End-User Development of Smart Home Rules Using Block-Based Programming: a Comparative Usability Evaluation with Programmers and Non-Programmers

MATEUS CARVALHO GONÇALVES, OTÁVIO NEVES LARA, RAPHAEL WINCKLER DE BETTIO, and ANDRÉ PIMENTA FREIRE , FEDERAL UNIVERSITY OF LAVRAS

The use of Smart Homes has grown considerably in the past decade. Enabling end-users to develop rules to program their homes and devices is very important to empower them. Several studies have analyzed trigger-action programming tools, primarily using form-based and data-flow approaches for programming interfaces. This paper aimed to evaluate the usability of a block-based tool prototype for end-user development of rules to control smart homes and to compare the difficulties encountered by non-programmers and programmers. Evaluations involved ten programmers and ten non-programmers in Brazil. A thematic analysis of 247 problem instances (80 from programmers and 167 from non-programmers) yielded the following themes, with problems related to condition blocks, action blocks, states and actions, time-related tasks, block configuration and personalization, information architecture, programming logic, the conceptual model of smart homes, simulator and debugging, help and technical problems. Despite most non-programmers being able to experiment with the blocks, their task completion rates were significantly lower than programmers'. The analysis showed aspects in which block-based programming can enhance the use for non-programmers. They also confirmed interaction aspects revealed by previous studies using form-based and data-flow approaches that also occur with block-based programming to design smart home rules. The results in this study are important to improve end-user development tools for smart homes.

1 INTRODUCTION

The use of Smart Home solutions has grown considerably in the past decade. The use of Internet of Things (IoT) resources and the availability of devices and technologies may enable the availability of many features to enhance the living experiences of users [23]. Such technologies can improve their houses' efficiency and allow more independence for people with disabilities [6, 24] and older people [43, 47].

Despite such technologies' growth, their use is still very unequal when comparing developing countries and more developed countries. Statista's [41] projections from 2019 estimated household penetration of smart home technologies in 32.2% of homes in the United States, 24.5% in the United Kingdom, 9.2% in Italy, whist much smaller percentages of households had some sort of smart home technology in developing countries, such as 3.4% in Brazil, 3.2% in Argentina and 0.9% in Sudan, for example.

Enabling end-users to develop rules and program on how homes and their devices will behave is very important to empower users to control and optimize their houses [14**?**].

Several tools have been made available to enable the creation of rules to be applied in smart homes. One of the most widely used tools is web-based IFTTT (If This Then That) [1], Azooma[2], Zipato[3], and others. Many of those tools can currently integrate with other solutions such as Alexa, Google Home, Apple Home and other solutions. The most common way of programming rules relies on the trigger-action approach. Users set a trigger (e.g. opening a door or having a particular time elapsed) to enact an action (e.g. switching the lights on or turning the TV off). IFTTT and Azooma, for example, use form-based interfaces to create rules. Zipato uses block-based programming, using jigsaw-like blocks that can fit into each other to connect actions, verifications and other elements to compose rules to be used to automate home behaviour.

[1]Available online at https://ifttt.com/
[2]Available online at https://atooma.com
[3]Available online at https://www.zipato.com/how-it-works/rule-creator

Authors' address: Mateus Carvalho Gonçalves; Otávio Neves Lara; Raphael Winckler de Bettio; André Pimenta Freire, , Federal University of Lavras.

Different studies have analyzed how trigger-action programming tools can be used for this end, primarily using form-based and data-flow approaches for programming interfaces (e.g. [4, 8, 13, 19, 27]). These studies have created a significant body of knowledge with opportunities and challenges to enhance end-user development to generate smart environments rules. Those issues include problems regarding end-users understanding and creating a suitable mental model of smart homes, difficulties in understanding programming logics, managing conflicting rules, debugging and simulating their rules' behaviour.

Block-based programming has been widely used in the context of teaching basic programming and computational thinking for non-programmers [26, 40, 46]. Block-based programming consists of using jigsaw-like pieces to visually represent programming concepts such as condition, commands, objects, variables, to avoid the need for non-programmers to memorize complex programming language commands and syntax. Despite the wide use in the context of programming and computational thinking teaching, there have been comparatively fewer studies [1, 28, 42, 45] investigating the use of block-based programming to support end-user development of rules for smart homes.

This paper aimed to evaluate the usability of a block-based prototype for end-user development of rules to control smart homes, comparing its use by programmers and non-programmers.

In this sense, the paper sought to answer the following guiding research question:

"What are the main characteristics of using block-based programming of smart-home rules by programmers and non-programmers and types of usability problems encountered by both groups?"

The study had a qualitative approach, and aimed to analyse the types of problems encountered by both user groups and compare the results with those encountered in studies investigating the use of trigger-action end-user development environments that use form-based and data-flow approaches.

Besides, the paper also contributes with a usability study of end-user development tools in a country with considerably lower penetration of smart home technologies than more developed countries in which most of the existing studies were developed, with a prominent presence of North-American, European and developed Asian countries.

The evaluations involved ten programmers and ten non-programmers in Brazil, performing tasks on a system developed with Google's Blockly [20] and a simulator to support debugging. The results include a thematic analysis of 247 problem instances (80 from programmers and 167 from non-programmers).

The remainder of this paper is organised as follows. Section 2 presents the main background concepts concerning smart homes, end-user development with block-based programming, and related work concerning end-user development of smart homes rules. Section 3.3 presents the methodological aspects, covering the study design, participants, evaluated systems, procedures for usability evaluation and data analysis. Section 4 reports on the results and types of problems encountered in the study. Section 5 discusses the main findings of the research and, finally, Section 6 presents conclusions and future work.

2 BACKGROUND

This section presents fundamental concepts concerning Smart Home technology and End-User Development with Blocks and related work in the literature concerning usability studies on end-user development for smart homes.

2.1 Smart Home Technology

Research on smart home devices and technologies has grown considerably, as the interest in this type of technology has grown with the availability of devices, sensors, controls and ubiquitous technology for home-usage.

The evolution of IoT (Internet of Things) technologies and devices and the availability of more appliances connected to the internet have boosted the use and research on smart homes.

Different authors approach the definition of "smart home" from different perspectives. Risteka Stojkoska and Trivodaliev [39], for example, defined smart homes as "the use of ICT in home control, ranging from controlling appliances to automation of home features (windows, lighting, etc.)". From a more technical perspective, Ricquebourg *et al.* [38] defined "smart homes" as "a house which is equipped with smart objects, a home network make it possible to transport information between objects and a residential gateway to connect the smart home to the outside Internet world". Katuk *et al.* [23] defined a smart home as a "home or living environment that uses technology to allow electrical appliances and systems to be controlled automatically".

Appropriate use of smart home devices and applications involves several challenges to provide good interaction for users. Many such systems may be challenging for users since they involve different devices and interaction with different protocols and connection requirements. Davidoff *et al.* [12], in an early study in 2006, defined a set of principles that smart homes should have to suit to users' needs in their homes: "P1 - allow for the organic evolution of routines and plans; P2 - easily construct new behaviours and modify existing behaviours; P3 - understand periodic changes, exceptions and improvisation; P4 - design for breakdowns; P5 - account for multiple, overlapping, and occasionally conflicting goals; P6 - The home is more than a location (needs even out of home); P7 - participate in the construction of family identity".

2.2 End-User Programming with Blocks

According to Lieberman *et al.* [25], the goal of End-User Development is "to allow people without programming experience to create or modify their applications". End-user programming has been used in several different scenarios, with different tools available for different contexts, like domain-specific languages, trigger-action programming and visual programming using strategies, such as blocks.

Block-based programming has been widely disseminated by the availability of tools such as *Scratch* [26]. This type of programming enables creating commands and logical blocks using jigsaw-like pieces that can be mounted to code a program. Google's App Inventor, for example, uses the principles of block-based programming to create mobile apps and has been extensively used as a tool for introductory courses and experiences with programming, including for computational thinking for children [31, 34, 40].

Zhang and Nouri [46] conducted a systematic literature review on the use of Scratch to teaching computational thinking to K-9 students, synthesising 55 empirical studies. Their results showed that the block-based programming tool helped young students to learn basic skills of computational thinking. However, they also found evidence that this kind of tool could pose difficulties in understanding more complex programming structures.

2.3 End-User Development for Smart Homes

The interest in the end-user programming (EUP) concept has increased in the past decade. For smart homes, in particular, this paradigm has been gaining popularity through the Event-Condition-Action (ECA) programming paradigm and its variants, like trigger-action. Paternò and Santoro [35] have pointed out that evolution in research with the Internet of Things has stimulated research on end-user programming approaches, methods and tools to support the use in applications, things and robots. An early study concerning Ambient Intelligence and Human-Computer Interaction [29] pointed out that the area would need more studies to explore different paradigms and editing tools to help create rules.

Our review focused primarily on studies that performed some kind of empirical study involving users. We also encountered other studies that presented literature reviews and proposed theoretical models. Among such studies, Fogli *et al.* [15] conducted a systematic mapping of the literature, analysing 48 papers from the literature regarding end-user development tools for the Smart Home. Their study yielded 11 tools for end-user development, and they performed qualitative analysis on six of the tools, according to the seven principles defined by Davidoff *et al.* [12]. In their study, the tool with the best performance according to the seven principles was the app *Tasker*. From the observation of issues and shortcomings in home automation solutions, Funk *et al.* [18] proposed a domain model for smart homes to model scenarios, intentions, references and actuations. However, they do not report any prototype implementation and evaluation with end-users.

This remainder of this section presents a review of related studies that analysed the interaction with end-user development tools focused on smart homes. The review presents studies that performed user studies, their findings, and their relation to the present study. The review analyses 1) studies that analyzed the needs and main usage behaviour of rules for smart homes, 2) usability studies of form-based and text-based trigger-rule platforms, 3) testing and debugging end-user programming of smart homes and 4) studies that analyzed the use of block-based programming.

2.3.1 *User needs and usage behaviour.*

Given the widespread use of end-user development approaches to programming rules for smart homes, there has been increased interest in understanding how users create such rules and user needs. This section describes studies that analyzed sets of rules and their applications and *in-loco* studies covering user needs.

Ur *et al.* [44] performed an analysis of more than 200,000 programs (named recipes) publicly available at the IFTTT website in 2015, created by more than 100,000 different users. The results showed that users used trigger-action rules for various activities, such as social networks and notifications for weather conditions and daily activities.

Jakobi *et al.* [22] conducted a longitudinal study in twelve households equipped with Do-It-Yourself (DIY) smart home kits to investigate how people used the system to tailor the house's functionality to their needs. They found that people's needs varied across time, with non-programmers participants being more engaged with more complex web-based user interfaces in the first months and then using mobile-based interfaces to configure their houses to adjust particular exceptions and other rules as time went by.

Brich *et al.* [4] conducted a contextual inquiry study with 18 participants to investigate their potential acceptance of home automation and the use of different notations for programming house behaviour - rule-based notations and process-oriented paradigms. Their results suggest that, even though rule-based notations can cover a good range of situations, they have many limitations regarding what they can afford for users. Process-oriented notations were pointed out by participants for complex scenarios, with more devices in the home and more expressive.

2.3.2 *Empirical studies using trigger-action programming for smart homes.*

Many studies encountered in our literature review concerned empirical studies using form-based and text-based tools using trigger-action to program smart homes by end-users. Following, we discuss the main characteristics of such studies, their findings, methods, contributions, gaps, and other relevant issues that helped contextualize and discuss the present study results.

Cabitza *et al.* [7] conducted a comparative test with 15 students of a Human-Computer Interaction course evaluating the IFTTT and Atooma applications. Both tools use a trigger-action rule-based approach to define rules for smart homes. Their study analysed a total of 114 tasks, classified according to how feasibly a non-expert could perform them. The results showed that 61% of the tasks would be feasible to be done by non-programmers, 20% would be conditionally feasible, and 18% would be not yet feasible. As conclusions, they provided four main recommendations for the design of End-User development tools for smart homes: 1) to design a multi-platform tool (including web-based and mobile platforms); 2) to categorise triggers and actions based on users' objectives; 3) allowing the combination of multiple conditions with multiple actions; 4) to provide clear descriptions of triggers and actions. Their study, however, did not evaluate any solution that employed block-based rulemaking.

Demeure *et al.* [14] conduct *in-situ* interviews with ten participants who used home automation in their houses in 2014. Their study investigated the types of technology they used, the devices and how they performed programming of actions and tested their scenarios. Most participants reported using structures consisting of Event-Condition-Action (ECA) either by text or forms. Of the technologies reported by participants, only Zipato had a rule creation system based on blocks programming inspired by Scratch [26]. However, the study did not provide more details about the use of this system by users. Demeure *et al.*'s study had important findings in other aspects. They pointed out the importance of enabling rules that consider time, as many participants reported this. They also pointed out difficulties in testing the rules created for home automation. Most interview participants reported testing using trial-and-error. Some participants reported using a "Test" button on their automation boxes, but this was not always available.

Huang and Cakmak [21] performed two studies with 60 and 42 participants, respectively, to analyse users' mental models with trigger-action programming using IFTTT. They found many difficulties related to inconsistencies in the interpretation of trigger-action programs and how users committed errors in creating programs with the expected behaviour.

The trigger-action rule-based concept was widely used in Ambient Assisted Living (AAL) to attend to elderly people necessities. Ghiani *et al.* [19] presents a web-based rule editor to personalize AAL scenarios in an "intuitive manner" and grouping rules concerning their main focus/goal, according to the authors. After collecting the requirements in a workshop, Chesta *et al.* [9] developed TARE, the Trigger-Action Rule Editor, a personalization platform for caregiving and monitoring the health and habits of older adults using a set of personas and scenarios to support the design. The GUI approach utilizes filling the clause needed through pathways of choices organized hierarchically in a tree structure to reach the user's element. Usability tests involving three elderly and four caregivers were also conducted to improve the tool's design, and one of the main enhancements based on problems of memorability and loss of control was the search functionality of rules.

In a study performed by Reisinger *et al.* [36, 37], two prototypes were designed to support usability tests with 16 participants and evaluate two paradigms, a form-filling to construct trigger-action rules and a flow-based approach (e.g., Node-RED). The qualitative results showed form-filling as more efficient and easier to use, and participants mentioned a security feel due to pre-defined directions. The User Experience Questionnaire confirmed the form-filling approach results, but participants pointed to data-flow as more attractive, exciting, creative and playful. Despite the better results of the first, combining the two would fit a better solution for different tasks.

Caivano *et al.* [8] conducted a study with a literature review and a user study with the tools Azooma, IFTTT and Tasker involving 20 users with varying levels of experience with digital technologies. Their results showed that different tools had a higher accuracy level and time-on-task. Participants had problems with unfamiliar terminology in selecting interactive elements. From the results, the authors provided a set of ten recommendations for end-user development of smart homes: 1) facilitate trigger and action retrieval, 2) facilitate trigger and action selection, 3) make rule creation flexible, 4) support re-use, 5) speak the user's language, 6) keep track of interaction history, 7) be coherent with device interaction metaphor, 8) provide different levels of complexity, 9) support management of conflicting rules and 10) support collaborative rule creation.

Palekar *et al.* [33] proposed a classification of the types of errors that users can make when using trigger-action programming. From experience from their previous studies, the authors categorized the following common errors: lack of action reversal, feature interaction (conflict between features), feature chaining, event+event rules, state+state rules, missing trigger, missing action, secrecy violation and integrity violation. They also reported problems with users not forming a wrong mental model about how programming smart homes would work.

2.3.3 Testing and debugging end-user programming of smart homes. One particular issue identified in recent studies concerns debugging and verifying rules by non-programmers in end-user development approaches of smart environments. Manca *et al.* [27] proposed a tool to help users identify whether a rules-based system achieves the intended behaviour and, if not, what are the reasons for this. They evaluated the tool with 20 participants. Their results showed that using the debugging tool helped reduce typical errors encountered by non-programmers with rules-based programming for smart homes. They found positive results to improve issues such as difficulties in perceiving the difference between events and conditions, helping users build appropriate mental models, encountering inconsistencies and conflicts between rules, and having examples and counterexamples.

Demeure *et al.* [13] also mentioned difficulties to help uses debug and simulate how the behaviour of a smart home would entail in comparison with the expected behaviour. They also point out problems to help identify why a particular rule does not work as expected.

In this sense, Zhao *et al.* [48] also proposed an initial prototype to help visualise the differences between trigger-action programs in syntax, behavior, and properties.

Some propositions have innovative interface designs, aiming to reach personalisation through interfaces that users are more familiarised on the web and smartphones to take off the weight of programming. Fogli *et al.* [16] designed ImAtHome, a smartphone app using the Apple HomeKit for iOS in which the whole house can be personalised from an initial set and the EUP tasks are presented like configurable elements as well. To evaluate the feasibility of the application, 14 participants executed a set of tasks, in which only two were focused in simple ECA rules. Bellucci *et al.* [2] presents a trigger-action rule-based system which is part of the T4Tags 2.0 do-it-yourself home toolkit. The programming tool allows combining multiple triggers, which provides expressiveness enough to encompass 94% of the scenarios extracted in a workshop for the research. Furthermore, T4Tags dispose of a web-based social platform to share the personal recipes (or codes) with the community.

Another study conducted by Corno *et al.* [11] also addressed the issue of debugging trigger-action programming in end-user development for the Internet of Things. They developed the tool EUDebug for IFTTT, using Semantic Colored Petri Net (SCPN) to simulate and debug triggers and actions. They evaluated the tool with 15 end users and obtained favourable results, showing advances in tools to support debugging and understanding the step-by-step process of execution.

2.3.4 Smart-home rules development with block-based programming. In one of the few studies encountered in this literature review concerning end-user programming for smart environments using block-based programming, Terrier *et al.* [42] proposed CCBL, the Cascading Context-Based Languages, a hierarchical block solution that possesses a root state which can prevent coordination problems existing in ECA rules, like redundancy, inconsistency and circularity. By comparing this solution with the ECA in user tests with 21 participants, between programmers and those with no programming skills, CCBL showed to be more effective against errors and faster to create codes for tasks centred on states both for programmers and non-programmers. Valsamakis *et al.* [45] aimed to empower older people and caregivers in their study by allowing them to program using blocks and share their codes as T4Tags described above. The study employed personas and scenarios to carry out the design and did not present any user study.

In an earlier study, Ash *et al.* [1] proposed the “Scratchable Devices”, a language to program smart home behaviours created by BYOB (Build Your Own Blocks), and they presented a set of use cases and explained the architecture of the project. However, they did not validate the idea with users nor mapped possible errors that the paradigm could endure.

Mattioli and Paternò [28] proposed recommendations for creating trigger-action rules in a block-based environment for IoT environments. They compared the use of block-based programming using two approaches for sequence-prediction: the definition of full rules and parts of rules relevant for the next step. They presented an evaluation and comparison of the two approaches.

The analysis of related work in this paper shows that there has been extensive research on the use of trigger-action programming using forms and data-flow approaches for end-user programming of smart environments. However, the analysis showed that, comparatively, fewer studies [1, 42, 45] have investigated the impact of using block-based programming in this context, despite the wide use of this approach in introductory programming teaching [26, 40]. The analysis also showed that there is very little research on end-user development for smart homes in developing countries, in which those technologies have had more limited dissemination in comparison to more developed countries.

3 METHODS

This section presents the methods used in the development of the study, covering the study design, participants, the evaluated prototype, procedures for usability evaluation and data analysis.

3.1 Study Design

This research consisted of an empirical study to evaluate an end-user development tool’s usability to create and simulate smart home rules. The tool has been integrated into a prototype structure in the laboratory to study the accessibility of smart homes for disabled and older users, called “Casa Assistiva” (assistive home). The end-user development tool used block-based programming, developed with Blockly [20], similarly to the Scratch software.

The usability study consisted of a remote evaluation of the prototype with the block-based rule editor and simulator (due to the social isolation measures taking place in Brazil due to the COVID-19 pandemic), with ten participants with at least basic experience with programming and ten participants with no previous experience with programming. This way, it was possible to compare programmers and non-programmers and analyse whether the block-based programming approach helped non-programmers start developing rules for smart homes.

Participants were shown a short 5-minute video to explain the tool’s basic features and see one example of a rule they could develop. The video only provided broad explanations about how to drag blocks and connect them, with a straightforward example, to minimise the learning effect before the evaluations.

After this, participants performed tasks and were asked to think out loud, saying what they were thinking to help the researchers understand their mental models of the tasks.

The research protocol for the he evaluations was approved by the university’s Research Ethics Committee, with id: CAAE 13310219.8.0000.5148. All users signed a consent form electronically authorizing the use of the collected data.

3.2 Participants

We selected two groups of participants to participate in the usability tests: programmers and non-programmers, aged 18 years or older.

Participants were recruited from the researchers’ social network and announcement at the university’s social media and communication channels. All participants were born and raised in Brazil and were native speakers of Portuguese.

Participants in the programmers' group should have completed at least an undergraduate-level course on Data Structures (second course in the Computer Science, Information Systems and Control Engineering program at the university). This requirement would show they had at least some experience programming more complex structures in conventional programming languages. Non-programmers could include any person that had not had any previous experience with programming.

In the **Programmers group**, the age of the ten participants ranged from 19 to 33 years, with an average of 23.5 and standard deviation of 3.9. The group included two females and eight males. Two participants were graduate students in Computer Science, and the other eight participants were undergraduate students in Computer Science. Most participants rated their experience with computers between 6 and 7 on a scale from 1 (not at all experienced) to 7 (very experienced). On a scale from 1 to 7 for rating programming experience, the median value in this group was 5. Programming languages participants had experience included C, C++, C#, Java, Pascal, Python, JavaScript, Haskell, Prolog. Most participants had never had any experience with programming devices and smart home environments (eight out of ten). Only three out of the ten participants had already taken part in usability evaluations.

In the **Non-Programmers group**, the age of the ten participants ranged from 18 to 38 years, with an average of 26.3 and standard deviation of 7.6. The group included five females and five males. Five participants were undergraduate students in Biology, Veterinary Medicine, Forestry Engineering, Portuguese Language and Mathematics. Two participants were graduate students, and three had a graduate degree. Concerning how participants rated their experience with computers on a scale from 1 (not at all experienced) to 7 (very experienced), the median value was 4. When rating their experience with programming on the same scale, nine participants rated their experience as 1 (not at all experienced) and only one rated as 2. One participant reported having had a brief experience with code.org. Four participants reported having had experiences programming devices such as VHS players, routers, TV automatic turn-off features and video-games. None of the participants had previous experience taking part in usability tests.

3.3 The Evaluated Prototype

In order to test the block language approach, was build a tool called Casa Assistiva EUD. This tool was written in Java and JavaScript by Anonymous [17] to make end-users create smart home's rules by themselves, without any need for previous knowledge of programming, bringing the development of a rule closer to human language. Casa Assistiva EUD is part of *Casa Assistiva* (*Assistive Home* in Brazilian Portuguese), a project to allow people with disabilities and older adults to have a more comfortable life with Smart Home geared towards them.

The block language was split into categories to help make easy the creation of rules:

- **Conditions**: condition structures (if / else if / else) to verify if a condition (stated on a Verification block) is true or false to activate a command;
- **Objects**: contains the elements of the home, such as light bulbs, fans, TVs and others;
- **Actions**: the actions to deal with this elements, such as turn on, turn off, open, close;
- **Verification**: a way to verify the current state of a element, the condition of a condition;
- **Time**: a special type of Verification blocks, related to time. They are two: "random" and "current time".

Casa Assistiva EUD uses the Blockly [20], a framework from Google to create the graphical blocks for the user interface. Blockly allow children and non-programmers to learn programming more intuitively. This idea in the development of Casa Assistiva EUD was to bring the advantages of block language to allow end-users to develop their own Smart Home rules.

The Casa Assistiva EUD provides a menu where we find the blocks separated into categories. These blocks may be dragged to the blank area, called assembly area, as seen in Figure 1. In the assembly area, the blocks are linked to become the rules. The blocks look like pieces of a puzzle to avoid unsuccessful attempts of link. In the example shown in Figure 1 (in Portuguese), the screen shows a list of elements available under the category Actions, with choices of actions such as Turn on, Turn off, Turn on during (5) (sec), Open, Close and Volume up. The rule in the assembly area corresponds to a rule with the condition "If the bedroom's light is turned on, then turn on the bedroom's fan".

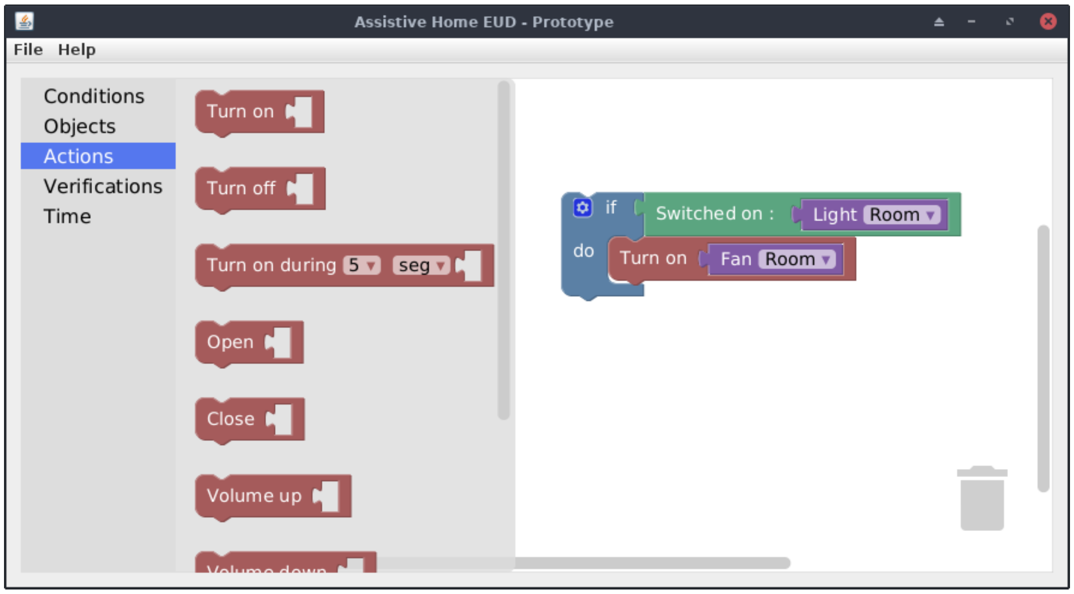

Fig. 1. Casa Assistiva EUD assembly area, with menus and blocks

Casa Assistiva EUD was connected with Eclipse Mosquitto[4], an implementation of the protocol Message Queuing Telemetry Transport (MQTT)[5]. In other words, this software is ready to connect in an actual Smart Home application using MQTT, executing the rules in a real home. However, to make the visualisation and test with users easier, we built a simulator to test the rules.

The simulator was built for two purposes. The first is to allow the user to have visual feedback on the status of objects made from the construction of rules when debugging them. For example, a light bulb is shown in black/white, and a light bulb is presented with a yellow colour when on. The second objective is to allow objects to be set in a specific state to test the algorithms. In this case, for example, the user can turn on / off the light by clicking on it.

The simulator, shown in Figure 2, depicts a real environment, in which there is a television, light bulb, fan, door, and socket, from each house's room: living room, bedroom, and kitchen. This tool also works as a remote control. When an element is turned on, the same component in real life is turned on. Figure 2 shows a limited simulation with prototype purpose contains a living room, a bedroom and a kitchen.

The bottom of the simulator, shown in Figure 2, displays components with global elements of the house.

[4]Available online at https://mosquitto.org/
[5]Available online at https://mqtt.org/

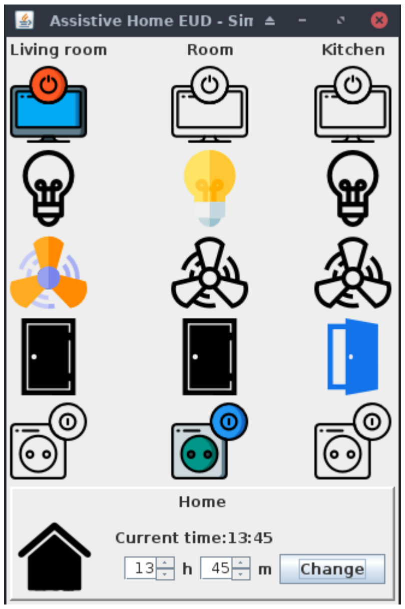

Fig. 2. Casa Assistiva EUD simulator

There are two components to simulate real-life events in debugging tasks with the simulator: the component “I am at home” (house button) and the component “change current time”.

The component “I am at home” simulates the complex task of identification if the user is at home. In real life, this feature would use images from cameras, together with the smartphone GPS position, or with a presence sensor information, and a suite of components to determine if the house’s real dweller is at home. The simulator uses the house button to identify it, only for tests.

The component “change current time” sets the time of the day to test rules with scheduled events. E.g. if there is a rule that the lights will turn off at 9 PM, the user may change the current time to 8:59 PM to see the change happening.

3.4 Procedures for Usability Tests

The user tests were conducted remotely. For this, we used Google Meet for video conferencing and Chrome Remote Desktop, a browser extension to enable the participants to remotely control the test facilitator’s computer once the application was installed in that computer to avoid hassle for the participants in installing the application in their computer. We also recorded the user’s face and the system screens - main application and simulator-with the participants’ previous authorisation.

At first, we presented the Informed Consent Form, explained the research purpose and collected participants’ signature by e-mail. We also asked for authorization to record before and right after it started.

Then, the tests proceeded with a short 5-minute video to present the Casa Assistiva EUD application. The video provided only an introduction about the test procedures itself and a short demonstration of the prototype, pointing its tools and parts - for example, we indicated the menu, the space to create the rules and the simulator, showing its goal verbally. At the end of the video, the presenter assembled a rule and tested it on the simulator to provide a quick practical example. It is important to highlight that none of the participants had explanations about programming logic or structure, and the video was shown only once per user.

The participants had no help during the tests. The mediator's role was to accompany the users and instigate them without giving any information or clue. They also had at most 30 minutes to complete each task (however, none of them used the maximum time) and the choice to pass a task whenever they wanted.

Three scenarios were created to assist the understanding of the tasks, described as follows.

(1) Whenever you get home, you like to sit on the living room sofa and watch TV to rest. Create a rule to tell the house to turn on the living room TV when you arrive.
(2) You love to cook. Every day at 8 p.m. you prepare something to eat and like to be distracted watching TV in the meantime. Create a rule to tell the house to turn on the kitchen light and TV at 8 p.m. to get there with everything on.
(3) Your bedroom is where you most enjoy spending time at home. Since your room is stuffy and gets hot when everything is closed, you would like the fan to turn on for 30 seconds each time you enter the room. You also enjoy being in the dark while watching TV. Create a rule to tell the home to turn on the fan for 30 seconds when you open the bedroom door, and another rule to turn off the light while the bedroom TV is on.

Figure 3 shows the ideal solutions to the three scenarios created with the blocks structures.

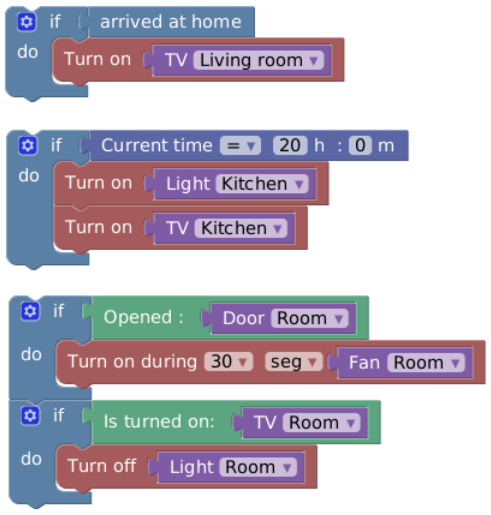

Fig. 3. Task scenarios ideal solutions

In the end, we applied a short post-task interview with the participants and asked them to fill out a satisfaction questionnaire, the System Usability Scale (SUS) [5].

3.5 Data Analysis

Data analysis employed a thematic analysis [3] of the usability problems identified in the videos' content analysis by recording excerpts in which participants committed errors and mistakes or expressed difficulties with a given feature.

The problems in the recorded videos were transcribed by the first author, who recorded a description of all usability problems identified in the videos. Transcriptions and coding were performed using online word processors, to allow collaboration between the authors. Besides describing each problem, the researcher also assigned a severity rating to the problem, based on the impact it had on the user's task. The classification was adapted from Nielsen's severity rating scale [32]:

(1) **Cosmetic problem**: need not be fixed unless extra time is available on project (nuisance, but does not impact on the task);
(2) **Minor usability problem**: fixing this should be given low priority (little impact on tasks, users can recover quickly);
(3) **Major usability problem**: important to fix, so should be given high priority (severe impact on tasks, recovery can occur, but with substantial effort);
(4) **Usability catastrophe**: imperative to fix this before product can be released (very severe impact, can prevent users from completing their tasks).

Following the identification of the problems, the next stage involved an open-coding of the issues identified.

In the first round, two of the authors (the first and the last author) performed an open-coding analysis to generate the initial sets of codes to categorise participants' problems, analysing 80 problems. This initial open-coding generated 31 categories.

After the first round of open-coding, the first version of the categorization instrument was used to assess inter-coder reliability. Inter-coder reliability was evaluated, employing a random sample of 30 usability problems by two coders. The calculated unweighted Cohen's kappa [10] in the first round was $\kappa = 0.325$, considered minimal [30].

Given the low agreement level in the first round, the two coders reviewed their coding and problems in which there were disagreements. The categorization instrument was revised, with the inclusion of new categories and revision of unclear, duplicated or imprecise descriptions. After this, a new categorization scheme was used in the second round of independent coding with the same two coders on another set of randomly selected 30 instances of usability problems. The calculated unweighted Cohen's kappa in the second round was $\kappa = 0.83$, considered strong [30].

After achieving an acceptable inter-coder reliability level, eventual disagreements were discussed and resolved between the two coders. The final version of the categorization instrument had a total of 40 categories. Using the consolidated categorization instrument, the two coders split the problem instances into two sets, which were coded independently.

Following the coding of the complete set of usability problems, the next stage of the thematic analysis involved identifying themes around the codes for the categories identified in the categorization round. The following themes were identified: Condition Blocks, Action Blocks, States and Actions, Time-related Tasks, Block Configuration and Personalisation, Menu Structure (Information Architecture), Programming Logic, Conceptual Model of Smart Homes, Simulator and Debugging, Help and Technical Problems. The full list of themes and codes are listed in Appendix A.

The description of the themes, categories and usability problems identified are presented in Section 4.

4 RESULTS

This section presents the main results obtained in the study, describing task completion rates, the types of problems identified and the results from the user satisfaction questionnaires.

4.1 Task Completion

Task completion analysis considered that tasks could be in three possible states: completed, partially completed, and failed. Completed tasks were considered the ones that generated the same home behaviour expected by the scenario, even if the user created reverse rules or used more blocks than necessary. Partially completed tasks were rules with conceptual models faults, states and actions errors and cases in which the chosen blocks and their number (for best solution) were correct, but some fit or configuration was wrong. Finally, failed tasks refer to rules that generate different home behaviours specified by the scenario or participants who quit the task.

Table 1 presents a summary with the task completion rates for programmers and non-programmers for each task undertaken during the tests and the total of tasks completed, partially completed and failed for the two groups. A Chi-Square test showed a significant difference in the proportion of task completion rates between programmers and non-programmers ($\chi^2 = 15.172, df = 2, p - value < 0.001$), with non-programmers having a lower rate of task completion.

Table 1. Task completion rates for programmers and non-programmers.

Task completion	**Programmers**			**Non-Programmers**		
	Completed	**Partially completed**	**Failed**	**Completed**	**Partially completed**	**Failed**
Task 1	4	6	0	2	4	4
Task 2	9	1	0	4	2	4
Task 3	7	3	0	3	4	3
Total	20 (66.6%)	10 (33.4%)	0	9 (30%)	10 (33.3%)	11 (36.7%)

4.2 Usability problems and user errors identified

This section presents a report of the main types of usability problems and user errors identified in the evaluations. The usability tests with ten programmers and ten non-programmers yielded a corpus of 247 problem instances, being 80 from programmers and 167 from non-programmers.

A Mann-Whitney test found a significant difference between the number of problem instances encountered by each programmer and non-programmer participant ($U = 5$, N=10,10, $p - value < 0.001$), with an average of 8 usability problems per programmer, and 16.7 problems per non-programmer user.

Regarding the severity levels of the problems encountered by users, we calculated the median severity rating of the problems encountered by each user. A Mann-Whitney test found a significant difference between the median severity rating of the problems encountered by each programmer and non-programmer ($U = 17.5$, N=10,10, $p - value < 0.01$), with non-programmers with higher median severity ratings for their problems than the problems encountered by programmers.

The following subsections are organised according to the thematic analysis's main emerging themes, as described in Section 3.5. In each subsection, the paper presents the categories of usability problems related to that theme, the number of instances and median severity of the problems in that category for programmers and non-programmers.

4.2.1 Condition Blocks. Table 2 presents a summary of the categories related to the theme "Condition Blocks" . The theme was the one with the most categories and the most frequent theme.

Table 2. Categories of usability problems related to Condition Blocks

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**instances**
Difficulty in understanding execution order of actions above and below (before/after) condition blocks	4	7 (7.87%)	4	19 (21.35%)	26
Affordance problem of condition block, attempt to fit object in place of verification	1	1 (1.12%)	3	14 (15.73%)	15
Lack of visibility/comprehension about being able to conjoin more than one action in the "do"	1	5 (5.62%)	4	10 (11.24%)	15
Affordance problem of condition block, attempt to fit action in place of verification	1	5 (5.62%)	2	6 (6.74%)	11
Affordance problem of condition block, attempt to fit object in place of action	-	-	1	8 (8.99%)	8
Lack of visible way to disconnect blocks	1	2 (2.25%)	2	5 (5.62%)	7
Affordance problem of condition block, attempt to fit verification in place of action	-	-	1	4 (4.5%)	4
Difficulty to identify if have difference conjoin condition blocks sequentially or not to build more than one rule	1.5	2 (2.25%)	-	-	2
Block "and" unavailable to create complex conditions	1	1 (1.12%)	-	-	1

Condition blocks had several problems related to lack of affordance issues. Those problems affected mostly non-programmers. Users often tried to fit blocks in the wrong places, as shown in Table 2. The worst cases included situations in which they attempt to fit actions into verification places, as shown in Figure 4. In this example, the user tries to connect “Open” next to “if”. When the system did not allow it, some participants move it to above the condition aiming to build a condition (which meant it would be executed sequentially, not as a condition) - see Figure 5.

Figure 6 illustrates another relevant type of problem, in which a user attempts to fit objects, e.g. “Door - Bedroom”, next to “if” as verification, once they had the same puzzle fitting. Although this was a recurring problem, most users quickly recovered from it. Also, the affordance of the “do” fitting showed to be ineffective. Many users hooked actions below the condition block instead of right after the sentence. Most of the participants recovered. However, others finished all three tasks building the rules with this incorrect procedure.

Nevertheless, users had difficulty in perceiving the possibility to conjoin actions inside the “do”. Exemplified by Figure 7, after assembling the first action, the barely visible space to group blocks led the users to build two rules with the same verification.

Users also became confused to disconnect some blocks groups. In every pair or group of assembled blocks, there is a base block that moves the composition. Participants frequently had difficulty disconnecting the blocks and frequently deleted them to start from the beginning, instead of only rearranging them with the new element.

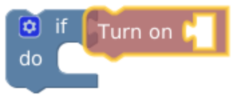

Fig. 4. Affordance problem of condition block, attempt to fit action in place of verification

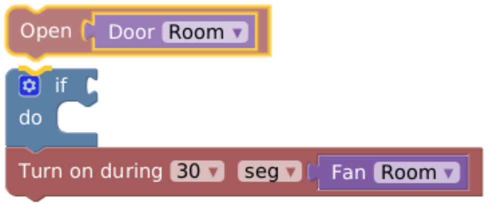

Fig. 5. Difficulty in understanding execution order of actions above and below (before/after) condition blocks

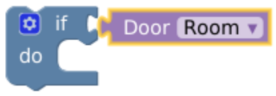

Fig. 6. Affordance problem of condition block, attempt to fit object in place of verification

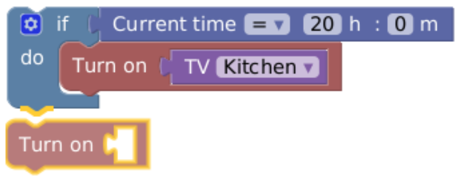

Fig. 7. Lack of visibility/comprehension about being able to conjoin more than one action inside the "do"

4.2.2 Action Blocks.

Table 3 summarizes the categories of usability problems related to the theme “action blocks”.

Users had some problems when trying to connect action blocks with objects. First, some participants tried to fit two objects in one action for tasks with two actions of the same type in a single condition block. All participants realized the necessity of picking two actions right after.

Besides, two non-programmers had difficulty understanding the slot to fit the objects in the actions. One of them only understood it in the third task, while the other solved it, despite taking a long time. This participant, in particular, was having connection problems, and the image was in low resolution. He stated the following about the slot: “This looks like a door picture for me, I didn’t think that I should connect (the object) here”.

Table 3. Categories of usability problems related to Action Blocks

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Attempt to fit two objects into one action block	1	2 (28.57%)	1	2 (28.57%)	4
Affordance problem of block fitting between objects and actions	-	-	2	3 (42.86%)	3

Table 4. Categories of usability problems related to States and Actions

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Difficulty to differentiate state and action for verifications	3	5 (41.67%)	4	7 (58.33%)	12

4.2.3 States and Actions. Table 4 presents a list of problems related to the theme “States and Actions”.

Rule conditions can be made with states of objects or actions executed on them, e.g. “is turned on” and “switched on”, as shown in Figure 8. If it is chosen incorrectly, the smart home can perform unexpected behaviours. Participants occasionally did not differentiate states and actions to pick up the correct verification to build the rules. Programmers were more insightful to reverse it when the situation occurred, while non-programmers generally picked a block and went until the end of the task. None of the participants who had this problem could test the rule pointedly to discover the error. The participants who managed to revert it looked again at the options right after picking the wrong verification.

Possible causes of the problem include the difficulty in understanding the difference between states (verifications) and conditions, the lack of contextual selection of applicable objects/actions and affordance of the connectors in the blocks.

4.2.4 Time-related tasks. Table 5 presents categories of problems with time-related tasks.

Table 5. Categories of usability problems with Time-Related Tasks

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Difficulty in interpreting the role and how to fit the block “Current time” in the context of a condition	-	-	4	4 (40.0%)	4
Difficulty in identifying blocks for time related action s	1	2 (20.0%)	2	1 (10.0%)	3
Difficulty in interpreting logical-mathematical operators to make verification in certain times	-	-	3.5	2 (20.0%)	2
Difficulty in comprehending how to use the “Random” time block	-	-	4	1 (10.0%)	1

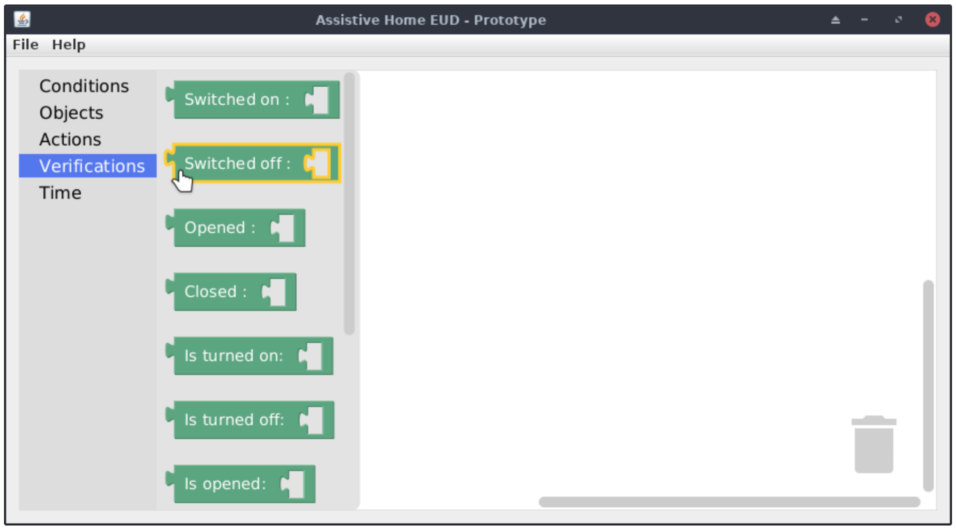

Fig. 8. “Verifications” in the menu, with states and actions conditions

These problems were associated with rules involving time-related elements, both in verifications and in actions. With 80% of the instances, non-programmers participants were more frequently affected by them than programmers.

Participants had difficulty interpreting the role and how to fit the block “Current time” in the context of condition blocks, as Figure 9 demonstrates. This issue became worse when it was combined with a problem of a misconception of the conceptual model to create the rules, which is discussed in Section 4.2.8. However, sometimes it was a momentary confusion of where to fit that block, trying to combine it with an action. This fact also describes another issue, the difficulty of identifying the Action block “Turn on during”.

Furthermore, non-programmer participant NP-02 had trouble interpreting logical-mathematical operators to define the time of a rule, including not seeing the “=” operator, which was the standard option - see Figure 10. The participant also reacted: “Wow... This bigger than and smaller than arrows are complicated to people with dyslexia”.

Lastly, one participant misused the “Random” time block, trying to make a rule to arrive at home at random times of the day instead of using the verification block “arrived at home”.

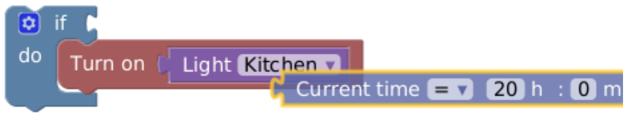

Fig. 9. Difficulty to interpret the role and how to fit the block “Current time” in the context of a condition: user trying to connect “Current time” in the action block

4.2.5 Configuration and personalisation of blocks operation. Table 6 presents the categories of problem instances related to the theme Configuration and personalisation of blocks operation.

Problems related to this occurred when users were personalizing condition blocks with combinations of “else if” and “else” or expectations about the blocks’ behaviours and operation in general.

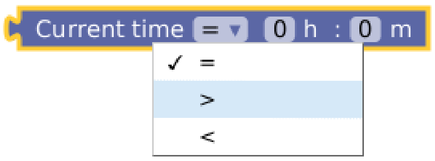

Fig. 10. Configuring time with logical-mathematical operators

Table 6. Categories of usability problems related to Configuration and personalisation of blocks operation

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Difficulty in understanding the configuration of the functioning of condition blocks and their parts	-	-	2	1 (33.34%)	1
Difficulty to identify how to leave the configuration pop-up of condition blocks	-	-	1	1 (33.33%)	1
Expected block functionality does not exist	-	-	1	1 (33.33%)	1

The evaluated prototype has standard options of condition blocks to choose from and allows the users to personalise them according to their needs, including and excluding clauses. Only a few programmers used this functionality, although more participants had entered in the pop-up configuration screen. The participants had problems understanding how to build the personalisation features and leave the pop-up screen, once the only way to quit was to click again on the gear button, “Get out of my screen, pal!”, said P-02 talking playfully. Figure 11 shows the configuration of condition blocks pop-up screen with a user trying to add an "else" in the "if" block out of the canvas instead of in it.

Another participant expected that placing a block over another would exchange all the connections and delete the previous one.

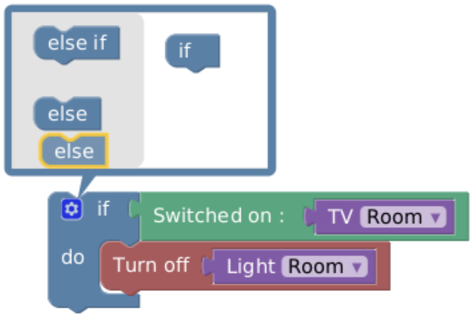

Fig. 11. User trying to personalize condition on the configuration pop-up screen

4.2.6 *Menu structure.*

Table 7 presents problems related to the menu structure (information architecture) with blocks and devices available.

Table 7. Categories of usability problems related to menu structures

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Problem with recognition, differentiation and understanding of what are the Actions and Verifications categories in the menu	1	8 (20.51%)	2	9 (23.08%)	17
Actions and Verifications options aren't similarly grouped to favor identification	1.5	6 (15.39%)	1.5	6 (15.39%)	12
Absence of means to view room options in the menu	1	2 (5.13%)	1	3 (7.69%)	5
Error in identifying verifications with similar labels	2	3 (7.69%)	-	-	3
Barely visible scrollbar	-	-	4	1 (2.56%)	1
Elements typography makes reading difficult	1	1 (2.56%)	-	-	1

The menu structure showed to be an essential tool to enable efficient programming of rules. In this theme, both programmers and non-programmers were similarly affected in the number of instances and severity.

Users, in general, had difficulty understanding, recognizing and differentiating the categories, primarily actions and verifications. It was customary to spend some time looking for a block in the wrong place and then switching to another. This confusion even made users pick the wrong blocks. For example, to verify if a door was opened, users commonly entered the Action category and picked up the "Open" block instead of the "Opened" in the Verification category. In other words, users were lost in the menu's categories even though it was relatively small. To non-programmers, this problem used to be worse since it has a lower recovery rate compared to programmers. In general, programmers had a better memorability of the items' purpose, while non-programmers repeated the same mistake a few minutes after recovering.

The organization of the blocks inside the categories is as important as the menu labels. Participants used to choose the wrong blocks because those with similar labels are apart in the menu like "Turn on" and "Turn on during". They also picked the wrong verifications with similar labels, like "is turned off" and "is turned on". Figure 1 presents the Action menu, and two blocks "turn on" can be seen (one "Turn on" and other for time - "Turn on during") separated by a "Turn off". Figure 8 shows the distance between states and actions verifications of the same word. These situations may lead some users to pick a block precociously on some occasion.

4.2.7 Programming logic. Table 8 presents the categories of problems related to the theme "Programming logic".

Both programmers and non-programmers had logic problems to create the rules. Programmers made mistakes on wrongly nesting conditions, but with quick recovery. They also had difficulty building reverse rules inside the "else", creating rules that did not represent their real wish. In the example shown in Figure 12, the participant built an "else turn off the TV - Living Room" to be task 1's reverse rule.

Non-programmers had more difficulty in interpreting the "else" clause and using it. Apart from the problem mentioned above, some of the participants also picked up conditions with "else" to make two rules or even fit an action meant to

Table 8. Categories of usability problems related to Programming logic

Category	Programmers		Non-Programmers		Total
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Logic error in condition chaining	3	3 (15.0%)	3	5 (25.0%)	8
Difficulty understanding the structure and functioning of the "else"	-	-	4	6 (30.0%)	6
Problem with the concept of the expression and functioning of the "if"	-	-	3	6 (30.0%)	6

be a second action in the “do”. In Brazilian Portuguese, “if” writes “se” and “else” is “senão”. However, “if not” writes as “se não” (with a space compared to “senão”). Thus, two participants had problems interpreting which was the correct meaning.

The misunderstanding of the “if” clause was also a common problem to build the blocks. One participant even left the verification space empty in the first task, although he knew that something was missing.

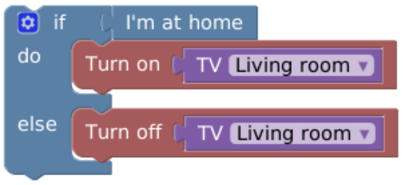

Fig. 12. Use of “else” (*senão*) to make reverse rule, instance of Logic error in condition chaining

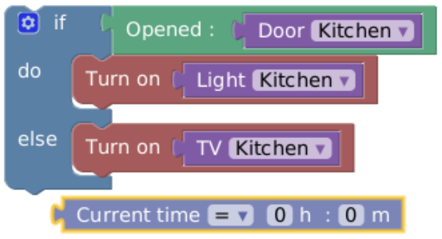

Fig. 13. Example of Programming Logic’s and Conceptual Model’s problems, respectively: (1) Difficulty understanding the structure and functioning of the "else"; (2) Conceptual difficulty for elaborating rules

4.2.8 Conceptual model of smart homes. Table 9 presents categories of usability problems that were related to incompatible users’ conceptual models of smart homes.

Problems in categories in this theme occurred when users employed incompatible conceptual models to create rules. In such cases, the rules users created behaved differently in how smart homes’ implementation takes place. It is worth noting that most study participants reported they had never experienced smart home technologies before.

The most recurring problem was to define “arrive at home” as opening the living room door and even as arriving at a time. Several participants did not even try to search for other options that could fit better for the task. Some others also

Table 9. Categories of usability problems related to the Conceptual model of smart homes

Category	Programmers		Non-Programmers		Total
	Median severity	Problem instances - N (%)	Median severity	Problem instances - N (%)	Total instances
Conceptual difficulty elaborating rules	3	3 (12.0%)	3	10 (40.0%)	13
Difficulty in comprehending that commands defined by system rules works together with another ways to interact with the appliances	4	4 (16%)	4	8 (32.0%)	5

understood that time verifications that affect a specific room would have to be combined with another condition in that room, for instance, turn on the kitchen light at a specific time and enter the kitchen, as shown in Figure 13.

Another conceptual problem was the difficulty of comprehending that rules defined by rules are one way to interact with the appliances, like manually, for example. Several participants expected that when a verification turns false, the reverse rule would be automatic. After testing the first task, P-01 said: "And now it won't turn off..." and then built the reverse rule.

In this way, non-programmers had difficulty testing rules in which the actions were not specified in the blocks. To turn off a light that was not on, some people decided to build a rule to turn on the light instead of merely turning it on manually before testing the rule. Figure 14 illustrates this, the "if" condition without verification has a "turn on" connected to "light - bedroom", which is a manual interaction that precedes the rule proposed in task 3.

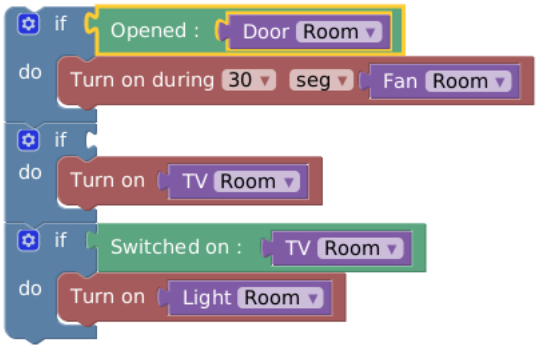

Fig. 14. Construction of rule to represent manual interaction, instance of Difficulty in comprehending that commands defined by system rules works together with another ways to interact with the appliances

4.2.9 Simulator and Debugging. Table 10 presents problems related to the use of the simulator and attempts to debug generated rules.

The simulator was a critical element in the task process. Several errors have been fixed after testing the rules. However, programmers were considerably more apt to use this tool to assist the rule building process.

Table 10. Categories of usability problems related to the Simulator and Debugging

Category	Programmers		Non-Programmers		Total
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Difficulty to recognize how to use the simulator to test the rules	4	3 (7.5%)	4	20 (50.0%)	23
Sequence of actions for task closure is not clear	2	3 (7.5%)	3	2 (5.0%)	5
Representative element of home appliance and actions aren't easy to recognize	2	3 (7.5%)	1	1 (2.5%)	4
Data loss with clock refresh when filling time forms	1	1 (2.5%)	2	2 (5.0%)	3
Simulator doesn't help to test timed tasks with seconds	1	3 (7.5%)	-	-	3
Simulator doesn't automatically debug problems in the rules	-	-	4	1 (2.5%)	1
Expected functionality doesn't exist in the simulator	-	-	1	1 (2.5%)	1

Many users had difficulty understanding how to use the simulator (only one was an expert). Often they finished a task without testing or even used the mechanism before making the block rules. Users who did not test the rules pointed out that they forgot of it, and the lack of use could lead to this behaviour.

Many users had difficulty understanding how to use the simulator (only one was an expert). Often they finished a task without testing or even used the mechanism before making the block rules. Users who did not test the rules pointed out that they forgot about testing. The lack of previous experience with programming and testing could lead to this behaviour.

Some participants showed difficulty to interpret figurative elements such as power plug and being at home. Task closure problems appeared in time changing, in which users had to fill the forms and forgot to click on the "Change" button and to begin tests. One participant searched for an option similar to compile the rules.

Users also wanted some more time-testing features, such as fast forward, seconds fields and the possibility that some fields worked in circles, like in clocks (e.g. minutes field go from 0 to 59 when pressing the 'up' button).

4.2.10 Help. Table 11 presents the problem and severity related to the theme "Help".

Table 11. Categories of usability problems related to Help

Category	Programmers		Non-Programmers		Total
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
Lack of detailed use description in Help	2	1 (100%)	-	-	1

Only one participant (P-09) searched for assistance in the "Help" button, but it had no content. Even so, she managed to do what she wanted in the end. This problem is beyond the block programming and violates several of Nielsen's heuristics once the button exists, but the function was not implemented.

4.2.11 Technical problems. Table 12 presents the problem related to technical problems and their impact on participants usage.

Table 12. Categories of usability problems related to technical problems

Category	**Programmers**		**Non-Programmers**		**Total**
	Median severity	**Problem instances - N (%)**	**Median severity**	**Problem instances - N (%)**	**Total instances**
User action cause unexpected behaviour of system inoperability	1	1	-	-	1

When moving a block, if the right mouse button was pressed, the block froze. One participant (P-03) discovered this bug by accident. Another inoperability that happened was an error message without appropriate description and that, but it could not be recorded and consequently documented as an error on the analysis. This message appears when users use the simulator, but the rule blocks had missing parts. Moreover, one of the problems discussed in the Simulator section would be fixed if this message was appropriate.

4.3 Acceptance evaluation

The SUS questionnaire achieved satisfactory results both from programmers and non-programmers evaluations. Table 13 also shows an expected clear difference between the two groups, as non-programmers had more usability problem instances, as well as lower acceptance ratings. Each statement was rated by participants on a scale from 1 (Strongly disagree), through 3 (neither agree nor disagree) to 5 (Strongly agree).

For non-programmers, question 9 had the highest difference with programmers, showing a significant difference in their confidence comparing to programmers.

Non-programmers also had worse ratings in terms of finding the system unnecessarily complex than programmers. They also had lower ratings than programmers in finding the system easy to use and finding the various functions well integrated.

Programmers had a similar rating to non-programmers' in thinking that most people would learn to use this system quickly, showing that they too would have concerns for the system's learnability.

We also calculated the SUS score using the average value, finding 93.75 for programmers, 75 for non-programmers and 82.5 for global evaluations. A Mann-Whitney test found a significant difference between SUS scores by programmers and non-programmers ($U = 16$, N=20, p-value < 0.001).

5 DISCUSSION

This section presents the discussion of the main findings from this study, how they relate previous studies in the literature, their implications, and the study's limitations.

5.1 Can Block-based Programming Help First-time End-Users?

End-user development using visual programming with blocks has shown significant promise in the context of basic programming and computational thinking education [26, 40, 46]. One of the points we investigated in the present study was whether block-based programming would translate into more immediate learning ease to develop rules for smart homes for non-programmers.

Table 13. Median ratings from programmers and non-programmers in the System Usability Scale questionnaire

Question	Programmers median	Non-programmers median	Global median
1. I think that I would like to use this system frequently	4.5	4	4
2. I found the system unnecessarily complex	1	2.5	2
3. I thought the system was easy to use	5	3.5	4
4. I think that I would need the support of a technical person to be able to use this system	1.5	2	2
5. I found the various functions in this system were well integrated	5	4	5
6. I thought there was too much inconsistency in this system	1	1	1
7. I would imagine that most people would learn to use this system very quickly	4	4	4
8. I found the system very cumbersome to use	1.5	2	2
9. I felt very confident using this system	5	3	4
10. I needed to learn a lot of things before I could get going with the system	1	1	1

However, we observed that the low task-completion rates and the problems encountered by non-programmers when attempting to perform the tasks show that this approach alone may not necessarily mean a less-steep learning curve. We acknowledge that the video users watched before the task was concise, as it did not intend to provide programming instruction. However, it was interesting to observe the outcome of non-programmers' attempts without training before using the tools, which was our intended outcome.

If block-based programming for end-user development of smart home rules is intended to enable non-programmers to develop rules with little training, more effort needs to be dedicated to designing the end-user development tools provided to them. As we discuss in some of the derived recommendations, contextualized help and demonstrations could help block-based programming be more effective to help users learn as they attempt to perform their tasks.

On the other hand, we observed that non-programmers at least attempted to use the blocks and formulate rules, even if they were incorrect. That behaviour could have been different if they were put off by another type of interface they might not feel comfortable in a first experience.

5.2 Supporting Participants to Elaborate Time-Related Rules

As pointed out by participants in the study conducted by Demeure *et al.* [14], participants in the present study also encountered issues that reinforce the importance of designing appropriate resources to support time-related tasks in smart homes.

Issues related to time blocks and operation with time were the source of many problems identified in the study. Problems in this theme reported in Section 4.2.4 included difficulty to interpret the role and how to fit the block "Current time" in the context of a condition, difficulty in identifying blocks for time-related actions, difficulty to interpret logical-mathematical operators to make verification in certain times and difficulty in comprehending how to use the "Random" time block.

The study showed the importance of designing time elements to match users' mental models of time operations. Textual descriptions related to time, with descriptions such as "random time" or "current time" did not seem to be adequate to depict what users would think of representations for these concepts.

Different approaches could include different symbols and representations of time that could be easier to recognise and learn for participants.

5.3 Organization of Devices, Conditions and Resources

As in other studies [1, 7], the present study also had many problems related to the organisation of elements in menu structures or sets of devices and resources.

End-user development tools for smart homes need more studies to devise information architecture organizational schemes that favour the types of tasks that users typically do in those systems. Inappropriate grouping of elements and similar conditions and devices often cause problems and difficulties for users.

The results in the present study showed many issues that need appropriate information architecture studies, including many problems reported in Section 4.2.6 related to menu structure, especially to organize concepts concerning conditions, verifications actions and objects.

5.4 Conceptual Models of Smart Homes and Low Exposure to the Technology

As well as previous studies [21, 27, 33], this study also revealed problems concerning inconsistencies in user's conceptual models of smart home and how they should work in context.

However, a surprising finding in this study was encountering such problem with incompatible conceptual models of smart home even with participants with previous experience with programming. Results from Section 4.2.8 revealed that problems with conceptual models of smart homes also occurred with programmers. This finding shows how important it is to illustrate how the rules will work in a real scenario when they are being made, even for programmers.

The problems with conceptual models, even with programmers, may also be related to the participants' low exposure to smart home technologies. As reported in Section 3, even amongst the participants who had experience with programming, most of which were enrolled in Computer Science-related degrees, very few had experience programming home devices. This finding may be related to the general low penetration of smart home technologies in the Brazilian market compared to other more developed countries.

It would be interesting to investigate if this also occurs in other developing countries. This research type could help develop design guidelines to help incorporate conceptual clues in designing end-user development tools for smart homes, which could benefit even programmers.

5.5 States and Actions in Block-based Programming

Many previous studies have discussed problems related to difficulties in telling the difference between actions (e.g. "opened") and states ("is open") [8, 21, 33].

Like those studies, this research also revealed several problems regarding confusion between states and actions when elaborating rules, both for programmers and non-programmers. The problems reported in Section 4.2.3 showed that participants in this study found problems in differentiating states and actions for verifications.

The results showed that this problem, commonly found in studies that analyzed form-based and data-flow interfaces was also prevalent in block-based programming. The design of the blocks representing states and actions was not distinct enough to help participants recognize if they chose the incorrect option for their problem.

More research is necessary to approach design solutions to help overcome this type of problem in block-based end-user development of smart homes.

5.6 Simulation and Debugging

Recent studies have started proposing specific approaches to help overcome difficulties with simulation and debugging of end-user development of rules for smart homes and smart environments [11, 13, 27, 48].

Even though the simulator used in the present study was much more limited in terms of debugging features than those proposed in those papers devoted explicitly to debugging, the user evaluations' experience provided valuable lessons to help understand those issues.

Our findings showed that users might have serious difficulties to simulate and test time-related rules. Even experienced programmers had difficulty knowing whether a particular time had elapsed or visualizing current times. The problems reported in Section 4.2.9 show that using the simulator and debugging rules were some of the most frequent problems in the study.

We also found that more investigation is needed to improve simulators that can enact direct actions in smart homes so that users can better understand the relationship between those direct actions and the rules they are devising.

We acknowledge, however, that the simulator developed in our project to enable debugging was the first attempt with limitations in its design. This way, further conclusions cannot be drawn from the study on this topic without more specific and detailed designs and debugging resources.

5.7 Design Recommendations

From the themes and categories of problems identified in the qualitative analysis of the 247 problem instances identified from programmers and non-programmers that took part in the present study, we have derived the following set of recommendations for the design of end-user development tools for rules in smart homes.

(1) **Attention to the affordance of blocks fitting design**: Programming blocks design should help users recognize incorrect fittings and provide recommendations for possibilities, according to the role of each part of a given block.
(2) **Help users recognise constraints in the way blocks work**: Systems should help users recognise that a given block, such as a choice of object to turn on, only accepts one parameter. In such cases, a clear indication should exist, as well as a description of a possible alternative (e.g. use another action block).
(3) **The composition of blocks should be easily reversible**: when participants are trying to formulate rules, they should easily couple and detach blocks to reverse their actions.
(4) **Help users recognise the difference between states and actions**: Designers should provide a clear distinction between actions (e.g. "got opened") and states (e.g. "is open") by having noticeable differences in the design of the components that represent them.

(5) **Design time-related components and rules compatible with users' mental models**: elements designating times (e.g. current time), comparisons and conditions relating to time should use metaphors and symbols well aligned to users' mental models.
(6) **Use appropriate Information Architecture techniques to organise and label blocks and elements**: the organisation and labelling of elements should favour the types of tasks performed by users when programming to avoid incorrect choices of blocks and elements.
(7) **Provide conceptual clues of how smart homes and rules work**: It is vital to provide conceptual clues of how the rules will enact in a smart home, such as short videos, animations or other types of demonstrations, to avoid inconsistencies in users' and system's conceptual model of smart homes.
(8) **Simulators and debugging tools should help identify block structures being executed**: When non-programmers are using block-based programming for the time, knowing what blocks are being executed when simulating may help them in understanding how the rules system works and to identify possible problems.
(9) **Simulators should explicitly show users how to provoke the enactment of the rules**: users should be able to recognise how to make the rules enact and to test them by direct means.

5.8 Limitations

This study had some methodological and practical limitations that we discuss in this section.

Due to the COVID-19 social isolation measures, it was impossible to conduct the studies in the laboratory. Unfortunately, this prevented us from allowing participants to test their rules in a real smart home scenario and with actual home appliances. Evaluations using video-conference also had limitations in the interactions and observations of participants' use of interactive devices, limiting the study to only observing their facial expressions. However, the simulation tool's use provided good insights into how to contribute to understanding simulation needs.

However, it is critical to note that this study was not explicitly aimed at proposing solutions for debugging and simulation, as other related studies [11, 13, 27, 48]. This way, the study was limited because the simulator did not provide enough debugging features that could enable a more profound analysis of how to design better solutions for debugging.

The study did not perform comparative analyses of the use of block-based programming and form-based and data-flow strategies for end-user development of smart home rules. Thus, it was limited in comparing specific features of the programming strategies in the same scenarios.

The participants involved in the study could have been more varied to enable the observation of more types of users with different abilities. For example, in the group of programmers, the majority of the users were undergraduate students in the Computer Science degree. It would have been interesting to examine other people's behaviour with programming experience with other background experiences.

Unlike other studies that performed longitudinal research or in-depth analysis with participants, the present study's tasks were shorter and less complex. This way, it was impossible to identify issues with conflicting rules and the long-term use of the generated rules in the home environment.

6 CONCLUSION

This paper presented a study on the usability of a tool for end-user development of smart home rules using block-based programming. The study involved ten programmers and 10 non-programmers in the evaluation, with three tasks with varying level of difficulty.

The tests yielded 247 problems and user error instances (80 from programmers and 167 from non-programmers), with significant differences in the average number of problems per user and the median severity of problems when comparing programmers and non-programmers. Despite most non-programmers being able to experiment with the blocks, their task completion rates were significantly lower than programmers'.

Two rounds of analysis of random sets of problems instances were performed independently until acceptable inter-coder reliability was achieved. After this, the thematic analysis procedure continued with 40 categories of problems grouped around the following emerging themes, that encompassed problems related to condition blocks, action blocks, states and actions, time-related tasks, block configuration and personalization, information architecture, programming logic, the conceptual model of smart homes, simulator and debugging, help and technical problems.

From the problems and themes identified in the research and recommendations from related studies, this paper derived the following recommendations: 1) attention to the affordance of blocks fitting design, 2) help users recognise constraints in the way blocks work, 3) the composition of blocks should be easily reversible, 4) help users recognise the difference between states and actions, 5) design time-related components and rules compatible with users' mental models, 6) use appropriate Information Architecture techniques to organise and label blocks and elements, 7) provide conceptual clues of how smart homes and rules work, 8) simulators and debugging tools should help identify block structures being executed and 9) simulators should explicitly show to users how to provoke the enactment of the rules.

The results in this study are essential to improve end-user development tools for smart homes, and to show aspects in which block-based programming can enhance the use for non-programmers, and to confirm interaction aspects revealed by previous studies using form-based and data-flow approaches that also occur with block-based programming to design smart home rules.

As future work, we intend to develop a new version of the end-user development tool prototype for smart homes employing block-based programming, using the design recommendations derived from the results presented in this paper and from other studies in the literature. After the development, we intend to conduct another round of usability tests to evaluate how they may improve the learnability of end-user development tools for non-programmers.

We also intend to conduct usability evaluations of block-based end-user development tools with the participation of children with initial programming and computational thinking skills.

Future studies will also include comparing the usability of block-based programming strategies with other approaches, such as IFTTT.

7 ACKNOWLEDGEMENTS

This work was supported by CAPES, CNPq (proc. 139936/2020-3), São Paulo Research Foundation (FAPESP) (proc. 2020/05187-5) and Universidade Federal de Lavras (PIBIC/UFLA).

A SET OF CODING CATEGORIES

This appendix lists the coding categories used in the analysis of the usability problems of the study.

Condition Blocks

- BC1 - Difficulty in understanding execution order of actions above and below (before/after) condition blocks;
- BC2 - Affordance problem of condition block, attempt to fit object in place of verification;
- BC2 - Lack of visibility/comprehension about being able to conjoin more than one action in the "do";
- BC3 - Affordance problem of condition block, attempt to fit action in place of verification;

- BC4 - Affordance problem of condition block, attempt to fit object in place of action;
- BC5 - Lack of visible way to disconnect blocks;
- BC6 - Affordance problem of condition block, attempt to fit verification in place of action;
- BC7 - Difficulty in identifying if have difference conjoin condition blocks sequentially or not to build more than one rule;
- BC8 - Block “and” unavailable to create complex conditions.

Problems with action blocks

- BA1 - Attempt to fit two objects into one action block;
- BA2 - Affordance problem of block fitting between objects and actions.

States and Actions

- EA1 - Difficulty in differentiating state and action for verifications.

Time-related tasks

- AT1 - Difficulty in interpreting the role and how to fit the block “Current time” in the context of a condition;
- AT2 - Difficulty in identifying blocks for time related action;
- AT3 - Difficulty in interpreting logical-mathematical operators to make verification in certain times;
- AT4 - Difficulty in comprehending how to use the “Random”.

Personalization and configuration of block operation

- CBP1 - Difficulty in understanding the configuration of the functioning of condition blocks and their parts;
- CBP2 - Difficulty in identifying how to leave the configuration pop-up of condition blocks;
- CBP3 - Expected block functionality does not exist.

Menu structure (Information Architecture)

- MEN1 - Problem with recognition, differentiation and understanding of what are the Actions and Verifications categories in the menu;
- MEN2 - Actions and Verifications options aren’t similarly grouped to favor identification;
- MEN3 - Absence of means to view room options in the menu;
- MEN4 - Error in identifying verifications with similar labels;
- MEN5 - Barely visible scrollbar;
- MEN6 - Elements typography makes reading difficult.

Programming logics

- LP1 - Logic error in condition chaining;
- LP2 - Difficulty understanding the structure and functioning of the "else";
- LP3 - Problem with the concept of the expression and functioning of the "if"

Conceptual model of smart homes

- MC1 - Conceptual difficulty elaborating rules;
- MC2 - Difficulty in comprehending that commands defined by system rules works together with another ways to interact with the appliances.

Simulator and Debugging

Difficulty recognizing how to use the simulator to test the rules	4	3 (7.5%)	4	20 (50.0%)	23

- SIM1 - Sequence of actions for task closure is not clear;
- SIM2 - Representative element of home appliance and actions aren't easy to recognize;
- SIM3 - Data loss with clock refresh when filling time forms;
- SIM4 - Simulator doesn't help to test timed tasks with seconds;
- SIM5 - Simulator doesn't automatically debug problems in the rules;
- SIM6 - Expected functionality doesn't exist in the simulator.

Help

- AJ1 - Absence of help with detailed description of how to use.

Technical problems

- PT1 - User action cause unexpected behavior of system inoperability.

REFERENCES

[1] Ash, J., Babes, M., Cohen, G., Jalal, S., Lichtenberg, S., Littman, M., Marivate, V., Quiza, P., Ur, B., and Zhang, E. (2011). Scratchable devices: user-friendly programming for household appliances. In *International Conference on Human-Computer Interaction*, pages 137–146. Springer.

[2] Bellucci, A., Vianello, A., Florack, Y., Micallef, L., and Jacucci, G. (2019). Augmenting objects at home through programmable sensor tokens: A design journey. *International Journal of Human-Computer Studies*, 122:211–231.

[3] Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101.

[4] Brich, J., Walch, M., Rietzler, M., Weber, M., and Schaub, F. (2017). Exploring end user programming needs in home automation. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 24(2):1–35.

[5] Brooke, J. (1996). Sus: a "quick and dirty'usability. *Usability evaluation in industry*, page 189.

[6] Buzzi, M., Leporini, B., and Meattini, C. (2019). Design guidelines for web interfaces of home automation systems accessible via screen reader. *Journal of Web Engineering*, 18(4):477–512.

[7] Cabitza, F., Fogli, D., Lanzilotti, R., and Piccinno, A. (2015). End-user development in ambient intelligence: a user study. In *Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter*, pages 146–153.

[8] Caivano, D., Fogli, D., Lanzilotti, R., Piccinno, A., and Cassano, F. (2018). Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. *Journal of Systems and Software*, 144:295–313.

[9] Chesta, C., Corcella, L., Kroll, S., Manca, M., Nuss, J., Paternò, F., and Santoro, C. (2017). Enabling personalisation of remote elderly assistant applications. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*, pages 1–9.

[10] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

[11] Corno, F., De Russis, L., and Monge Roffarello, A. (2019). Empowering end users in debugging trigger-action rules. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13.

[12] Davidoff, S., Lee, M. K., Yiu, C., Zimmerman, J., and Dey, A. K. (2006). Principles of smart home control. In *International conference on ubiquitous computing*, pages 19–34. Springer.

[13] Demeure, A., Caffiau, S., Dupuy-Chessa, S., Ta, H., and du Bousquet, L. (2019). End user development: Verifying home behavior. In *Joint Proceedings HCI Engineering 2019 - Methods and Tools for Advanced Interactive Systems and Integration of Multiple Stakeholder Viewpoints co-located with 11th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2019)*, pages 27–32.

[14] Demeure, A., Caffiau, S., Elias, E., and Roux, C. (2015). Building and using home automation systems: a field study. In *International Symposium on End User Development*, pages 125–140. Springer.

[15] Fogli, D., Lanzilotti, R., and Piccinno, A. (2016). End-user development tools for the smart home: a systematic literature review. In *International Conference on Distributed, Ambient, and Pervasive Interactions*, pages 69–79. Springer.

[16] Fogli, D., Peroni, M., and Stefini, C. (2017). Imathome: Making trigger-action programming easy and fun. *Journal of Visual Languages & Computing*, 42:60–75.

[17] for blind review, A. (2018). Home eud: an environment for a block-based programming language for the generation of rules to smart homes aimed at end-users. Monograph presented at the University (Anonymous for blind review).

[18] Funk, M., Chen, L.-L., Yang, S.-W., and Chen, Y.-K. (2018). Addressing the need to capture scenarios, intentions and preferences: Interactive intentional programming in the smart home. *International Journal of Design*, 12(1):53–66.

[19] Ghiani, G., Manca, M., Paternò, F., and Santoro, C. (2016). End-user personalization of context-dependent applications in aal scenarios. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, pages 1081–1084.

[20] Google Inc. (2020). Blockly Developers Google. https://developers.google.com/blockly. [Online; accessed 29-09-2020].

[21] Huang, J. 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*, pages 215–225.

[22] Jakobi, T., Stevens, G., Castelli, N., Ogonowski, C., Schaub, F., Vindice, N., Randall, D., Tolmie, P., and Wulf, V. (2018). Evolving needs in iot control and accountability: A longitudinal study on smart home intelligibility. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–28.

[23] Katuk, N., Ku-Mahamud, K. R., Zakaria, N. H., and Maarof, M. A. (2018). Implementation and recent progress in cloud-based smart home automation systems. In *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, pages 71–77. IEEE.

[24] Leporini, B., Rosellini, M., and Forgione, N. (2020). Designing assistive technology for getting more independence for blind people when performing everyday tasks: an auditory-based tool as a case study. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–17.

[25] Lieberman, H., Paternò, F., Klann, M., and Wulf, V. (2006). End-user development: An emerging paradigm. In *End user development*, pages 1–8. Springer.

[26] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15.

[27] Manca, M., Santoro, C., Corcella, L., et al. (2019). Supporting end-user debugging of trigger-action rules for iot applications. *International Journal of Human-Computer Studies*, 123:56–69.

[28] Mattioli, A. and Paternò, F. (2021). Recommendations for creating trigger-action rules in a block-based environment. *Behaviour & Information Technology*, 0(0):1–11.

[29] Mavrommati, I. and Darzentas, J. (2007). End user tools for ambient intelligence environments: an overview. In *International Conference on Human-Computer Interaction*, pages 864–872. Springer.

[30] McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia medica: Biochemia medica*, 22(3):276–282.

[31] Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., and Uche, C. (2011). Can android app inventor bring computational thinking to k-12. In *Proc. 42nd ACM technical symposium on Computer science education (SIGCSE'11)*, pages 1–6.

[32] Nielsen, J. (1994). Severity ratings for usability problems. Available online at https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/, last accessed on 30th September 2020.

[33] Palekar, M., Fernandes, E., and Roesner, F. (2019). Analysis of the susceptibility of smart home programming interfaces to end user error. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 138–143. IEEE.

[34] Papadakis, S., Kalogiannakis, M., Zaranis, N., and Orfanakis, V. (2016). Using scratch and app inventor for teaching introductory programming in secondary education. a case study. *International Journal of Technology Enhanced Learning*, 8(3-4):217–233.

[35] Paternò, F. and Santoro, C. (2019). End-user development for personalizing applications, things, and robots. *International Journal of Human-Computer Studies*, 131:120–130.

[36] Reisinger, M., Schrammel, J., and Fröhlich, P. (2017a). Visual end-user programming in smart homes: Complexity and performance. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 331–332. IEEE.

[37] Reisinger, M. R., Schrammel, J., and Fröhlich, P. (2017b). Visual languages for smart spaces: End-user programming between data-flow and form-filling. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 165–169. IEEE.

[38] Ricquebourg, V., Menga, D., Durand, D., Marhic, B., Delahoche, L., and Loge, C. (2006). The smart home concept: our immediate future. In *2006 1st IEEE international conference on e-learning in industrial electronics*, pages 23–28. IEEE.

[39] Risteska Stojkoska, B. L. and Trivodaliev, K. V. (2017). A review of internet of things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140:1454 – 1464.

[40] Sáez-López, J.-M., Román-González, M., and Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "scratch" in five schools. *Computers & Education*, 97:129–141.

[41] Statista (2019). Smart home market worldwide. Available online at https://www.statista.com/outlook/283/100/smart-home/worldwide, last accessed on 30th September 2020.

[42] Terrier, L., Demeure, A., and Caffiau, S. (2017). Ccbl: A language for better supporting context centered programming in the smart home. *Proc. ACM Hum.-Comput. Interact.*, 1(EICS).

[43] Tsuchiya, L. D., Braga, L. F., de Faria Oliveira, O., de Bettio, R. W., and Freire, A. P. (2020). Design and evaluation of a mobile smart home interactive system with elderly users in brazil. *Personal and Ubiquitous Computing*, Online first.

[44] Ur, B., Pak Yong Ho, M., 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 2016 CHI Conference on Human Factors in Computing Systems*, pages 3227–3231.

[45] Valsamakis, Y. and Savidis, A. (2017). Sharable personal automations for ambient assisted living. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 103–110.

[46] Zhang, L. and Nouri, J. (2019). A systematic review of learning computational thinking through scratch in k-9. *Computers & Education*, 141:103607.

[47] Zhang, Q., Li, M., and Wu, Y. (2020). Smart home for elderly care: development and challenges in china. *BMC geriatrics*, 20(1):1–8.

[48] Zhao, V., Zhang, L., Wang, B., Lu, S., and Ur, B. (2020). Visualizing differences to improve end-user understanding of trigger-action programs. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–10.