions to know the reason why a rule is (respectively: is not) verified. Moreover, this novel tool integrates a conflict analysis functionality able to highlight potential conflicts between rules that are simultaneously verified. In addition, we have also validated the new support in order to assess its effectiveness.

AN EXAMPLE SCENARIO

In order to show how the proposed solution works we can consider a scenario where a user, John, wants to more automatically manage the behaviour of his smart house, so that it can better fit the needs of his family (his wife and two children). To this purpose, John starts to formulate a first rule (Rule 1) saying that all the windows in the house should be open from 9 a.m. in the morning, in order to have some fresh air in the house. Afterwards, John defines another rule to ensure that all the windows will be safely closed in the evening after 9 p.m. (Rule 2). After saving these rules John runs the simulator, setting the value of 10 p.m. as the time of interest to be considered in the simulated context, and checks whether the two rules are verified or not. ITAD will notify John about the existence of a possible conflict between the two rules: when John activates the conflict analysis feature, the tool warns him that the windows of the smart home

could enter an undefined state after 9 p.m., because at that time the windows should simultaneously be open (according to Rule 1) and closed (according to Rule 2). Thus, thanks to this information, John is able to identify the issue, and then modifies Rule1 by adding another condition to better specify the time interval in which the windows of the house should be open, namely between 9 a.m. and 1 p.m.. In addition to the previously defined rules, John also wants to have all the windows closed when it is raining: then he adds another rule (Rule 3) to his rule private repository. After doing this, the conflict analysis tool will notify John that there could be a potential conflict during rainy mornings since, according to Rule 1 the windows should be open, according to Rule 3 the windows should be closed. Thus, thanks to the help of the system, John modifies Rule1 by adding a further condition saying that the windows should be open during the morning only during sunny days.

John adds further rules to manage other aspects of his smart home in a context-dependent manner. For instance, he adds a rule for switching off the TV after a specific hour in the evening (to prevent children from going late to bed). Then, he could add another rule for sending the output of the house surveillance camera to his mobile phone when children are alone at home and some motion is detected near the house. He also defines additional alarms and reminders. Among the various rules created over time, he has added a rule for a weekly analysis of the history of the energy consumption in the house during the past week. In particular, he would like to have this report shown on the TV of the living room each Saturday, when he generally has time to relax. To this aim, John adds the following rule (Rule 4): "WHEN Date becomes Saturday AND IF User is seated in front of living room TV, DO Show Energy Consumption on living room TV". After having added this rule to his repository, John decides checking whether the smart home would actually behave as expected. However, in that simulated context the rule would not be verified, therefore the tool highlights in red the associated rule. By selecting the ITAD 'why not' button, the simulator notifies about an inconsistency between the behaviour specified in Rule 4 (which in the current state would activate the report format the transition between Friday and Saturday, as expressed by the *event* "WHEN Date becomes Saturday" in the trigger), and the simulated context (which specifies the *condition* "Date IS Saturday", and then it would simulate a context in which the day is already Saturday). After changing the following trigger in the rule specification: "WHEN Date becomes Saturday" into "IF date is Saturday", John checks again the obtained rule through the tool simulator, which now shows that the rule is fully verified in the simulated context.

DEBUGGING TRIGGER-ACTION RULES

By observing users in various trials specifying trigger-action rules, and considering previous work in the area (e.g. Huang and Cakmak, 2015; Ghiani et al., 2017), we identified some features that an environment to enable users to define context-dependent personalization rules and manage their execution should satisfy:

- it should be able to perform the rules execution in specific contexts of use simulated through an interactive user interface;
- it should be able to provide indications about why or why not a rule can be triggered in a given context of use;
- it should be able to identify rules conflicting at design time. The environment should statically analyse the defined rules and identify which of them are in conflict (see rule 1 and rule 2 in Scenario section);
- it should be able to identify rules which may conflict depending on the values that their triggers can assume at run-time. Rules such as "IF it is raining, the windows should be closed" and "IF time is after 9 am the windows should be open" are in conflict only when both the conditions are verified (it is raining and it is after 9 am).

Context of Use Simulation

The goal is to allow end users without programming experience to be able to debug the execution of personalised trigger-action rules. In such rules an important issue is that it can be difficult to test whether the specified rules will behave as desired in the real context because often triggers depend on events and conditions that are not currently verified in the real environment (for example, a rule expressing that lights should be off at night cannot be verified if the current time is daytime). For this purpose, we found it important to support the possibility of *simulating* the state of the context of use by allowing users to assign values to various contextual aspects of interest (e.g. time, location, technology used), and then the tool will automatically check whether the indicated rules would be triggered for the specified values. The simulator presents the hierarchical structure of the context model in a tree-like manner (see Figure 3, left panel). After activating the simulator, the tool will automatically unfold just the context dimensions and entities involved in the triggers included in the rules for which the simulator has been activated, while the other context dimensions will be hidden: this has been done to limit the cognitive effort of users, who do not have to search within the context model for the elements involved in the defined triggers.

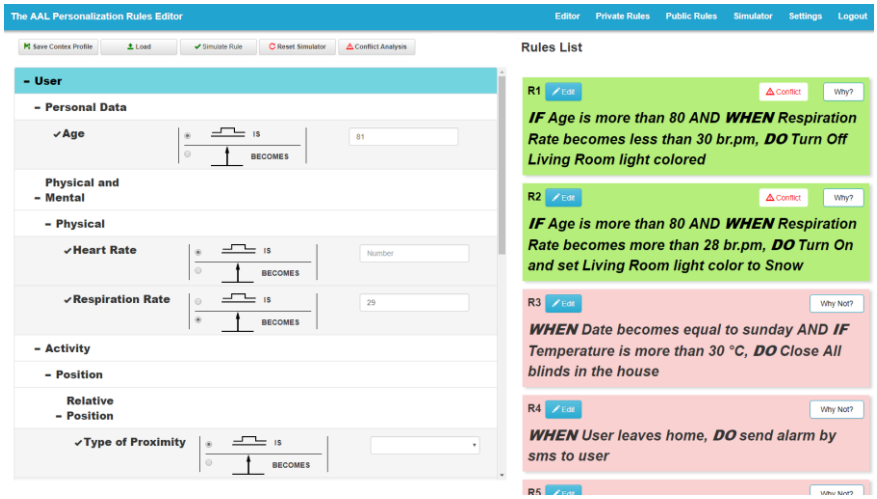

Figure 3: The environment for executing rules in a simulated context of use

Figure 3 shows the simulator user interface: on the left part there is the (editable) instantiation of the context of use used in the simulation, while in the right part there are the rules indicated by the user. In the considered example, the context instantiation used in the simulation uses the following values: "**Age is 81"** (condition) and "**Respiration Rate becomes 29"** (event). When the user selects the *Simulate Rule* button, the tool checks whether the defined rules can be triggered when the state of the context is the one indicated on the left side. The results are shown by changing the background colour of the verified rules to green, while the colour of the rules that are not verified becomes pink. Moreover, as a result of the simulation some buttons are displayed in each rule container: the "Edit" button allows users to activate the editing of a rule directly from the simulator; the "Why" (resp.: "Why not") button is displayed only if

a rule is verified (resp.: not verified): the "Why" button activates the highlighting of the contextual elements that are verified, the "Why not" button points out why the rule is not verified (Figure 4). Lastly, the "Conflict" button is displayed only if the rule is verified, it conflicts with other rule(s) and the priority of the verified rules is the same: when the user clicks on this button, this will invoke the conflict detection functionality which produces an output as the one showed in Figure 7: in particular, it does not show all the conflicts but only the ones that involve the considered rule.

In addition to the trigger values, the simulator also allows users to specify whether the value inserted for each contextual entity should be considered as associated with an event or a condition. To make this distinction clearer, the radio button representing the two possible choices has been augmented by two icons visually representing the difference between events and conditions. Differently from previous work, ITAD also helps users better understand the difference between events and conditions. Indeed, on the one hand a rule is presented as verified only if the user specifies the right option between event and condition and provides the correct values to use in the simulation. On the other hand, when a rule is not verified, in the explanation associated with 'why not' the tool highlights in red the elements that do not match, to inform the user about the reason why the rule is not verified. Figure 4 shows how the interface changes after selecting the "why not" button: on the right panel only the selected rule is displayed, while on the left panel only the context elements related to that rule are shown and expanded in the tree representation of the context model. The verified triggers are highlighted in green (both on the context tree and in the natural language description shown in the right-hand panel of the window), while parts of the trigger that are not verified will have a pink background.

As said, if a rule is not verified in a simulated context it could be for two reasons: either the values used in the simulation are not able to activate the trigger contained in the rule; or there is an inconsistency between the event or condition contained in the rule and those specified in the simulator; or a combination of the two situations. An example of the latter situation is shown in Figure 4. On the one hand the natural language description of the rule specifies an *event* involving respiration rate (see in the right panel of Figure 4: "WHEN Respiration rate BECOMES less than 30 bpm"). On the other hand, in the simulator (see left panel of Figure 4) not only a respiration rate value that does not verify the rule trigger is used (namely: 31), but that trigger is considered as a condition (in the left panel of Figure 4 the "IS" option of the radio button is selected), while in the rule specification it is modelled as an event. Thus, the simulator highlights these inconsistencies through a red border around the concerned elements in the left panel of the simulator (see Figure 4), and using a red text associated with the parts of the rule specification that are not verified, to indicate the reasons why the current rule fails.

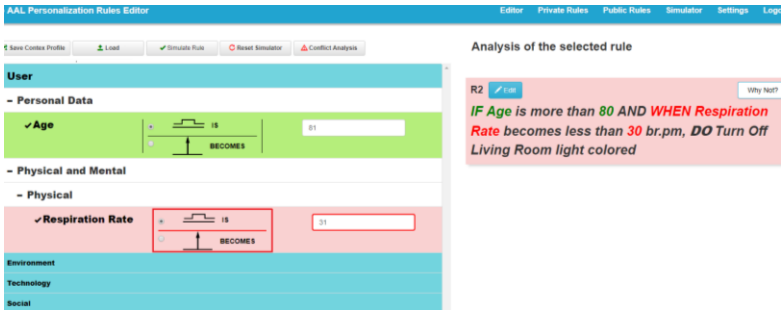

Figure 4: *Why Not* Example

Conflict Analysis

The conflict analysis functionality can be executed independently from the context of use simulation. The algorithm for detecting potential conflicts between rules has been defined taking into account the specific structure of the language describing the rules. As shown in Figure 5, in the rule specification language the structure of the actions part contains two important fields: reference (row 4 and row 15) and operator (row 8 and row 19).

```
1  "actions": [{
2      "root": "ChangeApplianceState",
3      "action": {
4          "reference": "appliances/kitchen/@radio",
5          "realName": "radio",
6          "type": "custom:applianceState"
7      },
8      "operator": "turnOn",
9      "parent": "Kitchen"
10 }]
11
12 "actions": [{
13     "root": "ChangeApplianceState",
14     "action": {
15         "reference": "appliances/kitchen/@radio",
16         "realName": "radio",
17         "type": "custom:applianceState"
18     },
19     "operator": "turnOff",
20     "parent": "Kitchen"
21 }]
```

Figure 5: An Example of Action Specification

The *reference* field identifies the target appliance or devices whose state will be modified when the rule is triggered; while the *operator* field describes the state changes that will be applied on the device (or appliance) identified by the reference field. For each such device there is a predefined and limited set of possible values (e.g. turn on/turn off, open/close, show/hide), thus the algorithm can easily identify any actions which can potentially conflict because they set different state changes.

Also the reference field considers a limited and predefined set of devices or appliances (which are indicated in the current context model), and it is also possible to identify a containment relationship among them. Indeed, the actions defined through the Rule Editor can refer to a set of devices/appliances grouped according to a specific category (e.g. all the lights, all the electric plugs, all the air conditioners, etc.). In this case it is possible to define a hierarchical containment relationship according to which an action involving *All lights* implicitly includes the *Living Room Light*. Thus, two actions (one referring to All lights and the other only to the Living Room light) may conflict if the specified state changes are different.

To summarise, the conflict detection algorithm first analyses the actions defined in all the rules, because the elements which may cause the conflict are those defined in the actions. Analysing the actions list, it derives the ones which refer to the same target element (e.g. the light in the kitchen) or those that are connected by a containment relationship (e.g. Living Room Light and All Lights). Then, from the list of actions identified in the previous step, the algorithm selects those that can potentially be in conflict due to the occurrence of conflicting state changes.

We say ‘potentially’ since the algorithm still has to verify if the rules can be triggered at the same time. Then, the algorithm considers the rules corresponding to the previously selected actions, and for each rule it extracts the associated triggers in order to understand if there actually is a range of values for which they are all verified.

```
"triggers": [{
    "dimension": "environment",
    "element": {
        "realName": "time",
        "type": "xs:time",
        "xPath": "environment/@time"
    }
    "operator": "before",
    "value": "12:00",
}]

"triggers": [{
    "dimension": "environment",
    "element": {
        "realName": "time",
        "type": "xs:time",
        "xPath": "environment/@time"
    }
    "operator": "after",
    "value": "10:00",
}]
```

Figure 6: Triggers example

Thus, the condition for the actual occurrence of a conflict is the simultaneous occurrence of multiple triggers belonging to different rules; these rules are composed of actions which have an effect on the same device or onto the same 'hierarchy' of devices. For instance, if we consider the rules: "between 9 pm and 5 am the living room light should be on" and "at 6 am turn off the living room light", these two rules refer to the same appliance (the living room light) and the actions are potentially conflicting (turn-on, turn-off). However, if we analyse the associated triggers, we can see that the rules cannot be executed at the same time because there is no overlap between the values which verify the triggers, thus they are not in conflict. Thus, the algorithm has to analyse three different situations:

- Triggers related to the same context entity but are verified for different, non-overlapping values, so there is no conflict. For example, when the first trigger is verified before 12:00 and the second one is verified when time is after 12:30;
- Triggers that are related to the same context entity but are verified for different values, partially overlapping. For example, a first trigger is verified if temperature is more than 12^{o} C degrees while the second one is verified if temperature is less than 19^{o} C degrees. In this case the algorithm identifies the overlapping between the values which trigger the execution of the rule and then it generates a sentence in natural language which describes why the rules conflict. The sentence is generated by taking into account the operators involved in the triggers definition:
    1) if the operators are both "greater than" it considers the highest value (trigger 1 verified if temperature > 12 and trigger 2 verified if temperature is > 15; then the rules are in conflict if temperature > 15)
    2) if the operators are both "less than" it considers the smallest value (trigger 1 verified if temperature < 12 and trigger 2 verified if temperature is < 15; then the rules are in conflict if temperature < 12)
    3) if the operators are different it considers the range of values in common between the two triggers (trigger 1 verified if temperature > 12 and trigger 2 verified if temperature is < 15 then the rules are in conflict if temperature is between 12 and 15).

- Triggers related to different context entities; in this case the rules conflict when all the triggers involved in the rule are verified. For example, rules can be: “if time is after 12 open the windows”, and “if it is raining close the windows”. Although the context entities are different (time and weather condition), the rules are conflicting when both triggers are verified (e.g. when time is after 12 and it is raining).

Figure 7 shows an example of the output produced by the tool. The rules that may conflict are 3:

rule1: IF Time is after 9, DO Open All window in the house

rule2: IF Time is after 21, DO Close All window in the house

rule3: IF Weather Condition is rainy, DO Close All windows in the house

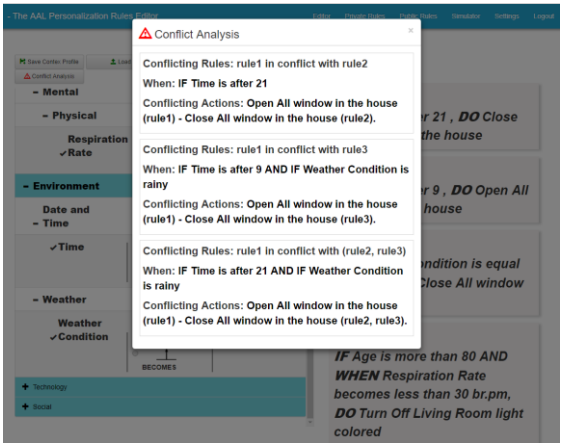

Figure 7: Conflict Detection and Analysis Tool

As we can see three conflicts are detected in the example. The results of the algorithm execution show three important pieces of information: the name of the conflicting rules; under which conditions they conflict; the actions which cause the conflict. In the example the first and the second rules conflict when time is after 21. The first and the third rules presented before conflict when time is after 9 and it is rainy. The first rule is in conflict with the second and the third ones when time is after 21 and it is raining.

It is worth pointing out that the tool shows the rules that are in conflict also taking into account possible priority values that have been associated with rules at design time. In this case, if there is a conflict on two rules having different priority values, the tool does not signal any conflict (since the conflict was actually resolved using the priorities); on the other hand, if there is a conflict on two or more rules having the same priority, it shows them, so that the user can properly act to resolve them.

USER TEST

We have carried out a user test to understand to what extent our approach for analysing and debugging trigger-action rules can support non-programmer users. In particular, we wanted to investigate whether ITAD was able to support users to identify and correct mistakes in their trigger-action rules more accurately than without its support.

Participants

20 participants (13 females) with age ranging between 23 and 61 (mean=34.3, median=32.5, std. dev.=9.9) were involved in the user study. They were recruited by sending an email message to various lists of the authors' research Institute. In particular, people mainly working in administrative roles, as well as their friends were recruited for the test. The main criterion for including people in the test was the fact that they must not be professional developers. As for the education of participants, 4 users held a High School degree, 7 a Master Degree, 2 a PhD and 7 a Bachelor. Their knowledge of programming languages was categorised in five different levels from no knowledge to very good knowledge: 1: No knowledge at all in programming; 2: Low Knowledge (which means: knowledge of HTML, CSS, and basic knowledge of JavaScript); 3: Medium Knowledge (knowledge of JavaScript, basic knowledge of either PHP or Java or C++); 4: Good Knowledge (good knowledge of either PHP or Java or C++); 5: Very Good Knowledge (knowledge of development languages at a professional level). Mean = 2.25, dev.st = 1.07, median = 2.5. The majority of participants had never used any customization tool for smart environments: five users were familiar with IFTTT, one user with Atooma. As compensation for their time and effort, participants received either a USB flash pen or a small backpack (estimated value: around 10 USD).

Test organization

The test was done in laboratory. A moderator observed the participants interacting with the authoring tool (one at a time), annotating any issue, remark, and question they had. The test was organised in four phases: introduction and motivations, familiarisation, test execution, questionnaire.

In the first phase participants received a brief introduction to the study, illustrating its main goals and motivations. Then, they were also provided with a description of the authoring tool and the trigger-action rule structure, as well as a brief explanation of the features of the tool, especially the ones supporting conflict analysis and rule debugging. In addition, we provided users with a short video illustrating the authoring tool capabilities and some example interactions for building trigger-action rules. Participants were strongly encouraged to analyse such introductory information remotely i.e. before the test, in order not to unnecessarily prolong the duration of the study. However, the information material (the slides and the video) was still available to them during the test, so that they could access it anytime, just in case.

In the second phase people familiarised with the tool. Participants could interact with it for some time, to understand its use and features. In this phase users were first asked to freely provide a rule they would define for their home and specify it using the tool. After that, the moderator provided users with a specific behaviour written in natural language, which they had to model using the tool: "Whenever the heart bpm (beats per minute) goes beyond 100 and the user is stationary, send to the caregiver an alarm call".

The next phase was devoted to the actual execution of the tasks planned for the test. Users were provided with the description of the considered scenario (smart home) and a list of ten rules, which were created beforehand. Users were first invited to quickly analyse them and then, taking into account such rules, they were asked to carry out three tasks. The tasks were aimed at checking to what extent users can solve some issues in the rules under two different conditions: *by using the tool* and *by not using the tool* ("control" case). It is worth noting that with 'issues' in the rules we mean possible inconsistencies between what was specified in the provided set of rule(s) and the behaviour that was required to describe. In each task the facilitator provided users with a natural language description of the desired behaviour, which they should specify using the trigger-action approach, possibly modifying one or more rules already created. In particular, users had to check if in the considered set of rules there was already one or more rules that correctly ad completely specified the expected behaviour (we prepared the set of rules in such a way that it was never the case). Then (in the 'control' case), they first had to write down (in natural language) the result of their analysis, identifying the rule(s) to edit and the kind of changes to do on one or more rules

for specifying the expected behaviour. In the other case they had to edit/fix the concerned rule(s) by exploiting the simulator functionality and the why/why not buttons developed in the tool. More in detail, the tasks given in the test were:

Task1: for this task, the rules given modelled the expected behaviour in a way that was more restrictive than the one requested, which was "As soon as the user enters in the kitchen at evening, the kitchen light will automatically switch on". Associated with this task, in the list of predefined rules, there were two rules, which users should modify to accomplish the task:

Rule1.1: WHEN User enters inside kitchen AND IF TIME is after than 24, DO Turn On Kitchen light

Rule 1.2: WHEN User enters inside kitchen AND IF Time is after 5 a.m., DO Turn On Kitchen light, Open Kitchen blind

Task2: For this task, the behaviour to specify was: "Whenever on Sunday the temperature goes beyond 30 degrees, close all the blinds". Associated with this task, users should modify one of the following two rules:

Rule 2.1: WHEN Date is Sunday AND IF Temperature is more than 30 °C, DO Open All blinds in the house

Rule 2.2: IF Date is Sunday AND Temperature is more than 30 °C, DO Turn On Kitchen light

Task3: In this task users had to model a behaviour which required editing two rules, since it expressed a "if-then-else" behaviour (which needs two rules). The behaviour to model was: "Whenever the respiration rate of an over-80 person goes beyond 28 breaths per minute (bpm), switch on the white light in the living room, otherwise switch off that light". Associated with this task we created two rules, which were also in conflict (for the interval 25-30 bpm, see below):

Rule 3.1: IF Age is more than 80 AND WHEN Respiration Rate is less than 30 br.pm, DO Turn Off Living Room light

Rule 3.2: IF Age is more than 80 AND WHEN Respiration Rate is more than 25 br.pm, DO Turn On Living Room light

Thus, for this task, not only users had to specify the desired behaviour, but we were interested to understand whether they were able to identify the existing conflict amongst the two previously specified rules. In addition, we included some other rule just for "noise" goals. They were the following ones:

IF Age is more than 80 AND Respiration Rate is less than 12 br.pm, DO Turn On Bedroom fan, send a reminder by sms

WHEN User leaves home, DO send alarm by sms to user

IF the user has low vision DO increase the character font of 20%

WHEN User leaves home, DO send reminder by mail

The tasks submitted to users were identified in such a way to cover some typical 'errors' that could occur in specifying rules, also varying the level of 'trickiness' of such errors. The first two tasks included 'errors' mainly associated with the fact that the involved type of triggers were right but not expressing the intended behaviour, either because the provided rule was too restrictive compared to the expected rule (Task1), or because (Task2) the provided rule incorrectly specified an event through a condition or vice versa. Task3 was dedicated to checking user's understanding of issues associated with conflicts existing in rules. The

execution order in which the three tasks were carried out was the same for all the users because the complexity of the three tasks was judged as increasing. For instance, the third task was judged as the most difficult one compared to the other ones, since users had to modify two rules to solve the task (remove conflicts). In addition, we could not counter-balance the two conditions (without vs. with the tool), since the use of the tool in the "with the tool" condition could have revealed the solution for the corresponding task in the "without the tool" condition (e.g. for the task associated with conflicts, the tool shows the existing conflicts). We provided the three desired behaviours descriptions in natural language in such a way that the formulation did not suggest any particular rule specification construct (e.g. by avoiding using IF and WHEN keywords which respectively indicate conditions and events).

In the last phase users filled in an online questionnaire indicating their expertise in programming, whether they had previously used any system for building trigger-action rules, and some personal data (age, gender, education). They also rated, on a 1-7 Likert scale, some aspects of the proposed environment and provided observations on positive/negative aspects they noticed on the assessed system and recommendations for its possible improvements.

Results

The data used for the analysis were different in the two conditions. In the without-tool condition, for each task, the users had to write down (and save) in a textual document the explanation of the errors/issues they found in the rules and e.g. how they would have corrected the rules consequently. Such explanations were afterwards analysed by the test moderator to assess whether the task could be classified as correct or not. In the with-tool condition, users exploited the features of the tool for solving the submitted tasks (e.g. properly editing and then saving the rules). Thus, the rules edited by the participants were directly available in the tool, and were used by the moderator to assess the correctness of the corresponding task.

Task1

Without the tool: 17 users out of 20 successfully identified the needed changes without using the tool. Regarding the remaining 3: Two of them did not report the need of modifying any rule. The remaining one wrongly reported that the change to do on Rule 1.1 would have been to delete "IF TIME is more than 24".

With the tool: Only one user did not realise the need of changing any rule. The other users correctly specified the rule by using the tool.

Task2

Without the tool: Only two users were able to recognize the modifications needed for successfully completing this task. For the remaining ones: two users left completely unchanged the set of rules, the other users wrongly edited one of the two rules R2.1 or R2.2 (all users changed the action part, none of them correctly changed the trigger part although the exercise would have been correctly solved by changing in one rule an event into a conditions and the opposite in the other provided rule).

With the tool: Using the tool, 8 users successfully edited the rules to express the desired behaviour.

Task3

Without tool: Only 8 users out of 20 realised that there were conflicts in the rules and correctly explained the modifications to do to specify the expected behaviour.

With tool: Using the tool, all users had the possibility to directly see the conflicts and, thanks to this support, all of them correctly edited the rules using the tool.

Figure 8 shows a summary of the results. By analysing more in depth the data resulting from this task we observed that people with higher programming knowledge were more successful in doing this task: this was not surprising due to their likely familiarity with the concept of conflicts.

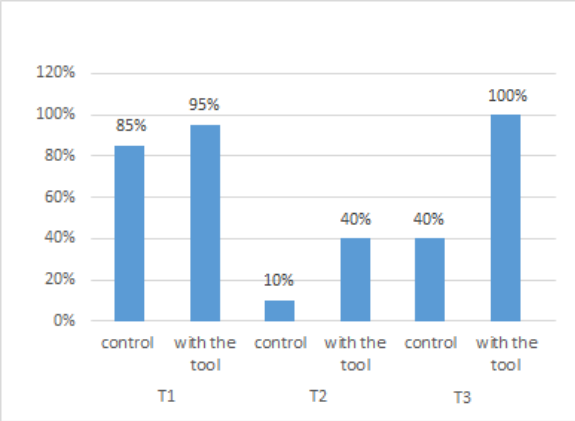

Figure 8: Percentage of successful users over the three tasks: not using the tool ('control' case) and using it

We have not considered time completion because it does not seem meaningful in this case, since the type of support in the two conditions was not comparable in terms of time, and we were more interested in the actual ability to find mistakes in the rules. Indeed, while in the 'control' case users were asked to just write down an explanation in natural language of what they would change in the provided rules and how, in the other case users had to exploit the features of the tool to edit, simulate, analyse and debug rules until they were satisfied with the results.

Questionnaire

Figure 9 shows a stacked bar chart reporting a summary of the results gathered from responses to the questionnaire submitted to participants. The aspects in the following were rated on a 1-7 Likert scale. The lowest and highest scores were associated to judgement labels depending on the question (e.g.: 1 = low usability, 7 = high usability; 1 = low usefulness, 7 = high usefulness).

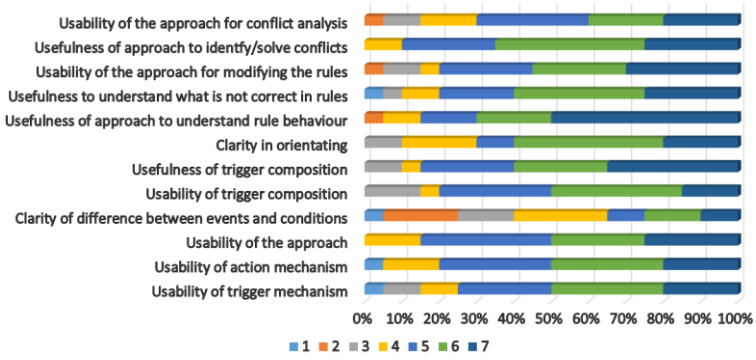

Figure 9: Stacked bar chart showing questionnaire results

From the post-task feedback shown in Figure 9, we can see that participants provided positive feedback regarding the tool, and found it easy to perform the tasks. Across all questions, the median ratings were at or above 5 on a 7-point Likert-scale (7 = best). Only for the clarity of difference between events and conditions it was slightly lower than 5 (it was 4).

Moreover, participants also answered a series of open-ended questions, whose results detail in the following.

Do you have suggestion to improve the approach usability?

Five users suggested better emphasising the difference between events and conditions, especially in terms of keywords used (one user suggested replacing "becomes" with "changes to"). One user recommended simplifying some terms used in the tool (e.g. "triggers") and in the hierarchies of triggers/actions, which can be a bit too technical for the unprofessional user. The same user found the tool too text-based, and suggested exploiting more visual elements for better explaining the tool functionalities. Another suggestion was to improve the choice of colour for accessibility goals. A user suggested using wizards for decreasing the cognitive effort needed, due to the fact in the tool UI there are many elements shown at the same time.

Do you have any suggestion for better highlighting the difference between events and conditions?

One user suggested providing a set of examples better explaining the difference within the tool, so that users can easily refer to them. Another user suggested explicitly using "event" and "condition" keywords to better highlight the difference. Two users suggested using tooltips associated with related UI controls for better illustrating the different meaning of the two concepts.

Do you have any comments about the usability of the mechanism for composing triggers?

The majority of users declared not having any particular difficulty in composing triggers using the tool.

Do you have any comments about the utility of the mechanism for composing triggers?

The vast majority of users found it very useful for modelling situations in which complex expressions of events and conditions have to be specified. One user complained about the absence of an "if-then-else" mechanism for expressing more directly this type of behaviour.

Commentato [C1]: non c'entra molto con la composizione dei trigger

When it was unclear where you were in rule editing process?

Three users found the editing process a bit difficult because the editing process is mainly controlled through the natural language string expressing the rule, which is always visible in the UI. As such, they found it difficult to understand when exactly it was possible to edit the rules. One user declared to have problems in going back to the start. One user suggested adding a UI navigation element of type "Where I am/What I still have to do" to better support users in orienting themselves, the same user also suggested visualizing the level of progress in completing a rule. *Do you have any comments on the utility of "why"/ "why not" to understand the behaviour expressed in the rules?*

The vast majority of users found both of them (especially the "why not") very useful to understand the behaviour of rules. One user was unsure about the utility of the button associated with the "Why". Another one, while acknowledging the utility of the support, suggested making it more usable and quick, since in the current version an additional step (click on the associated "why"/"why not" buttons) is needed to access the visualisation of the detected inconsistencies.

Do you have comments on the utility of "why"/ "why not" to understand what is not correctly expressed in a rule?

The vast majority of users agreed about the utility of the supported analysis functionalities. However, as in the previous case, one user suggested avoiding the use of buttons associated with why/why not functionalities as they make longer the interaction: the suggestion was to highlight the errors directly within the string visualising the rule in natural language. However, some users said that the tool does not sufficiently help understand that the error can be in the rule specification rather than in the values of the simulated context.

Do you have comments on the usability of the mechanism for editing the rules just analysed with the tool support?

Overall, the support was judged usable. However, some users suggested making the editing directly available within the simulator itself, without going back to the edit panel.

Do you have any comments about the utility of the mechanism for identifying and resolving conflicts?

All the users judged the support very useful, enabling users to identify quickly the rules that are involved in conflicts, which, in realistic cases, with tens of rules active at the same time, can be rather difficult to identify without any help.

Do you have any comments about the usability of the mechanism for identifying and resolving conflicts?

Several users found the support as improvable in terms of usability. Some of them complained about the way conflicts are currently shown, one user suggested rendering them in a more structured way (by e.g. enabling users to order, categorise, filter them, etc.). Other users suggested adding the possibility to directly edit rules in the window showing conflicts, without necessarily going back to the edit panel.

Do you have any further suggestions to improve the tool?

The most frequent suggestion was to better emphasise the difference between events and conditions. Users also suggested better rendering the point where two or more rules are in conflict (i.e. with a more proper feedback). One user suggested better visualising the state the user has currently reached (especially in the editing phase), and, to decrease user's cognitive effort, give more visibility to the options that are actually relevant in the current state and less visibility to others.

Improvements after the User Test

The results of the user test also motivated us to perform some improvements to the tool. One of them was to add a further explanation of the information provided by the debugging support, to better explain to the

user the reasons why rules are not verified after performing a simulation. As you can see from Figure 10 below, the tool now is also able to show in a textual manner the reasons why the rules are not satisfied, so allowing the users to more easily fix the identified errors. This was done in order to better drive the user's attention on the reasons of the inconsistencies detected by the tool, which can be either triggered by the set of context values considered in the current simulation (which does not activate the verification of the rule), or by the current specification of the considered rule(s) (which does not correctly describe the intended behaviour). Depending on where the problem is, the user can more easily identify accordingly what should be modified. In particular, for each trigger appearing in a rule there are three main aspects that should be verified simultaneously to have a rule verified in a simulated context: (a) the contextual entity referred in the trigger should also appear in the simulated context; (b) there should be some intersection/overlap between the value(s) involved in the specification of the trigger within the rule and the values associated with the same contextual entity in the simulated context; (c) the concerned contextual entity should be referred within an event (or within a condition) both in the rule specification and in the simulated context. Depending on whether one (or more than one) constraint is not verified, corresponding explanation(s) will be generated by the tool accordingly. For instance, in Figure 10, the lack of verification of t R2 in the considered simulated context produces an explanation of type b associated with the contextual entity "Age" and two other explanations (the first of type b and the second of type c) associated with the contextual entity "Respiration rate".

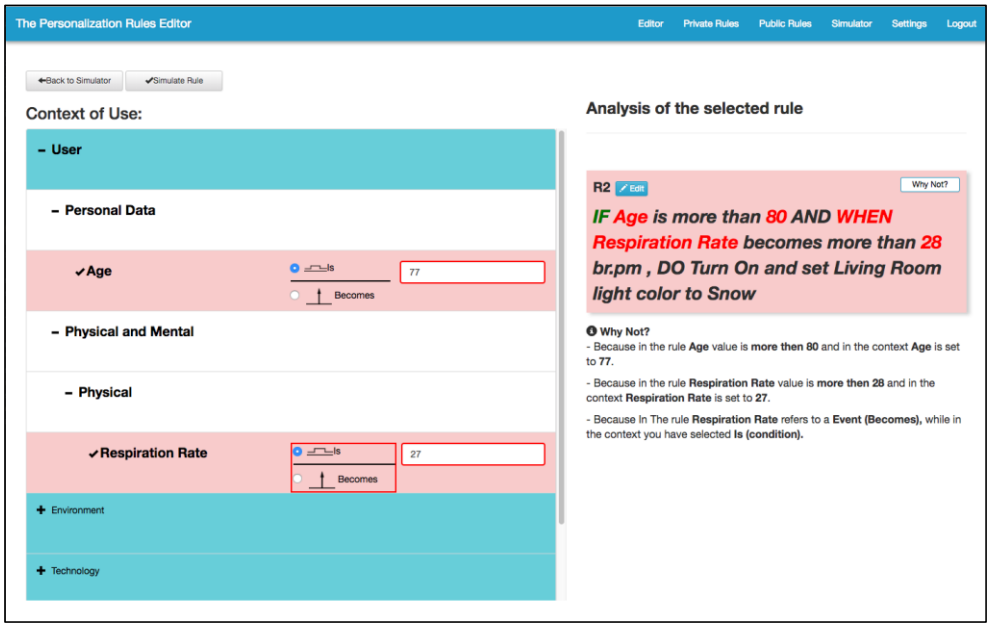

Figure 10: The tool also provides a textual explanation of why the rule is not verified

Discussion

From the results of the user study we can derive some more general implications for developing debugging mechanisms for Trigger-Action rules in IoT scenarios.

Perception of the differences between events and conditions

Previous studies (Huang and Cakmak, 2015; Ghiani et al., 2017) indicate that several users have difficulties in interpreting the distinction between events and conditions. This is an important factor affecting the correctness of the rules created. On the contrary, increasing the complexity of the rules in terms of the number of triggers and actions does not seem to impact the rule creation process much. This probably happens because the distinction between these two concepts is better understood by people accustomed to manipulating concepts in a formal way, which is not often the case for people without programming experience.

Limited user understanding of the difference between events and conditions is an issue that has also emerged in this study. Indeed, users made many more errors without tool support in the task where they had to find erroneous use of events and conditions. In this study we noticed some users' improvements in understanding such difference thanks to the support provided. The use of multiple, related cues for highlighting this aspect (providing the specific keywords "is/becomes" and "if/when" for characterising each concept , and even different icons to distinguish them) seems to help users better reflect on (and understand) them from the rule definition phase to rule simulation and debugging, thus diminishing the possibility of introducing incorrect behaviour in the resulting system. In addition, in the debugging support a rule is presented as verified only if the user specifies the right option between event and condition, and, when it is not verified, the tool highlights in red the elements that do not match in the explanation of why it is not verified.

Provide users ways to help them build suitable mental models of the system

Even a perfectly fine-tuned smart environment needs to evolve with changing user requirements, which can be expressed through personalization rules. In such a situation, users can have difficulties in forming an accurate model of a constantly evolving smart world because: on the one hand each rule has a specific purpose and offers only a partial view of the system behaviour in a specific case, on the other hand the behaviour of a collection of rules is difficult to grasp and therefore it is difficult to envision a global picture of the resulting IoT environment behaviour. Therefore, we need to provide users with tools to help them to build such an accurate mental model of the overall behaviour. One way to manage this tough problem is to allow users to limit this complexity by focusing onto specific simulated scenarios (identifying specific states of the context) of interest for them. To this end, we provide users with tools for simulating contexts and rule behaviour over specific logical/semantic values (i.e. without involving raw sensor data) in order to enable them to get an overview of the overall behaviour for the states that are particularly relevant to them.

Provide automatic support to detect and resolve inconsistencies and potentially conflicting interactions among rules.

Although trigger-action rules are based on a paradigm which, on a general basis, is quite immediate and easy to understand and use, when the number of rules increases in size, their combined behaviour is difficult to control. This is quite a common case if we consider the ease (on the user's side) of adding new rules to the system and the fact that users are likely to use EUD systems for rather long periods. As already highlighted, the use of what-if simulations in specific contexts can help users to better follow system behaviour in key scenarios. However, as it is a user-controlled simulation, it cannot ensure a full coverage of all the possible scenarios that the designed system can be confronted with, but likely only the scenarios which are most interesting/relevant for the user (in that specific moment). Therefore, users should be provided with additional automatic support able to ensure full coverage in the identification of unforeseen

interactions and conflicts between rules, which can easily increase beyond what users (even professional ones) can comfortably understand, while still allowing them to be in control.

Providing examples and counterexamples (e.g. through why and whynot), explanations and actionable feedback

As users live with their automations, they will likely experience surprising behaviour in a real context, prompting questions about why a certain (unwanted) behaviour occurred (or not) in a specific situation. In other cases, users would also just understand the rules that would be verified (or not) in a particular simulated context, and unfortunately sometimes users' expectations will not exactly match the foreseen system behaviour. When this happens (there is a mismatch in a real or in a simulated context between users' expectations and actual system behaviour), non-expert IoT users should be provided with tools that clarify the underlying rationale and logic of the system. Indeed, since rules collections are unstructured, it may be difficult for users to know which rule should be acted on in order to fix the current issue, especially if the program is quite complex (i.e. a large set of rules). In order to effectively localize issues in the currently specified behaviour, users should be supported by explanations, better if in natural language, of why or why not the rules can/cannot be correctly executed, possibly accompanied by concrete examples (or counterexamples) highlighting the situations in which a specific rule is verified or not.

Offer multiple cues to improve user understanding of how the system works

In order to facilitate end user comprehension of the IoT environment, we should provide users with multiple views and perspectives of the same information, by using multiple, different representations, as well as easily understandable visual cues. For instance, in this tool we support a context-based simulation in order to provide users with an additional perspective of the system behaviour as expressed by the rule set. In addition, when the simulation is started, the tool automatically hides the context variables that are not relevant for the verification of current rules, i.e. those aspects that are not involved in the rules' execution, so as to simplify the user model of the overall IoT environment. Easily perceivable colours, with a clear meaning are used to provide at a glance the parts of the rules that need further attention: red for the non-verified rules (or parts of rules) and green for the ones that are verified. In addition, each rule is specified through an interactive graphical UI, but it is also represented through a natural language expression in order to better speak the user's language. Moreover, the difference between events and conditions is rendered not only by the use of different keywords in the natural language specification, but also further emphasised by specific icons that are aimed at visually rendering such a key difference. By using all such related cues users should be able to see the connections between the different data and better understand the rules' execution behaviour, as well as increase their confidence in understanding how the rules editor works.

Limitations

The user study reported was useful and interesting to understand the usability of the debugging support and its potential. It was a laboratory study, and it would be interesting to consider in the future longitudinal studies assessing its use for long periods of time to investigate whether some further aspects emerge.

The study has been applied to a specific solution for supporting trigger-action programming. However, the approaches that exist at both research and commercial level in this area share some basic underlying concepts, even if they represent and support them differently. Thus we think that the results presented here can be generalised with limited effort to other trigger-action approaches.

Currently, the editing and debugging support are provided in two different parts of the environment, while it may be interesting to introduce the possibility of providing debugging support while editing as well.

CONCLUSIONS

The trigger-action approach is emerging as a useful programming paradigm to allow end user to personalise their IoT applications. Various tools have been proposed to support the development of such rules, but little attention has been dedicated to supporting their debugging, which is one of the most challenging aspects in programming, especially for non-professional developers.

This paper presents a novel solution to help end users understand whether the specified trigger-action rules behave as desired and without conflicts. It provides answers to common why/why not questions concerning rules execution in specific context states that can be interactively defined, as well as conflict analysis functionalities.

We also report on a user study that has provided positive feedback in terms of ability to detect conflicts and the user satisfaction expressed through a post-test questionnaire. The user study also yielded some suggestions for minor changes, some of which have been considered in a new tool version.

The results of the user test are encouraging, they indicate that debugging support of trigger-action programming is useful and possible even for non-professional developers, and also has the useful side effect that more users may become designers and developers and learn something through its adoption. While in the literature there are various studies concerning the actual possibility that rules written by non-professional developers do not indicate the desired behaviour, there is a lack of investigation of how to support them in order to prevent undesired effects. Our proposal aims to address this issue, and also shows that context simulation supported with interactive why and why not answers can help them to quickly check the actual behaviour resulting from their rules. This is also accompanied by the automatic support for conflict detection, which is another important aspect to consider when multiple rules are specified. Overall, although it does not cover all the potential issues, the type of solution proposed is useful for researchers and practitioners of IoT environments that can integrate it in order to improve their adoption.

Future work will be dedicated to addressing related issues such as loops between rules, further empirical validation, and introducing additional features, such as more interactive checking of rules directly during their editing.

REFERENCES

Atzori, L., Iera, A., and Morabito, G., 2010. The Internet of Things: A survey. Computer Networks, Volume 54, Issue 15, 28 October 2010, Pages 2787–2805. doi:10.1016/j.comnet.2010.05.010

Blackwell, A.F., 2002. First steps in programming: a rationale for attention investment models," IEEE Symposia on Human-Centric Computing Languages and Environments, pp. 2-10.

Bellotti, V, and Edwards, W.K., 2001. Intelligibility and Accountability: Human Considerations in Context-Aware Systems, Human-Computer Interaction, 16(2-4): 193-212.

Burnett M., and Scaffidi, C., End-User Development. The Encyclopedia of Human-Computer Interaction, 2nd Ed, cap.10 https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/end-user-development

Cheverst, K., Davies, N., Mitchell, K., and Efstratiou, C., 2001. Using Context as a Crystal Ball: Rewards and Pitfalls, Personal and Ubiquitous Computing: Volume 5 Issue 1.

Corno, F., de Russis, L., and Monge Roffarello, A., 2017. A High-Level Approach Towards End User Development in the IoT. In: CHI 2017: The 35th Annual CHI Conference on Human Factors in Computing Systems, Denver, CO (USA), May 6–11, 2017. pp. 1546-1552

Coutaz, J., and Crowley, J.L., 2016. A first person experience with end-user development for smart home. IEEE Pervasive Computing, vol. 15, no 2, May-June 2016: 26:39

Dax, J., Ludwig, T., Meurer, J., Pipek, V., Stein, M., and Stevens, G., 2015. FRAMES – A Framework for Adaptable Mobile Event-Contingent Self-report Studies. IS-EUD 2015: 141-155.

Desolda, G., Ardito, C., and Matera, M., 2017. Empowering End Users to Customise their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools, ACM Trans. Comput.-Hum. Interact. 24(2): 14:1-14:33, 2017

Dey, A.K., and Newberger, A., 2009. Support for context-aware intelligibility and control. CHI 2009: 859-868

Dey, A.K., Sohn, T., Streng, S., and Kodama, J., 2006. iCAP: Interactive Prototyping of Context-Aware Applications. Pervasive 2006: 254-271

Ghiani, G., Manca, M., Paternò, F., and Santoro, C., 2017. Personalization of Context-Dependent Applications Through Trigger-Action Rules. ACM Trans. Comput.-Hum. Interact. 24(2): 12:1-14:52

Huang, H., and Cakmak, M., 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. DOI=http://dx.doi.org/10.1145/2750858.2805830

Ko, A.J., and Myers, B.A., 2004. Designing the whyline, a debugging interface for asking why and why not questions about runtime failures. In Proceedings CHI'2004: Human Factors in Computing Systems (pp. 151–158). Vienna, Austria.

Ko, A.J., and Myers, B.A., 2005. A framework and methodology for studying the causes of software errors in programming systems. Journal of Visual Languages and Computing, 16(1), 41–84.

Ko, A.J., and Myers, B.A., 2009. Finding causes of program output with the java whyline. In CHI'2009: Human Factors in Computing Systems (pp. 1569–1578). Boston, MA.

Ko, A.J., Myers, B.A., Coblenz, M., and Aung H.H., 2006. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Transactions on Software Engineering, 33(12), 971–987.

Kubitza, T., and Schmidt, A., 2015. Towards a Toolkit for the Rapid Creation of Smart Environments. IS-EUD 2015: 230-235

Kulesza, T., Burnett, M.M., Wong, W.-K., Stumpf, S., 2015. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. IUI 2015: 126-137

Kulesza, T., Stumpf, S., Wong, W.-K., Burnett, M.M., Perona, S., Ko, A., and Oberst, J., 2011. Why-oriented end-user debugging of naive Bayes text classification. ACM Transactions on Interactive Intelligent Systems (TiiS) 1, 1 (2011).

Lieberman, H., Paternò, F., Klann, M., Wulf, V., 2006. End-User Development: An Emerging Paradigm. In End User Development, Henry Lieberman, Fabio Paternò, and Volker Wulf (eds.). Springer, The Netherlands, 1-8.

Lim, B. Y., 2012. Improving understanding and trust with intelligibility in context-aware applications. PhD thesis, Carnegie Mellon University.

Lim, B. Y., and Dey, A.K., 2010. Toolkit to support intelligibility in context-aware applications. UbiComp 2010: 13-22

Lim, B. Y., Dey, A.K., and Avrahami, D., 2009. Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems, Proceedings CHI 2009, pp. 2119-2128, ACM Press.

Lucci, G., and Paternò, F., 2014. Understanding End-User Development of Context-Dependent Applications in Smartphones. HCSE 2014: 182-198

Metaxas, G., and Markopoulos, P., 2017. Natural contextual reasoning for end users. ACM Transactions on Computer-Human Interaction (ACM TOCHI), Vol.24, Issue 2, Article N.13.

Mi, X., Qian, F., Zhang, Y., Wang, X.F., 2017. An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance, Proceedings of Internet Measurement Conference (IMC) '17, November 1–3, 2017, London, UK

Myers, B. A., Ko, A. J., Scaffidi, C., Oney, S., Yoon, Y. S., Chang, K., Kery, M. B., and Li, T. J.-J., (2017) Making End User Development More Natural. In: Paternò F., Wulf V. (eds) New Perspectives in End-User Development. Springer, Cham

Paternò, F., 2013. End User Development: Survey of an Emerging Field for Empowering People, ISRN Software Engineering, vol. 2013, Article ID 532659, 11 pages, 2013. doi:10.1155/2013/532659

Pausch R., Burnette T., Capeheart A.C., Conway M., Cosgrove D., DeLine R., Durbin J., Gossweiler R., Koga S., White J., Alice: Rapid Prototyping System for Virtual Reality. IEEE Computer Graphics and Applications, May 1995

Perera, C., Aghaee, S., and Blackwell, A.F., 2015. Natural Notation for the Domestic Internet of Things. Proceedings IS-EUD 2015: 25-41, Springer Verlag.

Pipek, V., 2005. From tailoring to appropriation support: negotiating groupware usage. University of Oulu. Retrieved from http://herkules.oulu.fi/isbn9514276302/isbn9514276302.pdf.

Pipek, V., and Wulf, V., 2009. Infrastructuring: Towards an Integrated Perspective on the Design and Use of Information Technology. Journal of the Association of Information Systems (JAIS), Volume 10, Issue 5, May 2009, 306-332.

Repenning, A., 1995. Bending the Rules: Steps toward Semantically Enriched Graphical Rewrite Rules, 1995 IEEE Symposium on Visual Languages, Darmstadt, Germany, pp. 226-233, Sept. 5-9, 1995.

Smith, D.C., Cypher, A., Spohrer, J., 1994. KidSim: Programming Agents Without a Programming Language, Communications of the ACM 37(7), 54-67, July 1994.

Sutcliffe, A.G., and Papamargaritis, G., 2014. End-user development by application-domain configuration. Journal of Systems and Software 91: 85-99.

Tetteroo, D., Vreugdenhil, P., Grisel, I., Michielsen, M., Kuppens, E., Vanmulken, D., and Markopoulos, P., 2015. Lessons Learnt from Deploying an End-User Development Platform for Physical Rehabilitation. In Proceedings CHI '15. ACM Press, pp. 4133-4142. DOI=http://dx.doi.org/10.1145/2702123.2702504

Ur, B., McManus, E., Yong Ho, M.P., and Littman, M.L., 2014. Practical Trigger-Action Programming in the Smart Home. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). 803-812.

Ur, B., Yong Ho, M.P., Brawner, S., Lee, J., Mennicken, S., Picard, N., Schulze, D., and Littman, M.L., 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

Yarosh, S., and Zave, P., 2017. Locked or Not?: Mental Models of IoT Feature Interaction. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). ACM, New York, NY, USA, 2993-2997. DOI: http://dx.doi.org/10.1145/3025453.3025617