# End User Development: Verifying Home Behavior

Alexandre Demeure[1], Sybille Caffiau[1], Sophie Dupuy-Chessa[1], Huong Ta[1], and Lydie DuBousquet[1]

Univ. Grenoble Alpes, CNRS, Inria, LIG, 38000 Grenoble, France
First.Last@univ-grenoble-alpes.fr

**Abstract.** End User Programming is a solution to enable inhabitants to create a smart home adapted to their lifestyle. With this purpose, it is necessary to design softwares adapted to end-users. This paper presents why inhabitants may need to evaluate the home behavior when she/he (1) specifies and (2) maintains and improves her/his programs, and how existing tools can meet these needs.

**Keywords:** End User Programming · Evaluation.

## 1 Introduction

End User Programming enables inhabitants to create a smart home according to their lifestyle [12]. A programmed smart home system may be defined as a system with an only one program running a set of rules. In order to help an end-user to program her/his home, several researches aim at proposing adapted programming languages or adapted interfaces (with metaphors [16] or specific interaction paradigms [5]). By using these interfaces, inhabitants may defined the set of rules (i.e programmed the home system) mainly expressed following the Event-Condition-Action paradigm [17]. However, this programming paradigm may imply that the programmed behavior does not correspond to the novice programmer's expectations [9,15].

Programming the expected behavior of her/his home is all the more challenging that the family life often implies to change the routines [7] and thus, to change the existing home program [7]. It is very important that some methods and softwares are developed to help end-users during the whole system life, i.e. to specify, to edit, to maintain the home programmed rules [6].

Considering the home behavior as a concrete system translation of the programmed rules, we propose to support the specification and the maintainability activities through the study of the home behavior. This paper presents why and how an inhabitant evaluates the behavior of her/his home in order to program her/his rules.

## 2 Why inhabitants need to verify the smart home behavior?

When considering any system that will be used by human, the first question to address is to understand users and their tasks. In our context, an end-user is an inhabitant who programs to specify the behavior of his/her smart home. As inhabitants have not (or have few) knowledge on programming, they need tools to bridge the gap between their ideas and what the system interprets from their programs.

### 2.1 Challenge 1: specifying rules

Most of the times, programs are expressed using rules based on Event-Condition-Actions (ECA) or Trigger-Action (TA) paradigms [15]. These rules aim at automating tasks, helping to perfom difficult tasks (e.g. closing all shutters during a storm) or adding features such as remembering to take out the trash. The concrete translation of the rules in the household environment is the smart home behavior.

In such context, several barriers need to be overcome in order to allow non programmers to program [10]. One of these barriers is that they have to learn the structure and the language constraints of the programming environment [13]. To ease this learning, End User Development approaches are designed to help inhabitants to express rules [14].

Specifying rules signifies to translate the expected smart home behavior into rules. Unfortunately, some studies [9,15] demonstrate that end users may misunderstand the semantics of rules and they may create bugs caused by the misunderstanding of the temporal paradigms [1]. For this programming step, the inhabitant needs to confront how the system interprets the rules with what he/she expected. This confrontation is done by **comparing the home behavior with the expected one**. In addition to check the expecting behavior, programming implies **to anticipate unforeseen behaviors**. Example 1 illustrates an unforseen behavior due to the missing of the undoing behavior (*Missing Reversal Bug* in [1]).

*Example 1.* Nic installed in his home some sensors and actuators that he controls by using a box. His box also allows him to create rules to program some personal features. The first rule he programed closes all shutters during the night. The first evening, all shutters shut down and Bob is very satisfied to be no longer obliged to travel over his three floors to close all the shutters. But the following morning, he observe that the shutters are not opened.

In Example 1 the unexpected behavior of the shutters is caused by an unforseen case. To improve the behaviour of his home, Nic will add the behavior of the shutters during the day. However, the modifications may impact the whole programmed behavior, bringing another challenge, the maintainability challenge in [6].

### 2.2 Challenge 2: maintaining and improving rules

For maintaining the set of rules, the comparison between the planned behavior (expressed in the rules) and the past one may help to check the modification implications. Let's consider again the example of Nic and the programmed behavior of his home shutters (Example 1, inspired from [8]).

*Example 2.* Nic is happy with the programmed behavior of his home shutters. He does not think about it anymore as it is managed by the system. But a summer night, while he is in his terrace with some neighbourgs at 10pm, he sees all shutters closing, including the shutter of the door giving access to his terrace. He quickly goes home and he opens (with a remote control) this shutter to be able to access to the terrace. Nic will modify his rules to include a behavior adapted for the summer period. As he is with friends, he will have to modify the shutter behavior at least the next day. So this modification requires to remember the context of this unexpected behavior and the way he programmed the previous rules.

To correct rules, an end user has **to identify the contexts (that occured in the past) in which the behavior is not the one expected**, the programming error and to make appropriate modifications. All these steps are new problems for end users that may cause behavior troubles.

Another problem is related to time: the programmed rules may have been written and run for a long time. This delay brings difficulties to identify the conditions of rules activation in term of data and to remind the programmed rules.

Moreover, even if inhabitants would like to modify rules to take into account some unexpected cases, they may want to keep the programmed behaviour over time. Thus, they need **to check poperties on the behavior according to the past**. For example, after correcting his rules, Nic wants to check (1) that in summer nights when he is in the terrace, the shutter of the terrace door stays opened and (2) that, for all the other time, the shutter behavior is the same that since the rules run.

## 3 Existing approaches to verify the programmed behavior

### 3.1 Approaches relying on users

Most of existing home automation systems enable end users to test the action part of a rule to verify whether it does correspond to what they expect [8,15]. In order to test their rules, end users try to reproduce the context in which their rules have to be tested. Currently, they use different strategies to achieve this. First, when possible, they can act on sensors to trigger the rules they want to test. This is easy for a button or a motion sensor but not for a thermometer or a smoke sensor for instance. A second observed strategy is to define virtual sensors when the system supports it. Virtual sensors are not binded to any real sensor

and their values can be set by end users (or a program such as a web service). This is then easier for end users to change sensors values and to see what rules are triggered. This approach is also explored by [3]. However sometimes these sensors values are not well known by users. For instance a luminosity sensor expresses luminosity in lux but this is not easy for end users to mentally map lux to actual luminosity they perceive or have in mind. As a result, the third strategy is simply to proceed by "trials and errors" and to wait for the "next time" the situation will occur.

A useful functionality, proposed by a couple of boxes (eeDomus, HomeSeer) [8] is the possibility to navigate between rules and their associated devices or services. This can be used to preventively check what programs and devices are going to be impacted by a modification of the system (e.g. removing a sensor, adding a new program that controls lights, etc.). In the same vain, AppsGate system [4] proposes a dependency graph that lets users monitor home states through relations between devices and programs. This aims to help user to remember that the state of an entity is modified by more than one program, which may result result in unexpected behavior due to conflicting commands.

### 3.2 Automatic and semi-automatic approaches

It is possible to check properties on rules at design time, i.e. when the end user is specifying them. Cano et al. [2] study the coordination of ECA rules and address three problems that may occur in programs using ECA paradigm: redundancy, inconsistency and circularity. **Redundancy** means that there are two or more rules in the system which replicate partially or totally a behavior. For instance, there can exist two rules triggered when the temperature goes below 15 degrees with the first rule applying actions A and B and the second rule applying action A. **Inconsistency** occurs when contradictory actions are sent to devices. This can occur if multiple rules are activated at the same time, and their execution order may render different final states of the system. The example 3, extracted from [2], illustrates inconsistency with three rules.

*Example 3.*

| |
|---|
| ON presence IF true DO lights_on |
| ON presence IF true DO TV_on |
| ON TV_light IF TV_on DO lights_off |

Last, **Circularity** occurs when rules get activated continuously without reaching a stable system state that makes them finish their execution. This happens when the action of a rule implies the trigger of another rule which in turn, directly or indirectly, triggers the first rule.

The system proposed by Cano et al. [2] is able to detect the three aforementioned problems. However it is based on a command line interface, which is adapted for programmers but not for end users.

Manca et al. [11] propose a system that targets end users by offering them a graphical user interface. This system aims to detect inconsistencies in rules although it seems to be limited to direct inconsistencies (when two rules can

be triggered at the same time and apply different actions to an actuator). As mentionned before, it also enables end users to simulate contexts and see how the system behaves.

Last, Zhang et al. [18] describe the Autotap system which enable users users to specify desired properties for devices and services. These properties are translated into linear temporal logic and used to produce compliant TAP rules from scratch and/or repairs existing ones.

## 4 Conclusion

Helping end users to define and verify their home behavior is challenging. We have seen that tools exist to automatically ensure that the end users' rules are consistent, avoid redundancies and circularities [2,11]. This is only a first stage in ensuring that the rules really encode the expected behavior.

Once this stage is reached, the problem is to enable end users to verify how the smart home behaves. To achieve that, a promising approach is to enable end users to simulate how the system behave according to simulated sensors and actuators. This approach is already explored by existing home automation boxes [8] and academic researches such as Manca et al. [11].

Thus, the existing verification approaches enable end users to compare the home behavior with the expected one (by simulating) and partially to anticipate unforeseen behaviors (when they are caused by programming errors such as circularity). Unfortunately, none approach helps to identify the contexts for which the behavior may not be the expected one or to check poperties on the behavior.

It is a difficult task for end user to enumerate and define all relevant values of sensors and actuators for which a rule should be tested, even when some of these values already occurred in the past. A challenge remains to provide a tool that help end users in providing theses values.

## References

1. Brackenbury, W., Deora, A., Ritchey, J., Vallee, J., He, W., Wang, G., Littman, M.L., Ur, B.: How users interpret bugs in trigger-action programming. In: Proc. CHI (2019)
2. Cano, J., Delaval, G., Rutten, E.: Coordination of eca rules by verification and control. In: International Conference on Coordination Languages and Models. pp. 33–48. Springer (2014)
3. Corcella, L., Manca, M., Paternò, F., Santoro, C.: A visual tool for analysing iot trigger/action programming. In: International Conference on Human-Centred Software Engineering. pp. 189–206. Springer (2018)
4. Coutaz, J., Crowley, J.: A first person experience with end-user development for smart homes. IEEE Pervasive Computing, special issue on Domestic Pervasive Computing **15**(2), 26–39 (2016). https://doi.org/https://doi.org/10.1109/MPRV.2016.24, http://ieeexplore.ieee.org/document/7445783/

5. Coutaz, J., Demeure, A., Caffiau, S., Crowley, J.L.: Early lessons from the development of spok, an end-user development environment for smart homes. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication. pp. 895–902. ACM (2014)
6. Dautriche, R., Lenoir, C., Demeure, A., Gérard, C., Coutaz, J., Reignier, P.: End-user-development for smart homes: relevance and challenges. In: Proceedings of the Workshop" EUD for Supporting Sustainability in Maker Communities", 4th International Symposium on End-user Development (IS-EUD). p. 6 (2013)
7. Davidoff, S., Lee, M.K., Yiu, C., Zimmerman, J., Dey, A.K.: Principles of smart home control. In: International conference on ubiquitous computing. pp. 19–34. Springer (2006)
8. Demeure, A., Caffiau, S., Elias, E., Roux, C.: Building and using home automation systems: a field study. In: International Symposium on End User Development. pp. 125–140. Springer (2015)
9. 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. ACM (2015)
10. Ko, A.J., Myers, B.A., Aung, H.H.: Six learning barriers in end-user programming systems. In: 2004 IEEE Symposium on Visual Languages-Human Centric Computing. pp. 199–206. IEEE (2004)
11. Manca, M., Santoro, C., Corcella, L., et al.: Supporting end-user debugging of trigger-action rules for iot applications. International Journal of Human-Computer Studies **123**, 56–69 (2019)
12. Mennicken, S., Vermeulen, J., Huang, E.M.: From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 105–115. ACM (2014)
13. Pane, John F, R.C.A., Myers, B.A.: Studying the language and structure in non-programmers' solutions to programming problems. International Journal of Human-Computer Studies **54**(2), 237–264 (2001)
14. Paternò, F., Santoro, C.: A design space for end user development in the time of the internet of things. In: New perspectives in end-user development, pp. 43–59. Springer (2017)
15. Terrier, L., Demeure, A., Caffiau, S.: Ccbl: a language for better supporting context centered programming in the smart home. Proceedings of the ACM on Human-Computer Interaction **1**(EICS), 14 (2017)
16. Truong, K.N., Huang, E.M., Abowd, G.D.: Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In: International Conference on Ubiquitous Computing. pp. 143–160. Springer (2004)
17. 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. ACM (2014)
18. Zhang, L., He, W., Martinez, J., Brackenbury, N., Lu, S., Ur, B.: Autotap: Synthesizing and repairing trigger-action programs using ltl properties