CCBL: A Language for Better Supporting Context Centered Programming in the Smart Home

LÉNAÏC TERRIER, IIHM team, Institute of Engineering Univ. Grenoble Alpes
ALEXANDRE DEMEURE, IIHM team, Institute of Engineering Univ. Grenoble Alpes
SYBILLE CAFFIAU, IIHM team, Institute of Engineering Univ. Grenoble Alpes

This paper presents CCBL (Cascading Contexts Based Language), an end-user programming language dedicated to Smart Home. We design CCBL to avoid the problems encountered by end-users programming with ECA (Event Conditions Actions), which is the dominant approach in the domain. We present the results of an experiment where we asked 21 adults (11 experimented programmers and 10 non-programmers) to express four increasingly complex behaviors using both CCBL and ECA. We show that significantly less errors were made using CCBL than using ECA. From this experiment, we also propose some categorization and explanation of the errors made when using ECA and explain why users avoid these errors when programming with CCBL.

CCS Concepts: • **Computer systems organization → Human-centered computing → Human computer interaction (HCI)** → Ubiquitous and mobile computing systems and tools; • **Software and its engineering → Software notations and tools → Context specific languages** → Domain specific languages;

KEYWORDS

End User Development; Smart homes; Home automation; CCBL; ECA; Context; programming language; experimentation.

ACM Reference format:

Lénaïc Terrier, Alexandre Demeure and Sybille Caffiau. 2017. CCBL: A Language for Better Supporting Context Centered Programming in the Smart Home. *PACMHCI / EICS'17 Paper*, 1, 1, Article 13, 2017. 18 pages.
DOI: http://dx.doi.org/10.1145/3099584

1 INTRODUCTION

Smart Home is becoming a reality [1, 4, 11]; with the raise of the Internet of Things (IoT), a plethora of communicant and programmable devices is being integrated into homes (e.g., smartphones, vacuum cleaner, controllable lamps, and programmable buttons). In addition, platforms such as IFTTT (https://ifttt.com/) are making available to end-users many online services that can be integrated into the home (e.g., music, news, or even smart device controls).

Smart Home is not just about controlling devices and services in isolation one from each other's, it is also the possibility to program them so that they work together to fulfill inhabitants' needs. This can be done via dedicated home automation systems [1, 4] that act as bridges for the different devices and services, or via online services such as IFTTT. In any case, the programming language that is proposed is a variant of Event Condition Action (ECA), sometimes also called Trigger Action programming (TAP). ECA offers a low threshold: it is easy for novice users to get started with, simple behaviors are easy to program (e.g., when I press the button, send a notification to my friend).

2573-0142/2017/05 - 13 $15.00

http://dx.doi.org/10.1145/3099584

While there is evidence that average users can successfully engage in programming with ECA [16], it has been demonstrated that users may form a wrong mental model when confronted with rules of the form "if condition then sustained action" (e.g., when I am in the room then turn on the lights) [8]. With such rules, many users tend to apply a non-sequitur reasoning: they think that if the condition becomes false, then the sustained action is reversed (e.g., when I am not in the room, then lights are turned off).

In this paper, we elaborate on these observations by Huang et al. [8]. We consider smart home behaviors that associate device states (e.g., the lights are on) to contexts (e.g., I am in the room). We consider behaviors of increasing complexity in terms of the number of contexts involved as well as the number of transitions between them. For instance, let us consider "the volume of the music" as a device and the three following contexts: "Being outside home", "Being at home" and "Being at phone". Then a "complex" behavior could be "User want the volume of the music to be normal when she is at home, except when she is at phone (smartphone), In that later case, she wants the volume to be low. When she isn't at home, she wants the volume to be off."

We introduce CCBL [15] (Cascading Context Based Language), a language dedicated to programming Smart Homes. CCBL focus on the notion of context rather than events as it is for ECA. We hypothesize that, with respect to ECA, CCBL lower the design barrier [9] for end-users to program correctly complex behaviors. We also hypothesize that users programming with ECA 1) will make errors due to confusion between conditions and event parts of rules and 2) will miss to express some transitions from one context to another.

In order to test our hypothesis, we set up an experimentation involving 21 participants (11 programmers and 10 non-programmers) who had to program four behaviors of increasing complexity. We compare two languages: ECA and CCBL to express aforementioned behaviors. We measure the errors made by users and categorize them to get clues on how participant structure their programs.

We now present background and related works. We then introduce CCBL and describe its cascade mechanism. In order to compare CCBL with ECA, we describe a user study where participants had to solve exercises of increasing complexity by using both CCBL and ECA. We present the results of this study and discuss the ways participant programmed in ECA and CCBL before concluding.

2 BACKGROUND AND RELATED WORKS

Our work is related to end-user programming in the Smart Home [3]. Menniken et al. [11] give an overview of the challenges in the domain. In this section, we first present the existing variants of ECA programming before providing an overview of related work.

2.1 Event Conditions Actions (ECA) and Triggers-Actions programming (TAP)

ECA programming consists of defining a set of rules. ECA programming languages are actually derived from practice; there is no unique formal definition that we can refer to. An ECA rule is as follows: ON event occurs IF conditions are satisfied THEN do some actions.

In this paper, we will use the definition proposed by Pane et al. [14]. He states that ECA programming consists of specifying a set of rules where each of them "autonomously reacts to actively or passively detected simple or complex events by evaluating a condition or a set of conditions and by executing a reaction whenever the event happened and the condition is true". This means that [2]:

- A rule is activated only by events;
- Its execution is independent of other rules in the system;
- It implements a reaction to the incoming event;
- It contains a guarding condition to execute such actions.

Trigger-action programming (TAP) is a variant of ECA. Recent works [8, 16] used this term although authors do no explain in which way it differs from ECA. Häkkilä et al. [7] seems to be the first to refer to it.

While Dey et al. [5] do not use the term (they write about if-then rules), they propose a very similar behavior and are cited by Ur et al. [16] as an example of system that support trigger-action programming.

While none of the authors of aforementioned works do provide formal definition of trigger-action programming, we propose the following one, inspired by Häkkilä et al. [7]: Trigger action is a variant of ECA, where it is possible to express rules of the form "IF conditions THEN actions", where conditions do not make any reference to an event. The semantics of this form of rule, expressed in standard ECA, is "ON conditions become true THEN do actions".

2.2 Related works

Pane et al. [13] study how non-programmers express solutions to programming problems and find that event based programming style is the most common way to be used. In the domain of context-aware application, Dey et al. [5] show that users tend to express themselves in terms of rules (if…then…) when having to describe context aware application. This is confirmed by Ur et al. [16] who asked 318 workers on Amazon's Mechanical Turk to list five things they would want a smart home to do. They found that 62.6% of the collected behaviors were about programming. All the programs could be expressed using a TAP structure.

García-Herranz et al. [6] show that both programmers and non-programmers are able to code behavior for a smart home by using triggers, conditions and actions. They also show that both programmers and non-programmers can make the difference between triggers and conditions. However, this later result is challenged by the results we obtain in this paper. We found that non–programmers do not really perceive a difference of usage between events and conditions when programming with ECA.

Huang et al. [8] distinguish between the different types of triggers and actions. A triggers can refer to an event or a state while an action can be instantaneous (e.g., send an email), extended (e.g., brew coffee) or sustained (e.g., turn on lights). Based on that categorization, they observe that many users tend to do non-sequitur reasoning as rules of the form "When state then do sustained action" are automatically associated with the rule "When not state then do reverse sustained action". For instance, the rule "When I am at home, then lights are on" is interpreted as implying that "When I am not at home, then lights are off". We designed CCBL to take explicitly into account non-sequitur reasoning.

Chandrakana et al. [12] considers 96 home automation rules written by end-users for the openHAB framework. OpenHAB is an open-source framework that is popular among Do-It-Yourselfers home automation builders. In openHAB, a rule is composed of 1) a trigger part that enumerate all the events that may trigger the rule, and 2) an action part that contains imperative code with possible use of classical workflow instructions such as if-then-else. The authors show that 77 of the 96 rules (80%) had a trigger part that contains less events that necessary. This means that it is difficult for users to enumerate all the necessary events that are relevant to trigger a rule.

While aforementioned works study rules quite in isolation from each other, Cano et al. [2] study the coordination problems in ECA rules. They propose three categories for them: 1) Redundancy, when two rules will at least partially do the same actions when an event occurs; 2) Inconsistency when two or more rules send contradictory actions to a same device; and 3) Circularity when rules get activated continuously, without reaching a stable state. We deigned CCBL so that inconsistencies are impossible to express and circularities easy to detect.

3 CCBL

We conceived CCBL [15] based on our experience with ECA. Our design rational is based on the notion of context instead of event as in ECA. The assumption is that contexts ordered by hierarchy and priority can support non-sequitur and general/particular case reasoning. Figure 1 illustrates a program composed of several contexts.

In CCBL, there are two types of contexts: Event Context and State Context. Event contexts are composed of an event expression and a set of actions. They are semantically close to ECA rule: if the event context is active, the trigger of its event leads to the execution of its set of actions. There is no duration associated to

event contexts. On the contrary, State contexts are associated to durations. State contexts are the core contribution of CCBL with respect to ECA. They provide the building blocks to structure contexts hierarchically. A State context is composed of three parts:

- A Boolean expression and optionally a start and/or an end event: The context is active either after the system detects the start event (if it is specified) or the sys-tem evaluate the Boolean expression as true (if there is no start event that is spec-ified). The context stays active until either the system detects the end event (if it is specified) or the system evaluate the Boolean expression as false (in any case). For instance, the starting event could be "System detects that Martin enters his home", the Boolean expression could be "System detects that Martin is at phone".
- A set of actions that change the state of devices or services when the context is active (e.g., "Set volume of the music to off and set lamp A to off").
- A list of sub-contexts. Sub-contexts can be Event Contexts or State Contexts. Each sub-context can only be active if the parent context is active. For instance, one can consider the sub-context "System detects that Martin is at phone" that apply only when the context "System detects that Martin is at home" is active.

By construction, CCBL imposes to have one root state context, this state root context aims to represent the "neutral state" of the smart Home (e.g., lights are off; doors are locked). Several contexts may set the state of a device but only one has access to the device at a time to prevent inconsistencies. Unlike what happens with ECA, it is not the last who speaks who get the access. In CCBL, contexts are ordered in a contexts priority list. When several contexts try to modify a device, only the one that has the maximum index in the priority list can actually do it. If this context become inactive, then the next context in the priority list sets the state of the device. The same process happens when a new context that tries to set the device become active. The order function used in CCBL is as follow, C1 > C2 means that:

- Either C1 is a descendant context of C2.
- Or C1 and C2 have the same parent context; C1 is indexed after C2 in the list of sub-contexts.
- Or C1 has an ancestor A1 and C2 have an ancestor A2 such as A1 and A2 have the same parent context, A1 is indexed after A2 in the list of sub-contexts.

Every context specifies either explicitly or implicitly the state of all devices and services. The implicit specification of the state of devices and services is what we call the cascade mechanism. We took the inspiration form the Cascading Style Sheet (CSS) language [10] that enables styling of HTML documents. To illustrate the cascade mechanism, let us consider the CCBL program illustrated in Figure 1, if we suppose that Martin is at home and that he is at phone, then two contexts try to set the state of the lamp: the root context and the one that has the Boolean condition "System detects that Martin is at home"). The lamp lights in white because "System detects that Martin is at home" is a descendant context of the root context. If Martin leaves home while still at phone, the lamp will be set to off, as the only remaining context that try to set it will be the root context.

We designed the cascade mechanism to make it easy for users to express what happens when the system exits a context. Indeed, the devices can never be in an undefined state, at least the default root context defines what their state is. Consequently, users only have to specify contexts for which at least one device state has to be changed.

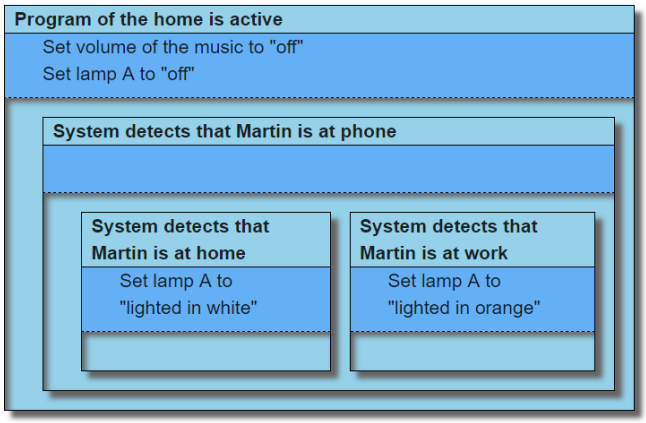

Fig. 1. Example of CCBL program. The lamp A is lighted in white when Martin is at phone at home and in orange if he is at phone at work. It is turned off otherwise.

4 EXPRESSIVE POWER OF CCBL

We believe that CCBL has the same expressive power as ECA. An ECA rule can be translated into a state context having C for condition and an event sub-context having E for condition and A for actions. The way CCBL handles a context having an event for condition is different from a condition based on a state: When the event E occurs, actions A are performed. The new devices states resulting from actions A are maintained until another context modifies them.

In this paper, we focus on the strength of CCBL over ECA: in other words the handling of state-based contexts. We illustrate CCBL and in particular its conciseness with respect to ECA through the following example: Ensuring that the security context has priority over others.

Let us suppose that the security is achieved by ensuring that the electrical outlet of the PC running the security system is always on. With CCBL, the users can achieve this by adding the security context (When security is active then Power outlet is ON) as the last child of the root context. Indeed, as the security context is the last child of the root context, this ensures by construction that the security context has the highest level of priority. The priority list therefore ensures that if the security context is active, then no others contexts will have access to the power outlet.

Now let us consider exactly the same example with ECA: 1) the users should first specify the rule for entering the security context, second 2) the users should find every other rules that modify the power outlet in order to prevent modification when security context is active. Third, the users have also to consider the point 2 when reprogramming the system. Each time a new rule is added or updated, the users should remember to check if this rule modifies the power outlet and act accordingly.

Last, let us now suppose users want to enrich the security context behavior as following: “When nobody is at home, then the door is locked”. Programming with CCBL, the users can add a “When nobody is at home” sub-context to the security context. The cascade mechanism and priority list ensure that when this sub-context ends, the system opens or closes the door lock with respect to what others active contexts specify. Programming with ECA, the users must add two rules: R1) for specifying what the system should do when the context becomes true (i.e. “When security context becomes active, then turns on the power outlet of the PC”), and R2) for specifying what the system should do when the context becomes false. Concerning this last point, the users have to analyze all others rules to know if the system should open or close the door lock. Moreover, when users add or update another rule of the system, they have to take care to update rule R2.

5 EXPERIMENT

We evaluated the use of two programming languages: ECA and CCBL. These languages were used to program Smart Home behaviors that associate device states to contexts. We compare correctness of programs edited with both languages. We also categorize errors made with ECA and try to understand their causes.

The 21 participants are French adults, aged between 18 and 51 (average: 31,5y-o standard deviation: 9,7y). None of them has a smart home system, 11 participants are programmers (engineer level) and 10 have no programming knowledge (they follow or followed studies after high school). As illustrated in Table 1. , participants were split into four groups according to their programming knowledge and the order in which they used languages during the evaluation (ECA and CCBL).

Table 1. Groups of participants

Order	Non-programmer	Programmer
CCBL then ECA	G1: 3 males, 2 females	G3: 4 males, 1 female
ECA then CCBL	G2: 1 male, 4 females	G4: 6 males

5.1 Evaluation method

Each evaluation session lasts nearly one hour. The participant and the evaluator are alone during the whole session time. It takes place either in the laboratory room or at the participant's home. The participant is placed in front of a computer, sits at a desk.

The evaluator first briefly exposes the objective of the study, he presents the smart home that will be considered for the rest of the session and explains what the sensors and actuators are. A presentation of the language was read at the beginning of the programming session. The presentation includes the core principles of each language and how the editor represents them (e.g., cascade for CCBL, event for ECA). The presenter illustrates these principles with the following example: "Martin wants lamps A to be lit in orange when he is at work, the lamp should be off otherwise.

The participant then programs a list of behaviors using CCBL and ECA. We implemented a software to support our experimentation (Figure 2). The software guides the evaluator and the participant through the experimentation process. It enables participants to edit programs and logs their actions into a database. At the beginning of the programming session, the evaluator lets the participant pick the interface for the first programming language (either ECA or CCBL) and answers questions. The evaluator asks the participant to program the following behavior: "Martin wants the lamp to be orange when he is phoning at work and green when he is phoning at home". The evaluator ensures that the participant understood the elements of the language by asking some questions (e.g., when is the lamp going to turn off?). The evaluator takes care that:

- For ECA: Participant understands that only an event can trigger a rule and that once an actuator is set, its state remains constant until the triggering of a rule sets it to a new value.
- For CCBL: Participant understands the concept of cascade and the nesting of contexts.

Participant has then to complete four exercises. The same process is repeated for the other programming language (pick up time and the 4 same exercises). At the end of the programming session, the evaluator asks the participant which language she prefers and why.

5.1.1 The prototype. The user interface of the prototype is illustrated in Fig. 1. A header indicates who the current participant is; what the current step of the process is and it enables evaluator to go to the next step. The panel on the left lists the two sensors and two sensors/actuators of the smart home:

- Alice sensor: It detects if Alice is at home, at Martin's home and if she is available. Her availability is independent of her location.
- Martin sensor: It detects if Martin is at home, at work, if he is phoning and if he sits in his sofa at home.
- Volume of the music: It sets the volume to off, low, normal or high. It can also be used as a sensor that indicates the state of the volume of the music.
- Lamp A: It sets the lamp to off, orange, green or white. It can also be used to indicate the lamp's state.

For each of them, the user interface lists related events (for ECA rules) conditions and actions (for both ECA and CCBL). The central part presents the program under edition. The footer part represent the exercise statements. As there is no standard way to represent ECA rules. We chose a representation that was consistent in terms of colors with the representation of CCBL programs. Figure 3 illustrates an ECA rule. Each rule must have one and only one event. It can have zero or several conditions and zero or several actions.

5.2.2 Exercises. The four exercises consist of associating values of the state of one device – either the color of the lamp or the volume of the music – to different contexts. We defined the four exercises to represent increasing levels of complexity. The complexity refers to the number of contexts taken into account as well as the number of possible transitions to switch from one context to another. Table 2 summarizes the exercise statements presented to participants (translated from French).

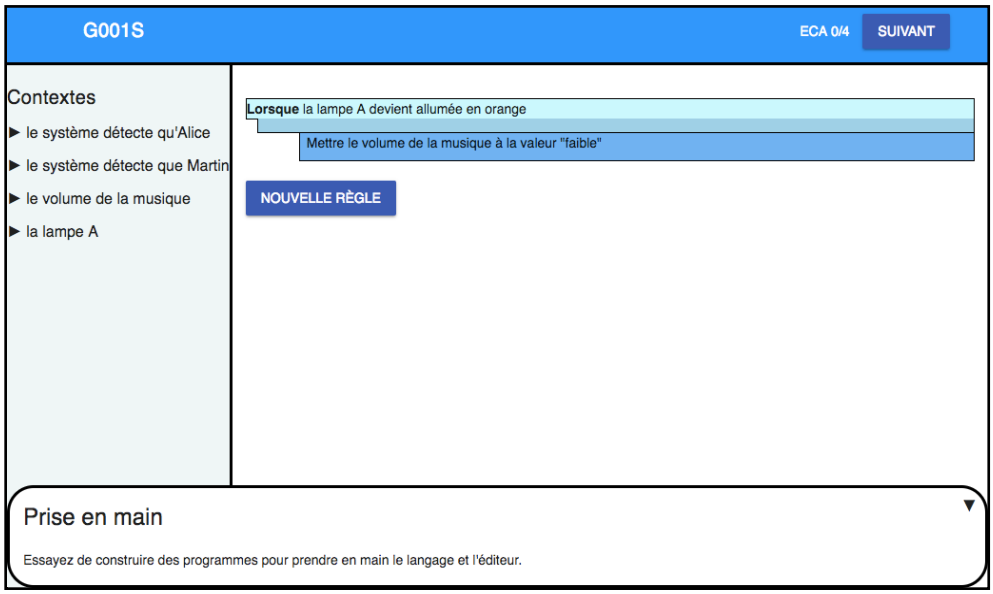

Fig. 1. User interface of the software that support the experimentation process (ECA rules editor).

Exercise 1 is the simplest one (Table 2 Ex 1). The objective is to test whether the participant really understand the fundamentals of each language. We want to check that participants do not use non-sequitur reasoning when programming with ECA and that they use the cascade mechanism when programming with CCBL.

Exercise 2 (Table 2 Ex 2) uses three contexts: "Outside", "At home" and "On sofa". The latter is a sub-context of "At home". We want to test if users are able to express general case ("Being at Home") and particular case ("Being on sofa"). This exercise is close to the exercise 1.

Exercise 3 (Table 2 Ex 3) also contains three contexts: “Outside”, “At home and at phone”, “At home and not at phone”. Figure 4 illustrates the HFSM (Hierarchical Finite State Machine) that presents the behavior of the volume of the music. A minimum of five ECA rules are needed to program the behavior. Indeed, when Martin enters his home, he can be at phone or not.

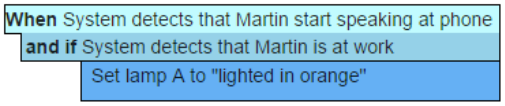

Fig. 3. An example of ECA rule. The header part represent the event that will trigger the rule. The middle part represents a set of conditions that must be fulfilled in order to execute the action (lower part of the rule).

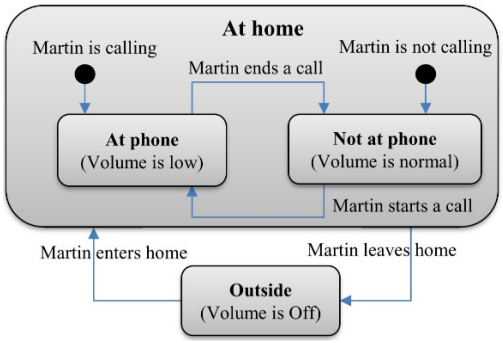

Fig. 4. HFSM of exercise 3. Three contexts are considered. When Martin enters home, the system distinguish if user is talking at phone or not.

Exercise 4 (Table 2 Ex 4) is the most complex one. It contains five contexts and programming its behavior necessitates a minimum of eleven ECA rules. We test if users identify all the triggers that may change the color of the lamp. We also test if users identify all the situations where the lamp must be turned off.

Table 2. Statements of the four exercises (originals in French)

Exercice	Statement
Ex 1	Martin wants the lamp to be lighted in white if and only if he is at home (lamp is off otherwise).
Ex 2	Martin wants the volume of the music to be low when he is at home, except when he sits in his sofa. In that later case, he wants the volume to be normal. When he isn’t at home, he wants the volume to be off.
Ex 3	Martin wants the volume of the music to be normal when he is at home, except when he is talking at phone (smartphone), In that later case, he wants the volume to be low. When he is not at home, he wants the volume to be off.
Ex 4	Martin wants to use his lamp to be aware of Alice when he is at home. When Alice is at home and not available, he wants the lamp to be lighted in orange. When she is at home and available, he wants the lamp to be lighted in green. When Martin receives Alice, he wants the lamp to be lighted in white. In all other cases, the lamp should be off.

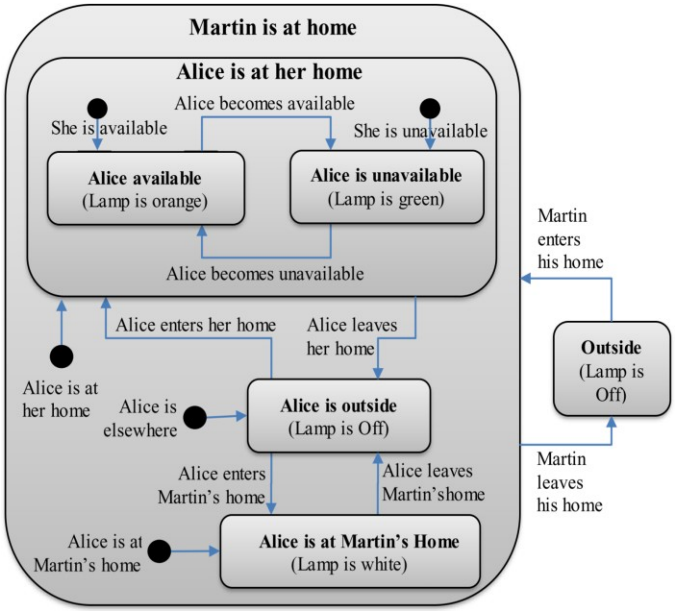

Fig. 2. HFSM of lamp behavior for exercise 4.

5.3 Collected data

We collected edited programs, SUS questionnaires and evaluator's notes. Edited programs are used to analyze the correctness: Do the program express correctly the expected behavior? We also analyze the structure of the program (e.g., does it contain a condition?) as well as the eventual errors. As few individuals compose each group, we analyze variance by computing p-value with a Fisher's exact test.

We collected one program per participant per exercise per language (21 x 4 x 2 = 168 programs). Table 3 summarizes the number of incorrect programs specified by participants for each exercise. Participants are classified with respect to their group (see Table 1). After the use of each language, we asked participants to fill a SUS-based questionnaire. At the end of the evaluation session, the evaluator asked participants which language they prefer and why.

Table 3. Number of incorrect programs edited with ECA and CCBL.

Group	Total	Ex 1		Ex 2		Ex 3		Ex 4	
	#	ECA	CCBL	ECA	CCBL	ECA	CCBL	ECA	CCBL
G1	5	0	0	3	0	5	0	5	1
G2	5	2	0	5	0	5	0	5	1
G3	5	0	0	2	0	4	0	5	0
G4	6	0	0	3	0	4	0	6	0

6 EXPERIMENTATION RESULTS

First, we show that participants do not always respect the ECA semantics and tend to express rules with a TAP semantics. Second, we show that even when taking into account TAP semantics, participants make fewer errors when programming with CCBL than with ECA. Third, we analyze the errors made by participant when using ECA and ECA with TAP semantics. Finally, we analyze the structures of CCBL programs and discuss the users' point of view.

6.1 From ECA to TAP semantic

We identified two programmers and two non-programmers who expressed at least one ECA rule in which they added a redundant condition that can only be true when the system triggers the event. For instance, we observed the following rule: "When Martin enters home and if Martin is at home then set the volume of the music to low". For theses participants the semantics difference between an event and a condition is not clear. These participants belong to groups G2 and G4 (see Table 1). They used ECA before CCBL, thus CCBL cannot have influence their mental model.

Several participants said during the sessions that they had problems to distinguish between events and conditions when programming with ECA. Sometimes, when reading the ECA programs, they transformed the event into a condition. For instance, "Martin enters his home" was pronounced "Martin is at home".

As stated before, an ECA rule is triggered by one event. When the event occurs, then conditions are evaluated. If conditions are evaluated as true, then actions are executed. However, we cannot understand several of the proposed ECA programs when using this semantics. Instead, we observe that participant blurred the distinction between event and condition. Participant considered the event as a condition. They used the following semantic: "If the conditions become true, then execute the actions". This is actually the semantics used in TAP.

Figure 6 illustrates this shift from ECA semantics to TAP semantics. It is a solution to exercise 3 proposed by a non-programmer. If we interpret this program with the ECA semantics, then it does not describe the behavior stated in Table 2 Ex 3. However, if we replace the events by their related states (e.g., "System detects that Martin starts speaking at phone" becomes "Systems detects that Martin is speaking at phone") and if we use TAP semantics, then the behavior is correct.

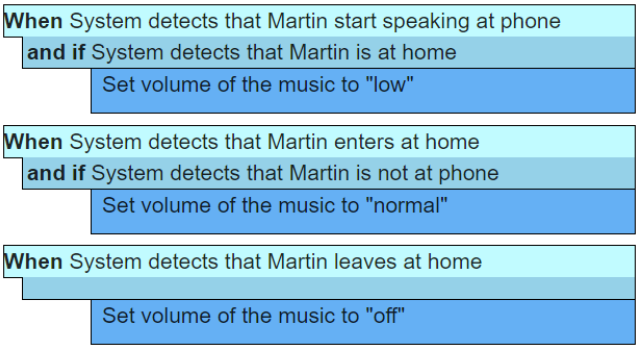

Fig. 6. Example of program where participant semantically blurs the distinction between events and conditions.

6.2 Related works

We analyze ECA programs edited by participants to check whether they used ECA or TAP semantics. We interpreted each program using both semantics and retained the semantics that made more sense with respect to the behavior that we asked participant to program. Table 4 summarizes the semantics (TAP or ECA) used by participants for each exercise. In some case, we were not able to determine which semantics was in use,

which explain why the sum of ECA and TAP semantics may be inferior to the number of participants in the group.

Table 4. Semantics used by participants for each exercise

		Ex 1		Ex 2		Ex 3		Ex 4	
Group	Total	TAP	ECA	TAP	ECA	TAP	ECA	TAP	ECA
G1	5	0	5	3	2	4	1	4	0
G2	5	1	2	4	0	4	0	4	1
G3	5	0	5	1	4	1	4	3	1
G4	6	0	5	2	3	1	4	3	3

The use of one or the other semantics (ECA or TAP) varies from one group of participants to another and according to the exercise. This variation is related to programming knowledge of participant at least for exercise 3 (p-value <0.002). Non-programmers use ECA or TAP semantics according to the exercises (p-value <0.007). This may be due to the increasing complexity of exercises or to the fact that participants tend to forget ECA semantics with time. However, with respect to data, we have no conclusive answer about how programmers use ECA or TAP semantics. We observe that they are more likely to keep using ECA semantics than non-programmers are.

6.3 Comparing CCBL and ECA (including TAP semantics)

Figure 7 summarizes the correctness of the programs specified by participants per exercise, language and programming knowledge. It takes into account that an ECA program can be correct either with the ECA semantics or with the TAP semantics. Non-programmers who correctly programmed exercises 3 and 4 used the TAP semantics, which suggest that this semantics better support users in programming behaviors that associate devices states to contexts.

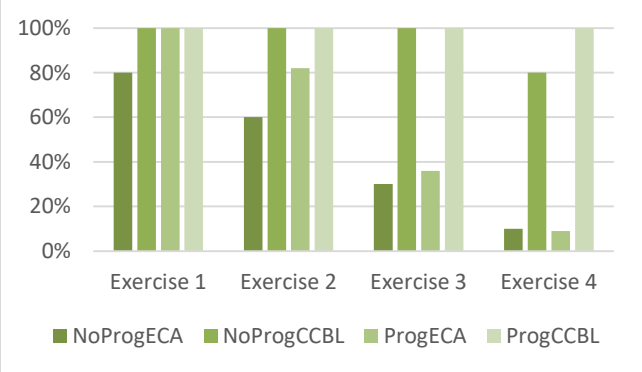

Fig. 7. Correctness of programs. We use ECA and TAP semantics to evaluate correctness of ECA programs.

When programming with ECA, the results are consistent with the level of difficulty: participants made more errors when exercises become more difficult. When programming with CCBL, only two non-

programmers made an error for exercise 4. For the exercises 1, 2 and 3, every programmers edited a correct CCBL program.

We analyze the errors made by participants and summarized in Table 3. We first test three possible factors of error: a) the order in which languages have been used (ECA then CCBL or CCBL then ECA), b) the programming knowledge of participants (programmers or non-programmers) and c) the language used for programming (ECA or CCBL). For each exercise, we compute the p-value (using Fisher's exact test) between the correctness of the programs and each of the three aforementioned factors of error (Table 5). The results do not allow us to conclude on factors a and b but do show that the correctness of programs is related to the language used for exercises 2, 3 and 4 (factor c).

Table 5. P-value for links of correctness with Order, Knowledge and Language.

	Ex 1	Ex 2	Ex 3	Ex 4
Order	0.63	0.84	1	0.96
Knowledge	0.5	0.67	1	1
Language	0.55	0.03	6.16E-6	1.55E-7

To conclude, participants make significantly fewer errors when programming with CCBL than with ECA (using ECA or TAP semantics).

6.4 Errors made by using ECA

From related work [2, 8, 12], we know that several types of errors exists when programming with ECA: non sequitur reasoning, using less triggers that necessary, redundancy, inconsistency and circularity. We observe no errors of inconsistency nor circularity. We observe only one error of redundancy (by a programmer in exercise 4). As the most complex exercise necessitates eleven rules to behave correctly, we postulate that this is not sufficient to produce this kind of errors.

Despite the fact that the evaluator tried to ensure that participants understood ECA semantics well before starting exercises, we observe that two non-programmers used non-sequitur reasoning for exercise 1. This suggest that this is a very natural way of reasoning. We observed other cases of non sequitur reasoning in exercise 2: participant specified what to do when Martin sits down on the sofa but forgot to mention what to do when he leaves the sofa. We observed similar errors in exercise 3: participants tend to forget to specify what to do when Martin hangs up the phone.

Many errors implied the use of less triggers than needed. This was especially the case for exercise 3 and 4. For instance, in exercise 4, the lamp lights in green when Martin is at home and Alice is at her home and available. This imply considering three events: "Martin enters home", "Alice enters her home" and "Alice becomes available". Most participants forgot about at least one of these events, which is consistent with what Chandrakana observed [12].

6.5 Errors made by using ECA with TAP semantics

We observe two types of errors when programming with TAP: 1) Reasoning in terms of general and particular cases and 2) Forgetting to specify what happens when contexts are exited, which can be related to non-sequitur reasoning. Figure 8 illustrates a participant using TAP semantics. Remember that every event has to be replaced with its related state in order to be interpreted correctly. The participant expresses a general case (Martin is at home) and associates it with the volume being "normal". He then expresses a particular case (Martin is at home AND he is at phone) and associates it with the volume being "low". However, this program

is incorrect. Indeed, when Martin hangs up the phone, if he is still at home, no rule will trigger so the volume will stay "low" instead of being set to "normal". We observe this error for 5/9 non-programmers that used TAP semantics for exercise 3 (three programmed it correctly and one made another error). Programmers were less likely to use TAP semantics, only two of them did it for exercise 3, and one of them made the same error (the other programmed it correctly).

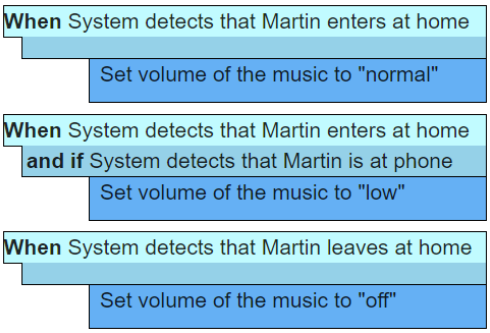

Fig. 8. Example of reasoning in terms of general case (rule at the top) and particular case (rule in the middle).

The other type of errors consists of forgetting to specify what happens when contexts are exited. We observe this type of error mainly for exercise 4. As illustrated in Figure 9, participants were able to specify (with TAP semantics) when the lamp should light in orange, green or white. Every participant also mentioned to turn it off "when Martin was not at home" (It was explicitly stated in the exercise) but forgot to mention that it was also the case when Alice was not at her home nor at Martin's home. There are two possible explanations for this type of error: 1) Participants applied non-sequitur reasoning or 2) Participants were not aware of all possible contexts (e.g., "Alice is not at her home nor at Martin's home).

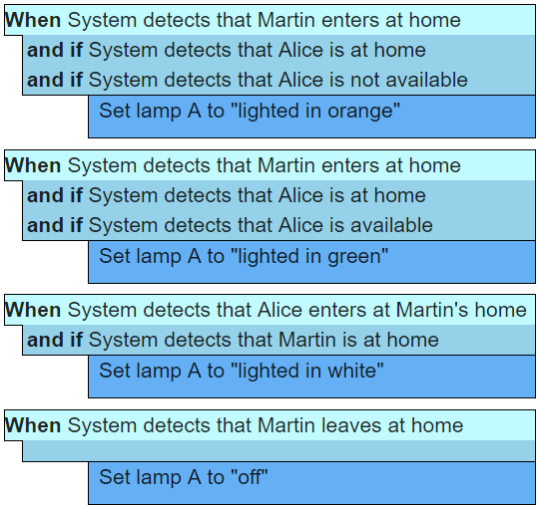

Fig. 9. Example of program where participant missed to express correctly when to turns light off (Miss the case when Alice is not at her home not at Martin's home).

6.6 Analysis of CCBL programs

The experimental results suggest that CCBL provides a threshold as low as that of ECA for simple problems. There was no significant difference observed between programming with ECA and with CCBL. This suggests that the CCBL cascade mechanism and the notion of sub-context do not introduce difficulties for the user that could lead to a higher threshold than for ECA.

Only two non-programmers made an error when using CCBL. Both did it at exercise 4. The first one forgot to mention that Martin had to be at home to light the lamp in orange, green or white. The second non-programmer expressed a very different behavior than what we asked him to do. We suspect that he was tired of the exercises and wanted to finish as soon as possible (it was his last exercise and he answered very quickly to the ending questions).

We analyze the structure of CCBL programs proposed by participant with respect to the errors made when they used ECA (with or without the TAP semantics). A source of errors when programming with ECA is non-sequitur reasoning, that is to say that user expect the system to do some reverse action when a context ends. In CCBL, there is always an active context, and the set of active contexts specifies the state of all devices. Consequently, users can always know what would be the state of a device when a context ends. Indeed, participants said that they preferred CCBL because "it is not necessary to think about reverse actions as it is with ECA".

For exercise 4, 8/11 programmer and 4/10 non-programmers did use the nesting of context to specify the default case (Figure 13-A). Some of them had an unnecessary context as children of the root context, "Martin is not at home", and associated it with "Set lamp A to off". This was done by 2/11 programmers and 3/10 non-programmers. At the end of the session, when we asked why they added this context, participants provided two explanations: 1) they forgot about the root context and 2) They added this "just to be sure" and because the instructions explicitly stated that when Martin was not at home then the lamp should be off.

Another source of error when programming with ECA and the TAP semantics was the use of general case / particular case reasoning. Participant who made the error illustrated in Figure 8 adopted the same mode of reasoning when programming in CCBL. Figure 10 illustrates how it can be coded using CCBL. This solution was proposed by 6/11 programmers and 5/10 programmers. Other participants add an additional unnecessary context "Martin is not at home" (4/AA programmers and 4/10 non-programmers) for the same reason that mentioned before. Last, one programmer and one non-programmers implemented the behavior with a decision tree (see Figure 11), which is a valid solution despite it does not use the cascade mechanism.

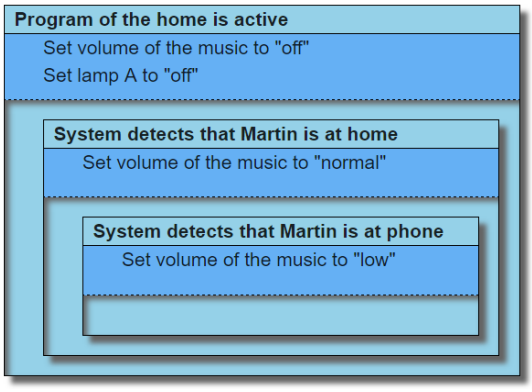

Fig. 10. Solution to exercise 3 with CCBL a general case / particular case strategy. The priority list mechanism ensures that only the most nested active sub-context can set the music volume.

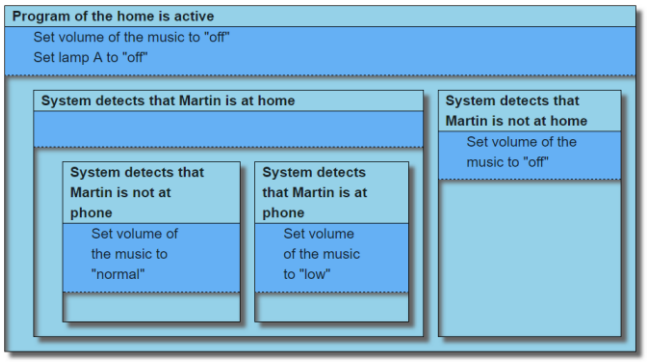

Fig. 11. Solution to exercise 3 with CCBL and a decision tree strategy. Only the root and the leaf contexts specify actions.

CCBL is quite tolerant in the ways users can structure programs. Figure 13 schematizes the three different types of correct solutions that both programmers and non-programmers proposed for solving exercise 4. We did not include screenshots due to lack of space and readability problems.

Figure 13-α is the optimal solution; it relies heavily on cascade mechanism and factorization of contexts. Figure 13-β is a quite straightforward translation in CCBL of the exercise 4 statement (Table 2 Ex 4), it can be read in a similar way than the ECA solution expressed in Figure 9 (with TAP semantics). On the contrary, the participant was able to concentrate on the contexts for which the lamp A had to be lit in orange, green or white. For all other contexts, the lamp is off as specified in the root context. An interesting insight is what the participant said at the end of the session: "There was the problem of specifying the invert actions, I only thought about it when programming with ECA but I would not be able to explain why".

Finally, the solution illustrated in Figure 13-γ follow a decision tree strategy elaborated by a programmer. This participant first tried to solve exercise 4 using ECA, he drew for that purpose the schema reproduced in Figure 12. Despite this schema, he forgot some rules in his ECA programs. However, it is interesting to observe that this schema is close to the structure of the CCBL program of Figure 13-C, all the more since it was written before the participant knew about CCBL. Actually, the same participant (a non-programmer) wrote these ECA and CCBL programs. However, the ECA program misses one rule ("When Alice is not as home and not at Martin's home then turn off lamp A").

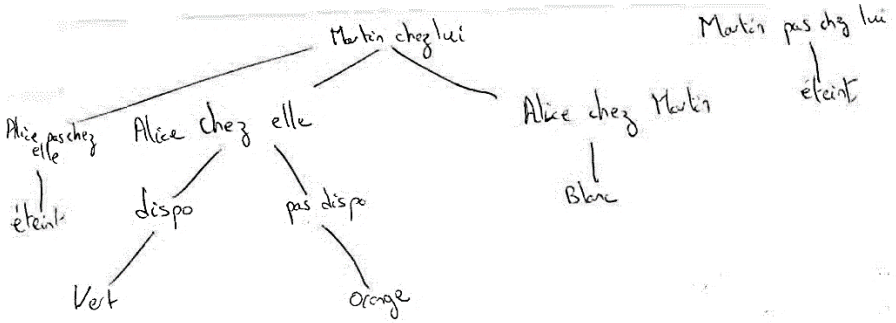

Fig. 12. Schema drawn by one programmer for exercise 4 in ECA. The two roots are "Martin is at home" and "Martin is not at home". Children of the first root are "Alice is not at home", "Alice is at home" and "Alice is at Martin's home".

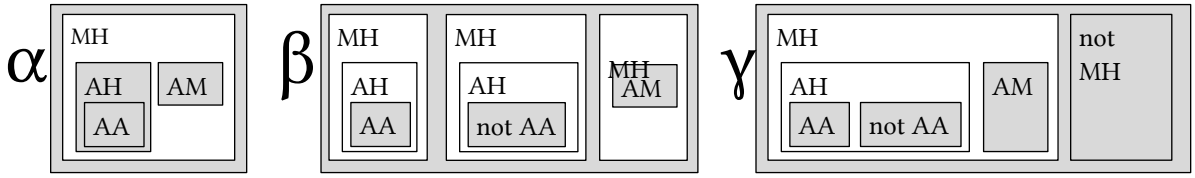

Fig. 13. Solutions to exercise 4 with CCBL. Contexts with grey background specify the color of the lamp. MH stands for "Martin is at home". AH stands for "Alice is at home". AM stands for "Alice is at Martin's home". AA stands for "Alice is available". Strategy α rely on general/particular case. Strategy β is close to TAP. Strategy γ implements on a decision tree.

6.7 Users' point of view

The software used for the experiment was instrumented to measure the editing time. Participants were faster using CCBL than ECA (average time in seconds for CCBL/ECA: ex 1: 81/142, ex 2: 105/174, ex 3: 46/209, ex 4: 240/494). CCBL was preferred by all programmers and 7/10 non-programmers. It seems that CCBL better fit the way of reasoning for the problems we tested. Participants stated: "CCBL is better because you do not have to care about counter-order", "Having a default case is much more simpler, I just have to concentrate on the case where it has to be changed", "CCBL give me the impression that it is going to take care of more things without I have to think about it. With ECA I always fear to miss something".

Participants also appreciated CCBL for being able to provide a good overview of the behavior: "with CCBL, things are better classified, with ECA all is at the same level so it is difficult to get an overview"; "with CCBL you see which conditions are more fundamental, such as Martin being at home. You don't have to repeat yourself as with ECA".

Participants perceived CCBL to be a higher level programming language than ECA. One programmer said, "It is like comparing languages with and without garbage collector, there are things you just do not want to bother with". One other mentioned that he "needed to concentrate much more when programming with ECA. You have to be really methodic if you don't want to forget a case".

Three non-programmers preferred programming with ECA because "it is more linear, more sequential, you get one sentence after each other's", "CCBL programs are harder to read because of the nested boxes, especially when programming exercise 4". This claims for a redesign of the user interface. However, in spite of this comments, these three participants succeeded in programming correctly the four exercises using CCBL and failed to program exercises 3 and 4 with ECA. One possible explanation is that they had a false impression of easiness of ECA; they did not perceive the difficulties of the behaviors to program.

8 LIMITS OF THE STUDY

Our evaluation is a first step on testing CCBL. Indeed, we tested only a category of programs: those centered on states. We need to conduct further user studies to evaluate how CCBL could support other categories of programs (e.g., more centered on events) with respect to ECA. Related to that point, we cannot not draw conclusions on comparing CCBL and TAP, although the results may suggest an advantage for CCBL. This requires further confirmation by conducting a comparative study of CCBL and TAP.

The exercises are realistic of home programming but we formulated them in a way that favors a state-based solution. We chose to do so because we targeted state-based programs in this user study. This could explain why users tended to interpret ECA rules with the TAP semantics (thus confounding events and states). This may have an impact on their performances. We envision a complementary user study with more neutral or balanced formulations (e.g., favoring either state-based or event-based solutions).

Finally, participants could not evaluate the correctness of their programs (except for the syntax). The number of incorrect ECA programs should decrease with the capacity of correcting the programs, although testing a program may also be a difficult task for participants. As a result, this user study is dedicated to the

very first steps of the programming process (e.g., we do not consider debugging). We assume that that the quicker the program is correctly specified, better it is for users.

9 CONCLUSION

In this paper, we introduce CCBL (Cascading Contexts Based Language), a new language dedicated for programming the Smart Home. We built CCBL around the notion of context rather than events as it is for ECA (Event Conditions Actions). CCBL support users' way of reasoning better than ECA by facilitating non-sequitur reasoning and general case / particular case reasoning.

We set up an experiment with eleven programmers and ten non-programmers. We asked them to program four Smart Home behaviors of increasing complexities that associate devices' states to context. Results show that both programmers and non-programmers do significantly fewer errors when programming with CCBL than when programming with ECA. Every participant was able to program correctly the four behaviors except two non-programmers that make errors for the exercise 4.

Results of our experiment also show that users, especially non-programmers, tend to blur the distinction between event and condition when programming with ECA. We observe that users tend to turn from the actual semantics of ECA, where rules are triggered by events, to the semantics of Triggers-Actions Programming (TAP). While TAP enables users to program rule of the form "if conditions than actions", our experiment show that users tend to forget to specify rules to reverse actions when behaviors are complex. We also show that users tend to apply a general case / particular case reasoning although neither TAP nor ECA actually support it, on the contrary of CCBL.

REFERENCES

[1] A.J. Bernheim Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu, and Colin Dixon. 2011. Home automation in the wild: challenges and opportunities. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11). ACM, New York, NY, USA, 2115-2124.

[2] Julio Cano, Gwenaël Delaval, Eric Rutten. Coordination of ECA rules by verification and control. 16th International Conference on Coordination Models and Languages, Jun 2014, Berlin, Germany. 16 p., 2014.

[3] Scott Davidoff, Min Kyung Lee, Charles Yiu, John Zimmerman, and Anind K. Dey. 2006. Principles of smart home control. In Proceedings of the 8th international conference on Ubiquitous Computing (UbiComp'06), Paul Dourish and Adrian Friday (Eds.). Springer-Verlag, Berlin, Heidelberg, 19-34.

[4] Alexandre Demeure, Sybille Caffiau, Elias Elias, Camille Roux. Building and Using Home Automation Systems: A Field Study. ISEUD 2015, May 2015, Madrid, Spain. 2015.

[5] Anind K. Dey , Timothy Sohn , Sara Streng , Justin Kodama, iCAP: interactive prototyping of context-aware applications, Proceedings of the 4th international conference on Pervasive Computing, p.254-271, May 07-10, 2006, Dublin, Ireland.

[6] García-Herranz, M., Haya, P., and Alamn, X. Towards a ubiquitous end-user programming system for smart spaces. Journal of Universal Computer Science 16, 12 (2010), 1633--1649.

[7] Jonna Häkkilä , Panu Korpipää , Sami Ronkainen , Urpo Tuomela, Interaction and end-user programming with a context-aware mobile application, Proceedings of the 2005 IFIP TC13 international conference on Human-Computer Interaction, September 12-16, 2005, Rome, Italy

[8] Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '15). ACM, New York, NY, USA, 215-225.

[9] Andrew J. Ko , Brad A. Myers , Htet Htet Aung, Six Learning Barriers in End-User Programming Systems, Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing, p.199-206, September 26-29, 2004

[10] Håkon Wium Lie, Cascading Style Sheets, Thesis submitted for the degree of Doctor Philosophiœ, Faculty of Mathematics and Natural Sciences, University of Oslo, Norway, 2005.

[11] Sarah Mennicken, Jo Vermeulen, and Elaine M. Huang. 2014. From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '14). ACM, New York, NY, USA, 105-115.

[12] Chandrakana Nandi and Michael D. Ernst. 2016. Automatic Trigger Generation for Rule-based Smart Homes. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (PLAS '16). ACM, New York, NY, USA, 97-102.

[13] John F. Pane, Chotirat "Ann" Ratanamahatana, and Brad A. Myers. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Hum.-Comput. Stud.* 54, 2 (February 2001), 237-264.

[14] Paschke A: ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics, IBIS, Technische Universität München, Technical Report 11 / 2005.

[15] Lénaïc Terrier, Alexandre Demeure and Sybille Caffiau: CCBL: A new language for End User Development in the Smart Homes. 6 pages short paper to appear in IS-EUD 2017, WORK-IN-PROGRESS category.

[16] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14). ACM, New York, NY, USA, 803-812.